南 阳 理 工 学 院

本科生毕业设计(论文)

学院(系)： 计算机与软件学院

专 业： 软件工程

学 生： 武扬岚

指导教师： 张政

完成日期 2024 年 04 月

南阳理工学院本科生毕业设计（论文）

基于SpringCloud和Kubernetes的短链接SaaS平台的设计与实现

**Design and implementation of short-link SaaS platform based on SpringCloud and Kubernetes**

总 计：毕业设计（论文）47页

表 格： 20个

插 图： 29副

南 阳 理 工 学 院 本 科 毕 业 设 计（论文）

基于SpringCloud和Kubernetes的短链接SaaS平台的设计与实现

Design and implementation of short-link SaaS platform based on SpringCloud and Kubernetes

学 院(部)： 计算机与软件学院

专 业： 软件工程

学 生 姓 名： 武扬岚

学 号： 2015105456

指导教师(职称)： 张政 （讲师）

评阅教师(职称)：

完 成 日 期： 2024年04月

南阳理工学院

Nanyang Institute of Technolog

基于SpringCloud和Kubernetes的短链接SaaS平台的设计与实现

软件工程 武扬岚

1. 随着互联网的高速发展，现在Web站点越来越多，并这些站点的Url通常都较长，这就造就了一些问题，如：Url较长不方便用户复制而造成体验感差，点击率较低；在进行推销产品时，由于短信、推文或博文发送推广广告时字数的限制，不利于产品的推广和营销；原始的Url不具备链接追踪分析的能力；原始的Url将原始的域名直接暴露出了，存在被推广渠道方封禁的风险等。所以，为了解决这些痛点变开发了短链接SaaS平台。该系统采用Mysql数据库进行数据持久化存储操作，采用Redis数据库进行数据缓存、分布式锁、消息队列等，并通过SpringCloud的服务注册与发现、负载均衡、断路器等特性进而去构建弹性伸缩的微服务架构，同时利用Kubernetes的容器编排、水平扩展、自动化部署能力、自我修复能力，构建弹性的部署机制，保证平台在面对高负载时能够自动扩展，提供稳定可靠的服务。
2. 短链接SaaS平台；SpringCloud；Kubernetes；微服务

Design and implementation of short-link SaaS platform based on SpringCloud and Kubernetes

**Software Engineering Major WU Yang-Lan**

1. With the rapid development of the Internet, there are more and more Web sites now, and the URLs of these sites are usually longer, which creates some problems, such as: the URLs are long and inconvenient for users to copy and cause poor experience, and the click rate is low; When promoting products, due to the limitation of the number of words when sending promotional advertisements in text messages, tweets or blog posts, it is not conducive to the promotion and marketing of products; The original URL doesn't have the ability to analyze link tracking; The original URL directly exposes the original domain name, and there is a risk of being banned by the promotion channel. Therefore, in order to solve these pain points, a short-link SaaS platform was developed. The system uses Mysql database for data persistent storage operations, Redis database for data caching, distributed locks, message queues, etc., and builds an elastic scaling microservice architecture through SpringCloud's service registration and discovery, load balancing, circuit breaker and other features, and at the same time uses Kubernetes' container orchestration, horizontal scaling, automatic deployment capabilities, and self-healing capabilities to build a flexible deployment mechanism. Ensure that the platform can automatically scale in the face of high loads and provide stable and reliable services.
2. short link SaaS platform; SpringCloud;Kubernetes; Microservices

目录

[1 绪论 1](#_Toc504035385)

[1.1 课题研究背景及意义 1](#_Toc1647211927)

[1.2 国内外研究现状分析 1](#_Toc1479163612)

[1.3 课题主要内容和目标 2](#_Toc1032129212)

[1.4 论文组织结构 2](#_Toc1770249265)

[2 相关技术介绍 4](#_Toc1340951317)

[2.1 后端框架介绍 4](#_Toc1675393201)

[2.1.1 SpringCloud框架介绍 4](#_Toc527949743)

[2.1.2 Mybatis-Plus框架介绍 4](#_Toc1996384844)

[2.2.3 ShardingSphere介绍 4](#_Toc955572380)

[2.2.4 Quartz介绍 5](#_Toc1422278394)

[2.2 前端框架介绍 5](#_Toc592493201)

[2.2.1 Vue框架介绍 5](#_Toc151558068)

[2.3 中间件介绍 5](#_Toc320843534)

[2.3.1 MySQL数据库介绍 5](#_Toc85838321)

[2.3.2 Redis数据库介绍 5](#_Toc1723133910)

[2.3.3 Nginx介绍 6](#_Toc1894645575)

[2.4 运维部署相关介绍 6](#_Toc420661309)

[2.4.1 Docker 容器化技术介绍 6](#_Toc538454439)

[2.4.2 Kubernetes 容器管理平台介绍 6](#_Toc307667815)

[3 需求分析 8](#_Toc1979828376)

[3.1 可行性分析 8](#_Toc1863888814)

[3.1.1 技术可行性 8](#_Toc1035338109)

[3.1.2 经济可行性 8](#_Toc2015089969)

[3.1.3 社会可行性 8](#_Toc1799995793)

[3.1.4 操作可行性 9](#_Toc927157662)

[3.2 非功能性需求分析 9](#_Toc597482602)

[3.2.1 系统性能 9](#_Toc256558442)

[3.2.2 系统可靠性 9](#_Toc1978055165)

[3.2.3 系统安全性 9](#_Toc2126302595)

[3.3 功能性需求分析 10](#_Toc492344438)

[3.3.1 主要功能模块分析 10](#_Toc578477575)

[3.3.2 核心用例分析 10](#_Toc814133056)

[3.3.3 核心用例描述 11](#_Toc1515957155)

[4 系统设计 15](#_Toc945916077)

[4.1系统概要设计 15](#_Toc190067398)

[4.1.1系统架构设计 15](#_Toc1154575097)

[4.1.2系统功能模块设计 15](#_Toc281420987)

[4.2系统详细设计 16](#_Toc1083537815)

[4.2.1创建短链接流程设计 17](#_Toc358730145)

[4.2.2编辑短链接流程设计 18](#_Toc1190949886)

[4.2.3短链接跳转流程设计 18](#_Toc1747143962)

[4.3系统数据库设计 19](#_Toc1704663903)

[4.3.1 数据库概念结构设计 19](#_Toc706883094)

[4.3.2 数据库逻辑结构设计 22](#_Toc704625654)

[4.4系统部署上线设计 27](#_Toc1418537220)

[5 系统实现 28](#_Toc2139091193)

[5.1系统部署实现 28](#_Toc681946324)

[5.1.1系统多环境打包 28](#_Toc351643429)

[5.1.2系统镜像打包及上传 29](#_Toc196114659)

[5.1.3系统上线部署 30](#_Toc1859159315)

[5.2系统功能实现 32](#_Toc1003543355)

[5.2.1系统登陆注册功能实现 32](#_Toc216603947)

[5.2.2系统短链接生成功能实现 34](#_Toc477755564)

[5.2.3系统短链接跳转功能实现 36](#_Toc196408015)

[5.2.4系统短链接统计功能实现 38](#_Toc347142666)

[5.3系统非功能实现 39](#_Toc1861202210)

[5.3.1系统日志和链路追踪功能实现 39](#_Toc978741268)

[6 系统测试 42](#_Toc2127238903)

[6.1测试简述 42](#_Toc1196487465)

[6.2功能测试 42](#_Toc327953747)

[6.2.1登录功能测试 42](#_Toc1475587627)

[6.2.2短链接创建功能测试 42](#_Toc1060091433)

[6.2.3短链接跳转功能测试 43](#_Toc1432378919)

[6.3非功能测试 43](#_Toc700808763)

[6.4测试总结 43](#_Toc351581042)

[结束语 44](#_Toc1295059997)

[参考文献 45](#_Toc1326607234)

[致谢 47](#_Toc1112558684)

# 

# 绪论

## 课题研究背景及意义

近些年来，随着互联网的高速发展，互联网现在已经深深的融入到我们的生活之中，并成为生活的一部分，为我们的生活提供便利，但这也带来了一系列的问题。如当一个C端用户在分享一个链接的时候，往往系统生成的链接都较长，C端用户难以复制去浏览造成体验感较差；当B端用户在推广自己的产品的时候，由于短信和推文发送的内容字数受限，这就要将链接尽可能变短，同时长链接的用户点击率、转化率低，不利于产品的营销和推广，并且原始链接会暴露服务的真实域名，容易被渠道方封禁；同时原始链接不具备用户点击信息的追踪和统计功能，难以去了解用户偏好。

现代社交媒体和网站普遍使用短链接来缩短并共享长链接。然而，大规模使用短链接也带来了一些挑战，如海量生成、海量存储、海量跳转、管理和维护成千上万个短链接等，这些任务变得较为困难。

为了应对这些挑战，基于SpringCloud和Kubernetes的短链接SaaS平台的设计与实现成了一个有意义的研究方向。将SpringCloud和Kubernetes结合起来，可以为短链接SaaS平台提供可靠的架构。SpringCloud作为基础架构用于实现平台的微服务化[1]，其包括服务注册与发现、负载均衡、远程调用等功能。Kubernetes则可以用于容器的编排和自动化部署[2]，可以提高系统的弹性和可靠性，降低维护成本，同时也能更好地支持平台的横向扩展和高可用性。

因此，基于SpringCloud和Kubernetes的短链接SaaS平台的设计与实现对于提升系统性能和可靠性具有重要意义。

## 国内外研究现状分析

随着互联网的高速发展，短链接平台[3]现在似乎已经成为一个必不可少的平台，所以国内外对短链接平台的研究也越来越多。在国内外已经有很多个成熟的短链接平台，如百度短网址、新浪短网址、腾讯短链接、bitly、tinyurl、Twitter短链接等。但在这些短链接平台或多或少都会存在一些不足，有些只供自己平台内部分享、跳转使用，有些则是需要进行完全付费才能够使用，有些则是部分高级功能进行收费，有些则是会加入一定量的广告而进行盈利，有些则是在短链接跳转时存在一定延迟等。但有一点是肯定的，这些平台大都采用了单体架构或前后端分离架构进行开发，在项目开发和上线发版的时候不够灵活，难以应对随时可能变更的新需求。这些平台的服务在部署上线的时并没有采用CI/CD流水线方式进行部署，运维人员对服务的部署和维护成本较高。同时在流量低谷和高峰期时难以去动态地对服务进行阔缩容，难以应对不同时期的流量请求。

总的来说，国内外的短链接平台都在致力于去构建一个完美的平台，现如今基于Spring Cloud和Kubernetes的短链接SaaS平台的设计与实现是一个具有挑战性和前景的研究方向。通过合理的架构设计和技术选择，可以实现高性能、可伸缩和可靠的短链接服务。旨在为用户提供高效、稳定和灵活的短链接平台，以满足广大使用者的需求。

## 课题主要内容和目标

本课题旨在使用SpringCloud和Kubernetes构建一个多模块、弹性伸缩、快速开发和部署、可靠高效的短链接平台。

其主要功能有平台用户管理、短链接生成、短链接跳转、短链接存储、短链接删除和失效、短链接访问监控、短链接分组管理、短链接回收站管理、短链接白名单列表、接口流量风控处理、接口限流处理、接口幂等性处理、分布式调用链路日志处理、网关日志收集等。并采用ShardingSphere分库分表[4]进行海量数据存储，Redis缓存进行减轻数据库的压力，Redisson分布式锁保证分布式应用线程安全[5]，Redis的Sream消息队列进行异步和流量削峰等。

该平台采用Maven多环境打包部署方式，分为Dev环境、Docker部署环境、Kubernetes部署环境，每个环境都有自己独立的配置文件。在使用Docker或Kubernetes部署时需要编写前端服务和后端多模块服务对应的Dockerfile，并将镜像上传至DockerHub镜像仓库。在使用Kubernetes部署时，需要编写对应的部署脚本文件，并且部署一台负载均衡服务器用于将请求流量分配到Kubernetes集群的不同节点上。

该课题的实验设备为8C16G的MacBook-Air一台，2C4G的腾讯云服务器一台。Dev环境为本地Docker启动的Mysql、Redis、Nacos中间件，Docker部署环境为腾讯云2C4G云服务器，Kubernetes环境为本地VMware[6]构建的3台CentOS虚拟机搭建的Kubernetes集群。

## 1.4 论文组织结构

本文分为六章，每章的主要内容介绍如下：

第1章为绪论章节，阐述了本系统的开发背景及开发意义。然后分析了国内外短链接平台的发展现状以及如应如何去进行改进，最后详细阐述了整个系统的主要功能、主要难点、部署方式、课题研究的设备情况。

第2章介绍系统相关技术，本章节重点阐述这些技术的背景、特点以及它们所具备的优势。

第3章介绍系统分析，本章节首先对整个系统进行可行性分析，然后分析非功能行需求和功能性需求，并绘制用例图和编写用例描述。

第4章介绍系统设计，本章节首先对系统的整体架构进行设计，然后按照功能进行模块拆分，并对一些核心功能进行流程图设计，接着对系统进行实体类的分析并设计相应的表结构，最后对系统的部署上线做相应设计。

第5章介绍系统的实现，本章节首先对系统的部署做了说明，之后分别从功能实现和非功能实现上做了说明。

第6章介绍系统测试，本章首先对系统测试做简要介绍，然后分别从功能测试和非功能测试对系统进行测试，最后总结了本次测试。

# 相关技术介绍

## 后端框架介绍

### 2.1.1 SpringCloud框架介绍

Spring Cloud[7]是一个基于Spring Boot框架用于构建分布式系统的开源框架，它提供了一系列工具和组件，用来帮助开发者快速构建、管理和部署分布式系统的应用。它提供了全套的分布式系统解决方案，包括服务注册与发现、配置中心、服务网关、智能路由、负载均衡、断路器、监控跟踪、分布式消息队列、消息总线等。用其构建的应用架构图大体如图2-1所示。



图2-1 应用架构图

其作为Spring生态系统的一部分，SpringCloud在企业级分布式系统中得到广泛的应用，并且持续得到开发者社区的支持和更新，为企业级分布式系统的发展提供了坚实基础。随着这些年微服务架构的持续普及，SpringCloud作为企业级微服务架构的解决方案，将继续发挥重要作用，并为企业提供可靠的技术支持和解决方案。

2.1.2 Mybatis-Plus框架介绍

MyBatis-Plus是一个基于MyBatis的增强工具包[8]，完全兼容Mybatis，旨在简化Web应用程序的开发过程。它在MyBatis的基础上进行了一些扩展，为MyBatis提供了一些高效、实用、开箱即用的功能，使用它可以有效地节省我们开发人员的开发时间。

MyBatis-Plus提供了一套开箱即用的数据库操作接口极大的简化了单表的CRUD操作，并支持Lambda查询、条件构造器、代码生成器、自动分页操作、SQL注入防护等操作使开发更加便捷高效。它还提供了丰富的插件扩展接口，可以通过插件扩展实现自定义功能。

总之，MyBatis-Plus是一种强大而灵活的持久化框架，通过简化数据访问层的开发，为项目开发提供了便利。此外，它作为一个开源工具，拥有庞大的社区支持和持续的更新，保证了其与时俱进的能力。在企业级的Java应用开发中被广泛应用，使我们更加高效地使用MyBatis，更加高效的开发应用程序。

### 2.2.3 ShardingSphere介绍

ShardingSphere是一个开源的分布式数据库中间件解决方案，它提供了数据分片、读写分离、弹性伸缩、灵活配置、多数据源、数据加密、分布式事务、分布式治理等能力，可以对MySQL、PostgreSQL、SQLServer、Oracle等数据库进行增强。其设计目标是帮助开发者在高并发场景下充分发挥大型数据库的潜力，同时解决单机数据库在存储和计算方面的瓶颈，规避因架构变化带来的技术风险。

ShardingSphere的内核可插拔化多种功能，例如加密、分片都可灵活组合使用，这使得它能够兼容各种关系型数据库。

总的来说，ShardingSphere 是一个强大的分布式数据库解决方案[9]，它可以帮助开发者构建高性能、安全的分布式数据库系统。

### 2.2.4 Quartz介绍

Quartz是一个强大且开源的作业调度库，它允许开发人员在 Java 应用程序中创建和管理定时任务[10]。它支持简单或复杂的作业调度，其特性有灵活的时间表达式、支持任务优先级、支持多种数据源、支持集群部署和故障转移[11]、以及对任务执行结果进行监控和管理等。它提供了一系列的工具和接口，使得开发者可以轻松地创建和管理定时任务，通过使用 Quartz，开发者可以轻松地在Java应用程序中创建和管理定时任务，从而实现更加复杂和灵活的业务逻辑。

## 2.2 前端框架介绍

### 2.2.1 Vue框架介绍

Vue是一个用于构建用户界面的JavaScript框架，它基于标准的HTML、CSS 和 JavaScript，并提供了一种声明式和基于组件的编程模型，帮助开发者高效地开发用户界面。其核心特性有：（1）声明式渲染：Vue 扩展了标准 HTML，提供了一种模板语法，允许我们基于 JavaScript 状态声明式地描述 HTML 输出；（2）响应式：Vue自动跟踪JavaScript状态的变化，并在变化发生时高效地更新 DOM。

Vue有一个强大的生态系统，包括Vue Router和Vuex等辅助库，使得开发个性化的应用程序更加方便。Vue 在开发者社区中备受欢迎，拥有丰富的学习资源和支持，现在在项目开发中大部分应用都采用Vue进行项目开发。

## 2.3 中间件介绍

### 2.3.1 MySQL数据库介绍

MySQL是一个非常流行的开源关系型数据库管理系统（RDBMS），使用结构化查询语言（SQL）进行数据查询。它支持多种操作系统和语言，包括 PHP、PERL、C、C++、JAVA等。在默认情况下，其单表的文件大小限制为4GB，但可以根据操作系统的能力增加到理论上的8000万TB的限制，故表中数据最多可以支持5000万行或更多。并提供多种存储引擎，包括InnoDB、MyISAM和Memory，用户可以根据具体需求场景选择适合的存储引擎来优化数据库性能。

MySQL是一个流行的、经过时间考验的[12]，但同时也是现代的、功能齐全的关系型数据库管理软件。全球的大部分企业都采用它进行数据存储、处理，应用于Web应用程序和大型网站，如Facebook、Twitter、YouTube、淘宝和微博等。

### 2.3.2 Redis数据库介绍

Redis（Remote Dictionary Server）是一个开源的Key-Value内存数据库，它使用C语言编写，并支持网络、持久化机制、事务、发布/订阅模式[13]等，并提供了主从模式、哨兵模式和集群模式三种集群部署模式。

在实际应用中，Redis通常被用作缓存服务器，将热点数据存储在内存中，以提高读写性能和响应速度。与传统的缓存方案相比，Redis具有以下优点：一是由于采用了内存存储方式，Redis具有极高的读写性能和响应速度；二是Redis支持多种数据结构和丰富的操作命令，可以实现各种复杂的应用场景；三是Redis支持多种数据持久化方式，可以将数据异步写入磁盘中，以保证数据的可靠性和持久化；四是Redis支持分布式集群部署，可以支持大规模数据存储和高并发读写操作。因此使用Redis缓存技术在我们的系统中发挥了很重要的作用。

### 2.3.3 Nginx介绍

Nginx是一个由俄罗斯的Igor Sysoev开发的一个开源的、高性能的、可靠的、小型化的、灵活的、易于配置的HTTP服务器和反向代理服务器，它也可以作为一个IMAP/POP3/SMTP代理服务器。它的设计目标是提供一个高性能、稳定、低消耗资源的服务器，同时还要求易于配置和扩展。  
 Nginx的安装和配置相对简单，其配置文件通常位于/etc/nginx/nginx.conf，其中包含了服务器的全局设置和各个服务器块的配置，服务器块定义了如何处理特定的请求，包括监听的端口、服务器名称、根目录、日志文件位置等。

