**郑州轻工业学院**

本科毕业设计（论文）

题 目 基于自定义RPC框架及增量同

步的云笔记的设计与实现

学生姓名 韩俊涛

专业班级 软件工程（软件开发14-01）

学 号 541413260106

学 院 软件学院

指导教师(职称) 孙安宝（资深软件开发工程师）

邓璐娟（教授）

完成时间 2018年6月16日

郑州轻工业学院

**毕业设计（论文）任务书**

**题目 基于自定义RPC框架及增量同步的云笔记的设计与实现**

**专业 软件工程(软件开发) 学号 541413260106 姓名 韩俊涛**

**主要内容、基本要求、主要参考资料等：**

### 主要内容

借助于网络和云服务设计一款云笔记，系统的实现探索自定义RPC框架技术和增量同步节省资源的模式，云笔记实现以下功能和服务。采用Java和Vue编写，后台根据Netty和ZooKeeper构建的轻量级RPC框架来分模块进行编写，同时以RESTful API形式对外提供接口，供前端调用。在数据缓存方面使用Redis以AOP形式进行缓存，减轻数据库的访问的压力。前端采用Vue来构建单页面应用，同时使用增量同步技术来减少每次同步数据的数据量，减轻服务端的压力。用户可以新建文章分类和标签，让笔记更易管理，同时可以分享笔记。

### 基本要求

1. 设计和实现云笔记功能；
2. 主要业务功能通过功能测试和性能测试；
3. 项目能够运行并演示；
4. 完成毕业设计论文。

### 主要参考资料

**完 成 期 限： 2018年6月16日**

**指导教师签名：**

**专业负责人签名：**

**2018年 1 月 15 日**

基于自定义RPC框架及增量同步的云笔记的设计与实现

摘 要

随着网络的快速发展和智能设备的普及以及云存储解决方案的成熟，云笔记类应用越来越受到用户的重视，云服务的使用和存储方式也逐渐便捷和人性化。由于云笔记的服务都是有偿使用，本人本着对云笔记和新技术的兴趣以及服务于热爱技术的朋友们的目的，设计和实现了一款基于RPC框架和增量同步的云笔记。采用Dubbo作为服务治理框架，功能丰富，结合Zookeeper实现服务的注册与发现以及管控，实现远程过程调用与服务治理；采用微服务设计构建整个应用，数据同步时采用增量同步技术来节省资源。设计的云笔记具有记录、分类、检索、分享笔记等功能。系统的功能测试和性能测试说明设计的业务功能满足了需求，性能指标也达到了要求，可以给用户良好的体验，达到了任务目标，下一步可以部署到服务器上试用并改进推广。

关键词云笔记；RPC框架；微服务；增量同步

**Design and implementation of cloud notes based on custom RPC framework and incremental synchronization.**

**ABSTRACT**

With the rapid development of network and the popularization of intelligent equipment and the maturity of cloud storage solutions, the application of cloud notes has been paid more and more attention by users, and the way of use and storage is becoming more and more convenient and humanized. At present, the cloud notes on the market are very rich, can support a variety of document formats, multi - person collaborative editing, and an editor, multi - terminal user experience to attract users of cloud notes plays an important role. This paper focuses on the design and implementation of cloud notes, and combines the popular micro service architecture and search engine to build a simple distributed application, which makes the design of cloud notes lighter and more humanized.

**KEY WORDS** cloud note; RPC; Microservices; distributed service system

目录

[摘 要 I](#_Toc516584023)

[**ABSTRACT** II](#_Toc516584024)

[1 绪论 1](#_Toc516584025)

[1.1 研究背景 1](#_Toc516584026)

[1.2 云笔记国内外现状 1](#_Toc516584027)

[1.3 系统主要特点 2](#_Toc516584028)

[2 相关技术和理论 3](#_Toc516584029)

[2.1 RPC技术 3](#_Toc516584030)

[2.2 微服务架构 5](#_Toc516584031)

[2.2.1 微服务 5](#_Toc516584032)

[2.2.2 微服务设计 5](#_Toc516584033)

[2.3 缓存技术 6](#_Toc516584034)

[2.3.1 缓存 6](#_Toc516584035)

[2.3.2 Redis缓存 7](#_Toc516584036)

[2.4 Elasticsearch搜索引擎 7](#_Toc516584037)

[3 增量同步云笔记可行性分析 9](#_Toc516584038)

[3.1 技术可行性 9](#_Toc516584039)

[3.2 经济可行性 9](#_Toc516584040)

[3.3 社会可行性 9](#_Toc516584041)

[4 增量同步云笔记需求分析 10](#_Toc516584042)

[4.1 功能性需求 10](#_Toc516584043)

[4.2 非功能性需求 12](#_Toc516584044)

[4.2.1 安全性需求 12](#_Toc516584045)

[4.2.2 性能需求 12](#_Toc516584046)

[5 增量同步云笔记概要设计 14](#_Toc516584047)

[5.1 系统架构设计 14](#_Toc516584048)

[5.2 主要功能流程 15](#_Toc516584049)

[5.2.1 登录认证流程 15](#_Toc516584050)

[5.2.2 记录笔记流程 16](#_Toc516584051)

[5.2.3 检索笔记流程 17](#_Toc516584052)

[5.2.4 ELK数据同步流程 18](#_Toc516584053)

[5.2.5 分享笔记流程 20](#_Toc516584054)

[6 增量同步云笔记详细设计与实现 21](#_Toc516584055)

[6.1 增量同步云笔记详细设计 21](#_Toc516584056)

[6.1.1 数据库设计 21](#_Toc516584057)

[6.1.2 认证模块设计 23](#_Toc516584058)

[6.1.3 缓存模块设计 25](#_Toc516584059)

[6.1.4 搜索模块设计 26](#_Toc516584060)

[6.1.5 笔记模块设计 28](#_Toc516584061)

[6.2 增量同步云笔记实现 31](#_Toc516584062)

[6.2.1 认证模块实现 31](#_Toc516584063)

[6.2.2 笔记模块实现 34](#_Toc516584064)

[6.2.3 搜索模块实现 37](#_Toc516584065)

[6.2.4 缓存模块实现 37](#_Toc516584066)

[7 系统测试 39](#_Toc516584067)

[7.1 功能测试 39](#_Toc516584068)

[7.1.1 认证模块测试 39](#_Toc516584069)

[7.1.2 缓存模块测试 39](#_Toc516584070)

[7.1.3 搜索模块测试 39](#_Toc516584071)

[7.1.4 笔记模块测试 40](#_Toc516584072)

[7.2 性能测试 41](#_Toc516584073)

[7.3 本章小结 42](#_Toc516584074)

[结束语 44](#_Toc516584075)

[致 谢 45](#_Toc516584076)

[参考文献 46](#_Toc516584077)

# 1 绪论

## 1.1 研究背景

随着互联网的蓬勃发展，大量有用或者无用的信息填满了我们的生活，我们有时很难去捕捉一些对自己很有用却转瞬即逝的信息，有些信息即使记录下来又没有统一的工具进行管理，不方便信息的收集以及日后的查看。现在很多课程与知识大都迁移到互联网上，例如网络课堂、公开课这些面对面课程，更加需要一款使用方便、功能丰富的记录型工具来记录这些即时信息，最好提供一些分类的方式，可以让用户快速检索到自己需要的笔记。

现在市场上出现了很多类型的云笔记，这些云笔记功能丰富，提供了非常人性化的操作与界面，用户使用也十分方便。但这些云笔记都是商业软件，需要付费使用，并且不开源，对于那些想了解云笔记内部所用技术以及架构设计的开发人员来说，是无法直接得知这些技术细节的。针对云笔记的问题和需求，本项目在参考目前市面上的一些云笔记的基础上，结合目前比较热门的Dubbo以及Elasticsearch搜索引擎，由Vue和Element-UI构建前端页面，设计一款云笔记的开源实现，构建一个简单的分布式应用。

## 1.2 云笔记国内外现状

云笔记可谓是一种非重量级的云服务，它以云计算为基础，提供给用户跨时空、跨平台的数据信息管理功能[1]。目前云笔记可谓非常流行，各大厂家纷纷推出自己的云笔记产品，而且他们的云笔记产品各有自己的特色。

有道云笔记提供多种客户端，并且客户端操作十分方便也很美观，并且支持云协作功能。蚂蚁笔记开放源代码，开发人员可以看到该笔记的设计理念，代码结构等，同时可以提Issue和Pull Request参与笔记的开发当中，社区氛围浓厚。前段时间发布的腾讯文档功能也是十分强大，它提供与腾讯QQ、TIM、微信等无缝对接，一处编辑，多端同步，同时支持多人在线编辑，以及权限编辑和分享功能。在格式方面，支持Word文档、Excel表格等，编辑极其方便，十分强大。

对于国外的云笔记产品，目前比较为人所知的是Evernote和OneNote。Evernote，一款国外堪称最完美的云储存型笔记软件，其基本功能是信息获取和信息组织，它不仅具备强大的笔记捕捉功能，还采用了独特的分类方式，而且支持filter的自动整理、即时搜索[2]。Evernote 是比较强大好用的，后来此应用进入中国市场，推出适合大陆用户的云产品印象笔记，也是十分好用的。OneNote属于微软Office办公组件，功能自然十分强大，用户基数也很多，体验很好。

## 1.3 系统主要特点

该云笔记结合目前市场上的云笔记的特点与优势，运用目前热度较高的微服务架构，利用Dubbo来构建简单的分布式应用，并集成Redis作为缓存，包括登录认证信息缓存，笔记相关数据缓存等。前端采用Vue来进行渲染。主要有以下特点：

1. 系统模块化：将一些公共模块和系统核心模块分离开来，以RPC形式调用服务，便于系统功能的扩展和服务的发布。
2. 高效笔记：云笔记提供易操作的界面，使笔记记录更加高效和轻松。同时支持Markdown语法，因为Markdown几乎是所有人的最爱。
3. 知识管理： 云笔记提供了标签分类和文件夹分类的功能，不再让笔记管理杂乱无章，它将是笔记管理的利器。
4. 分享：云笔记集成分享和评论功能，并且评论功能接入Github的Issues系统，即两者评论同步，方便与Github好友分享知识、想法和经历。
5. 搜索：云笔记集成Elasticsearch搜索引擎，提供更快速，更精确的搜索服务。

# 2 相关技术和理论

## 2.1 RPC技术

远程过程调用(RPC)是构建分布式服务系统的重要技术[3]。它简单、灵活、强大[4]，通过网络，从远程计算机程序上请求服务，而不必了解底层网络技术的协议[5]。简单的理解，就是在不必知道调用细节的前提之下，调用远程计算机上运行的某个对象，使用起来就像调用本地的对象一样。RPC隐藏了底层网络通信细节[6]。目前流行的RPC框架有：Dubbo、gRPC等。

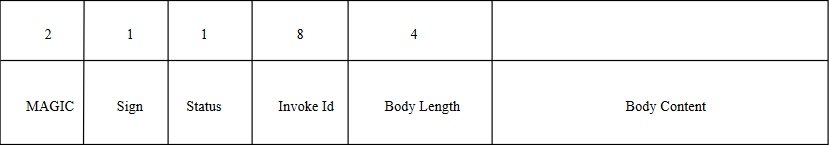
远程过程调用(RPC)是随着网络和分布式系统的发展而发展起来的[7]。当两个服务不在一个内存空间，即对于Java应用来说，不在同一个JVM内存里面，所以之间无法直接调用。这个时候需要通过网络来表达调用的语义和传达调用的数据，并尽可能使远程调用的语义尽可能接近本地调用的语义[8]。

RPC机制的实质是实现OSI七层模型中的会话层功能，它在两个试图进行通信的场点之间建立一条逻辑信道(即会话连接)，并利用这个信道交换信息，不用时就释放连接[9]。

在实际应用中，Client和Server之间的调用往往复杂的多，因为交互复杂的原因，考虑到网络传输、协议优化以及编解码技术等，如果对性能要求较高，还要考虑序列化以及反序列化技术。基于以上的问题，网络模块用基于异步的、事件驱动的Netty框架实现[10]。Netty是由JBOSS提供的基于NIO实现的Java开源框架,该框架提供异步的、事件驱动的网络应用程序框架和工具,用以快速开发高性能、高可靠性的网络服务端和客户端程序[11]。

如果采用Netty作为底层通信框架，那么还有许多问题需要考虑，例如心跳机制。在Client和Server保持TCP长连接的时候，由于网络的不稳定或者其他原因，Server端需要知道Client端是否存活，以便继续进行交互，此时就需要引入心跳机制。Client端需要定时发送特殊数据包给Server端，Server然后回应特殊报文，从而保持TCP长连接。在实际生产环境应用中，还需要有断线重连机制，需要检测通道是否可用。

由于Netty底层采用TCP/IP协议传输数据，而TCP/IP协议发送的数据是以数据包形式发送的，当我们传输大量数据时，接收方需要解析这些数据包，如果发送方和接收方之间没有约定好的协议，那么数据没法进行解析了。所以我们自定义协议来完成这样的需求，同时解决解决TCP 粘包，拆包问题。那么对于一个数据包，需要考虑协议开始标志、消息标志位、消息id、消息长度以及要发送的数据。图2-1是自定义简单的RPC协议。协议开始标志是一个十六进制的魔数（MAGIC），代表消息的开始，消息标志位（Sign）, 用来表示消息类型，消息id（Invoke Id）唯一标识消息，要发送的消息放在消息体（Body Content）内。定义好RPC协议后，需要考虑数据的编码与解码，编码与解码如果要实现完整的话，还是很复杂的，设计到心跳机制检测，ACK机制，TCP 粘包，拆包问题等，好在这里面的一些问题Netty已经有了较好的解决方案，十分方便。



**图2-1 自定义的RPC协议**

由于Java本身自带的序列化机制效率低下，所以采用Google开源的Protobuf进行序列化和反序列化，后期如果需要其他特殊的序列化需求，需要将序列化和反序列化扩展为接口，同时实现几种序列化和反序列化，根据配置来选择具体的序列化与反序列化方式，这样会使系统的扩展到能力更强。

随着分布式系统的快速增长，相对于目前热门的基于Http协议的微服务架构来说，RPC虽然说相对复杂，无论是协议还是服务治理，但RPC 服务的自动化程度比较高，能够实现强大的服务治理功能，和语言结合更友好，性能也十分优秀。现在已经有了很多可以跨语言调用的框架，例如Thrift、gRPC等，弥补了以前RPC框架不能跨语言调用的不足，但这类的RPC框架没有服务治理等相关的功能。传统的RPC框架例如Dubbo功能丰富，提供远程调用以及服务发现及治理功能，同时提供管控台来对服务降级等操作。但缺点是语言耦合度较高，跨语言支持难度较大[12]。相信随着RPC框架的优化以及功能的增强，会将跨语言调用以及强大的服务治理发现功能整合起来，更为高效和强劲。

本项目采用Dubbo作为服务治理框架。Dubbo是Alibaba开源的分布式服务框架，它功能完善，使用极其方便。对于RPC框架来说，服务的注册与发现占有相当重的地位，当服务越来越多时，服务地址的管理会非常混乱和困难，所以Dubbo可以结合Zookeeper实现服务的注册与发现，即实现一个注册中心，使服务的位置透明，摆脱手工管理服务地址的重大问题，动态的注册与发现服务。

Dubbo包含大量特性，包含集群容错、负载均衡、泛化引用、本地伪装、服务降级、优雅停机等特性，这些特性有些成熟度很高，已被大量用于生产环境，一些特性成熟度不高，主要用于测试和新功能研发。同时支持多种配置方式，包括传统的XML配置和基于注解的配置方式，使用起来非常方便，灵活性很大。在注册中心的选择上，可以选择多种注册中心，不过基于Zookeeper的注册中心成熟度高，主要用于生产环境。

## 2.2 微服务架构

### 2.2.1 微服务

近几年来微服务在互联网行业热度很高，我们或多或少有所耳闻。它提倡将单一应用程序划分成一组小的服务，服务之间互相协调、互相配合，最终构建整个应用[13]。随着互联网的快速发展，轻量、快速、高效等符合互联网的发展趋势与需求，而传统应用属于单体式的应用，模块间耦合性太大，后期维护困难，同时代码的体积过大，会导致部署困难。因为只要修改一点代码，就需要重新上传代码部署，这显然问题很大，十分不符合当今互联网的特性，所以微服务才这么受欢迎。

微服务可以降低系统的部署难度和提高容错。在微服务架构下，每个微服务都是独立部署的，这样做可能增加了一个系统的部署次数，但降低了发生错误的概率，提高容错。因为当一个项目很大的情况下，逻辑复杂，模块间依赖严重，由于没有拆分模块，每次改动都需要大量的测试，浪费人力财力，同时不可避免造成不可预料的问题，线上排错也是十分难受可见微服务是以后发展的趋势。

### 2.2.2 微服务设计

从微服务的概念来说，其理念是十分宽泛的，很难有统一的架构和框架，但因为有大量的企业采用微服务的架构，自然也就形成了一些比较公认的设计原则，业界关于微服务应用有以下四个设计原则：

1. AKF扩展拆分

AKF概念主要阐述的是微服务的拆分方式，比如常见的负载均衡，数据的横向扩展，或者根据业务的不同进行微服务拆分，具体拆分的方式根据项目的大小、业务的需求进行拆分，过度的拆分会导致业务过度分离，造成项目管理的混乱。

1. 前后端分离

以前的单体应用前端代码和后端代码混合在一起，部署在一个应用里面，代码体积庞大，这样就违背微服务的设计理念，所以前后端分离就很必要。现在前端各种框架层出不穷，而且现在的前端框架的功能很强大，对构建大型单页面应用支持很好。

1. 无状态服务

将有状态的服务转换为无状态的服务，比如原来我们的应用在一个系统中进行缓存、Session等数据的管理，不利于系统的扩展，现在可以将这些独立出来做成计算类服务，认证、缓存等做到统一管理，这样系统就可以进行动态伸缩和扩展，而不必考虑因业务扩展导致的一系列的问题。

1. REST风格

RESTful API 作为现在应用常用的API设计风格已经不新鲜了，例如最近很火的代码托管网站Github就是使用Rest风格设计的API，这种架构风格将所有事物都抽象为资源，所以对资源就会有增删改查操作，而且所有操作都应为无状态的。在项目中，仅仅使用RESTful API是远远不够的，因为RESTful API只提供JSON数据，没有了视图层，将视图渲染交给了前端，这就带来了许多问题。比如我们以前在写JSP界面的时候，会把数据放到request里面，把登录的用户信息放在session里面，因为jsp本质上也是servlet，所以可以在界面直接拿到这些数据，现在前后端一分离，怎样对用户的身份进行识别就是首要考虑的问题。

以前我们是通过session来进行对用户的身份验证，由于http协议是无状态的，所以我们需要在服务器端将用户的信息保存下来，这份登录信息会在响应时传递给浏览器，浏览器将其保存在cookie中(Java中的jsessionid),下次再请求时就能识别了。

现在我们的API可能不只是在浏览器上调用，比如app等也需要调用，这是如果只用session的话就有局限性了，所以我们可以用 Token 进行身份鉴权，由于Token我们可以根据自己的需要进行自定义，只要API提供方和调用方约定好如何生成Token和解析Token，更加轻量化，扩展性更强。

## 2.3 缓存技术

### 2.3.1 缓存

当我们项目的数据量不是很大的时候，缓存的作用不是很明显，对数据的增删改查直接操作数据库就可以了，不会对数据库产生多大的压力。但当我们的应用的数据量非常大的时候，一些数据更新不是很频繁，并且查询所占的比重明显比增删改操作占的比重大，此时如果每次都从数据库里面去读这些不常变化的数据，就会非常浪费数据库的资源，因为从连接数据库到断开连接这一过程所占资源较多，系统还会响应慢，这明显是有问题的。

现在市面上的缓存系统多种多样，包括开源的以及各个公司自研的，种类繁杂，提供的功能也很丰富。而有些缓存系统则需要几个缓存系统组合到一起使用，充分发挥各个缓存系统的特性，提高缓存系统的整体性能。

### 2.3.2 Redis缓存

Redis是一种以Key-Value存储结构的内存数据库，所以Redis的查询速度是很快的，我们可以用它来作为缓存系统的存储数据的工具。在考虑设计缓存系统时，需要思考缓存的设计方案以及其合理性，例如缓存key的生成，缓存过期时间的设置，序列化机制的实现，缓存是否采用AOP机制，尽量减少对业务的侵入性，是否支持集群配置，扩展能力强不强？

上面的需求都是一个可用的缓存系统必须要面对的挑战，缓存系统好不好用，高效不高效，其实取决于一些核心的问题，只有把这些问题解决的完美，那么系统的可用性也就提高了。

例如如何为每一条缓存记录生成唯一标识，以便进行快速检索，那么就需要根据不同缓存key的设计有所不同。有些系统可能根据每次生成的sql语句作为key，查询的结果作为value存入Redis，有些系统可能会根据每次调用的方法的名称以及参数来生成key等等，其实key的生成多种多样，主要能够快速生成和快速识别即可。

单节点Redis来构建缓存系统其实只是测试级别，如果数据量很大的情况下，就需要搭建Redis集群。现在Redis-Cluster已经非常稳定了，性能很高。

## 2.4 Elasticsearch搜索引擎

Elasticsearch是一个基于Lucene的分布式搜索和分析引擎，它能够以接近实时的速度处理数据[14]。Elasticsearch提供全文搜索、结构化数据实时统计、数据分析等强大功能，同时与Logstash、Kibana框架结合构建分布式数据分析处理ELK平台，可以将MySQL、系统日志、Kafaka等产生的数据增量同步到Elasticsearch中，性能十分强劲。

由于Elasticsearch提供了简单好用的RESTful API，应用接入Elasticsearch搜索引擎是十分简单的，同时支持多种语言的客户端，无论是Java还是C#，都可以与Elasticsearch进行交互，以完成数据的搜集与检索功能。

云笔记集成Elasticsearch的强大的搜索特性，利用Logstash将存储在MySQL中的数据增量同步到Elasticsearch中，以便前台检索数据。考虑到Elasticsearch里面的数据是以JSON格式存储的，而不是在传统数据库中以字段形式保存，这就要求在进行增量同步之前，需要先将数据转化为JSON格式的数据。Elasticsearch与传统数据库相比，概念有了本质的变化，它引出Index、Type、Document、Filed、Mapping等概念，这些概念可以与传统数据库的概念相对应，所以才可以将存在传统数据库中的数据同步到Elasticsearch中，不过需要定义Mapping关系，然后利用Logstash提供的自定义配置功能，进行数据同步。

利用Logstash进行数据同步需要在配置文件中定义好Logstash三大模块的内容，这三大插件分别是input、filter、output。input代表数据的来源，可以是日志文件、数据库、Kfaka等，不过不同的来源定义的配置不一样，在云笔记系统中，数据来源是MySQL数据库，所以需要定于MySQL的连接配置，执行的查询SQL的路径以及执行策略。filter是Logstash最复杂的，同时也是有用的插件，因为它提供了许多插件，用来过滤数据。它之所以如此重要，是因为无论是日志文件、MySQL还是其他的数据来源，数据格式繁杂，或者不符合导入到Elasticsearch中的格式要求，所以需要将数据格式进行过滤和转换，所以filter的重要性不言而喻。output代表数据输出，该模块集成了大量的输出插件，可以将数据输出到指定文件、Elasticsearch，也可以输出到指定网络地址。在该模块中还可以做相关预警配置，比如将预警信息发送到微信和email等，功能也是十分强劲。

数据同步分全量和增量，对于小数据量的系统而言，全量同步消耗不了多少资源，似乎也没增量同步的需求，但对于数据量超大的系统而言，全量同步往往不太现实，这是就要考虑到增量同步。何谓增量同步？就是每次同步数据的时候不是全部同步，而是只同步变化的数据，这样就会节省资源，数据更新也会更及时、快速。在Logstash同步数据时，推荐使用增量同步。对于云笔记系统而言，由于只是同步MySQL中的数据，所以在设计表结构的时候，需要增加一个gmt\_modified字段来表示每条数据的更新时间，Logstash中有一个sql\_last\_value表示最后更新时间，每次读取数据库表时更新此值，这样只有对于MySQL中更新后的数据才会被Logstash同步，这样就做到增量同步。