Nginx被广泛用于Web服务器、反向代理、负载均衡、缓存、安全防护等多种场景，它的高级功能和模块化架构使得它非常灵活和强大，是现代Web架构中不可或缺的一部分。

## 2.4 运维部署相关介绍

### 2.4.1 Docker 容器化技术介绍

Docker是一种开源的容器化平台，它允许开发者打包应用程序和所有相关的环境到一个统一的容器中，然后部署到任何支持Docker的环境中。

Docker容器化技术的优势包括：轻量化、速度快、可移植性强、灵活性高、资源利用率高[14]、便于部署等。与虚拟机相比，Docker容器的速度非常快，因为它们轻量级且启动快。它不使用虚拟机管理程序，使用可用资源更有效率，可以在单个主机上运行比虚拟机更多的容器，减少资源浪费。它能将构建的应用程序作为一个单一元素轻松地移动到任何地方，并且它们的性能也保持不变。

总之通过Docker，开发者可以更加方便地构建、打包、交付和运行应用程序，从而提高开发和部署效率。

### 2.4.2 Kubernetes 容器管理平台介绍

Kubernetes是一个开源的容器编排平台，最初由Google开发的Borg系统演变而来[15]，现由Cloud Native Computing Foundation (CNCF)维护，它主要用于管理容器化应用程序在集群环境中的运行和协调。其集群为两个部分：控制平面与计算设备（或称为节点），其整体架构如图2-2所示。

在Kubernetes中有Pod、Service、Ingress、Volume、Namespace、Lable、ReplicaSet、Deployment、StatefulSet、持久存储卷（Persistent Volume，PV）和持久存储卷声明（Persistent Volume Claim，PVC）等相关概念。其中Pod是一组紧密关联的容器的抽象，共享网络和存储空间，提供了容器之间通信和资源共享的环境。另外Service是Kubernetes中的一种抽象，它用于定义一组Pod的访问策略，可以轻松实现负载均衡和服务发现，为应用组件提供了统一的入口和可靠的网络访问机制。

Kubernetes的ReplicaSet功能可确保指定数量的Pod副本在任何时候都在运行，以提高应用的可用性和稳定性。而Deployment则允许用户声明式地管理Pod和ReplicaSet的部署，包括滚动更新、回滚操作等。

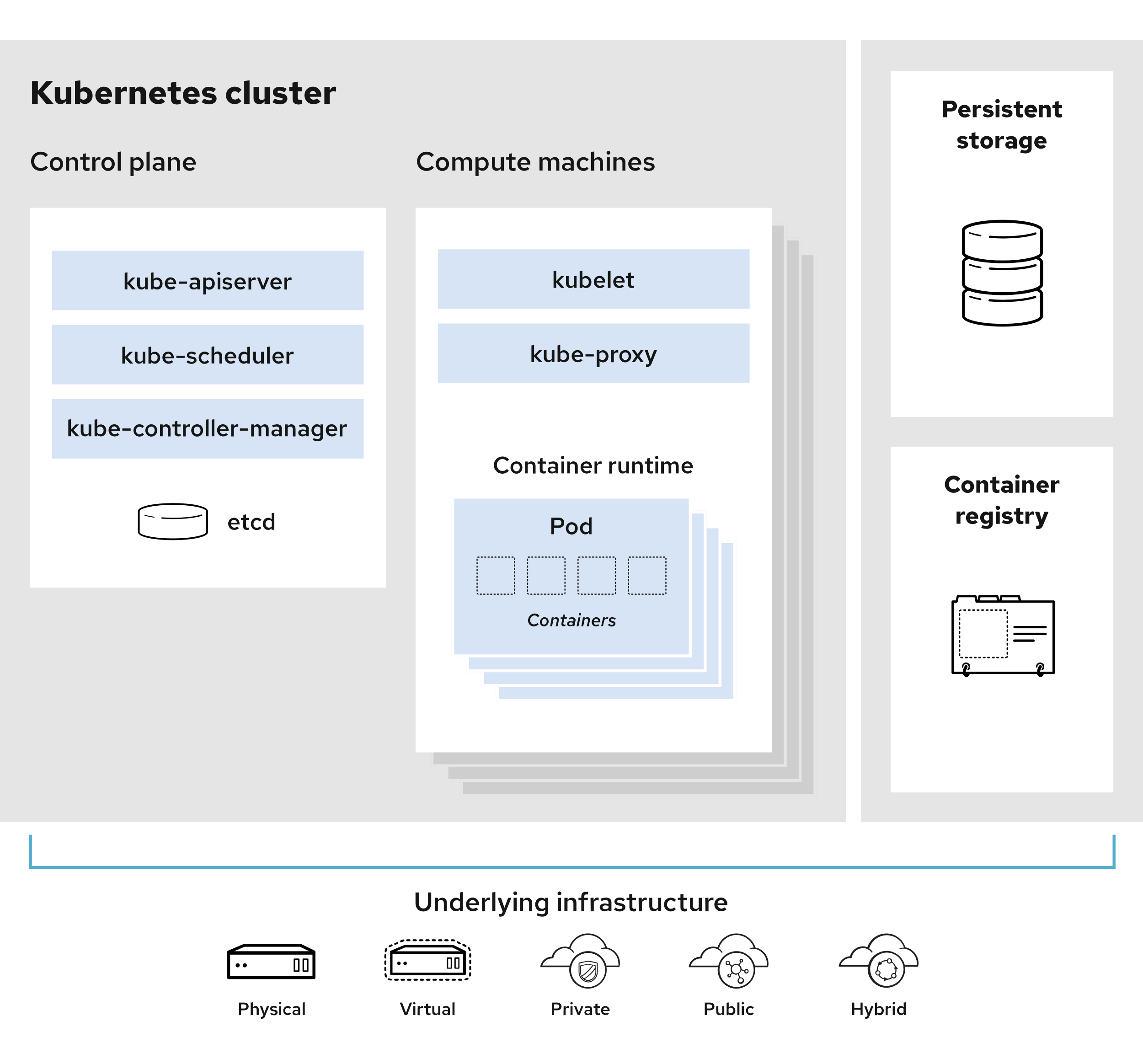


图2-2 整体架构图

我们在部署Kubernetes应用时，只需要将应用的配置文件修改为Kubernetes环境下，并生成相应的Docker镜像，然后为应用编写相应的部署Yml文件，最后执行部署Yml文件，就可以轻松将应用部署在Kubernetes集群中。

Kubernetes是一个强大的容器管理平台，它通过自动化部署、扩展和管理容器化应用程序的过程，大大简化了容器化应用程序的管理。

# 需求分析x

## 3.1 可行性分析

### 3.1.1 技术可行性

SpringCloud和Kubernetes是当下流行的微服务和容器化技术，它们可以提供高可用性和可伸缩性的基础设施。SpringCloud可以用于构建各种微服务，提供服务注册与发现、负载均衡、断路器、网关等全套解决方案。Kubernetes可以用于部署、扩展和管理容器化应用，提供自动化的容器编排、服务发现与负载均衡等功能。Spring Cloud和Kubernetes提供了一套完整的解决方案，使得开发者能够在Kubernetes环境下构建和运行高效、可扩展的SaaS平台。同时Redis性能高效，作为缓存和分布式锁保证分布式系统的正常运行。 Redis、SpringCloud和Kubernetes的社区很稳定，开发者可以从中获得丰富的文档、示例和社区支持，从而更容易地解决在开发过程中遇到的问题。

因此，基于SpringCloud和Kubernetes构建的短链接SaaS平台在技术上是可行的。通过使用这些技术，可以实现高可用性、弹性扩展和灵活部署等优势。

### 3.1.2 经济可行性

本短链接SaaS平台采用都是开源免费的主流技术，包括SpringCloud、Kubernetes、Mysql、Redis、Vue等，不会有额外的支出。短链接平台在开发过程采用本地开发，并不会有任何开销。采用Docker部署的话，只需要一台2C4G的云服务即可，而学生购买云服务器会获得很大的优惠。采用Kubernetes部署的话，使用的是本地虚拟机搭建的Kubernetes集群，不会有任何开销。

同时短链接SaaS平台的市场也正在快速发展，因此短链接平台具备很大的市场前景。在技术的持续发展和更新过程中，平台可以持续地对其进行优化和升级，从而提高用户的体验，从而吸引到更多的用户，从而为企业带来更多的经济效益。

综上所述，该短链接平台具有较低的成本和无限的潜力，具有很高的经济可行性。

### 3.1.3 社会可行性

提供便捷的短链接服务：短链接在现代社会中得到广泛应用，可以方便地分享和传播信息。通过这样的SaaS平台，用户可以快速生成短链接，减少了输入繁琐Url的步骤，提高了用户的使用效率，同时利于产品的营销和推广[16]。

提供数据分析和统计功能：通过对用户生成的短链接进行统计和数据分析，可以帮助用户了解链接的使用情况和效果，并进行优化和改进。同时，平台还提供相关的数据报表和图表等功能，方便用户对数据进行可视化分析和展示。

### 3.1.4 操作可行性

前端页面采用Vue框架，设计美观且简洁，易于用户上手，能够满足用户的使用需求。同时，平台的功能设计也符合用户使用习惯，例如登录注册、短链接管理、短链接回收站管理等功能，均符合用户的期望。在平台开发过程中，也采用了较为常见的UI设计原则和人机交互原则，能够提高用户对平台的认知度和使用效率。因此，该短链接平台在操作可行性方面表现较为良好，有利于用户的使用和推广。

## 3.2 非功能性需求分析

### 3.2.1 系统性能

系统的性能要求是用户极其关注的，良好的性能可以为用户带来舒适的体验感。短链接在生成时，我们并不关心反向解密的难度，更是关注的是哈希的运算速度和冲突概率。而MurmurHash算法是一种非加密型哈希函数，对比与MD5、SHA等对称加密算法，生成的效率要高很多[17]。将注册过的用户名放入Redis的布隆过滤器中，当有新用户需要去注册时直接去查询布隆过滤器即可，避免去查询数据库而造成性能损耗。短链接跳转接口采用Redis进行缓存，同时做了缓存穿透、缓存击穿、缓存雪崩的措施，该接口的QPS可达到10000左右。

### 3.2.2 系统可靠性

系统开发采用SpringCloud微服务架构，并对一些重要接口进行限流处理，并使用熔断器和断路器模式对服务进行容错处理，确保系统在出现异常情况时依然可以正常运行。系统采用Nacos做为注册中心，确保流量请求平均分配到不同的服务实例上，保证系统的负载均衡能力，提高系统的可用性。系统通过Kubernetes进行部署，它的容器编排功能可以在集群中自动创建和管理多个副本，当某个容器出现故障时，其他副本可以接替其工作，从而保证系统的可靠性。该系统有完备的微服务调用链日志，可以及时发现和处理问题，如出现线上故障，可以根据日志进行漏洞修复，确保在24小时之内解决问题。

### 3.2.3 系统安全性

系统安全性的高低，是一个系统能否获得用户依赖的关键。在用户注册时，输入的信息都会保存在数据库中，但如果不加以防备，可能会被不法人士盗取进行牟利，所以这里用ShardingSphere的加密功能对密码、身份证、电话等敏感信息进行加密，同时对页面展示用户数据通过脱敏算法进行脱敏处理，所以相对来说用户信息是安全的。系统通过认证和授权机制限制对系统资源的访问，确保只有经过授权的用户和服务才能使用平台功能。

## 3.3 功能性需求分析

### 3.3.1 主要功能模块分析

本系统的用户有B端用户和C端用户两类群体，这两类用户群体在系统上会有如下操作：

B端用户：

1. 系统的登陆与注册：用户可以使用用户名、真实姓名、手机号、邮箱等进行注册用户，注册之后可以进行系统登陆，之后便可以进行各种操作。

（2）短链接分组管理：用户注册成功会默认创建一个默认短链接分组，同时用户可以创建短链接分组、修改短链接分组、删除短链接分组、对这些短链接分组进行排序等操作。

（3）短链接管理：用户可以在一个分组中创建短链接，也可以对短链接进行批量创建。同时可以对短链接进行修改，如修改短链接分组、短链接失效时间等。

（4）短链接回收站管理：用户可以将短链接移动至回收站，将短链接从回收站删除，将短链接从回收站恢复等。

（5）短链接监控及统计：用户可以对短链接的访问进行监控，如查看短链接用户访问量、访问IP、访问设备等信息。

C端用户：

（1）短链接跳转：用户可以根据获取的短链接进行跳转，并访问原始链接。

### 3.3.2 核心用例分析

本系统中包含B端用户和C端用户两类群体，B端用户主要负责短链接的管理、维护和监控，C端用户则主要关注短链接的跳转功能。针对这两类用户，可以进行用例分析。通过用例分析，可以更好地了解系统的功能和用户的需求，以便进行系统设计和开发。

1. B端用户主要有注册与登陆、短链接分组管理、短链接管理、短链接回收站管理、短链接监控和统计等这些功能。其用例图如图3-1所示。

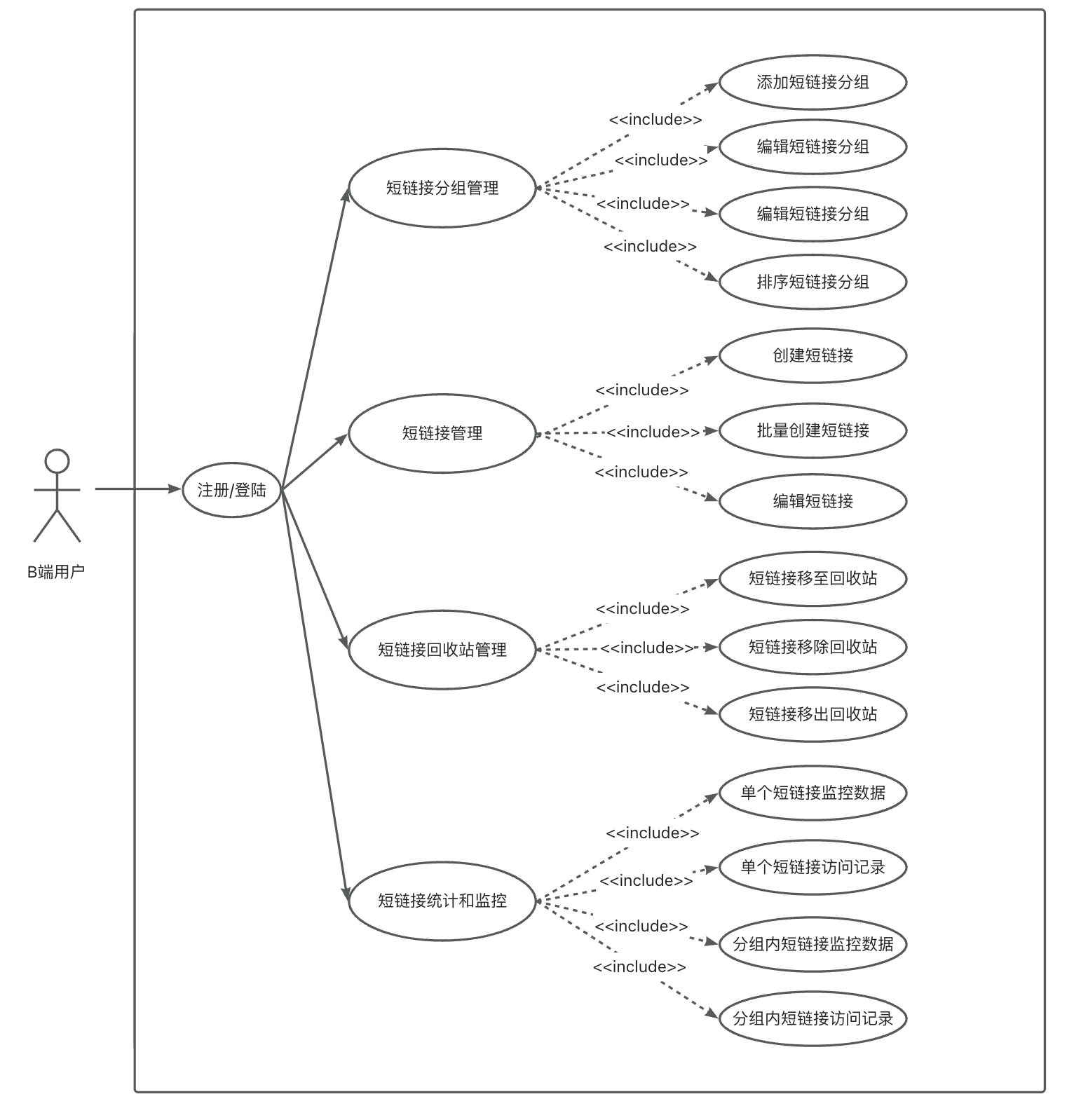


图3–1 B端用户用例图

1. C端用户主要有短链接的跳转功能。其用例图如图3-2所示。

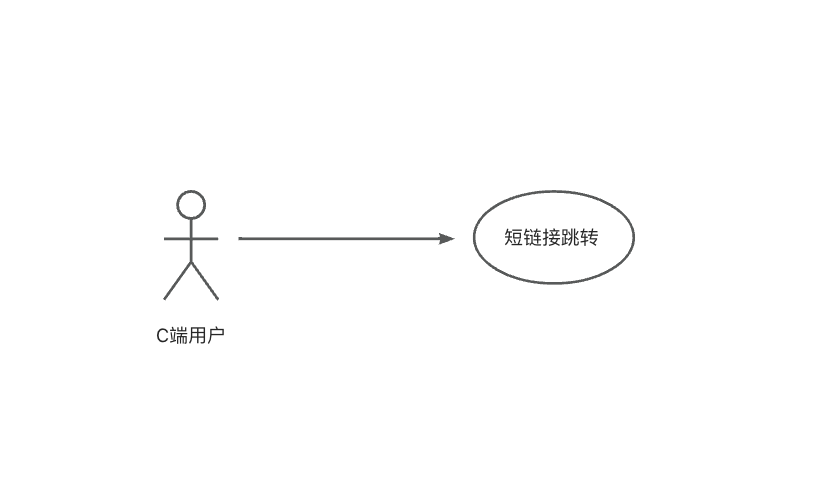


图3–2 C端用户用例图

### 3.3.3 核心用例描述

（1）B端用户注册用例描述

B端用户点击注册进入注册页面，根据注册所需信息进行填写。填写完成之后会有后端验证，如不符合要求则出现错误提示信息，如注册失败等，用户可根据错误提示填写正确格式，注册成功。B端用户注册用例描述如表3-1所示。

表3-1B端用户注册用例描述

| 用例名称 | B端用户注册 |
| --- | --- |
| 描述 | B端用户在注册页进行注册 |
| 参与者 | B端用户 |
| 前置条件 | B端用户需根据注册要求提示信息认真填写 |
| 结果 | 注册是否成功 |
| 基本事件流 | 1.B端用户进入注册页面  2.根据要求输入用户名、密码、真实姓名、邮件、手机号等信息  3.系统对输入的数据进行验证  4.系统根据验证结果返回相应的页面，如果成功则保存并跳往登陆页面，失败则返回到用户注册页面并给出提示信息 |
| 其他事件流 | 无 |
| 异常事件流 | 由于输入信息不符合要求而导致注册失败 |
| 数据格式 | Json字符串 |
| 注释 | 无 |