# 3 增量同步云笔记可行性分析

## 3.1 技术可行性

现在开源软件十分流行，开源社区非常活跃。云笔记采用了大量的开源软件，得益于这些自由免费的开源的软件，系统的构建与运行才会如此容易。这些开源技术大多是基于一些开源协议，这些开源协议给使用者一定的使用权限来使用这些开源的代码。有些技术更新迭代非常快，社区活跃，比如Elasticsearch搜索引擎，代码贡献者已超过上千，且有中英文社区和书籍提供免费支持，所以在技术上是没有问题的。

## 3.2 经济可行性

云笔记由于使用了很多的开源软件，且本身又开源，所以在软件的使用上不会产生费用。在系统的设计上，需要一些工具、纸张等资源的支持，需要花费一些费用。对于应用的部署、发布，由于需要部署在服务器上，所以需要购买服务空间，这需要花费一定的费用。项目部署后，用户可以免费使用。

## 3.3 社会可行性

云笔记是在开源协议下进行发布的，使用者在遵守国家相关法律的前提下可以自由使用，复制以及修改。

用户在注册为云笔记的用户时，需要检测用户信息的有效性、完整性，非法用户不能注册为云笔记用户。在注册时用户需要同意相关协议才能进行注册，不同意不能进行注册，这些协议有关用户的隐私管理方式、不良信息处理办法等相关信息。

在处理不良信息的方式上，系统内置了不良信息过滤功能，对于色情、反动等信息都会进行过滤，云笔记在保证用户自由言论的同时，不会危害国家、社会以及个人。云笔记尊重每一个用户的创作版权，所以创作版权均归用户所有，而且云笔记提倡原创，这是对其他创作者的尊重。

# 4 增量同步云笔记需求分析

## 4.1 功能性需求

对于云笔记系统而言，用户最关注的功能无疑是如何快速方便地写笔记，对笔记的高效检索和分类，以及分享自己所写的笔记。这些是云笔记的核心功能，也是本系统主要关注的功能。云笔记用户的用例图如图4-1所示。



**图4-1 云笔记用户用例图**

从图4-1可以看出，游客需要注册才能拥有对笔记操作的权限，因为云笔记需要根据不同的用户来归纳和区分笔记的，所以用户需要系统的认证授权。

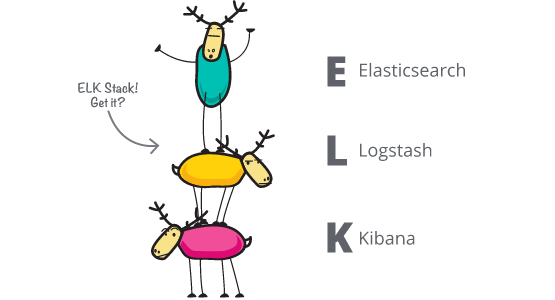
认证与授权：用户第一次使用云笔记时，需要在云笔记注册帐号，然后进行登录认证。本应用的认证和授权中心将数据保存在Redis中，并采用Token机制，抛弃了传统Java Web中的Session机制，这样做的好处是不仅可以做单点登录，还可以利用Token机制的特点，使信息传递更加安全。利用Token进行认证，可以将认证和授权与业务逻辑进行剥离，形成一个单独的认证与授权中心，也与特定技术进行分开，减少没必要的逻辑耦合，有利于系统的扩展。

记录笔记：记录笔记作为云笔记最主要的功能，应该考虑设计是否合理，体验是否良好，操作是否简便。现在很多人都喜欢用Markdown语法记录文字，是否提供Markdown语法支持等。在本应用中，支持Markdown语法，用户可以用Markdown来记录文字，并且提供实时渲染和预览功能。对于有上传图片需求的用户，本应用提供插入图片功能，使记录笔记功能更加丰富。

分类笔记：用户可以记录很多笔记，这些笔记可能有一些共同的属性，但大部分所包含的内容差别还是很大的，如何管理这些相似性不大的笔记呢？所以云笔记引入了文件夹和标签的概念。每个注册的用户有几个默认的文件夹来存放笔记，如果用户认为这些文件夹种类太少不足以标识笔记，用户可以自己新建文件夹。对于文件夹来说，支持层级文件夹是十分有必要的，因为一个文件夹的属性可能是其他文件夹属性的集合，若不支持层级文件夹，对文件夹的管理也会是个大问题，会显得十分混乱。对于标签而言，是对文件夹功能不灵活的补充，因为一个笔记只能属于一个文件夹，但它可能会有很多不同的属性，所以引入标签功能十分有必要。对于检索笔记来说，标签也是一个很好的归类标志。

分享笔记：自己写的笔记有些是私密的，不想别人看到，但有些笔记记录的事情很有趣，很想与别人分享，而笔记默认别人是不可以访问的，所以就需要增加分享功能。云笔记提供分享功能，如果用户不想让某一篇笔记再进行分享了，或者别人想浏览这份笔记需要一定的权限，例如为笔记加入阅读密码功能，别人需要输入密码进行查看，这样不仅做到了共享性，还做到了私密性、安全性，这样将两者结合在一起，体验很好。

检索笔记：云笔记需要提供检索功能来检索笔记。对于小数据量而言，可能直接按照筛选条件查询数据库速度不会很慢，但对于数据量稍大或者很大的情况而言，直接查询数据库是不太好了，特别是那种模糊查询，数据库没有多大的优势，此时需要利用搜索引擎来检索数据。本应用采用Elasticsearch作为搜索引擎，Logstash作为数据收集平台，Kibana作为数据分析和可视化平台，共同构成ELK数据处理平台，为数据检索提供快速且稳定的服务。Elasticsearch里面的数据来自MySQL存入的数据，利用Logstash增量式地将更新的数据导入Elasticsearch中，这些数据将会转化、聚合，以供用户快速且准确地检索。图4-2为ELK平台的形象架构图。



**图4-2 ELK平台形象架构图**

## 4.2 非功能性需求

### 4.2.1 安全性需求

对于一个普通的应用而言，保证用户的信息不轻易泄露，用户在该应用产生的数据不轻易被其他不法分子盗取，这对于系统的安全性提出了很高的要求。云笔记提供的功能主要是面向用户的，用户会在该应用上产生大量的数据，这些数据会贯穿应用的界面，后台服务，搜索服务、数据库层以及缓存层，所以需要系统的各个方面来保证用户数据的安全性以及系统整体的安全性以及健壮性。

1. 数据存储安全： 用户数据主要存放在传统数据库、NoSQL、搜索引擎中，需要保证这些系统整体的安全性，不轻易被攻击，提高这些系统的访问权限以及数据同步时处理业务的安全防护。
2. 个人隐私的保护：现在用户越来越重视自己在互联网的隐私，用户的个人信息在系统中应该得到加密防护，使用比较安全的加密算法使用户的个人隐私不轻易被泄露。
3. 数据过滤防护：用户在云笔记会保存大量的数据，这些数据有可能是违反相关法律法规的，或者包含色情、反动以及一些不健康的信息，这些数据都需要按照相关法律法规予以拦截处理，保证整个系统的健康性。

### 4.2.2 性能需求

性能这个主题一直都是软件技术人员不懈追求的目标，对于一些高性能、高可用的架构一直都在推陈出新，因为性能会直接影响用户体验、系统并发量以及整体吞吐量。对于云笔记而言，虽然是简单的分布式应用，但对性能还是有要求的。系统一旦有并发量，就需要考虑很多因素，例如并发资源共享，数据的完整性等，这些都是必须要解决的问题，因为这些问题会直接影响并发系统的可用性。为了提高数据库数据的正确性，防止脏数据的产生，可以考虑锁机制，比如经常被使用的数据库的乐观锁、悲观锁。在代码层面，考虑对临界资源的操作的敏感性，同样使用同步锁机制，编程语言为这些提供了支持，功能也十分强大。如果有更大的需求，比如在分布式中要考虑临界资源的竞争情况，可能需要采用分布式锁来提供支持。

本系统对于性能的指标是：在不考虑网络严重延迟的情况下，单次请求时间不能超过一秒；页面渲染不能超过2秒，且不能出现渲染错误，数据错乱情况；对于数据的更新操作需要保证其正确性，不能因为多个用户同时更新导致更新失败或者更新数据错误。

# 5 增量同步云笔记概要设计

## 5.1 系统架构设计

系统的总体架构图如图5-1所示。



**图5-1 系统总体架构图**

本云笔记大量采用模块化的概念，对于前端界面而言，采用Vue构建的单页面应用就是完全基于模块来进行编写的，例如文件夹模块、标签模块、显示笔记列表模块以及笔记的编辑与显示模块等，将这些彼此关联性不大的模块独立出来，大大减轻了功能之间的耦合度，对于模块之间的数据传递，由于已经将这些模块剥离开来，就不能再因为数据的原因有过多的联系，所以这里利用Vuex状态管理模式来管理模块之间需要传递的数据。Vuex的功能就好像是一个数据存储的中转站，对于模块而言，不需要关心数据从哪里来，更不用关心数据怎么产生的，直接从Vuex里面拿来用即可，因为如果Vuex里面有数据，就会是模块想要的数据。至于数据的流向和更新，其实都是经过Vuex来关联各个模块，一个模块产生数据，一个模块更新数据，一个模块利用数据，总之都是操作Vuex里面的数据，实在是大大地提高了模块的独立性，对整个系统的扩展性的提高大有裨益。

对于后端而言，将公共代码和公共配置独立为模块，同时将接口模块也独立出来，利用Dubbo框架进行服务治理，将以前Service层统统做成服务，这样就可以将服务与API层剥离开来，可以进行单独部署，远程调用。这样服务独立出来后，可以细分多个服务，有些服务规模较大，实现比较复杂，例如搜索服务，更应该独立出来，将搜索服务单独部署，因为Elasticsearch会涉及到集群等分布式概念，这样会减轻系统的部署压力，更会增强系统的扩展能力。

## 5.2 主要功能流程

### 5.2.1 登录认证流程

非注册用户注册云笔记后，即可开启登录认证流程。由于本应用抛弃了传统的基于Session的认证，而采用基于Token的认证机制，这里主要包括Token的生成、存储、发送与检测等。如图5-2所示。



**图5-2 登录认证活动图**

1. 用户在云笔记系统进行请求时，需要将Token发送给服务器。
2. 服务器端会检测Token的有效性，有效则系统认证成功，进行下一步操作。
3. 如果认证不通过，用户需要重新进行登录。
4. 检测用户和密码的有效性，无效则继续进行登录操作。
5. 用户名和密码检测通过后，服务器会生成Token，用户在请求时需要将此信息发给服务器。

这里有一点需要注意，用户登录云笔记后，在进行后续操作时，都需要将Token信息放在Header的Authorization中，服务器会检测该操作的合法性，如果Token无效，将会进行重新登录操作，继续进行认证。

### 5.2.2 记录笔记流程

用户在登录云笔记后，可以进行新建笔记操作。活动图如图5-3所示。



**图5-3 记录笔记活动图**

1. 由于云笔记引入了文件夹-标签管理形式，所以在创建笔记之前需要选择文件夹，在设计文件夹的时候，考虑到用户可能不想选择文件夹，或者忘了选择文件夹，所以每个用户都有一个默认的文件夹，没有选择文件夹的时候就以默认文件夹为当前笔记的文件夹。
2. 用户接下来需要选择笔记的类型，云笔记主要提供Markdown编辑器和富文本编辑器，选择笔记的类型以渲染不同的编辑器，最终保存的文档格式也不同。
3. 接下来用户就可以进行笔记的编写了，填写好笔记的一些基本信息，这里用户可以添加标签，支持添加多个标签，并且可以动态删除。
4. 保存笔记到数据库中。