（2）B端用户创建短链接用例描述

B端用户进入系统后，点击创建短链接按钮，会跳转到创建短链接页面。需要输入要跳转的原始链接，系统后端服务会自动获取该链接的图标和描述信息，同时需要选择要创建的短链接分组，并选择短链接的有效期。创建成功之后，会跳转到短链接展示页面，并展示刚才创建的短链接。B端用户创建短链接用例描述如表3-2所示。

表3–2B端用户创建短链接用例描述

| 用例名称 | B端用户创建短链接 |
| --- | --- |
| 描述 | B端用户在首页点击创建短链接按钮进行短链接的创建 |
| 参与者 | B端用户 |
| 前置条件 | 1.B端用户已经登陆  2.B短用户输入的原始链接真实有效 |
| 结果 | 短链接是否成功创建 |
| 基本事件流 | 1.B端用户登陆进入首页  2.B端用户点击创建短链接按钮  3.根据要求输入原始链接、短链接分组、短链接有效期等信息  4.系统对输入的数据进行验证  5.系统根据用户输入的信息，将原始链接采用哈希算法进行短链接的生成  6.系统将刚生成的短链接展示在短链接列表页面上 |
| 其他事件流 | 无 |
| 异常事件流 | 由于系统采用Hash算法生成的短链接重复次数过多而导致创建失败 |
| 数据格式 | Json字符串 |
| 注释 | 无 |

（3）B端用户编辑短链接用例描述

B端用户进入系统后，在短链接列表里点击某个短链接的齿轮按钮，会跳转到编辑短链接页面。可以对要跳转的原始链接、短链接分组、短链接的有效期等做编辑。编辑成功之后，会跳转到短链接展示页面，并展示刚才编辑过的短链接。B端用户编辑短链接用例描述如表3-3所示。

表3–3B端用户编辑短链接用例描述

| 用例名称 | B端用户编辑短链接 |
| --- | --- |
| 描述 | B端用户可以对已经创建的短链接进行编辑 |
| 参与者 | B端用户 |
| 前置条件 | 1.B端用户已经登陆  2.需要编辑的短链接已经创建 |
| 结果 | 短链接是否编辑生成 |
| 基本事件流 | 1.B端用户登陆进入首页  2.B端用户选择某个短链接并点击编辑按钮  3.根据要求编辑原始链接、短链接分组、短链接有效期等信息  4.系统对输入的数据进行验证  5.系统根据用户输入的信息对短链接进行编辑操作，如删除Redis中的缓存信息、采用分布式读写锁保证用户可以正常访问该短链接并且可以收集到访问信息等  6.系统将刚编辑的短链接展示在短链接列表页面上 |
| 其他事件流 | 无 |
| 异常事件流 | 1.需要编辑的短链接不存在  2.短链接正在被其他C端用户访问，获取读写锁失败 |
| 数据格式 | Json字符串 |
| 注释 | 无 |

（4）B端用户查看短链接访问情况用例描述

B端用户进入系统后，在短链接列表里点击某个短链接的统计按钮，会跳转到短链接统计页面，可以选择要查看日期区间内的统计信息。可以查看短链接的访问历史记录，包含访问时间、访问IP、访问地区、访问设备、访问网络、访客类型等。还可以对短链接访问做统计，如访问曲线、访问Top10省份、高频访问IP、24小时分布、一周分布等。B端用户查看短链接访问情况用例描述如表3-4所示。

表3–4B端用户查看短链接访问情况用例描述

| 用例名称 | B端用户查看短链接访问情况 |
| --- | --- |
| 描述 | B端用户可以对已创建的短链接查看其访问情况 |
| 参与者 | B端用户 |
| 前置条件 | 1.B端用户已经登陆  2.该短链接已经被C端用户访问 |
| 结果 | 短链接访问情况的信息展示 |
| 基本事件流 | 1.B端用户登陆进入首页  2.B端用户选择某个短链接并点击统计按钮  3.系统查询数据库，对统计数据做聚合统计计算  4.系统将短链接的统计情况展示在短链接统计页面上 |
| 其他事件流 | 无 |
| 异常事件流 | 无 |
| 数据格式 | Json字符串 |
| 注释 | 无 |

（5）C端用户点击短链接跳转用例描述

C端用户可以点击短链接进行原始链接的跳转，系统会对访问信息做采集工作。C端用户点击短链接跳转用例描述如表3-5所示。

表3–5C端用户点击短链接跳转用例描述

| 用例名称 | C端用户点击短链接跳转 |
| --- | --- |
| 描述 | C端用户可以点击短链接进行原始链接的跳转 |
| 参与者 | C端用户 |
| 前置条件 | 无 |
| 结果 | 1.跳转到原始链接  2.短链接失效或要跳转的原始链接不存在 |
| 基本事件流 | 1.C端用户点击短链接  2.后端服务对请求进行处理并做追踪和统计功能  3.重定向到原始链接 |
| 其他事件流 | 无 |
| 异常事件流 | 1.短链接失效跳转到404页面  2.短链接不存在跳转到404页面 |
| 数据格式 | 无 |
| 注释 | 无 |

# 系统设计

## 4.1系统概要设计

### 4.1.1系统架构设计

系统整体采用B/S架构。后端服务整体采用微服务架构进行设计，系统被拆分成Admin（后台管理）模块、Project（中台）模块[18]、Task（定时任务）模块、Gateway（网关）模块。这些模块在编写时，遵循Web的三层MVC设计模式。同时这些服务都会被注册进注册中心Nacos中，服务之间的调用采用OpenFegin进行远程调用，并对外暴露Gateway模块，提供统一的API接口。前端采用Vue框架进行项目构建，在服务部署时将代码打包进Nginx服务器中，提供反向代理和页面展示功能[19]。其系统整体架构图如图4-1所示。

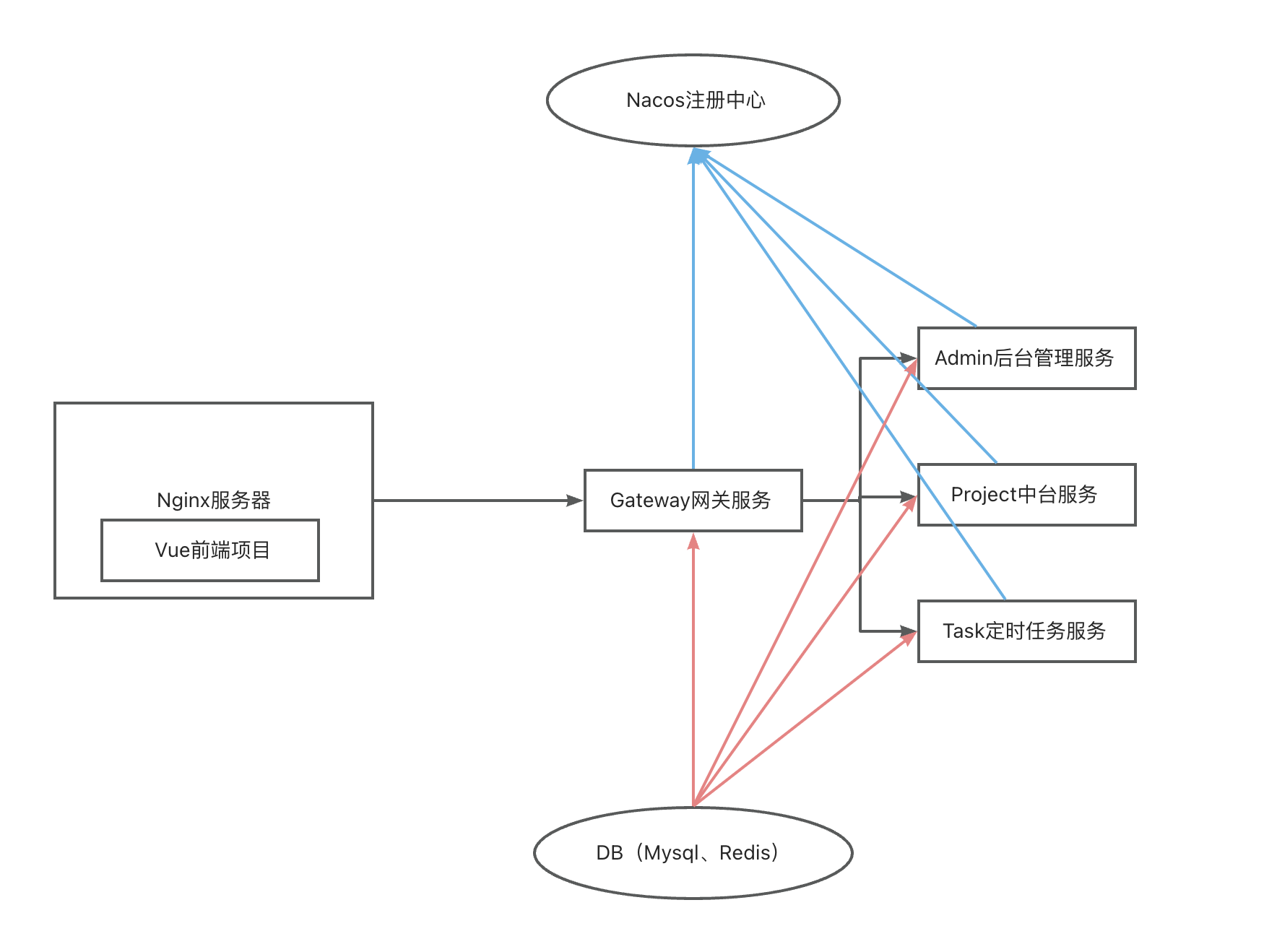


图4-1系统整体架构图

### 4.1.2系统功能模块设计

对短链接SaaS平台进行整体功能分析之后，可以设计出该系统的功能模块图如图4-2所示。

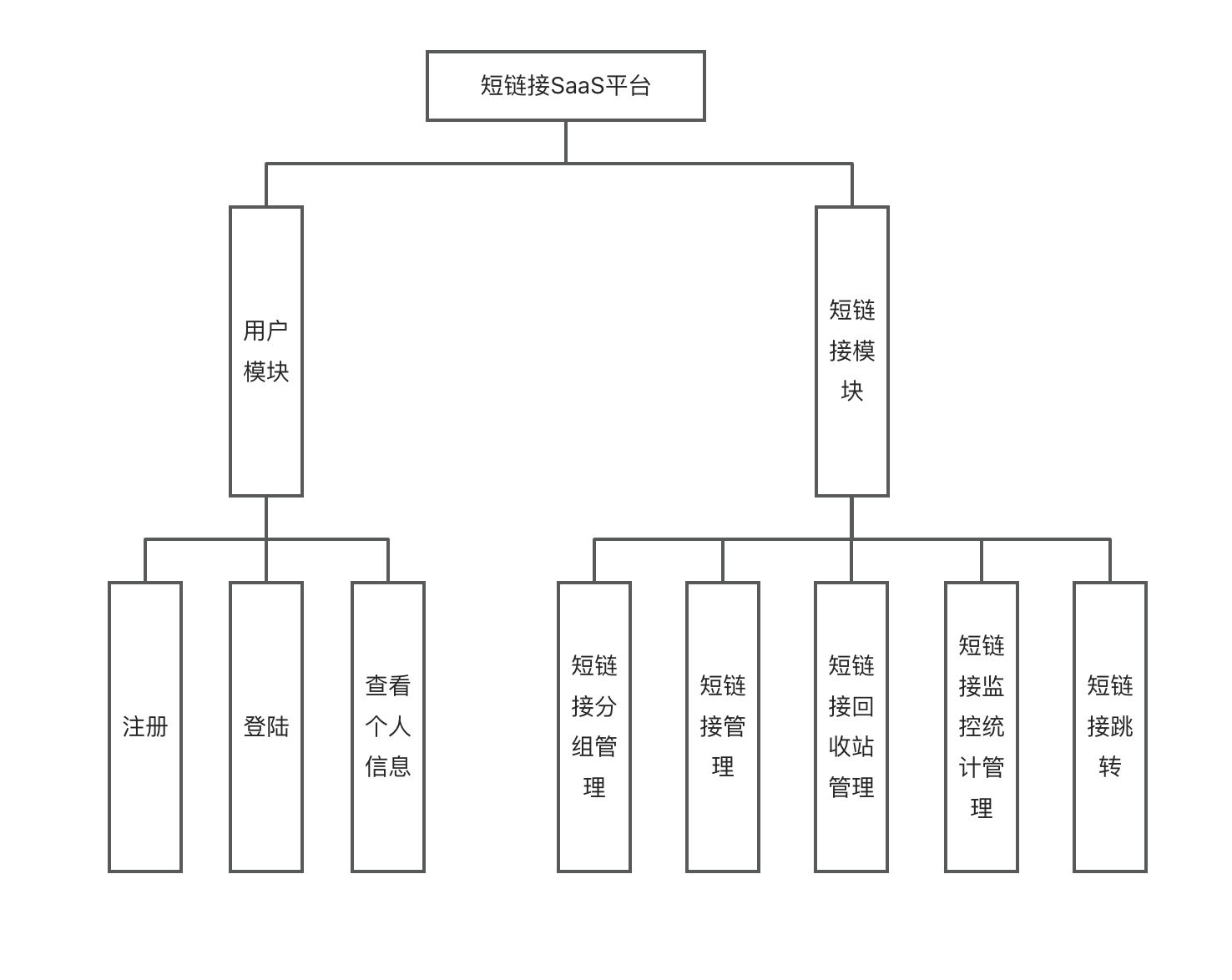


图4-2 系统功能模块图

用户模块主要有用户的登陆、注册和查看个人信息功能。主要用于用户对平台进行注册，之后可以进行平台的登陆操作，以便于使用系统中的各项功能。同时还能查看个人信息，这些信息进行脱敏处理，防止个人信息泄露[20]。

短链接模块主要有短链接分组管理、短链接管理、短链接回收站管理、短链接跳转、短链接统计监控功能。短链接分组管理主要负责对用户的多个短链接进行分组，同时每个用户最多创建15个短链接分组。短链接管理主要负责短链接的创建、批量创建、编辑功能，短链接编辑可以修改分组、有效期、原始链接等信息。短链接回收站管理主要负责短链接的逻辑删除，移至回收站只是暂时删除，从回收站移除才是真实删除。短链接监控和统计管理主要负责追踪用户访问短链接的基本信息和对这些信息进行统计聚合。短链接跳转主要负责短链接的核心跳转功能，并保证高性能和高可用。

## 4.2系统详细设计

根据上述的功能模块的划分，设计了短链接SaaS平台的一些核心流程图，通过流程图可以更加清晰直观把项目流程展示出来。

### 4.2.1创建短链接流程设计

用户登陆系统之后，可以在短链接分组下进行短链接的创建，需要输入原始链接、短链接分组、短链接有效期等信息。这些信息输入完成之后点击创建按钮，就会进行短链接的创建。创建短链接流程图如图4-3所示。

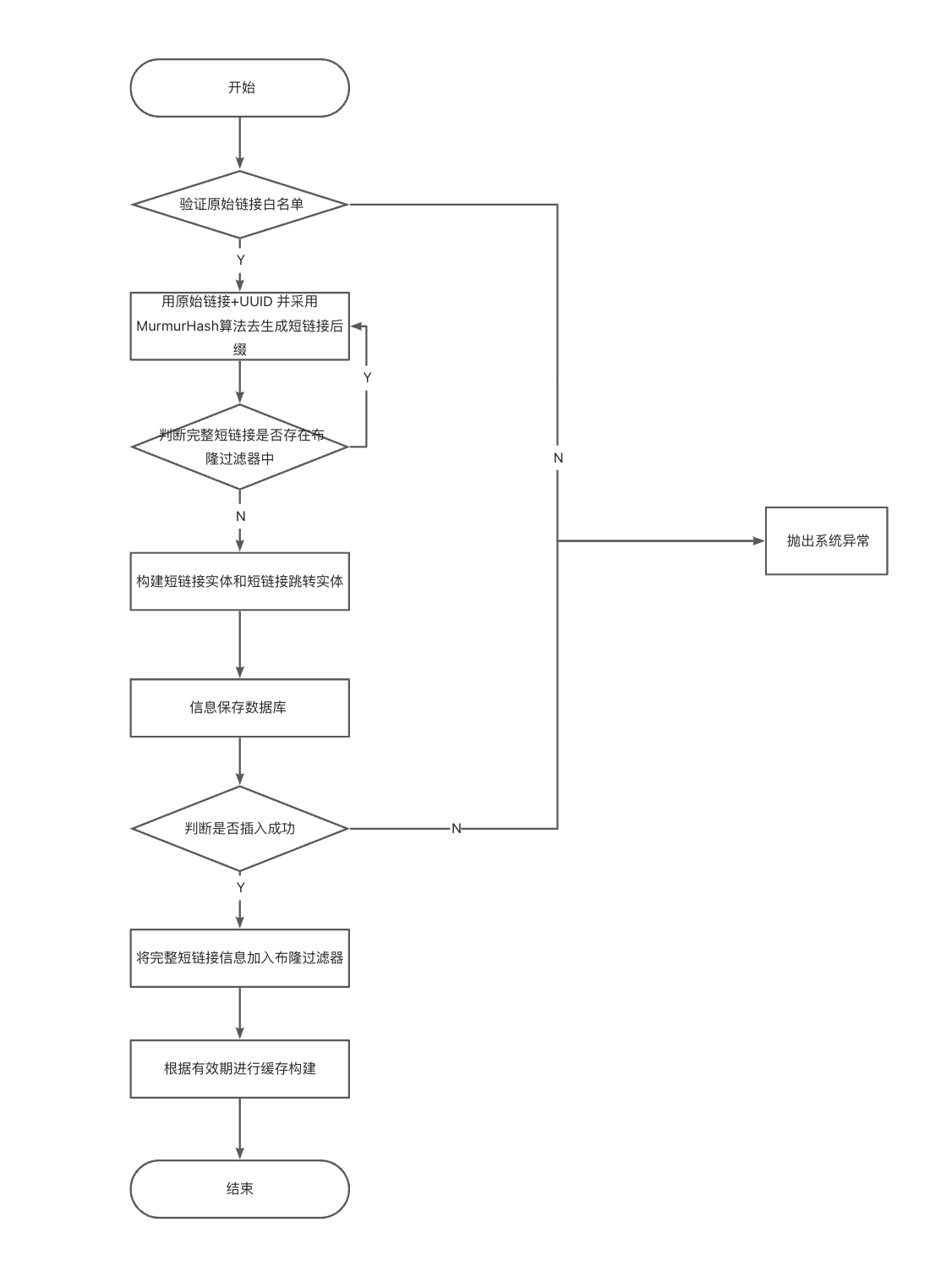


图4-3 创建短链接流程图

### 4.2.2编辑短链接流程设计

用户登陆系统后，可以对分组下的某个短链接进行编辑。可以修改这个短链接的原始链接、短链接分组、短链接的有效期等信息。修改完之后点击保存按钮，则会对该短链接进行修改，如果此时有很多用户在访问该短链接，则会提示修改失败。编辑短链接流程图如图4-4所示。

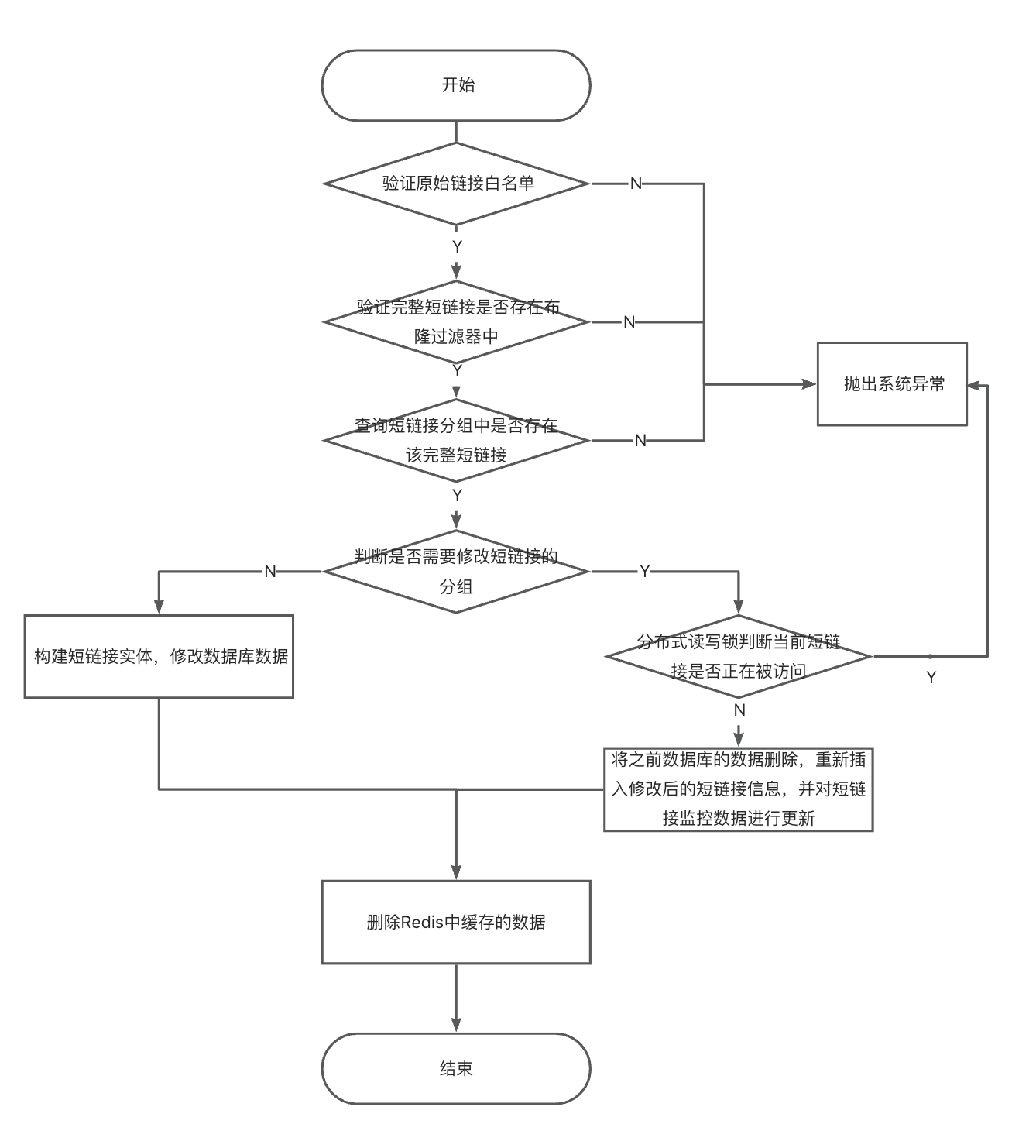


图4-4 编辑短链接流程图

### 4.2.3短链接跳转流程设计

用户创建完短链接之后，可以通过复制链接或者二维码分享给其他人。其他人点击或者扫码之后，请求会到达本平台上，平台会对请求进行解析，之后会返回短链接的原始链接并进行重定向。在请求解析的过程采用双重判定锁，避免重复查询数据库去构建缓存，减轻数据库压力。短链接跳转流程图如图4-5所示。

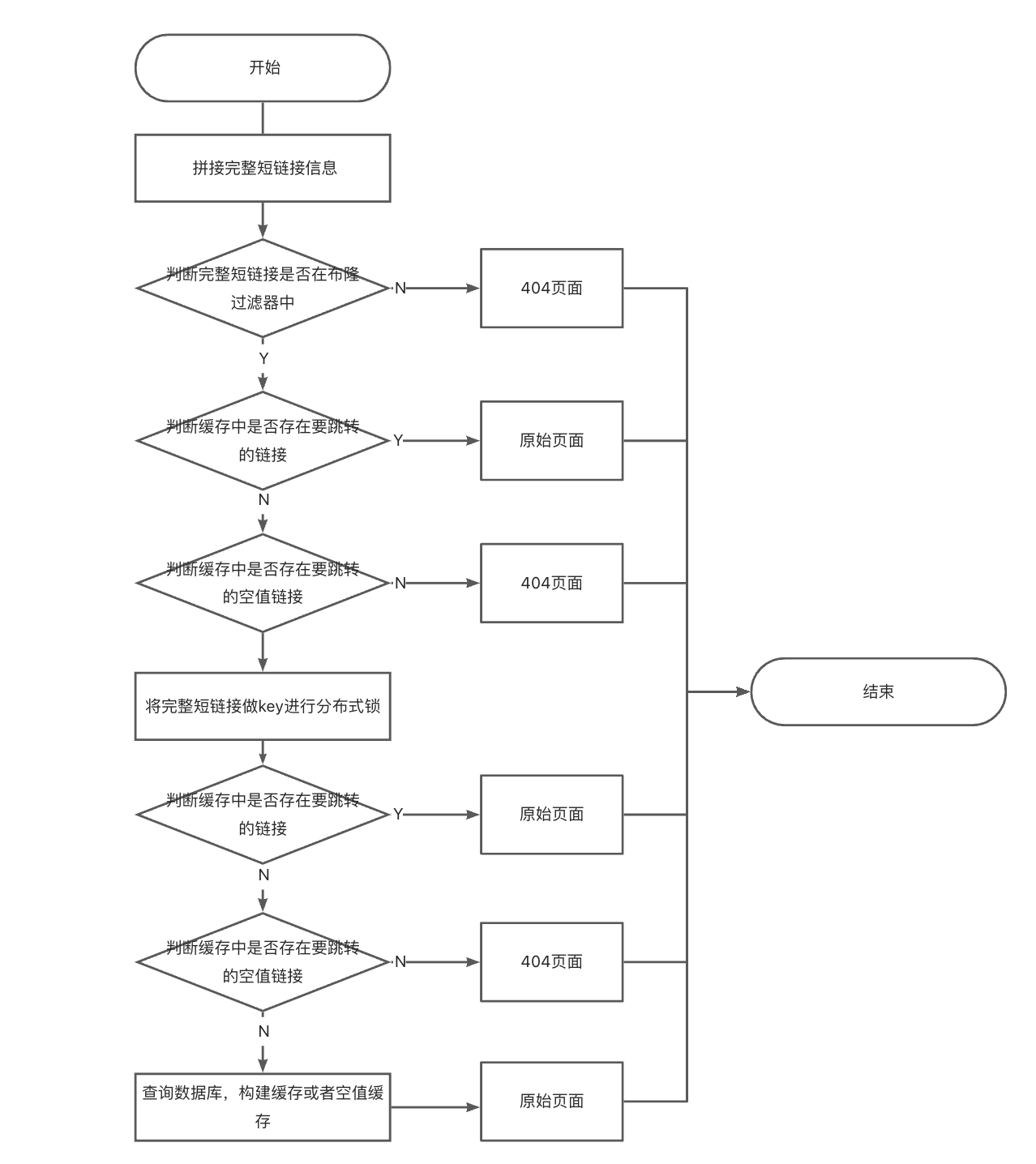


图4-5 短链接跳转流程图

## 4.3系统数据库设计

### 4.3.1 数据库概念结构设计

用户实体记录了用户的基本信息。属性包括主键ID、用户名、密码、真实姓名、注册手机、注册邮箱、注销时间戳等信息。如图4-6所示。

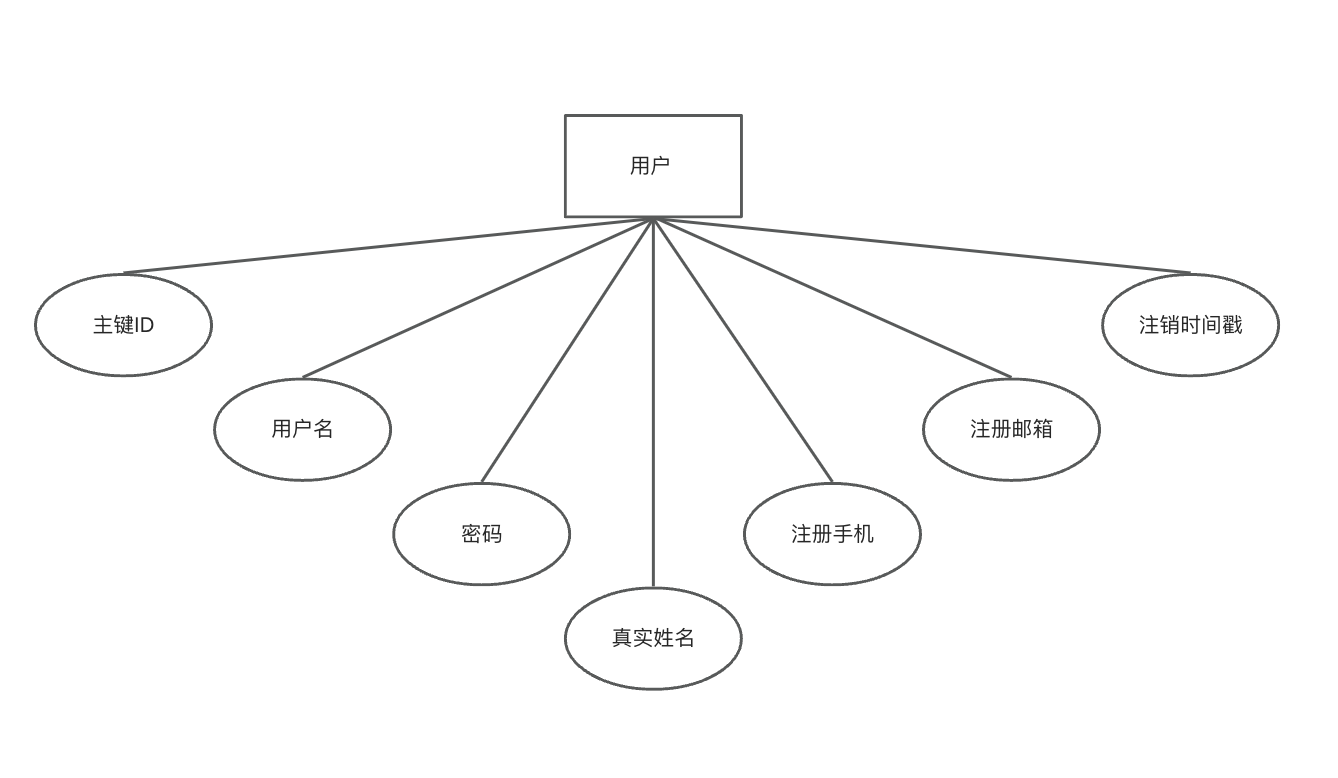


图4-6 用户实体属性图

短链接分组实体记录了短链接分组的基本信息。属性包括主键ID、分组标识、分组名称、创建用户名、分组排序等信息。如图4-7所示。

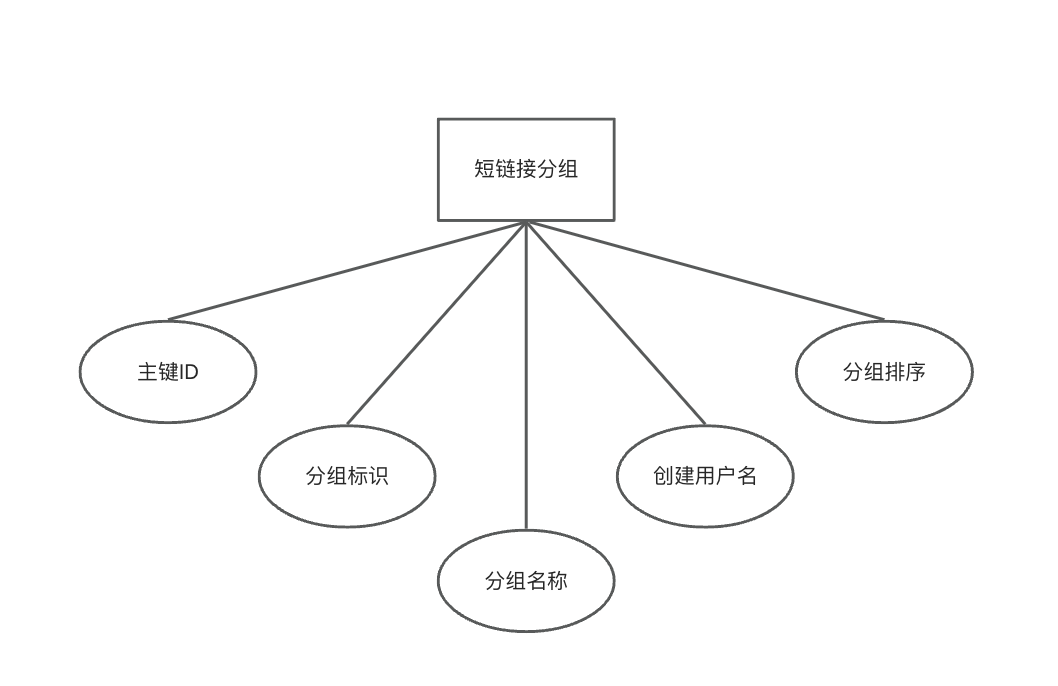


图4-7 短链接分组实体属性图

短链接基本信息实体记录了短链接的基本信息。属性包括主键ID、分组标识、原始链接、域名、短链接、完整短链接、创建类型、有效期、网站图标、描述、网站标题、启用标识、历史UV、历史PV、历史UIP、点击量等信息。如图4-8所示。

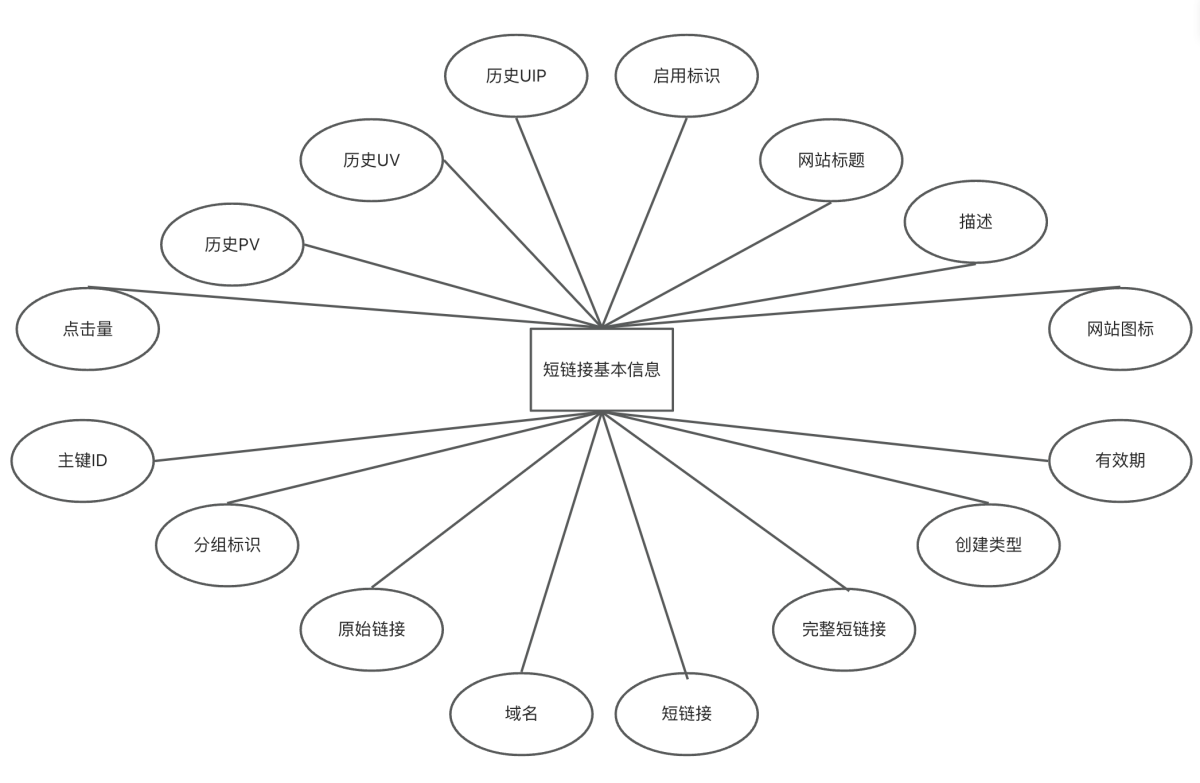


图4-8 短链接基本信息实体属性图

短链接监控统计信息实体记录了短链接监控统计的基本信息。属性包括主键ID、分组标识、用户信息、完整短链接、访问IP、访问地区、访问浏览器、访问网络、访问设备、访问日期、访问星期、访问小时等信息。如图4-9所示。