### 5.2.3 检索笔记流程

云笔记提供强大的检索功能，用户可以输入要检索的数据进行快速检索，这得益于Elasticsearch强大的搜索功能。活动图如图5-4所示。

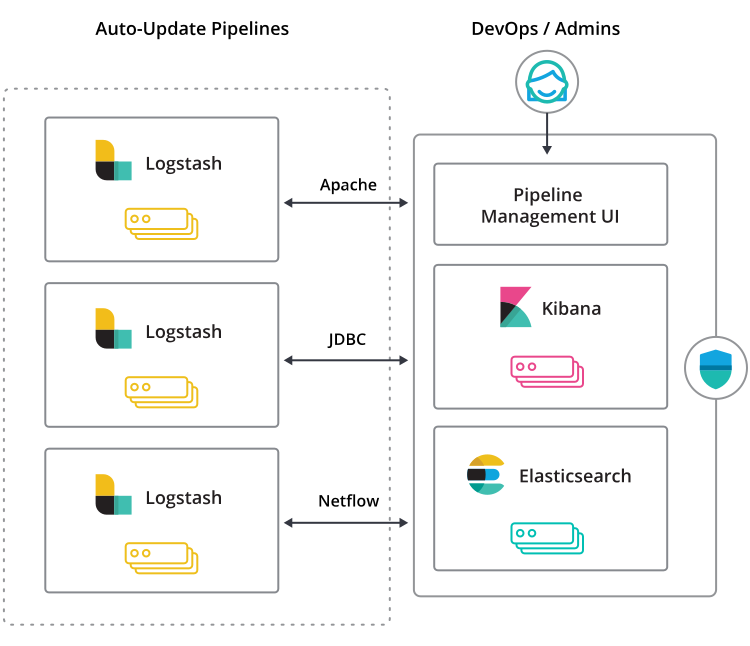


**图5-4 检索笔记活动图**

1. 用户首先要输入检索关键字。
2. 对搜索关键字进行检测。
3. 将数据发给服务器，服务器对数据进行处理。
4. 通过调用Elasticsearch Client将数据发送给Elasticsearch集群，进行数据的检索。
5. 将检索结果返回给用户。

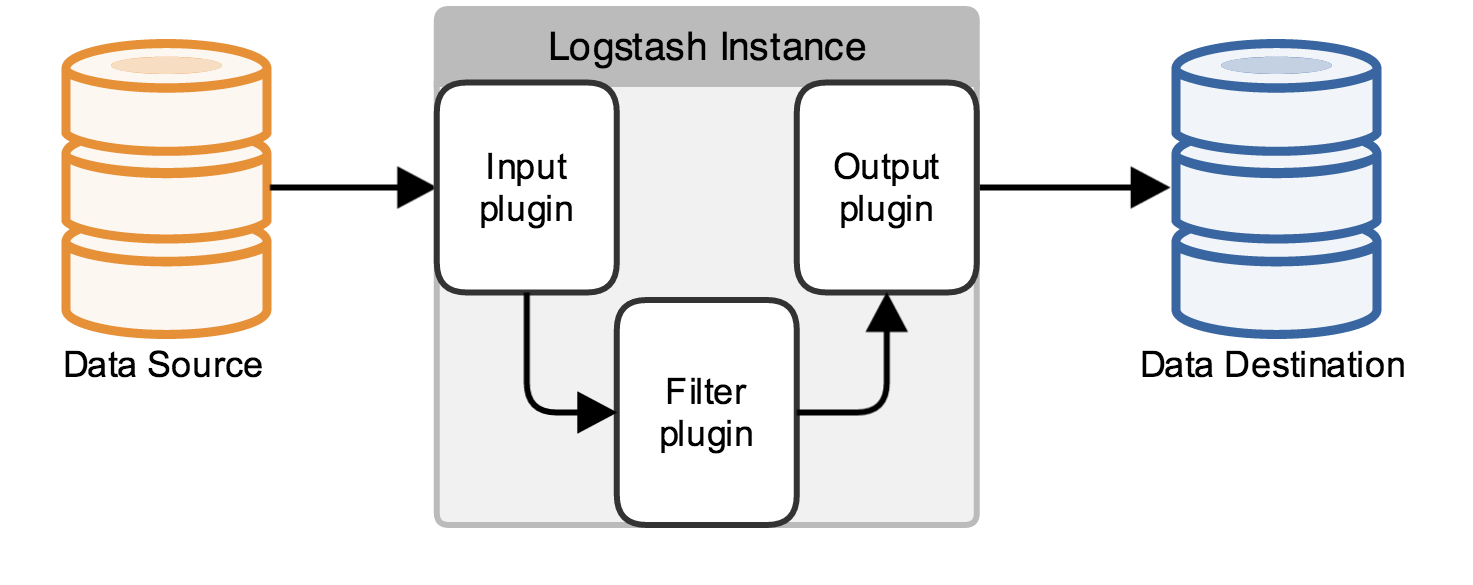
### 5.2.4 ELK数据同步流程

云笔记提供检索笔记的前提就是Elasticsearch中需要存储用户已发布的云笔记，而云笔记是保存在MySQL数据库中，而没有直接保存在Elasticsearch中，所以要支持检索笔记，那么就需要将数据从数据库同步到Elasticsearch中，完成这一步操作的是Logstash，图5-5为Logstash的工作流程。



**图5-5 Logstash工作流程**

在本应用中，需要将MySQL数据库中的数据通过JDBC形式同步到Elasticsearch中，所以我们需要自己编写Logstash的input、filter、output三大插件的内容（详细内容见下一章），来进行的数据的同步，如下图5-6所示。对于本应用而言，Data Source为MySQL数据库，Data Destination为Elasticsearch。



**图5-6 Logstash工作流程**

1. 首先通过input插件与数据库建立连接，执行sql查询语句，将数据从数据库查询出来。
2. 然后由filter插件进行数据的过滤和处理，因为数据库表中的数据和将要同步到Elasticsearch中的数据会有很大差别，需要进行专门的处理。
3. 最后由output插件决定数据的输出位置和格式，这里直接输出到Elasticsearch中。

### 5.2.5 分享笔记流程

用户可以分享自己的笔记给其他用户。活动图如图5-7。



**图5-7 分享笔记活动图**

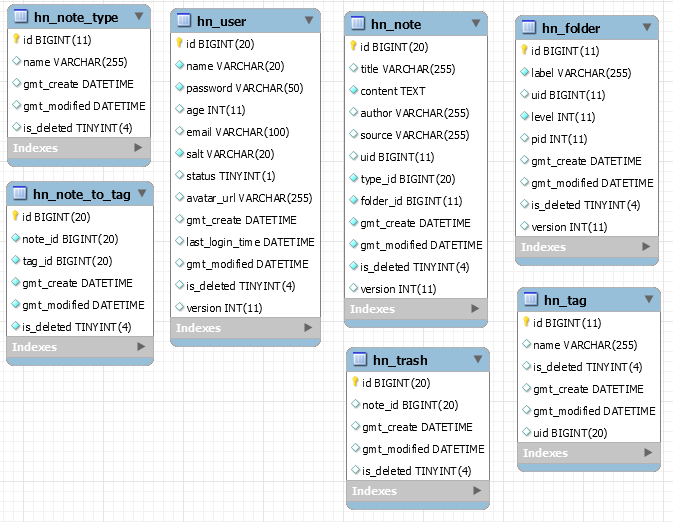
1. 用户选择某一篇笔记。
2. 然后用户点击分享按钮。
3. 系统会自动渲染云笔记，生成分享页面。
4. 初始化评论页面。

# 6 增量同步云笔记详细设计与实现

## 6.1 增量同步云笔记详细设计

### 6.1.1 数据库设计

云笔记经过对整体架构的设计以及对需求分析和概要设计进行仔细研究，并且对表中的字段以及字段的名称、类型、约束条件、默认值、说明等属性都做了详细的论证与设计，最终将云笔记所需要的表确定了下来。数据库结构模型如图6-1所示。考虑到系统对并发的支持，所以本项目采用了数据库乐观锁（Optimistic Locking）机制来保证数据的正确性。在关键表中额外添加一个数据版本（version）字段，默认值为1，每次更新时读取此字段，若此字段与读取值一致，那么予以更新，这个字段同时增加1，否则失败。



**图6-1 数据库结构模型**

下面将云笔记的所需要的表以及相关属性列举出来。

用户表包含用户id、名称、密码、年龄、邮箱、盐值、状态标识、头像URL等字段。具体表结构如表6-1所示。

**表6-1 用户表**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 名称 | 类型 | 约束条件 | 默认值 | 说明 |
| id | bigint(20) | 无重复 |  | 用户唯一主键标识 |
| name | varchar(20) | 不为空 |  | 用户名称 |
| password | varchar(50) | 不为空 |  | 密码 |
| age | int(11) | 可以为空 |  | 年龄 |
| email | varchar(100) | 不为空 |  | 邮箱 |
| salt | varchar(20) | 不为空 |  | 盐值，加密时使用 |
| status | tinyint(1) | 不为空 | 1 | 状态标识 |
| avatar\_url | varchar(255) | 可以为空 |  | 头像URL |
| gmt\_create | Datetime | 不为空 |  | 创建时间 |
| last\_login\_time | Datetime | 不为空 |  | 上次登录时间 |
| gmt\_modified | Datetime | 不为空 |  | 更改时间 |
| is\_deleted | Tinyint | 不为空 | 0 | 删除标识 |
| version | Int | 不为空 | 1 | 乐观锁版本号 |

笔记表包含笔记id、标题、内容、作者、来源、盐值、用户id、类型id、文件夹id等字段。具体表结构如表6-2所示。

**表6-2 笔记表**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 名称 | 类型 | 约束条件 | 默认值 | 说明 |
| id | bigint(20) | 无重复 |  | 笔记唯一主键标识 |
| title | varchar(20) | 不为空 |  | 标题 |
| content | varchar(50) | 不为空 |  | 内容 |
| author | varchar(20) | 可以为空 |  | 作者 |
| source | varchar(100) | 不为空 |  | 来源 |
| uid | bigint(20) | 不为空 |  | 用户id |
| type\_id | bigint(20) | 不为空 | 2 | 类型id |
| folder\_id | bigint(20) | 不为空 |  | 文件夹id |
| gmt\_create | Datetime | 不为空 |  | 创建时间 |
| gmt\_modified | Datetime | 不为空 |  | 更改时间 |
| is\_deleted | Tinyint | 不为空 | 0 | 删除标识 |
| version | Int | 不为空 | 1 | 乐观锁版本号 |

文件夹表包含文件夹id、文件夹名称、用户id、层级标识、父级文件夹id等字段。具体表结构如表6-3所示。

**表6-3 文件夹表**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 名称 | 类型 | 约束条件 | 默认值 | 说明 |
| id | bigint(20) | 无重复 |  | 文件夹唯一主键标识 |
| label | varchar(50) | 不为空 |  | 文件夹名称 |
| uid | bigint(20) | 不为空 |  | 用户id |
| level | int(11) | 不为空 | 0 | 层级标识 |
| pid | bigint(20) | 不为空 | 0 | 父级文件夹id |
| gmt\_create | Datetime | 不为空 |  | 创建时间 |
| gmt\_modified | Datetime | 不为空 |  | 更改时间 |
| is\_deleted | Tinyint | 不为空 | 0 | 删除标识 |
| version | Int | 不为空 | 1 | 乐观锁版本号 |

标签表包含标签id、标签名称、用户id等字段。具体表结构如表6-4所示。

**表6-4 标签表**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 名称 | 类型 | 约束条件 | 默认值 | 说明 |
| id | bigint(20) | 无重复 |  | 标签唯一主键标识 |
| name | varchar(50) | 不为空 |  | 标签名称 |
| uid | bigint(20) | 不为空 |  | 用户id |
| gmt\_create | Datetime | 不为空 |  | 创建时间 |
| gmt\_modified | Datetime | 不为空 |  | 更改时间 |
| is\_deleted | Tinyint | 不为空 | 0 | 删除标识 |

### 6.1.2 认证模块设计

前文已经提到过云笔记的登录认证模块基于Token的认证机制，抛弃了传统基于session的认证方式。基于Token的认证机制对于开发者更加透明，因为登录认证模块已经被独立出来，Token的生成、检测、存储等操作开发者都需要自己编写，所以扩展能力很强。对于本应用而言，Token的相关操作皆是由自己编写的，认证的顺序图如图6-2所示。



**图6-2 认证顺序图**

本应用生成Token采用JWT加密方式，JWT总共分Header、Payload、Signature三部分。Header主要包含两部分，分别是Token的类型，这里选择JWT，另一部分声明加密的算法，通常直接使用 HMAC SHA256，然后将Header进行Base64加密。Payload是关于实体（通常是用户）和附加元数据的声明，这部分是存放数据的地方，比如用户id什么的，然后将有效Payload用Base64进行编码。Signature代表签名部分，要创建签名部分，必须要有已经编码的Header、Payload、一个密匙（secret）和加密算法，这里采用HMAC SHA256算法将三者进行加密，形成完整有效的Token。