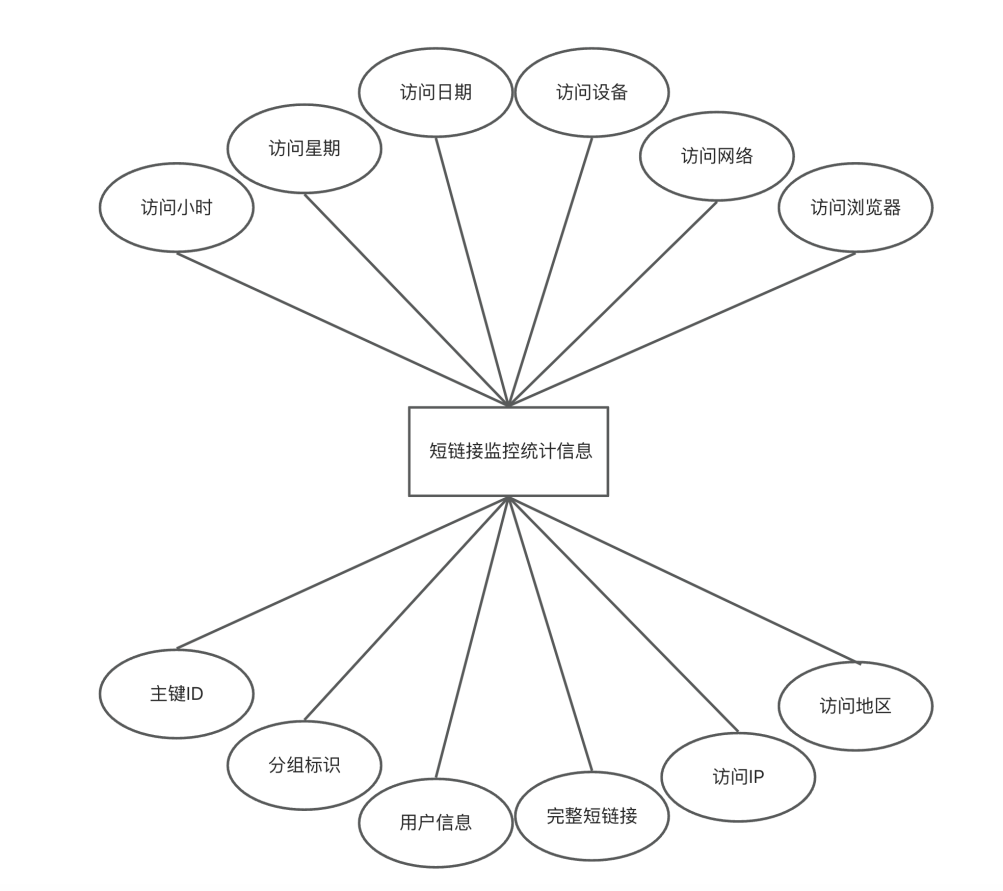


图4-9 短链接监控统计信息实体属性图

各个实体间的关系如图4-10所示。

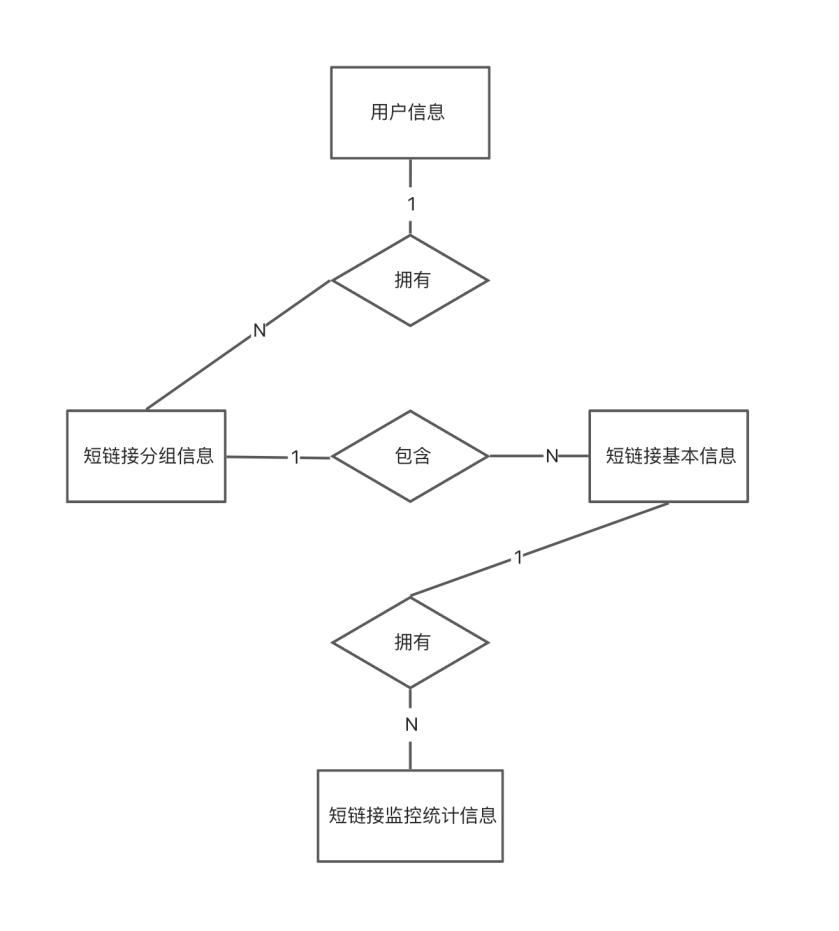


图4-10 实体E-R图

### 4.3.2 数据库逻辑结构设计

逻辑模型包含了数据库和表的设计，下面是本系统数据库的逻辑模型。如下：

（1）用户表（t\_user）

用户表主要用来存储系统用户的基本信息，具体的表结构如表4-1所示。

表4–1用户表

| 字段 | 数据类型 | 长度 | 可空否 | 说明 |
| --- | --- | --- | --- | --- |
| id | bigint | 20 | N | 主键值 |
| username | varchar | 256 | Y | 用户名 |
| password | varchar | 512 | Y | 密码 |
| real\_name | varchar | 256 | Y | 真实姓名 |
| phone | varchar | 128 | Y | 注册手机 |
| email | varchar | 512 | Y | 注册邮箱 |
| deletion\_time | bigint | 20 | Y | 注销时间戳 |
| create\_time | datetime | 0 | Y | 创建时间 |
| update\_tate | datetime | 0 | Y | 更新时间 |
| del\_flag | tinyint | 1 | Y | 删除标识 |

（2）分组表（t\_group）

分组表主要用来存储用户的短链接分组信息，具体的表结构如表4-2所示。

表4–2分组表

| 字段 | 数据类型 | 长度 | 可空否 | 说明 |
| --- | --- | --- | --- | --- |
| id | bigint | 20 | N | 主键值 |
| gid | varchar | 32 | Y | 分组标识 |
| name | varchar | 64 | Y | 分组名称 |
| username | varchar | 256 | Y | 创建者用户名 |
| sort\_order | int | 3 | Y | 分组排序 |
| create\_time | datetime | 0 | Y | 创建时间 |
| update\_tate | datetime | 0 | Y | 更新时间 |
| del\_flag | tinyint | 1 | Y | 删除标识 |

（3）短链接表（t\_link）

短链接表主要用来存储短链接的基本信息，具体的表结构如表4-3所示。

表4–3短链接表

| 字段 | 数据类型 | 长度 | 可空否 | 说明 |
| --- | --- | --- | --- | --- |
| id | bigint | 20 | N | 主键值 |
| gid | varchar | 32 | Y | 分组标识 |
| origin\_url | varchar | 1024 | Y | 原始链接 |
| domain | varchar | 128 | Y | 域名 |
| short\_url | varchar | 8 | Y | 短链接 |
| full\_short\_url | varchar | 128 | Y | 完整短链接 |
| created\_type | tinyint | 1 | Y | 创建类型 |
| valid\_date\_type | tinyint | 1 | Y | 有效期类型 |
| valid\_date | datetime | 0 | Y | 有效期 |
| describe | varchar | 1024 | Y | 描述 |
| favicon | varchar | 256 | Y | 网站图标 |
| title | varchar | 256 | Y | 网站标题 |
| enable\_status | tinyint | 1 | Y | 启用标识 |
| click\_num | int | 11 | Y | 点击量 |
| total\_pv | int | 11 | Y | 历史PV |
| total\_uv | int | 11 | Y | 历史UV |
| total\_uip | int | 11 | Y | 历史UIP |
| create\_time | datetime | 0 | Y | 创建时间 |
| update\_tate | datetime | 0 | Y | 更新时间 |
| del\_flag | tinyint | 1 | Y | 删除标识 |
| del\_time | bigint | 20 | Y | 删除时间戳 |

（4）短链接跳转表（t\_link\_goto）

短链接跳转表主要用来存储完整短链接和短链接分组标识之间的跳转信息，解决因分表而造成的读扩散问题，具体的表结构如表4-4所示。

表4–4短链接跳转表

| 字段 | 数据类型 | 长度 | 可空否 | 说明 |
| --- | --- | --- | --- | --- |
| id | bigint | 20 | N | 主键值 |
| gid | varchar | 32 | N | 分组标识 |
| full\_short\_url | varchar | 128 | N | 完整短链接 |

（5）基础访问监控表（t\_link\_access\_stats）

基础访问监控表主要用来存储短链接的访问基础监控信息，具体的表结构如表4-5所示。

表4–5基础访问监控表

| 字段 | 数据类型 | 长度 | 可空否 | s说明 |
| --- | --- | --- | --- | --- |
| id | bigint | 20 | N | 主键值 |
| gid | varchar | 32 | Y | 分组标识 |
| full\_short\_url | varchar | 128 | Y | 完整短链接 |
| date | date | 0 | Y | 日期 |
| hour | int | 11 | Y | 小时 |
| weekday | int | 11 | Y | 星期 |
| pv | int | 11 | Y | 访问量 |
| uv | int | 11 | Y | 独立访问数 |
| uip | int | 11 | Y | 独立IP数 |
| create\_time | datetime | 0 | Y | 创建时间 |
| update\_tate | datetime | 0 | Y | 更新时间 |
| del\_flag | tinyint | 1 | Y | 删除标识 |

（6）访问地区监控表（t\_link\_locale\_stats）

访问地区监控表主要用来存储短链接的访问地区监控信息，具体的表结构如表4-6所示。

表4–6访问地区监控表

| 字段 | 数据类型 | 长度 | 可空否 | 说明 |
| --- | --- | --- | --- | --- |
| id | bigint | 20 | N | 主键值 |
| gid | varchar | 32 | Y | 分组标识 |
| full\_short\_url | varchar | 128 | Y | 完整短链接 |
| date | date | 0 | Y | 日期 |
| country | varchar | 64 | Y | 国家名称 |
| province | varchar | 64 | Y | 省份名称 |
| city | varchar | 64 | Y | 市区名称 |
| adcode | varchar | 64 | Y | 城市编码 |
| cnt | int | 11 | Y | 访问量 |
| create\_time | datetime | 0 | Y | 创建时间 |
| update\_tate | datetime | 0 | Y | 更新时间 |
| del\_flag | tinyint | 1 | Y | 删除标识 |

（7）访问浏览器监控表（t\_link\_browser\_stats）

访问浏览器监控表主要用来存储短链接的访问浏览器监控信息，具体的表结构如表4-7所示。

表4–7访问浏览器监控表

| 字段 | 数据类型 | 长度 | 可空否 | 说明 |
| --- | --- | --- | --- | --- |
| id | bigint | 20 | N | 主键值 |
| gid | varchar | 32 | Y | 分组标识 |
| full\_short\_url | varchar | 128 | Y | 完整短链接 |
| date | date | 0 | Y | 日期 |
| browser | varchar | 64 | Y | 访问浏览器 |
| cnt | int | 11 | Y | 访问量 |
| create\_time | datetime | 0 | Y | 创建时间 |
| update\_tate | datetime | 0 | Y | 更新时间 |
| del\_flag | tinyint | 1 | Y | 删除标识 |

（8）访问操作系统监控表（t\_link\_os\_stats）

访问操作系统监控表主要用来存储短链接的访问操作系统监控信息，具体的表结构如表4-8所示。

表4–8访问操作系统监控表

| 字段 | 数据类型 | 长度 | 可空否 | 说明 |
| --- | --- | --- | --- | --- |
| id | bigint | 20 | N | 主键值 |
| gid | varchar | 32 | Y | 分组标识 |
| full\_short\_url | varchar | 128 | Y | 完整短链接 |
| date | date | 0 | Y | 日期 |
| os | varchar | 64 | Y | 访问操作系统 |
| cnt | int | 11 | Y | 访问量 |
| create\_time | datetime | 0 | Y | 创建时间 |
| update\_tate | datetime | 0 | Y | 更新时间 |
| del\_flag | tinyint | 1 | Y | 删除标识 |

（9）访问设备监控表（t\_link\_device\_stats）

访问设备监控表主要用来存储短链接的访问设备监控信息，具体的表结构如表4-9所示。

表4–9访问设备监控表

| 字段 | 数据类型 | 长度 | 可空否 | 说明 |
| --- | --- | --- | --- | --- |
| id | bigint | 20 | N | 主键值 |
| gid | varchar | 32 | Y | 分组标识 |
| full\_short\_url | varchar | 128 | Y | 完整短链接 |
| date | date | 0 | Y | 日期 |
| device | varchar | 64 | Y | 访问设备 |
| cnt | int | 11 | Y | 访问量 |
| create\_time | datetime | 0 | Y | 创建时间 |
| update\_tate | datetime | 0 | Y | 更新时间 |
| del\_flag | tinyint | 1 | Y | 删除标识 |

（10）访问网络监控表（t\_link\_network\_stats）

访问网络监控表主要用来存储短链接的访问网络监控信息，具体的表结构如表4-10所示。

表4–10访问网络监控表

| 字段 | 数据类型 | 长度 | 可空否 | 说明 |
| --- | --- | --- | --- | --- |
| id | bigint | 20 | N | 主键值 |
| gid | varchar | 32 | Y | 分组标识 |
| full\_short\_url | varchar | 128 | Y | 完整短链接 |
| date | date | 0 | Y | 日期 |
| network | varchar | 64 | Y | 访问网络 |
| cnt | int | 11 | Y | 访问量 |
| create\_time | datetime | 0 | Y | 创建时间 |
| update\_tate | datetime | 0 | Y | 更新时间 |
| del\_flag | tinyint | 1 | Y | 删除标识 |

（11）访问日志监控表（t\_link\_access\_logs）

访问日志监控表主要用来存储短链接的访问日志监控信息，具体的表结构如表4-11所示。

表4–11访问日志监控表

| 字段 | 数据类型 | 长度 | 可空否 | 说明 |
| --- | --- | --- | --- | --- |
| id | bigint | 20 | N | 主键值 |
| gid | varchar | 32 | Y | 分组标识 |
| user | varchar | 64 | Y | 用户信息 |
| full\_short\_url | varchar | 128 | Y | 完整短链接 |
| ip | varchar | 64 | Y | 访问IP |
| locale | varchar | 256 | Y | 访问地区 |
| os | varchar | 64 | Y | 访问操作系统 |
| browser | varchar | 64 | Y | 访问浏览器 |
| network | varchar | 64 | Y | 访问网络 |
| device | varchar | 64 | Y | 访问设备 |
| create\_time | datetime | 0 | Y | 创建时间 |
| update\_tate | datetime | 0 | Y | 更新时间 |
| del\_flag | tinyint | 1 | Y | 删除标识 |

（12）今日统计监控表（t\_link\_stats\_today）

今日统计监控表主要用来存储短链接的今日统计监控信息，具体的表结构如表4-12所示。

表4–12今日统计监控表

| 字段 | 数据类型 | 长度 | 可空否 | 说明 |
| --- | --- | --- | --- | --- |
| id | bigint | 20 | N | 主键值 |
| gid | varchar | 32 | Y | 分组标识 |
| full\_short\_url | varchar | 128 | Y | 完整短链接 |
| date | date | 0 | Y | 日期 |
| today\_pv | int | 11 | Y | 今日PV |
| today\_uv | int | 11 | Y | 今日UV |
| today\_uip | int | 11 | Y | 今日IP数 |
| create\_time | datetime | 0 | Y | 创建时间 |
| update\_tate | datetime | 0 | Y | 更新时间 |
| del\_flag | tinyint | 1 | Y | 删除标识 |

## 4.4系统部署上线设计

系统选用Docker作为容器，DockerHub作为其镜像仓库，采用Kubernetes集群对其进行部署。同时为了保证系统的高可用，另搭建一台LoadBalancer服务器，负载均衡到Kubernetes集群中。为保证Kubernetes中Pod的数据重启之后不会消失，搭建一台NFS文件系统进行文件的持久化存储。

对于Java后端服务来说需要基于JDK镜像进行服务镜像的构建，对于Vue前端服务来说需要基于Nginx镜像进行服务进行的构建。每个服务都需要手动编写Dockerfile进行镜像的构建，并上传到DockerHub仓库中。对于Mysql、Redis等数据库服务原则上部署在实体机上更高效，但这里为了统一和方便也部署进Kubernetes集群中。在采用Kubernetes进行部署时，需要编写对应的Yml部署文件，文件中包含Pod、Service、Deployment等描述信息，之后只需执行kubectl apply -f 部署文件即可。本平台的部署架构图如4-11所示。

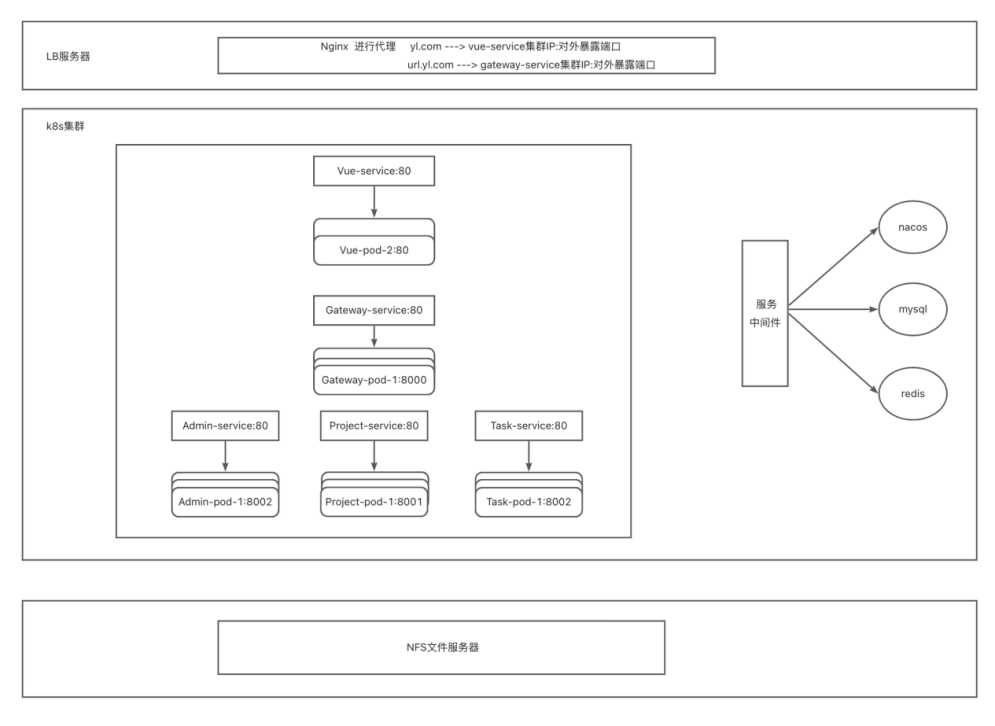


图4-11 部署架构图

# 系统实现

## 5.1系统部署实现

### 5.1.1系统多环境打包

系统多环境打包是指在软件开发过程中，针对不同的环境（开发环境、测试环境、生产环境等）进行打包部署。这样做的目的是为了在不同的环境中使用不同的配置，例如数据库连接信息、API端点、日志级别等。这样可以提高开发效率，同时确保在不同环境下软件的行为一致性和稳定性。本平台在Maven中配置多环境打包的核心配置如下所示。

<profiles>

<profile>

<id>dev</id>

<properties>

<surroundings>dev</surroundings>

</properties>

<activation>

<activeByDefault>true</activeByDefault>

</activation>

</profile>

<profile>

<id>k8s</id>

<properties>

<surroundings>k8s</surroundings>

</properties>

</profile>

</profiles>

<build>

<finalName>${project.artifactId}</finalName>

<plugins>

<plugin>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-maven-plugin</artifactId>

<executions>

<execution>

<goals>

<goal>repackage</goal>

</goals>

</execution>

</executions>

</plugin>

</plugins>

<resources>

<resource>

<filtering>true</filtering>

<directory>src/main/resources</directory>

<excludes>

<exclude>application-dev.yaml</exclude>

<exclude>application-k8s.yaml</exclude>

</excludes>

</resource>

<resource>

<filtering>true</filtering>

<directory>src/main/resources</directory>

<includes>

<include>application-${surroundings}.yaml</include>

</includes>

</resource>

</resources>

</build>

### 5.1.2系统镜像打包及上传

在Kubernetes环境下分别对每个模块进行打包之后，还需要分别编写对应的Dockerfile，将项目打成Docker镜像，并上传到DockerHub私服上。

其中前端Vue项目的Dockerfile如下。

FROM nginx:1.19  
MAINTAINER ylan "pepsiwyl@gmail.com"  
ENV TZ=Asia/Shanghai  
RUN ln -snf /usr/share/zoneinfo/$TZ /etc/localtime && echo '$TZ' > /etc/timezone  
ENV APP\_PATH=/app  
WORKDIR $APP\_PATH  
COPY ./dist /usr/share/nginx/html/  
COPY ./nginx/default.conf /etc/nginx/conf.d/default.conf  
EXPOSE 80

其中Nginx的核心配置文件如下。

upstream gateway\_server{  
 server short-link-gateway-service;

keepalive 2000;  
}  
server {  
 listen 80;   
 server\_name web.yl.com;   
 charset utf-8;  
 access\_log /var/log/nginx/host.access.log main;  
 error\_log /var/log/nginx/error.log error;  
 location / {  
 root /usr/share/nginx/html;  
 index index.html index.htm;  
 try\_files $uri $uri/ /index.html =404;  
 }  
 location /api/ {  
 proxy\_set\_header X-Real-IP $remote\_addr;  
 proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for;  
 proxy\_pass http://gateway\_server;  
 proxy\_set\_header Host $host:$server\_port;  
 rewrite ^/api/(.\*) /$1 break;  
 }  
}

其中镜像的打包和上传命令如下。

docker build -t pepsiwyl920/short-link-vue:1.0.0 .  
docker push pepsiwyl920/short-link-vue:1.0.0

### 5.1.3系统上线部署

本平台部署于本地搭建的Kubernetes集群中，其中集群IP地址分别为172.16.255.11，172.16.255.12，172.16.255.13，其中172.16.255.11为主节点。同时在172.16.255.11上部署了NFS文件存储系统，用来存储Kubernetes中Pod的持久化数据。同时在172.16.255.11上搭建了LoadBalancer服务器，负载均衡到Kubernetes集群中。Kubernetes集群中主要部署了Mysql、Redis、Nacos等中间件和Project、Admin、Task、Gateway、Vue等项目。

其中LoadBalancer服务器中Nginx的核心配置如下。

upstream vue\_server{  
 server 172.16.255.11:30080 weight=1;  
 server 172.16.255.12:30080 weight=1;  
 server 172.16.255.13:30080 weight=1;  
 keepalive 2000;  
 }  
 upstream gateway\_server{  
 server 172.16.255.11:30000 weight=1;  
 server 172.16.255.12:30000 weight=1;  
 server 172.16.255.13:30000 weight=1;  
 keepalive 2000;  
 }  
 server {  
 listen 80;  
 server\_name web.yl.com;  
 location / {  
 proxy\_set\_header X-Real-IP $remote\_addr;  
 proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for;  
 proxy\_pass http://vue\_server;  
 proxy\_set\_header Host $host:$server\_port;  
 }  
 location /api/ {  
 proxy\_set\_header X-Real-IP $remote\_addr;  
 proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for;  
 proxy\_pass http://gateway\_server;  
 proxy\_set\_header Host $host:$server\_port;  
 rewrite ^/api/(.\*) /$1 break;  
 }  
 }  
 server {  
 listen 80;  
 server\_name url.yl.com;  
 location / {  
 proxy\_set\_header X-Real-IP $remote\_addr;  
 proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for;  
 proxy\_pass http://gateway\_server;  
 proxy\_set\_header Host $host:$server\_port;  
 }  
 }

其中Project（中台模块）的Kubernetes部署文件如下。

# Deployment  
apiVersion: apps/v1  
kind: Deployment  
metadata:  
 name: short-link-gateway-deployment  
 labels:  
 app: short-link-gateway  
spec:  
 replicas: 1  
 selector:  
 matchLabels:  
 app: short-link-gateway  
 template:  
 metadata:  
 name: short-link-gateway  
 labels:  
 app: short-link-gateway  
 spec:  
 containers:  
 - name: short-link-gateway  
 image: pepsiwyl920/short-link-gateway:1.0.0  
 imagePullPolicy: IfNotPresent  
 ports:  
 - containerPort: 8000  
 volumeMounts:  
 - mountPath: /app/logs  
 name: logs  
 volumes:  
 - name: logs  
 nfs:  
 server: 172.16.255.11  
 path: /nfs/data/app/logs  
 restartPolicy: Always  
---  
# service  
apiVersion: v1  
kind: Service  
metadata:  
 name: short-link-gateway-service  
 labels:  
 app: short-link-gateway  
spec:  
 selector:  
 app: short-link-gateway  
 type: NodePort  
 ports:  
 - name: short-link-gateway  
 protocol: TCP  
 port: 80  
 targetPort: 8000  
 nodePort: 30000  
---

平台部署完成之后，如图5-1和5-2所示。

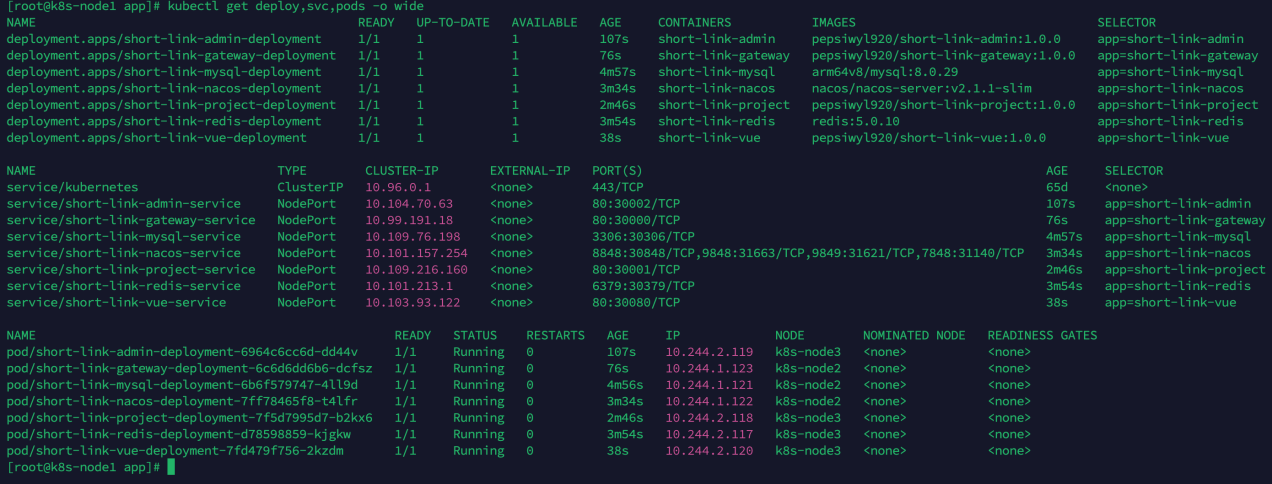


图5-1 服务部署成功

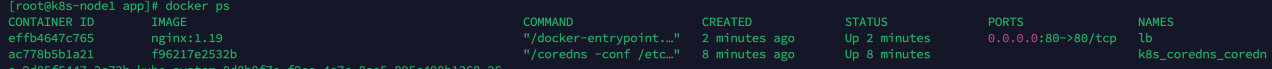


图5-2 LoadBalancer部署成功

## 5.2系统功能实现

### 5.2.1系统登陆注册功能实现

为保证安全，需要对用户进行校验，首次使用需要先注册，只有在平台注册过的用户才能使用该平台。平台首页为登陆页面和注册页面，登陆页面如图5-3所示，注册页面如图5-4所示。



图5-3 系统登陆



图5-4 系统注册

在系统注册时，把用户名作为分布式锁的key，防止同一时刻有相同用户名的用户注册，同时采用布隆过滤器过滤掉数据库中已经存在的用户名。在系统登陆时，采用布隆过滤器进行过滤，避免系统不存在的用户进行登陆时查询数据库操作。登陆成功之后，系统生成token返回客户端，并将token存入redis中设置过期时间。系统用户重复登陆时，会对token进行续约操作。

用户登陆的核心代码如下。

@Override  
 public UserLoginRespDTO login(UserLoginReqDTO requestParam) {  
 if (!hasUserByUsername(requestParam.getUsername())) {  
 throw new ClientException(USER\_LOGIN\_ERROR);  
 }  
 Map<Object ,Object> hasLoginMap = stringRedisTemplate.opsForHash().entries(LOGIN\_PREFIX + requestParam.getUsername());  
 if (CollUtil.isNotEmpty(hasLoginMap)) {  
 stringRedisTemplate.expire(LOGIN\_PREFIX + requestParam.getUsername(), 30L, TimeUnit.MINUTES);  
 String token = hasLoginMap.keySet().stream()  
 .findFirst()  
 .map(Object::toString)  
 .orElseThrow(() -> new ClientException("用户登录错误"));  
 return new UserLoginRespDTO(token);  
 }  
 LambdaQueryWrapper<UserDO> queryWrapper = Wrappers.lambdaQuery(UserDO.class)  
 .eq(UserDO::getUsername, requestParam.getUsername())  
 .eq(UserDO::getPassword, requestParam.getPassword());  
 UserDO userDO = baseMapper.selectOne(queryWrapper);  
 if (Objects.isNull(userDO)){  
 throw new ClientException(USER\_LOGIN\_ERROR);  
 }  
 String token = UUID.randomUUID().toString();  
 stringRedisTemplate.opsForHash().put(LOGIN\_PREFIX + requestParam.getUsername(), token, JSON.toJSONString(userDO));  
 stringRedisTemplate.expire(LOGIN\_PREFIX + requestParam.getUsername(), 30L, TimeUnit.MINUTES);  
 return UserLoginRespDTO.builder().token(token).build();  
 }

用户注册的核心代码如下。

@Override  
public void register(UserRegisterReqDTO requestParam) {  
 if (hasUserByUsername(requestParam.getUsername())) {  
 throw new ClientException(USER\_NAME\_EXIST);  
 }  
 RLock lock = redissonClient.getLock(LOCK\_USER\_REGISTER\_KEY + requestParam.getUsername());  
 boolean isLocked = lock.tryLock();  
 if (isLocked) {  
 log.info("分布式锁{}获取成功, 开始执行注册逻辑", LOCK\_USER\_REGISTER\_KEY + requestParam.getUsername());  
 try {  
 int inserted = baseMapper.insert(BeanUtil.toBean(requestParam, UserDO.class));  
 if (inserted < 0) {  
 throw new ClientException(USER\_SAVE\_ERROR);  
 }  
userRegisterCachePenetrationBloomFilter.add(requestParam.getUsername()); groupService.saveGroup(requestParam.getUsername(),GroupSaveReqDTO.builder().name("默认分组").build());  
 } catch (DuplicateKeyException ex) {  
 throw new ClientException(USER\_EXIST);  
 } finally {  
 lock.unlock();  
 }  
 return;  
 }  
 throw new ClientException(USER\_NAME\_EXIST);  
}

### 5.2.2系统短链接生成功能实现

系统核心的功能就是短链接生成，用户在生成短链接时，需要输入原始链接、短链接分组、有效期等信息，而网站的描述信息和图标等是通过后端服务自动爬取。系统生成短链接如图5-5所示，生成成功之后会跳转到首页并展示刚刚生成的短链接，如图5-6所示。其核心代码如下。

private String generateSuffix(ShortLinkCreateReqDTO requestParam) {  
 String shortUri;  
 int customGenerateCount = 0;  
 while (true) {  
 customGenerateCount++;  
 if (customGenerateCount >= 10) {  
 throw new ServiceException(SHORT\_LINK\_GENERATE\_ERROR);  
 }  
 shortUri = HashUtils.hashToBase62(StrBuilder.create(requestParam.getOriginUrl()).append(UUID.randomUUID()).toString());  
 String fullShortUrl = StrBuilder.create(Objects.isNull(requestParam.getDomain()) ? NetUtils.removalProtocol(defaultShortLinkDomain) : NetUtils.removalProtocol(requestParam.getDomain())).append(URL\_SPLIT).append(shortUri).toString();  
 if (!shortUriCreateCachePenetrationBloomFilter.contains(fullShortUrl)) {  
 return shortUri;  
 }  
 }  
}

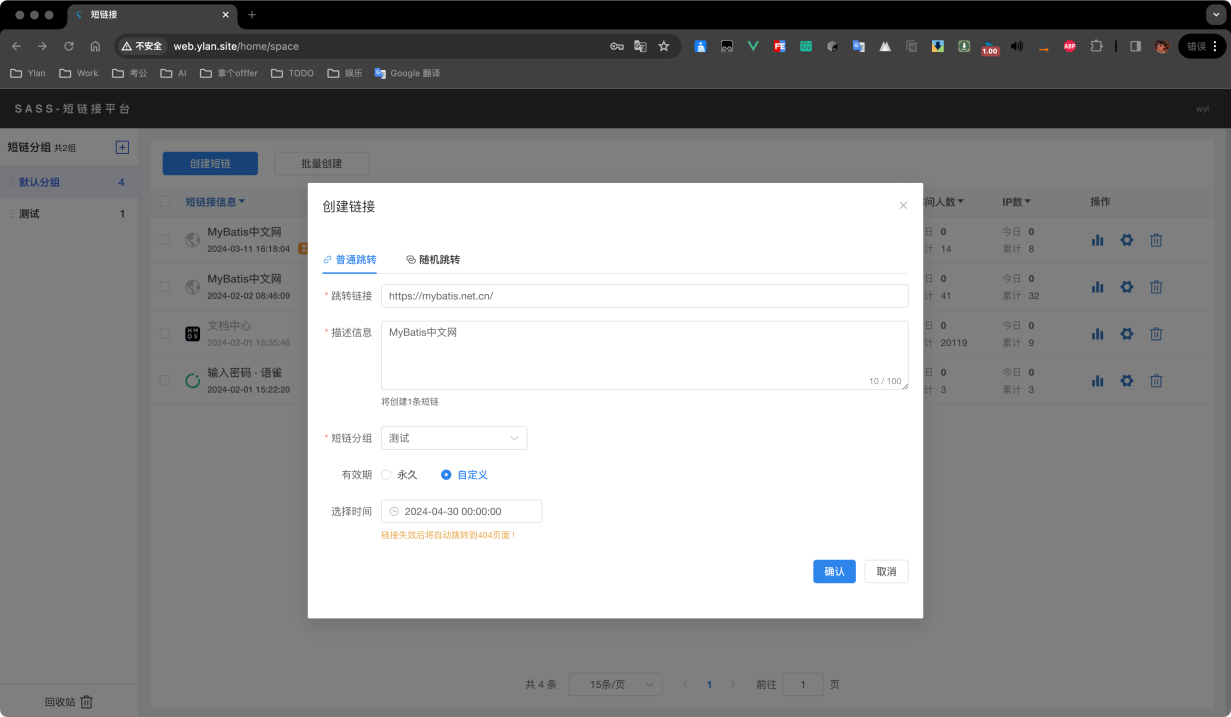


图5-5 系统短链接生成

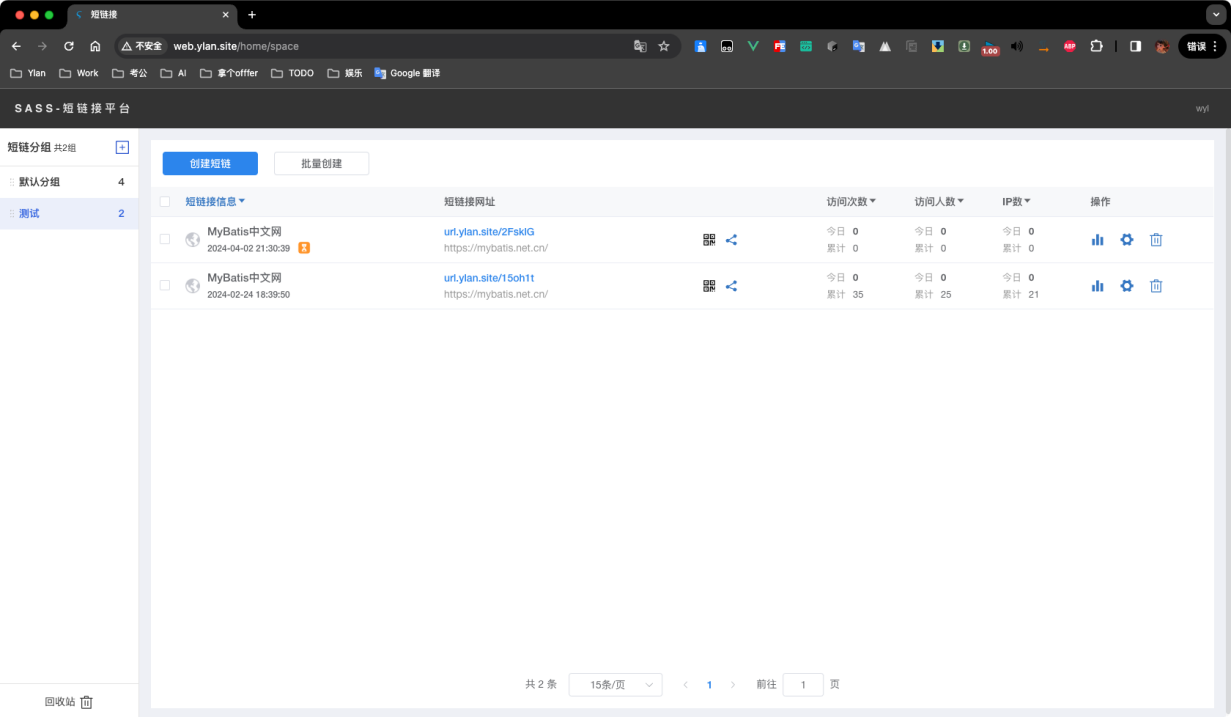


图5-6 系统短链接生成成功

### 5.2.3系统短链接跳转功能实现

用户可以通过点击复制链接或者生成二维码将短链接分享给他人，如用于产品推广、营销等，如图5-7所示。当有用户点击或者扫码该短链接之后，请求会到达系统，系统会做处理并将浏览器重定向至原始链接，如图5-8所示。