下图6-3是登录认证拦截器的一部分实现代码，在该代码片段中主要拦截了用户的请求，如果该次请求有认证的标识，那么就需要进入认证流程。对于未拥有Token的用户，系统要求该用户进行登录操作，然后后台服务会按照上面的Token生成机制生成Token并存入Redis，并将其返回给浏览器，浏览器会将其存入Cookie中，当下次访问后台服务的时候，需要将Token放在HTTP的Header头Authorization中，后台会利用拦截器检测Header头Authorization中信息，取出Token，然后开始解析Token，获取存放在其中的信息，再根据此信息从Redis中取出Token信息，与Header头中的Token进行信息比对，判断Token的有效性，进而验证是否已经登录。



**图6-3 登录认证拦截器**

### 6.1.3 缓存模块设计

云笔记采用Redis来存储缓存数据。对于设计缓存模块而言，由于目前的缓存系统不需要很复杂，而需要更加方便、更加优雅的方式来缓存数据，所以经过仔细设计与分析，决定采用AOP形式来实现缓存。

首先需要考虑哪些类的方法需要缓存数据、删除缓存，为了标识这些方法，我们需要自定义注解，这里直接定义两个注解：@RedisCache 和@RedisEvict，@RedisCache代表这个方法需要缓存，@RedisEvict代表这个方法的缓存需要清除。接下来就需要写一个切面还实现缓存的存入与更新。这里直接使用Aspectj实现APO拦截。切面类图如下6-3所示。

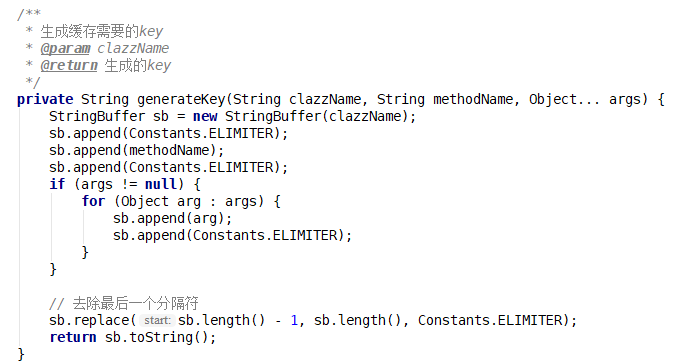


**图6-3 RedisCacheAspect类图**

缓存的顺序图如图6-4所示。对于缓存的存入功能而言，需要先生成缓存key。考虑到两次查询调用的类名、方法名和参数值相同，我们就可以确定这两次查询结果一定是相同的（在数据没有变动的前提下），部分代码如图6-5所示。因此，我们可以将这三个元素组合成一个字符串做为key, 就解决了标识问题。接下来有个key，然后去Redis查询，判断缓存是否命中。如果缓存命中，那么将数据反序列化，将其返回；如果缓存未命中，那么去数据库查询数据，然后将数据进行序列化，这时需要判断是否设置了超时时间，如果没有设置，那么默认无限期；如果设置了，那么对数据设置时间。这里的序列化为由实体或者集合转为JSON，反序列化为由JSON转为实体或者集合。

上面只是一种缓存key的生成机制，其实key的生成机制会有很多种，比如由用户指定缓存key等。根据缓存key生成机制的不同，可以将缓存key的生成抽象出来，根据用户的不同设置来动态地选择生成key的方式，这样使缓存设计更加灵活，扩展性更强。

**图6-4 Redis Cache顺序图**



**图6-5 生成Cache Key**

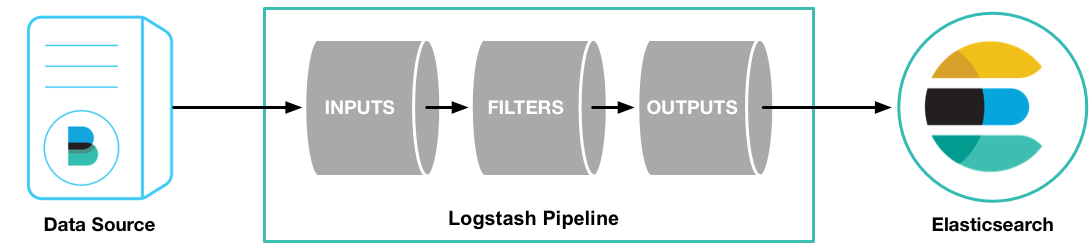
### 6.1.4 搜索模块设计

搜索模块因为采用了Elasticsearch作为搜索引擎，所以需要分两部分来设计，一部分是数据检索部分，另一部分是数据同步部分。数据检索需要依赖同步过来的数据，所以先介绍同步数据部分。搜索顺序图如图6-5所示。



**图6-5 搜索顺序图**

Logstash作为ELK平台的一部分，它具有实时同步数据的能力。Logstash包含input、filter和output三大插件。输入插件从数据源获取数据，过滤器插件根据用户指定的数据格式修改数据，输出插件则将数据写入到目的地[15]。如图6-6所示。



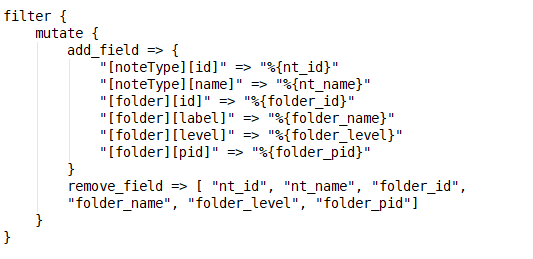
**图6-6 Logstash同步数据流程**

首先需要在Logstash源码目录下新建hnote-mysql.config配置文件，在这个配置文件中配置三个插件的内容。

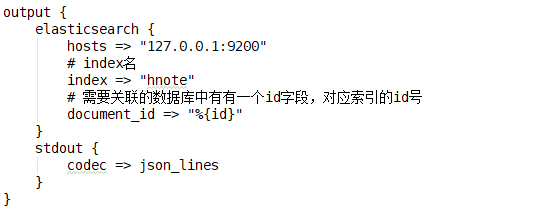
输入插件可以将一个数据源的数据同步到Logstash 管道中。云笔记是将MySQL中的数据读取到 Logstash 管道中，所以只需配置MySQL相关信息即可。具体需要配置以下信息：数据库连接信息、用户名和密码、驱动jar包的位置、驱动类型、分页相关配置、执行策略、查询sql路径、索引类型等信息。其中路径尽量配置成相对路径，因为在Linux和Windows中路径不一致，会造成部署出错。因为还需要用到增量同步技术，所以在查询语句中需要加入判断条件，判断数据更新时间大于上一次Logstash同步时间（:sql\_last\_value）。



过滤器插件可以对同步到Logstash 管道中的数据进行分析与处理，并且提供丰富的插件。这里使用Mutate插件。Mutate允许用户对字段进行更改。用sql查询语句查出的结果是平铺的，没有任何结构而言，而存入到Elasticsearch中的数据类型是JSON格式，而是多级节点的JSON。例如从数据库查询出来的一条记录包含笔记的基本信息、笔记的类型、笔记的文件夹，而类型和文件夹不能直接平铺存到Elasticsearch中，而是需要构造为对象，形成节点。鉴于此需求，就需要用到logstash-filter-mutate的add\_field和remove\_field。例如对于folder\_name字段，在add\_field中可以以"[folder][label]" => "%{folder\_name}"形式转换，看起来和二维数组，folder作为节点名称，label作为其属性，这样就问题就解决了。然后利用remove\_field将folder\_name这个无用字段移除，以免将无用字段转化。



输出插件这里的输出位置为Elasticsearch，所以只需要配置Elasticsearch相关信息即可。具体需要配置以下信息：Elasticsearch的地址和端口、索引名称以及document\_id。这个document\_id代表需要关联的数据库表的主键，对应Elasticsearch索引id。



配置完Logstash，接下来就需要设计数据检索部分。因为后台语言用的是Java，所以就直接引入Elasticsearch RestHighLevelClient，编写Elasticsearch配置文件，然后通过Spring以JavaConfig形式加载配置信息，编写Elasticsearch客户端的工厂类，获取RestHighLevelClient对象，最后编写Elasticsearch搜索客户端，提供搜素服务。当API层调用搜索服务时，可以指定匹配类型，常用的有全文匹配（match）、紧邻搜索（match\_phrase）、精确匹配（term）。部分实现代码如图5-222所示。



### 6.1.5 笔记模块设计

笔记模块主要涉及对笔记、标签、文件夹等增删改查操作以及对笔记的分享。由于整个项目采用前后端分离设计，后端只提供RESTful API，前端调用相应的API来对页面进行渲染。

对于后端而言，目前提供文件的上传和下载功能，文件夹相关API、笔记相关API、标签相关API，回收站相关API，这些API皆是围绕笔记进行设计的。文件夹API、笔记API和标签API提供各自的增删改查功能。回收站主要对删除的笔记进行管理。

笔记API类图6-7所示。包括获取最新笔记列表、模糊查询笔记（Elasticsearch 检索入口API）、根据标签id检索笔记、根据笔记id获取单条笔记信息、新建笔记、删除笔记和更新笔记。此时会发现关于笔记的搜索API相对来说设计的比较多，这是因为云笔记以浏览笔记为主要设计主体，让用户以各种方式能够检索到笔记，提高了云笔记整体的检索能力，而不至于因为找不到想要的笔记而渐渐失去对云笔记的兴趣，这也是云笔记的设计宗旨。



**图6-7 笔记类图**

文件夹API类图如图6-8所示。由于文件夹的表设计比较特殊，它设计到父子文件夹的关联关系，从数据库查出来的文件夹信息是平铺的，并没有层级结构，而前台数据渲染时需要层级结构，此时就需要将平铺的数据转化为具有层级结构的JSON结构，这操作本来是放在后端处理的，后来想想在前端处理也是可以的，遂将此递归转换操作放在了前端，只不过原理一致。



**图6-8 文件夹类图**

标签API类图如图6-9所示。由于标签与笔记为多对多关系，删除笔记的标签只需删除标签与笔记的关联记录即可。这里还提供了根据标签id查询笔记的API, 又给笔记检索多出了一种方式。



**图6-9 标签类图**

文件API类图如图6-10所示。上传功能类图主要是前端编辑器上传的图片，这里直接调用第三方接口将图片存储到图片服务器上，下载功能主要是根据笔记的类型动态生成相应的文件，供用户下载和浏览。



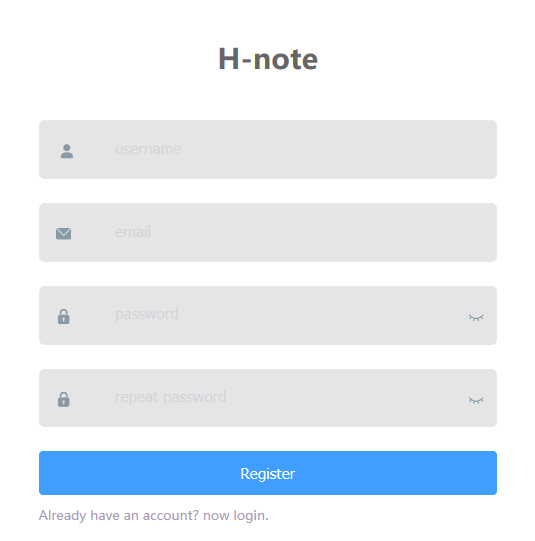
**图6-10 文件类图**

对于前端而言，由于选用了Vue全家桶，完全进行模块化编写，并利用Vuex和Vue Router对数据和路由进行管理，减少了模块之间的依赖，彼此之间独立性更强。前端主要分为API、模块、三方库、路由、Vuex 等部分。主界面主要有三个模块构成，分别是文件夹和标签模块，笔记列表和搜索模块，笔记编辑和展示模块。这些模块依赖Vuex和路由。

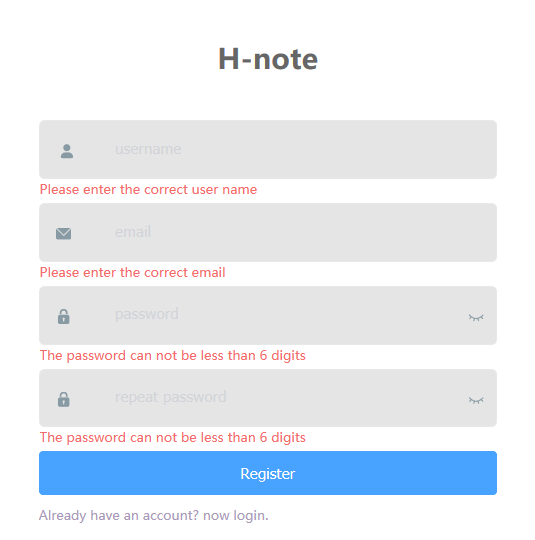
## 6.2 增量同步云笔记实现

### 6.2.1 认证模块实现

用户首次使用云笔记时需要注册帐号，注册新用户需要填写用户名、邮箱、密码、重复密码信息，如果信息验证正确，用户可以用已注册的帐号信息进行登录。注册页面如图6-11所示。在注册时如果信息填写不正确，页面会有相应的提示，此时无法继续注册，需要将不符合要求的信息改正后再继续注册。如图6-12所示。

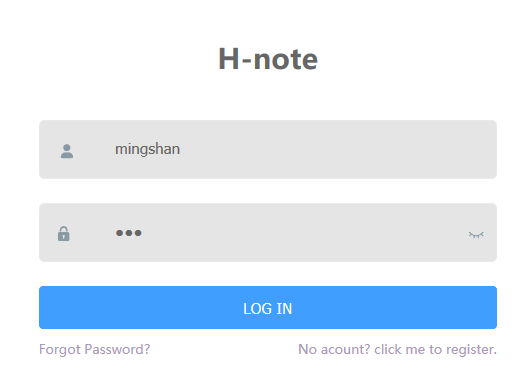


**图6-11 注册页面**

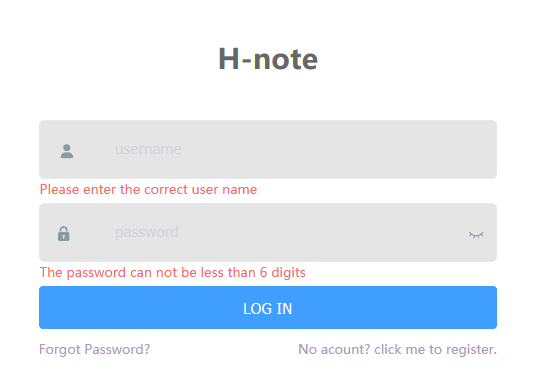


**图6-12 注册检测页面**

用户注册后即可进入登录页面进行登录。登录界面如图6-13所示。用户在注册页面输入用户名和密码后，进行检测，检测页面如图6-14所示。



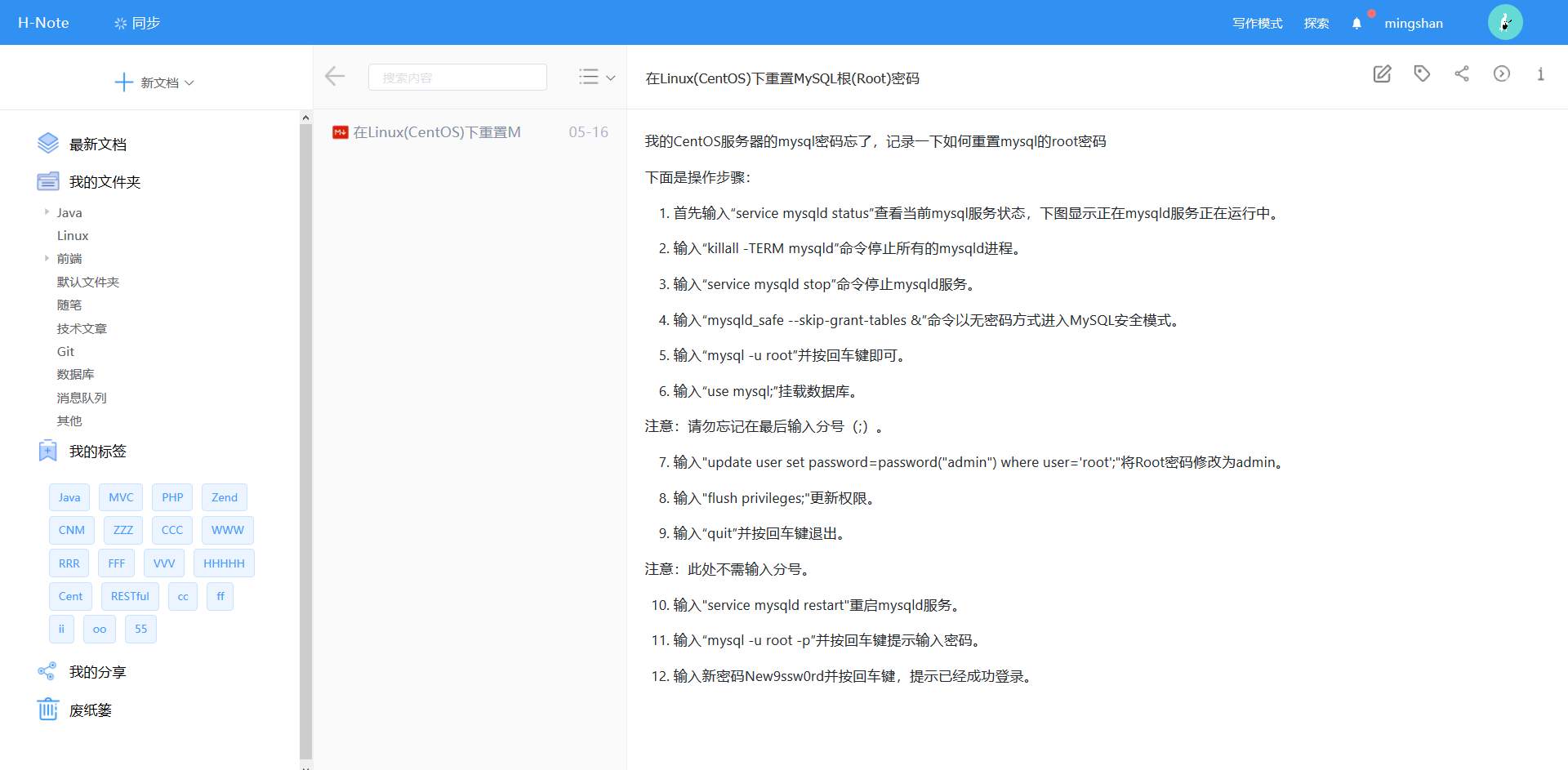
**图6-13 登录页面**



**图6-14 登录页面**

### 6.2.2 笔记模块实现

系统登录成功后，就会进入云笔记主界面。如图6-15所示。主界面包括头部区域和主内容区域。头部区域包含LOGO、写作模式、通知、用户名称以及头像。主内容区域包括左中右三部分。左侧边栏上部分为创建文档、文件夹等的按钮，用户可以点击按钮展开下拉框，进行更多操作。左侧边栏下部分主要有最新文档、我的文件夹、我的标签、我的分享、废纸篓这几栏。主内容区域中间这一部分包含两小部分，一部分为导航和搜索框，下面为笔记列表的展示。主内容区域右边为笔记编辑与展示区域。



**图6-15 云笔记主页面**

当用户点击某一云笔记时，右边区域会自动渲染笔记详细信息，然后点击最左侧编辑按钮，进入编辑状态，如图6-16所示。点击标签图标，在标题栏下方会展开标签列表，用户可以动态的增删标签。



**图6-16 云笔记编辑页面**

用户若想将笔记移动到另一个文件夹，那么只需要在该笔记点击鼠标右键，弹出选择框，选择移动到选项，弹出选择框，如图6-17所示。在该弹出框中会显示选择的笔记的类型图标、标题，以及当前用户创建的所有的文件夹。用户只需点击想要移动的文件夹，然后点击确定按钮即可将笔记移动到新文件夹。

用户可以创建新文件夹，只需在弹出框中填写信息保存即可。如图6-18所示。假如用户没有选择父文件夹，那么创建的文件夹就作为根文件夹，否则文件夹就作为已选择文件夹的子文件夹。最下方会有已选择父文件夹的提示，用户可以将其删除，最后可以点击确定按钮保存文件夹。