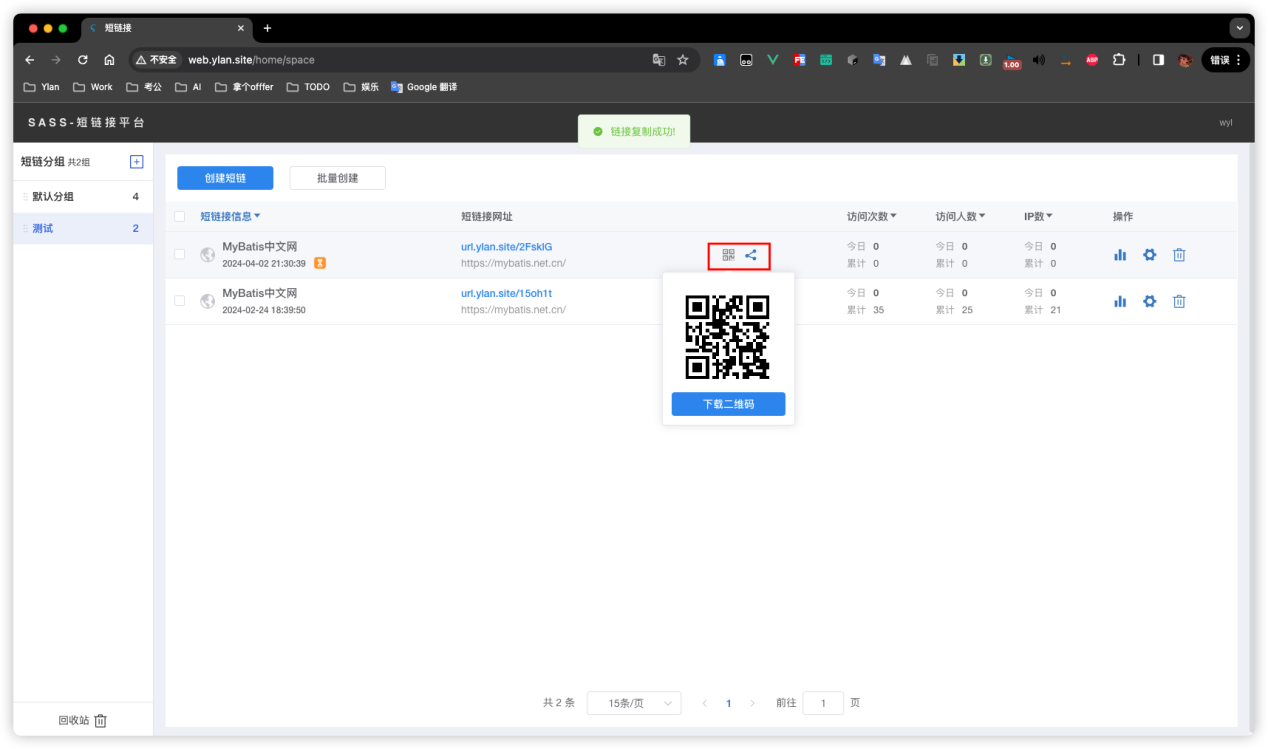


图5-7 系统短链接分享

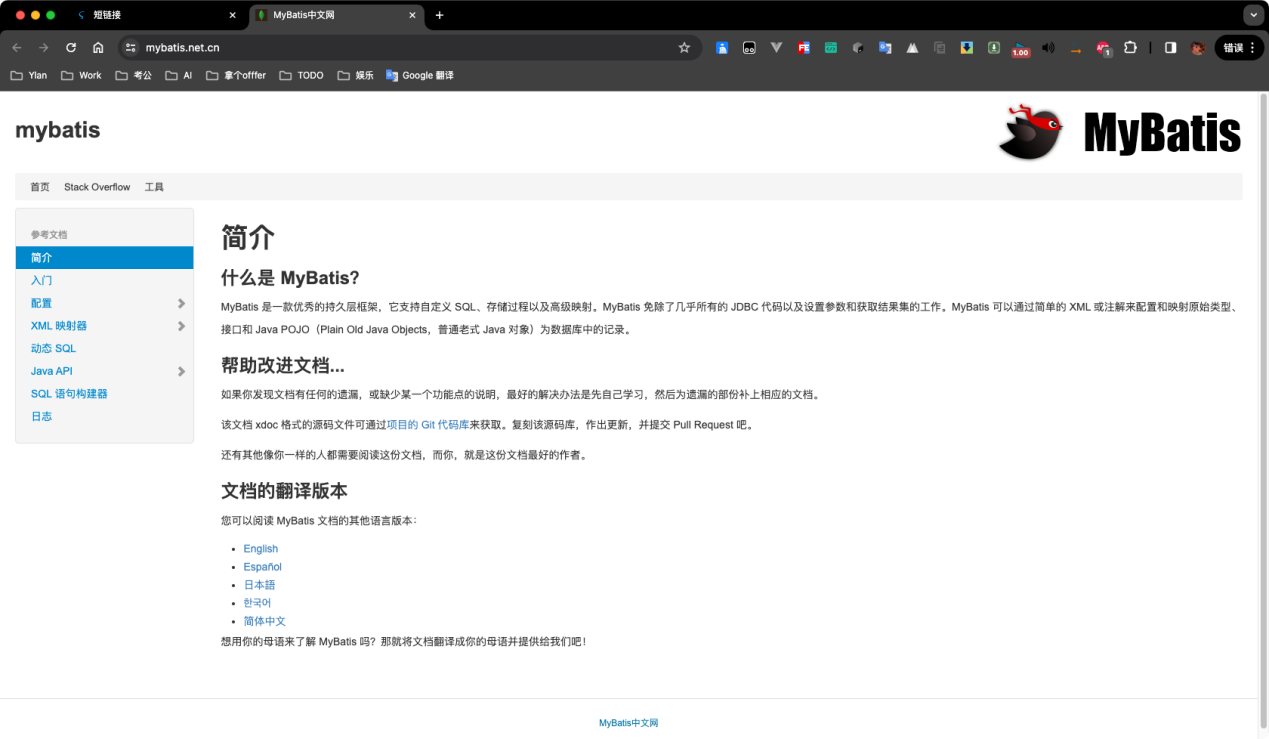


图5-8 系统短链接跳转成功

其核心代码如下。

RLock lock = redissonClient.getLock(String.format(LOCK\_GOTO\_SHORT\_LINK\_KEY, fullShortUrl));  
lock.lock();  
try {  
 originalLink = stringRedisTemplate.opsForValue().get(String.format(GOTO\_SHORT\_LINK\_KEY, fullShortUrl));  
 if (StringUtils.isNotBlank(originalLink)) {  
 ShortLinkStatsRecordDTO statsRecord = shortLinkStatsService.buildLinkStatsRecordAndSetUser(null, fullShortUrl, request, response);  
 shortLinkStatsService.shortLinkStats(statsRecord);  
 jumpLink(request, response, fullShortUrl, originalLink);  
 return;  
 }  
 gotoIsNullShortLink = stringRedisTemplate.opsForValue().get(String.format(GOTO\_IS\_NULL\_SHORT\_LINK\_KEY, fullShortUrl));  
 if (StringUtils.isNotBlank(gotoIsNullShortLink)) {  
 jumpNotFound(request, response, fullShortUrl, originalLink);  
 return;  
 }  
 LambdaQueryWrapper<ShortLinkGotoDO> linkGotoQueryWrapper = Wrappers.lambdaQuery(ShortLinkGotoDO.class).eq(ShortLinkGotoDO::getFullShortUrl, fullShortUrl);  
 ShortLinkGotoDO shortLinkGotoDO = shortLinkGotoMapper.selectOne(linkGotoQueryWrapper);  
 if (Objects.isNull(shortLinkGotoDO)) {stringRedisTemplate.opsForValue().set(String.format(GOTO\_IS\_NULL\_SHORT\_LINK\_KEY, fullShortUrl), "-", 30, TimeUnit.MINUTES);  
 jumpNotFound(request, response, fullShortUrl, originalLink);  
 return;  
 }  
 LambdaQueryWrapper<ShortLinkDO> shortLinkQueryWrapper = Wrappers.lambdaQuery(ShortLinkDO.class)  
 .eq(ShortLinkDO::getGid, shortLinkGotoDO.getGid())  
 .eq(ShortLinkDO::getFullShortUrl, fullShortUrl)  
 .eq(ShortLinkDO::getEnableStatus, 0);  
 ShortLinkDO shortLinkDO = baseMapper.selectOne(shortLinkQueryWrapper);  
 if (Objects.isNull(shortLinkDO) || (VailDateTypeEnum.CUSTOM.getType().equals(shortLinkDO.getValidDateType()) && shortLinkDO.getValidDate().before(new Date()))) {  
 stringRedisTemplate.opsForValue().set(String.format(GOTO\_IS\_NULL\_SHORT\_LINK\_KEY, fullShortUrl), "-", 30, TimeUnit.MINUTES);  
 jumpNotFound(request, response, fullShortUrl, originalLink);  
 return;  
 } stringRedisTemplate.opsForValue().set(String.format(GOTO\_SHORT\_LINK\_KEY, fullShortUrl), shortLinkDO.getOriginUrl(),  
 LinkUtil.getLinkCacheValidTime(shortLinkDO.getValidDate()), TimeUnit.MILLISECONDS  
 );  
 ShortLinkStatsRecordDTO statsRecord = shortLinkStatsService.buildLinkStatsRecordAndSetUser(shortLinkDO.getGid(), fullShortUrl, request, response);  
 shortLinkStatsService.shortLinkStats(statsRecord);  
 jumpLink(request, response, fullShortUrl, shortLinkDO.getOriginUrl());  
} finally {  
 lock.unlock();  
}

### 5.2.4系统短链接统计功能实现

用户点击或者扫码分享的短链接后，系统会对用户行为进行分析，记录下访问时间、访问IP、访问地区、访问浏览器、访问设备、访问网络、访客类型等。

系统会对这些信息进行统计分析得出如访问历史记录、访问曲线、高频访问IP、访问地区图、24小时分布图、一周分布图、操作系统比例、浏览器比例、网络比例、访客类型比例等。如图5-9、5-10、5-11、5-12所示。

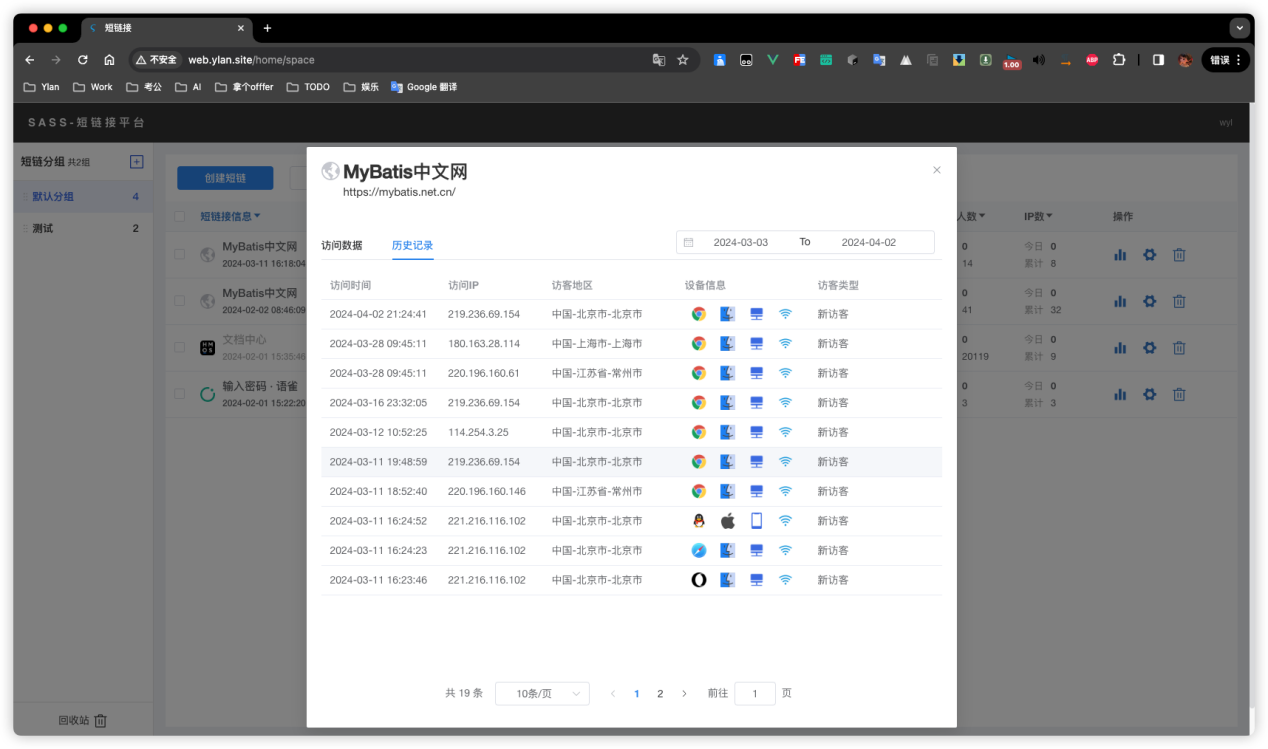


图5-9 系统短链接历史记录

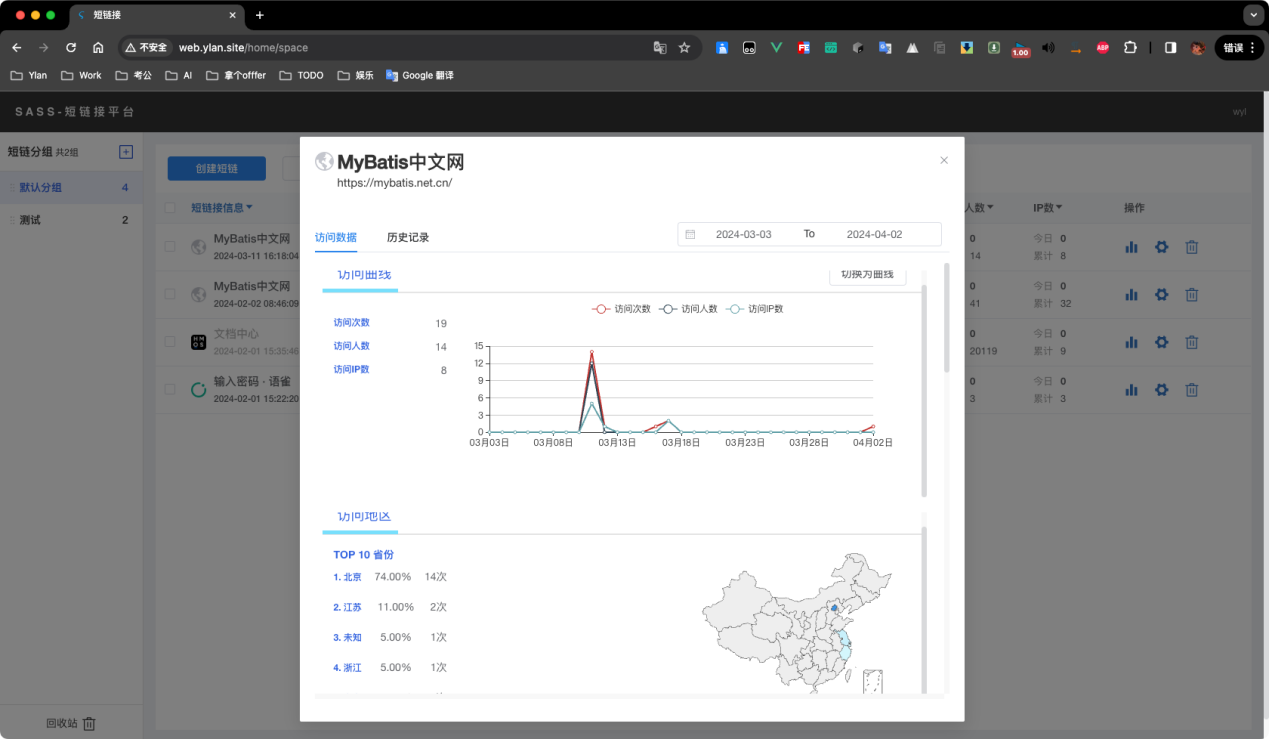


图5-10 系统短链接访问数据

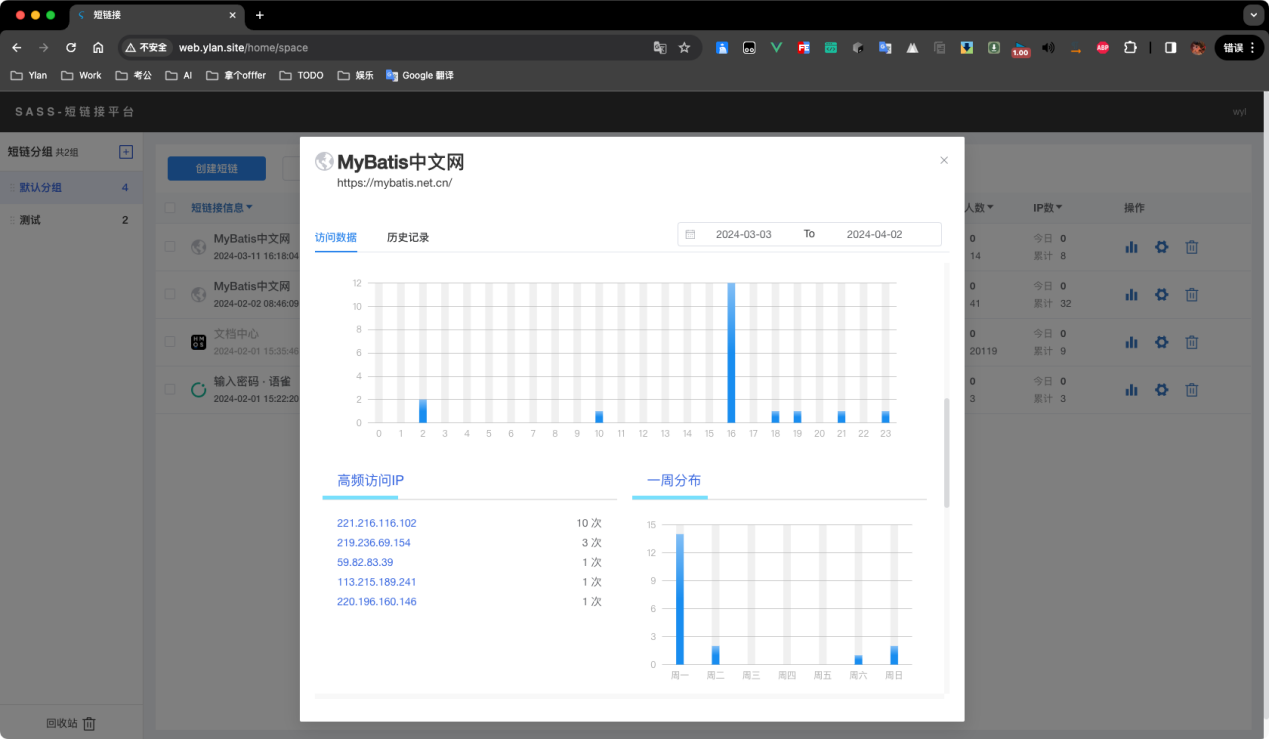


图5-11 系统短链接访问数据

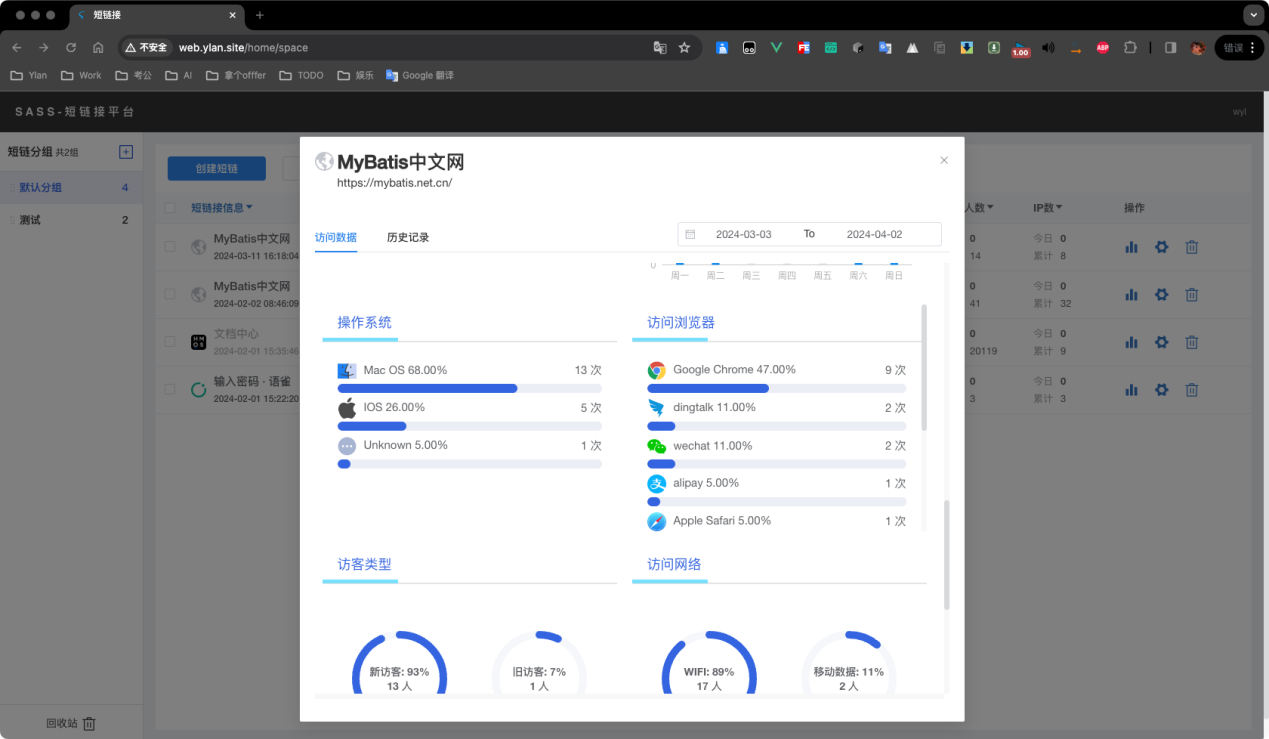


图5-12 系统短链接访问数据

## 5.3系统非功能实现

### 5.3.1系统日志和链路追踪功能实现

系统实现了运行日志和接口日志的持久化，其中运行日志以Log日志文件的形式进行记录，接口日志以数据库记录的形式进行记录，同时各模块间的调用链路用TraceID进行记录。这种实现方便我们在遇见线上问题时，能快速定位和解决问题，是系统维护的重要保证。