**图6-17 更换文件夹页面**



**图6-18 新增文件夹页面**

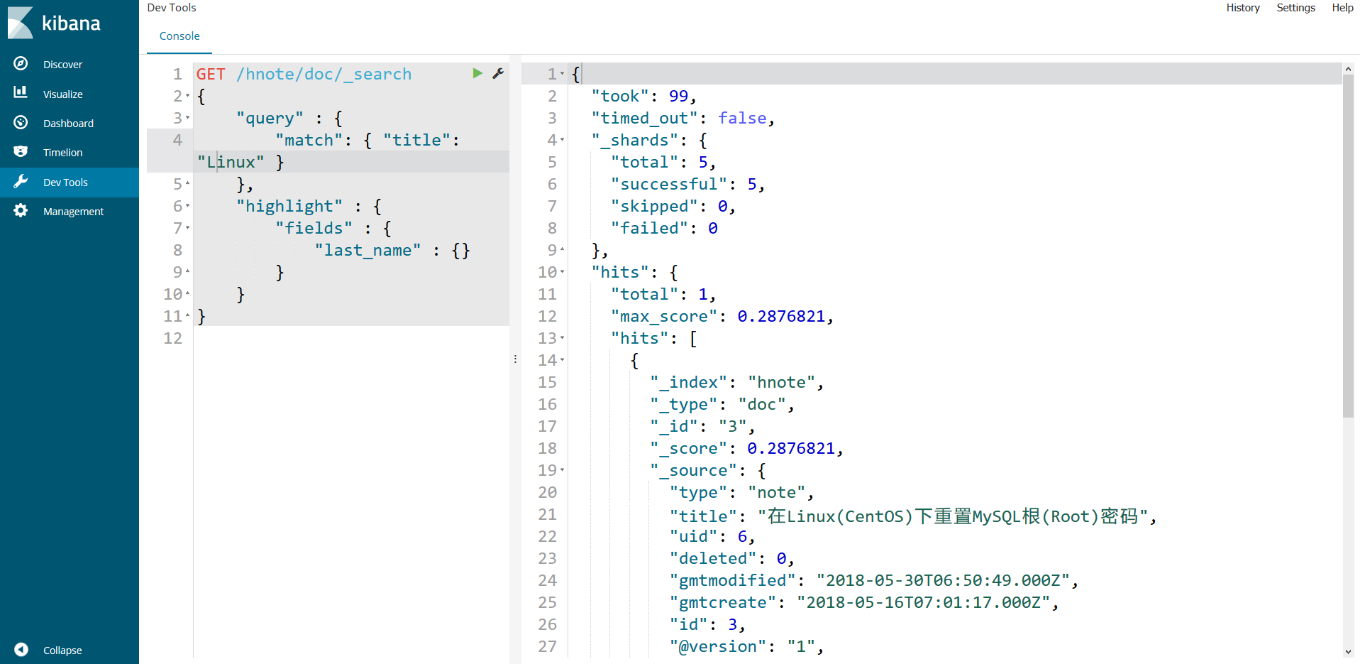
### 6.2.3 搜索模块实现

进入云笔记的主界面，在搜索框中输入想要搜索的内容，按下回车键后将会向后台发起搜索请求，然后将搜索结果显示在下方列表中。如图6-19所示。



**图6-19 搜索页面**

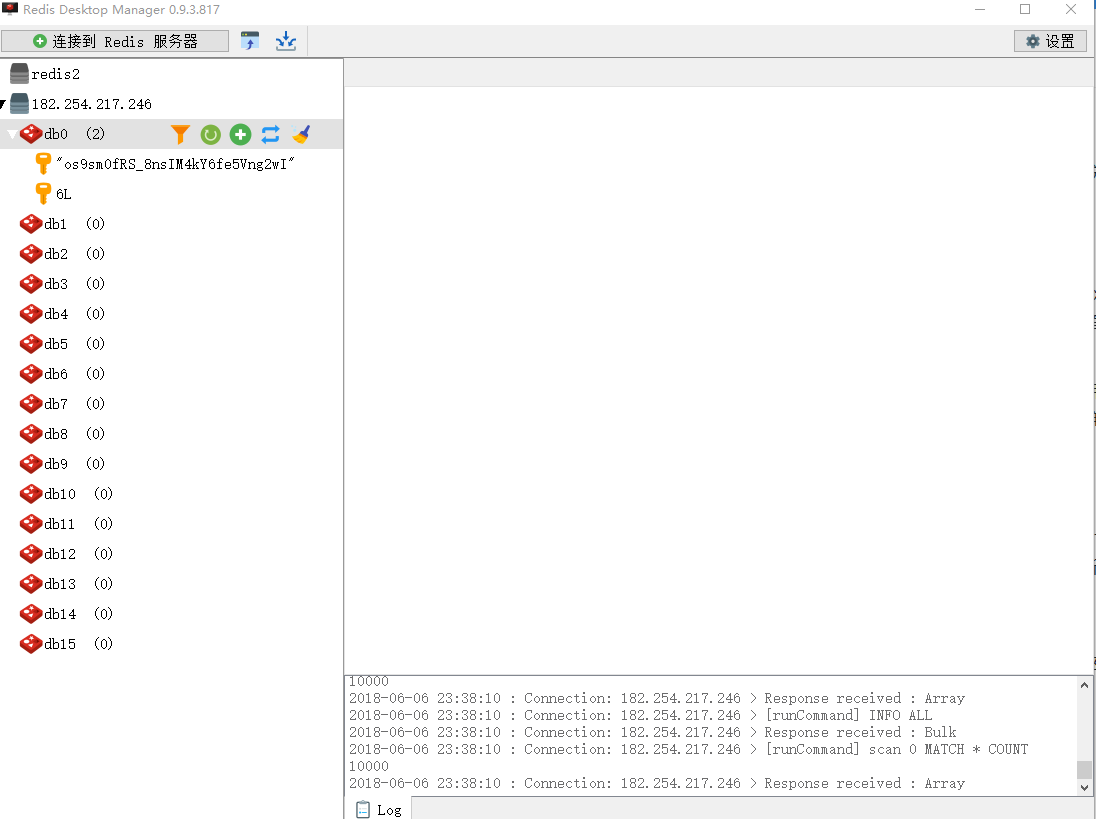
如果想详细查看存入到Elasticsearch的数据信息，可以启动Kibana，然后在Dev Tools页面进行查询，如图6-20所示。



**图6-20 查询索引页面**

### 6.2.4 缓存模块实现

用户认证信息还有与业务相关的数据都缓存在Redis中，用Redis桌面客户端可以进行查看缓存在Redis中的数据。如图6-21所示。



**图6-21 Redis数据展示页面**

# 7 系统测试

## 7.1 功能测试

### 7.1.1 认证模块测试

登录模块的测试包含对输入数据验证的测试、Token有效期测试、登录测试等，这里以登录认证进行举例，如表7-1所示。

**表7-1 登录认证功能测试**

|  |  |
| --- | --- |
| 测试用例描述 | 登录认证 |
| 测试数据 | 已注册用户 |
| 测试步骤描述 | 打开浏览器输入mingshan.me/hnote/login进入登录页面。 |
| 预期测试结果 | 用户名或密码错误时提示错误原因，正确时进入到mingshan.me/note/detail。 |
| 实际测试结果 | 用户输入mingshan和123，可以进入主界面。输入mingshan和2334，提示错误信息。 |

### 7.1.2 缓存模块测试

缓存模块的测试包括缓存数据测试、缓存有效期测试、清除缓存数据测试、更新缓存数据测试等，这里选取缓存数据测试为例，如表7-2所示。

**表7-2 缓存数据功能测试**

|  |  |
| --- | --- |
| 测试用例描述 | 缓存数据 |
| 测试数据 | 用户名和密码 |
| 测试步骤描述 | 1. 清除Redis中key为6的信息 2. 打开浏览器输入mingshan.me/hnote/login进入登录页面。 3. 输入mingshan和123，进入到系统主界面，此时打开Redis，查询key为6的信息 |
| 预期测试结果 | 可以查询到key为6的用户信息 |
| 实际测试结果 | 用户进入登录页面，输入mingshan和123后，查询Redis中，发现有数据。 |

### 7.1.3 搜索模块测试

搜索模块测试包括输入关键词（包含中英文）搜索测试，测试步骤如表7-3所示。

**表7-3 搜索功能测试**

|  |  |
| --- | --- |
| 测试用例描述 | 搜索测试 |
| 测试数据 | 关键词（中英文） |
| 测试步骤描述 | 在输入框中输入关键词“Linux”，然后回车。 |
| 预期测试结果 | 列表区域会显示一条笔记记录。 |
| 实际测试结果 | 在搜索框中输入Linux，列表区域会显示一条笔记记录。 |

### 7.1.4 笔记模块测试

笔记测试模块包含浏览笔记测试、创建笔记测试、修改笔记测试、删除笔记测试、恢复笔记测试、分享笔记测试等，这里主要测试创建笔记和分享笔记，首先是测试创建笔记，测试步骤如图7-4所示。其次是分享笔记测试，测试步骤如图7-5。

**表7-4 创建笔记功能测试**

|  |  |
| --- | --- |
| 测试用例描述 | 创建笔记测试 |
| 测试数据 | 笔记基本信息 |
| 测试步骤描述 | 1. 选择创建markdown选项，点击进入创建笔记页面。 2. 输入标题和内容，点击保存。 3. 点击最新笔记查看记录。 |
| 预期测试结果 | 最新笔记列表会显示出刚才所新家的笔记。 |
| 实际测试结果 | 在新建页输入笔记标题和内容，点击保存，最新笔记列表会显示出刚才所新建的笔记。 |

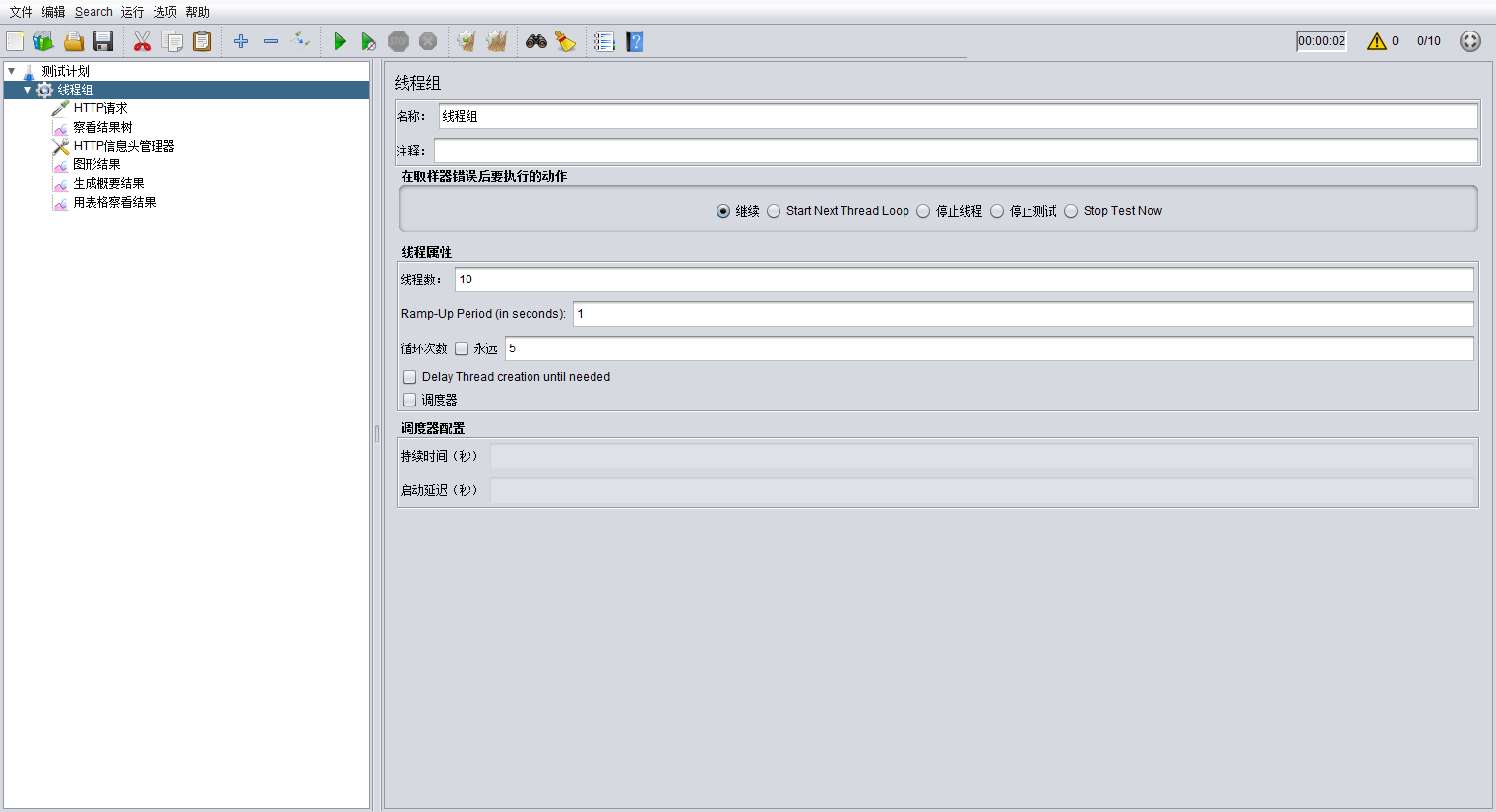
**表7-5 分享笔记功能测试**

|  |  |
| --- | --- |
| 测试用例描述 | 分享笔记测试 |
| 测试数据 | 笔记信息 |
| 测试步骤描述 | 1. 点击一个笔记，进入笔记详细页面 2. 点击分享按钮，进入分享页面 3. 点击初始化评论按钮，页面渲染评论信息 |
| 预期测试结果 | 分享页面正常渲染。 |
| 实际测试结果 | 在列表页面点击一个笔记，然后点击分享按钮，分享页面正常渲染。评论信息正常从Github中接入。 |

## 7.2 性能测试

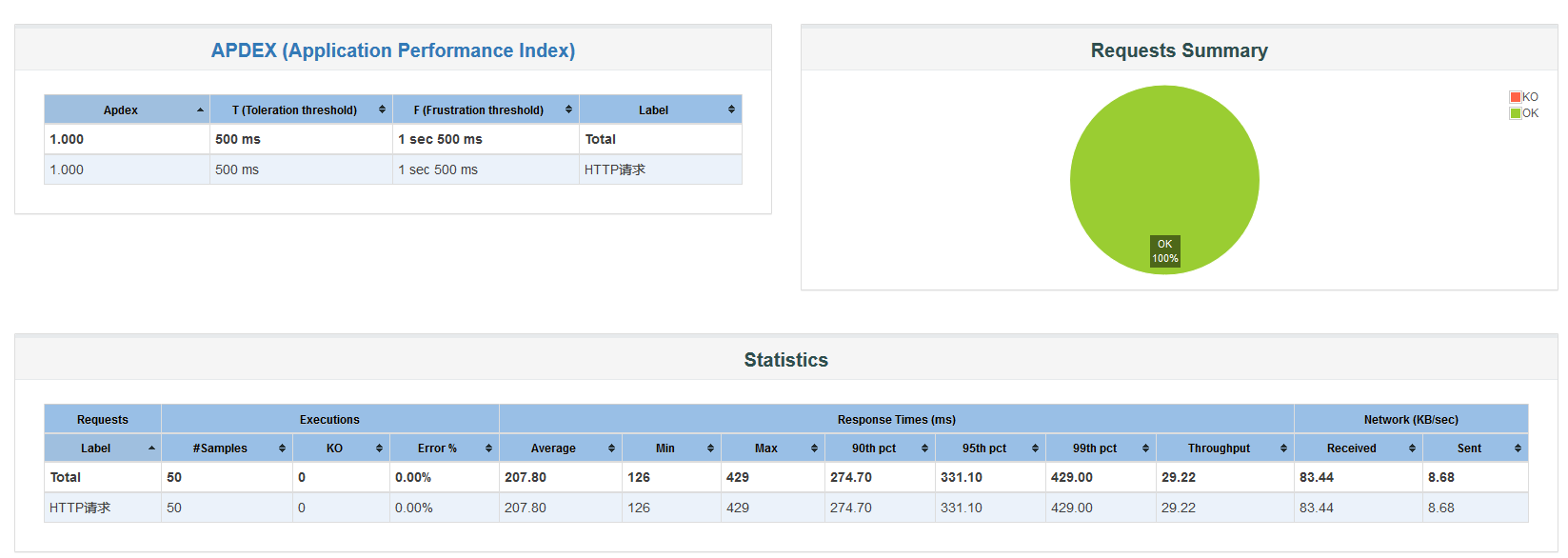
本项目采用JMeter作为性能测试工具。JMeter作为Apache开源的一款性能测试工具，提供的功能非常多，这里用它来测试接口的响应速度，并生成性能报告。将JMeter作为一个辅助测试工具，用其来完成常用的web测试、数据库测试、JMS 测试，可以在很大地程度上提高测试人员的效率[16]。