其中接口日志信息持久化的核心代码如下。

private Mono<Void> buildGatewayLog(ServerWebExchange exchange, GatewayFilterChain chain) {  
 ServerHttpRequest request = exchange.getRequest();  
 Route route = exchange.getAttribute(ServerWebExchangeUtils.GATEWAY\_ROUTE\_ATTR);  
 GatewayLogDO gatewayLog = new GatewayLogDO();  
 String traceId = MDC.get(TRACE\_ID);  
 gatewayLog.setTraceId(traceId);  
 String ip = NetUtils.getActualIp(request);  
 gatewayLog.setIp(ip);  
 String schema = request.getURI().getScheme();  
 gatewayLog.setSchema(schema);  
 String requestMethod = request.getMethod().toString();  
 gatewayLog.setRequestMethod(requestMethod);  
 String requestPath = request.getPath().pathWithinApplication().value();  
 gatewayLog.setRequestPath(requestPath);  
 String requestContentType = null;  
 MediaType mediaType = request.getHeaders().getContentType();  
 if (!Objects.isNull(mediaType)) {  
 requestContentType = mediaType.toString();  
 }  
 gatewayLog.setRequestContentType(requestContentType);  
 Date requestTime = new Date();  
 gatewayLog.setRequestTime(requestTime);  
 String targetServer = route.getId();  
 gatewayLog.setTargetServer(targetServer);  
 String routeConfig = JSON.toJSONString(route);  
 gatewayLog.setRouteConfig(routeConfig);  
 if (MediaType.APPLICATION\_FORM\_URLENCODED.isCompatibleWith(mediaType) || MediaType.APPLICATION\_JSON.isCompatibleWith(mediaType)) {  
 return writeBodyLog(exchange, chain, gatewayLog);  
 } else {  
 return writeBasicLog(exchange, chain, gatewayLog);  
 }  
}

其中运行日志信息持久化的核心配置如下。

<configuration scan="true" scanPeriod="60 seconds" debug="false">  
 <springProperty scope="context" name="springAppName" source="spring.application.name"/>  
 <property name="pattern"  
 value="[YlanShortLinkProject] ===> [%-5level]-[%d{yyyy-MM-dd HH:mm:ss.SSS}]-[%thread]-[%X{traceId}]-[%L-%c.%M]-[%msg] %n"/>  
 <property name="LOG\_HOME" value="logs"/>  
 <appender name="STDOUT" class="ch.qos.logback.core.ConsoleAppender">  
 <encoder class="ch.qos.logback.classic.encoder.PatternLayoutEncoder">  
 <pattern>${pattern}</pattern>  
 <charset>utf-8</charset>  
 </encoder>  
 </appender>  
 <appender name="FILE" class="ch.qos.logback.core.rolling.RollingFileAppender">  
 <Prudent>true</Prudent>  
 <rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy"> <FileNamePattern>${LOG\_HOME}/%d{yyyy-MM-dd}/YLanShortLinkProject-%d{yyyy-MM-dd}-%i.log</FileNamePattern>  
 <MaxHistory>30</MaxHistory>  
 <totalSizeCap>5GB</totalSizeCap>  
 <cleanHistoryOnStart>true</cleanHistoryOnStart>  
 <timeBasedFileNamingAndTriggeringPolicy class="ch.qos.logback.core.rolling.SizeAndTimeBasedFNATP">  
 <maxFileSize>30MB</maxFileSize>  
 </timeBasedFileNamingAndTriggeringPolicy>  
 </rollingPolicy>  
 <encoder class="ch.qos.logback.classic.encoder.PatternLayoutEncoder">  
 <pattern>${pattern}</pattern>  
 <charset>utf-8</charset>  
 </encoder>  
 </appender>  
 <root level="info">  
 <appender-ref ref="STDOUT"/>  
 <appender-ref ref="FILE"/>  
 </root>  
</configuration>

平台收集到的接口日志信息如图5-3所示。

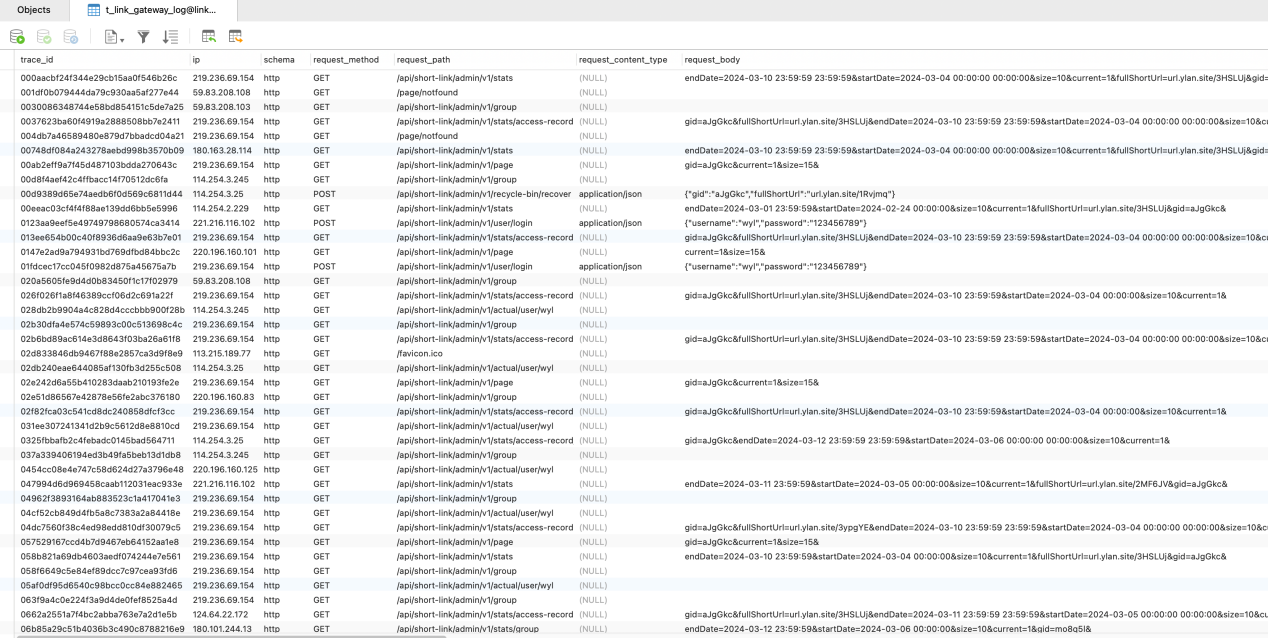


图5-3 接口日志信息

平台收集到的运行日志信息如图5-4所示。

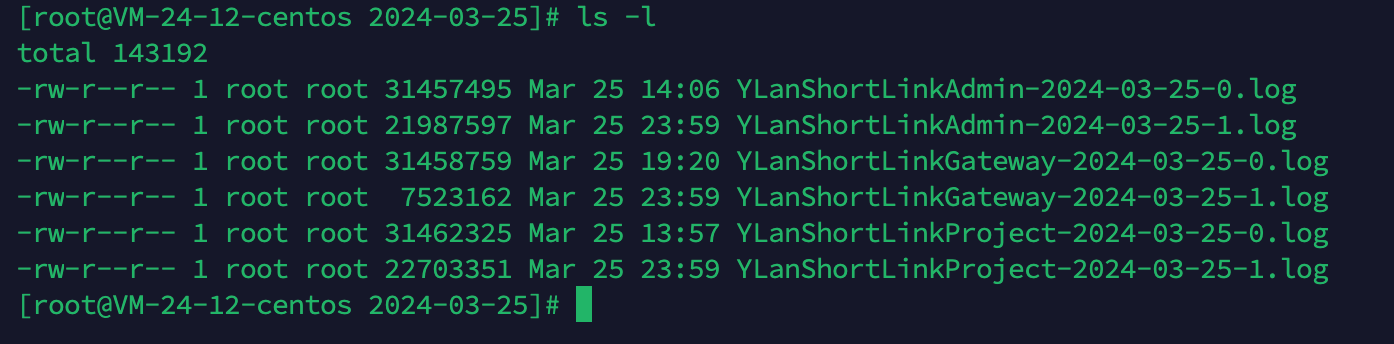


图5-4 运行日志信息

# 系统测试

## 6.1测试简述

系统测试是软件测试的重要阶段之一，旨在确保软件系统的功能、性能、可靠性、安全性和易用性等方面的质量。系统测试通过对整个系统进行全面、深入、模拟真实环境的测试，检查系统是否能够满足业务要求。

系统测试方法分黑盒测试和白盒测试。黑盒测试是基于系统功能需求和规格说明书，不考虑系统内部实现细节，对系统进行测试的一种测试方式。黑盒测试旨在检查系统是否能够正确地处理输入，输出结果是否符合预期，功能是否完备，以及系统性能是否满足要求等。黑盒测试可以有效地检测系统功能是否满足用户需求，但无法深入了解系统内部的实现细节和错误。白盒测试则是基于系统内部实现细节的测试方式。白盒测试旨在发现代码中的逻辑错误和代码漏洞等问题。白盒测试可以深入了解系统内部实现细节和错误，但需要更多的时间和精力。

## 6.2功能测试

### 6.2.1登录功能测试

用户使用平台时首先需要保证登录功能的可用，用户通过输入用户名和登录密码进行登录，之后可以正常进入平台。登录功能测试如表6-1所示。

表6-1登录功能测试

| 测试用例编号 | 测试用例-01 | | |
| --- | --- | --- | --- |
| 功能描述 | 平台的登陆功能 | | |
| 用例目的 | 检测平台的登录功能是否正常 | | |
| 前提条件 | 网络畅通 | | |
| 输入/动作 | 期望结果 | 实际情况 | 是否通过 |
| 输入账号密码进行登录 | 登陆成功 | 登陆成功 | 通过 |
| 登录后状态 | 进入主页 | 进入主页 | 通过 |

### 6.2.2短链接创建功能测试

用户登陆进平台后可以在一个分组内进行短链接的创建，需要输入原始链接、有效期、短链接分组等信息，是平台的核心功能。短链接创建功能测试如表6-2所示。

表6-2短链接创建功能测试

| 测试用例编号 | 测试用例-02 | | |
| --- | --- | --- | --- |
| 功能描述 | 平台的短链接创建功能 | | |
| 用例目的 | 检测平台的短链接创建功能是否正常 | | |
| 前提条件 | 网络畅通，用户已登陆 | | |
| 输入/动作 | 期望结果 | 实际情况 | 是否通过 |
| 输入原始链接和有效期 | 创建成功 | 创建成功 | 通过 |
| 成功之后状态 | 展示短链接列表 | 展示短链接列表 | 通过 |

### 6.2.3短链接跳转功能测试

用户收到分享的短链接后，可以扫码或点击然后即可跳转到原始链接。短链接跳转功能测试如表6-3所示。

表6-3短链接跳转功能测试

| 测试用例编号 | 测试用例-03 | | |
| --- | --- | --- | --- |
| 功能描述 | 平台的短链接跳转功能 | | |
| 用例目的 | 检测平台的短链接跳转功能是否正常 | | |
| 前提条件 | 网络畅通 | | |
| 输入/动作 | 期望结果 | 实际情况 | 是否通过 |
| 点击短链接 | 跳转成功 | 跳转成功 | 通过 |

## 6.3非功能测试

兼容性测试，通过使用不同浏览器(chrome、edge等浏览器)打开网页进行测试，发现系统在不同浏览器上都能够正常运行，页面展示和功能都可以正常使用。经过测试，系统兼容所有IE11以上的主流浏览器，不会出现页面错位、样式错乱、功能无法使用等问题。

性能测试，同时，也进行了性能测试，通过使用性能测试工具对系统进行了测试，发现系统的响应速度和页面加载速度都在合理的范围内。系统的交互界面设计简洁明了，用户体验良好，操作流程简单易懂，用户可以轻松完成相关操作。

在测试过程中，未发现系统存在明显的性能问题或用户体验问题，系统的非功能性表现较为稳定，满足了系统设计的相关需求。

## 6.4测试总结

当项目开发完成后，需要对系统进行测试，以确保开发的功能可以适应用户的需求，尽量规避用户在使用过程中，产生的不良情况。在测试过程中要考虑到各种突发情况，要将发现的漏洞及时反馈给开发人员，及时的修改。本系统经过测试，大部分功能都能满足使用者的需要，性能良好。

结束语

本平台历时三个月左右，所有的基础核心功能都已研发完毕，并进行了详细的系统测试，同时拥有完整的上线部署方案。本平台涉及到了市场调研、需求分析、系统设计、前端开发、服务端开发、系统测试、系统部署和运维，这也是我第一次对一个项目的完整生命周期有全局的把控。在这期间我遇见了很多难题，有需求分析上的，有技术选型上的，有技术实现上的，同时也有系统缺陷修复上的，我不断阅读各种技术资料，去各种网站寻求解决方案，最终完成了本平台的研发和部署上线。

本平台采用SpringCloud微服务架构，每个模块可以独立开发、维护和部署，大大加强了开发的灵活性。采用Kubernetes进行部署，保证了系统的高可用性。在技术选型上拥抱开源，追随热门技术。通过本项目，我结合了现代云原生技术和微服务架构，为用户提供了一个高可用、高可扩展性的短链接服务。

在开发本平台时，发现了自己的许多不足，今后要在实践中不断完善自己的编程知识，提升自己的编程能力。虽然本平台已开发完毕，但这还只是一个初始版本，后期还需要进行功能拓展和系统优化。我也会继续关注新技术的发展，不断优化和完善这个平台，为用户创造更大的价值。希望这个平台能为用户提供便捷、安全的短链接服务，为他们的工作和生活带来便利。

# 参考文献

1. 陆伟强.基于微服务的民机工业软件架构设计[J].计算机与现代化,2023,(07):73-78+126.
2. 邹佶汛.基于云平台的民航业信息系统运维自动化研究[J].互联网周刊,2024,(06):54-56.
3. 王耀暖.短网址技术在多维分析系统中的应用研究[D].浙江海洋大学,2023.DOI:10.27747/d.cnki.gzjhy.2023.000404.
4. 吴振海,何格新.Mysql快速全同步复制技术的设计和应用[J].湖南工程学院学报(自然科学版),2023,33(02):38-44+66.DOI:10.15987/j.cnki.hgbjbz.2023.02.005.
5. 陈鹏. 基于分布式缓存和消息中间件的选课系统的设计与实现[D].重庆大学,2020.DOI:10.27670/d.cnki.gcqdu.2019.002758.
6. 王国辉.VMware超融合架构技术在广播电台中的应用[J].电视技术,2022,46(09):148-151.DOI:10.16280/j.videoe.2022.09.040.
7. 杜俊,李金忠,何新武,等.基于Spring Cloud微服务架构的非物质文化遗产展示与学习系统的设计与实现[J].广东轻工职业技术学院学报,2024,23(01):14-21.DOI:10.13285/j.cnki.gdqgxb.2024.0003.
8. 时业茂,颜晓宏,刘卫.基于Spring Boot整合SSMP框架实现图书管理系统[J].电脑编程技巧与维护,2023(06):82-84.DOI:10.16184/j.cnki.comprg.2023.06.044.
9. 周傲英.互联网时代分布式关系数据库的有益探索[J].计算机研究与发展,2024,61(03):539.
10. 罗智. 面向分布式环境的任务调度研究与应用[D].重庆邮电大学,2023.DOI:10.27675/d.cnki.gcydx.2022.000485.
11. 李永.基于Windows平台的故障转移集群研究与应用[J].信息技术,2023(08):47-50.DOI:10.13274/j.cnki.hdzj.2023.08.009.
12. 杨成飞.我国省级工程建设项目审批管理系统中高可用MySql架构的部署与应用讨论[J].中国建设信息化,2022(24):73-75.
13. 杨洪.基于Redis的共享缓存分析[J].数字技术与应用,2024,42(01):110-112.DOI:10.19695/j.cnki.cn12-1369.2024.01.35.
14. Theo Combe,Antony Martin,Roberto Di Pietro．To Docker or Not to Docker: A Security Perspective[J]．Cloud Computing, IEEE,2016,3(5):54-62
15. Shuto Masuda,Fujun He,Akio Kawabata,Eiji Oki．Distributed Server Allocation Model with Preventive Start-Time Optimization against Single Failure[C]//IEEE International Conference on High Performance Switching and Routing.2020
16. 迟五一．中小企业新产品营销推广策略的分析与研究[J]．营销界（理论与实践）,2020,000(5):P.1-1
17. 郑卫斌,张德运,丁会宁,李继华,高磊．基于哈希表的高性能URL过滤器研究[J]．小型微型计算机系统,2005,026(2):178-180
18. 赵彦博.中间业务云平台的探索与使用[J].科技风,2023,(08):67-70.DOI:10.19392/j.cnki.1671-7341.202308023.
19. 崔娟,王伟民,冯继虎,李怀堂,王林柱．基于Nginx反向代理解决公网上服务跨域问题的研究[J]．现代信息科技,2023,7(8):79-82
20. 鞠鑫,丁松松,张俊杰．卫生健康行业数据安全标准建设方法论探索[J]．信息安全研究,2023,9(1):87-92

致谢

大学四年时光转瞬即逝，回想起刚入学时的青涩和懵懂，对知识的渴望和对未来的憧憬，如今经过大学的熏陶，我已经变的自信成熟。大学给了我对未来美好生活的希望，我将一步步的去追求。

首先，我要感谢我的导师张政老师，他既是我的论文指导老师也是我的社团指导老师。他在我完成毕业设计和论文的过程中给予了我悉心、细致的指导和支持。同时他对社团责任心满满，给社团提供了很多优秀的学习资源，时刻关心学生的情况。他是我学习上的领路人，让我在学习的道路上越走越远。

其次，我要感谢我的大学舍友们，我们一起在学习上、生活上互相帮助，共渡难关。我还要感谢我的社团同学们，我们在一起并肩作战、共同进步，让我感受到了团队的力量。这几年的点点滴滴令我难以忘怀，是我大学期间的最大财富。

然后，我要感谢我的家人和朋友们，是你们在我人生中的每一个阶段给予了无私的支持和鼓励。你们是我坚强的后盾和精神支柱，鼓舞着我不断前进。同时还有感谢生命中的那个她，在过往的日子里，我们有过矛盾，有过坎坷，但终究是快乐的。

最后，我要感谢我实习期间的每位同事，是你们的支持和帮助让我度过了这段特殊的时光。在此，我想对所有曾经帮助过我的同事们表达我最诚挚的感谢之情。愿我们未来的道路上继续携手并肩，共同努力，共创美好未来。

最后我想对自己说，做自己想做的事情，避免内耗去活着，是自在的，是自由的。

谨以此文，感谢你们一直以来的支持和鼓励！