首先需要配置线程组的一些相关信息，例如线程数、循环次数等，然后配置HTTP相关参数，这里配置了10个线程，循环5次。配置页面如图7-1所示。



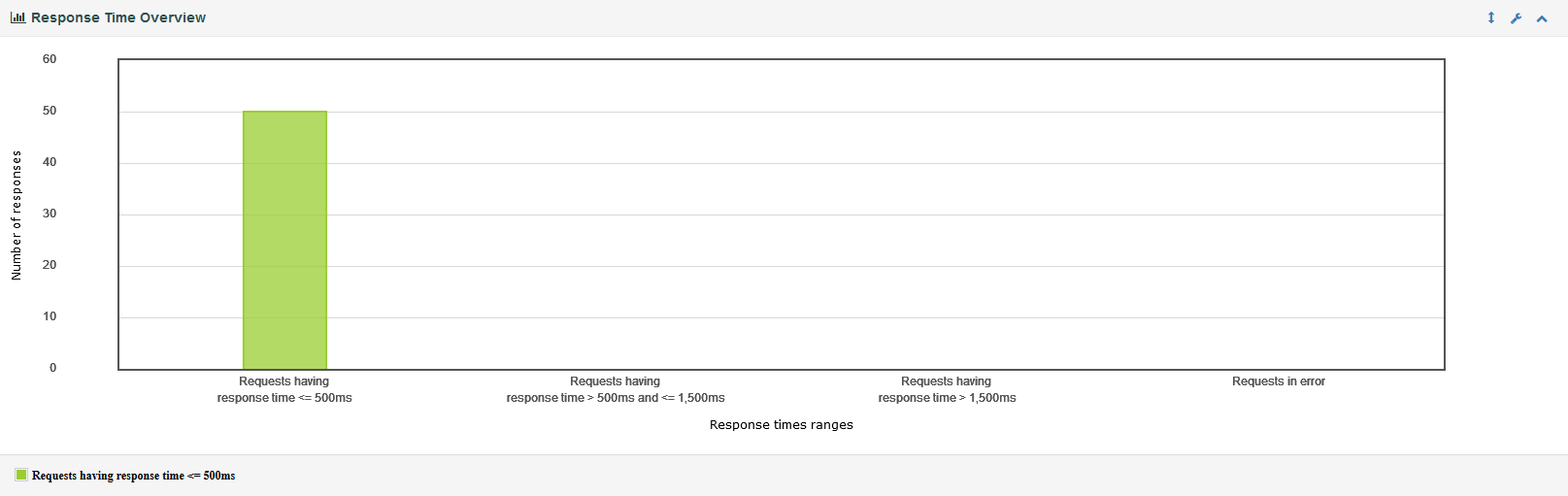
**图7-1 线程组配置页面**

配置完相关参数，就可以利用JMeter生成性能测试报告了。首先需要执行相关命令，生成HTML报告。性能报告概览页面如图7-2所示。APDEX区域为云笔记性能指数。Requests Summary代表请求的状态，成功或者失败百分比。Statistics表示最终得出的数据分析结果。APDEX区域的HTTP请求对应的Apdex为1，说明符合要求，Requests Summary区域显示为100%，说明所有请求都是成功的。根据Statistics区域显示的结果来看，平均响应时间为207.80ms，最小响应时间为126ms，最大为429，最大响应时间没有超过500ms，远远小于1s，说明系统的响应时间符合需求设定。下面来看云笔记响应的图表分析。



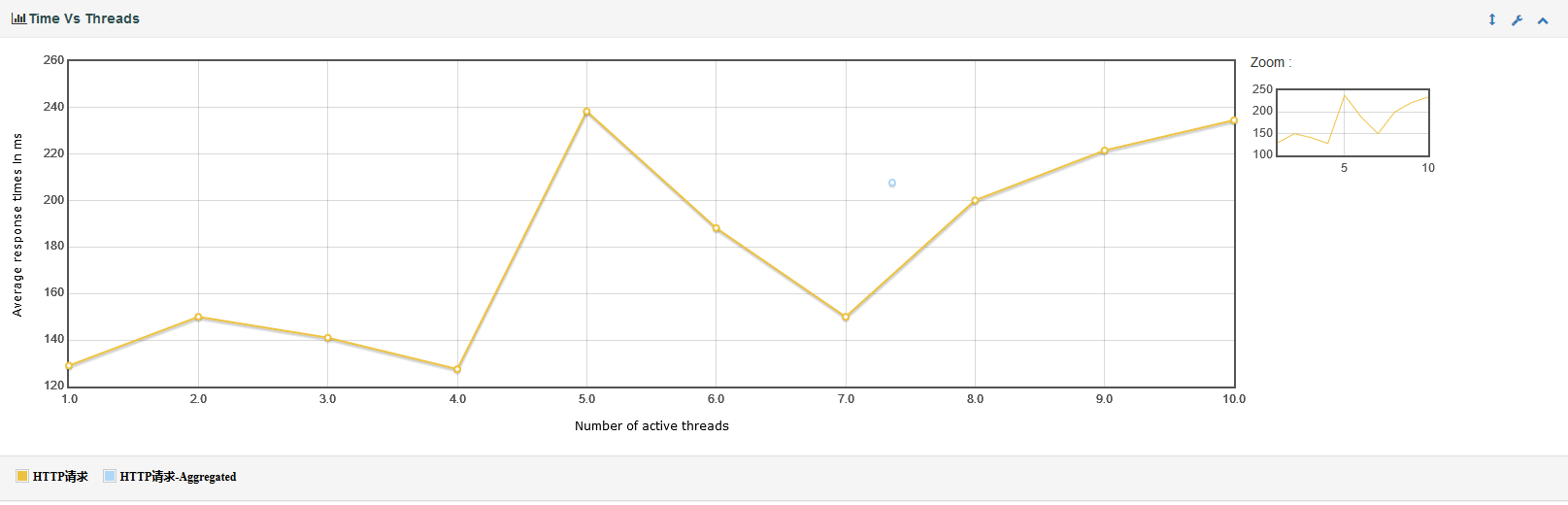
**图7-2性能测试概览页面**

性能测试结果包含许多图表分析。首先来看Response Time Overview，如图7-3所示。从此图可以看出，系统响应时间全部小于500ms，没有大于500ms的，性能很好。



**图7-3 响应时间统计图**

因为测试开始设定的线程数为10，循环执行5次，所以对线程的执行状态分析就十分有必要，图7-4显示的是线程与时间的折线图。从图中可以看出线程数量为10对应的时间平均为230ms，已经达到了需求分析的要求。



**图7-4 线程时间关系图**

## 7.3 结果与结论

本章从系统的功能测试和性能测试来对系统进行整体测试和分析。功能测试挑选了系统的一些主要功能点来进行测试，通过编写测试用例，分析测试预期结果和实际结果的差异，发现这些功能都符合预计结果，证明系统的预期功能已经实现，可以提供云笔记相关服务。性能测试利用JMeter性能测试工具，配置线程组来进行性能测试分析，通过对系统响应时间、线程时间关系等的分析，性能已经达到了需求分析的要求，能够给用户良好的体验。

结束语

经过几个月的学习与研究，云笔记的相关功能趋于完善与稳定，各个模块的设计也更加合理。由于此项目用到了很多比较新和热门的技术和理论，例如单页面应用，Elasticsearch搜索引擎等。在项目的编写中遇到了很多挑战，但没有让我有困难的感觉，因为在了解这些新技术，新理论的时候，能够了解它们的使用方式，弄清它们的实现原理，并将其应用到自己的项目中，为项目提供了另一种良好的解决方案，这实在是一件令人感到愉快的事情。我想这也是我们对待新技术、新理论该有的态度，不应因为困难而逃避，更不应因为无知而漠视。

对项目的优化更能提高自己对项目整体结构的理解，对一些代码上的设计可能会有另一种思考与总结，这也是我收获最大的地方。意识到自己在代码编写的不足和疏漏，总比一直写千篇一律的代码有价值、有意义些。下一步将会给项目引入更多的功能，同时会优化有关逻辑与整体架构，给开源社区贡献一点小小的力量。

致 谢

转眼间大学生活就要结束了，回望大学四年，收获很多，我从一个对开发语言什么都不懂的学生，变成了可以独立完成毕设系统的开发者，同时我也结识了很多志同道合的朋友，遇到了和蔼可亲的老师，无论是学习、生活还是实习，他们都在我的大学生活里扮演了重要的角色。

在做毕业设计的时候遇到了很多困难，好在有和蔼可亲的导师邓璐娟教授为我答疑解惑，给我的系统提了很多建设性的建议，在论文的编写上也给了我很大的启发，让我看到了自己的不足之处并及时改正。再次感谢学院在我离校之际给我机会去做自已喜欢的事情，也感谢我的导师在百忙之中对我进行指导。

最后要感谢我的父母，以及我身边的每一个不能离开的人，是你们让我在漫长岁月中找到喜欢的自己，做自己喜欢的事，成为一个价值有所依托的人。今后我将尽我所能，成为一个更好的人，同时也为社会之发展民族之振兴尽点微薄之力。

参考文献

[1] 艾瑞市场研究公司.2012年中国云存储行业及用户行为研究报告[R].2012．

[2] 罗秀娟.基于云笔记Evernote的科研工作者个人知识管理探究[J].图书馆学研究,2013(17):65-68.

[3] 王博,陈莉君.Hadoop远程过程调用机制的分析和应用[J].西安邮电学院学报,2012,17(06):74-77.

[4] A. L. Ananda,B. H. Tay,E. K. Koh. A survey of asynchronous remote procedure calls[J]. ACM SIGOPS Operating Systems Review,1992,26(2).

[5] 顾祥荣. 远程调用技术研究及新交换平台设计[D]. 上海交通大学, 2007.

[6] 王斌斌. 基于Protobuf的RPC系统的设计与实现[D].电子科技大学,2016.

[7] 姜虹. 基于Web Services的RPC体系结构的研究[D]. 西安工业大学, 2008.

[8] Andrew D. Birrell,Bruce Jay Nelson. Implementing Remote procedure calls[J]. ACM SIGOPS Operating Systems Review,1983,17(5).

[9] 刘心松, 邱元杰, 杨锋.Linux下RPC的分析及改进[J].小型微型计算机系统, 2001, 22(7):881-884.

[10] 张艳军,王剑,叶晓平,李培远.基于Netty框架的高性能RPC通信系统的设计与实现[J].工业控制计算机,2016,29(05):11-12+15.

[11] 李林锋.Netty权威指南[M].北京:电子工业出版社,2014:1-505.

[12] 张羽. 一种分布式服务治理框架的设计与实现[D]. 北京交通大学, 2016.

[13] 王健. Docker技术将引爆容器生态系统[J]. 软件和集成电路, 2016(z1):48-49.

[14] 钟丽娟. 时间序列数据相似性与聚合top-k查询算法研究与应用[D].浙江大学,2016.

[15] Elastic. Stashing Your First Event [EB/OL]，<https://www.elastic.co/guide/en/logstash/current/first-event.html，2018-02-17/2018-06-08>.

[16] 陈绍英，夏海涛，金成姬．web性能测试实战[M]，北京：电了工业出版社，2006