

毕业设计（论文）



题 目 校内讨论问答社区的设计与实现

姓 名 童沧真

学 号 31501124

专业班级 计算机1502班

所在学院 计算学院

指导教师（职称） 彭彬（讲师）

二○一九 年 五 月 十八 日

校内讨论问答社区的设计与实现

【摘要】 如何构建一个符合现在主流架构的校内web社区使它能够满足同学日常交流使用需要，可以支撑校方发布通知，并能支持选课级别的流量，对设计者来说是一件巨大的挑战。本文设计实现了一个基于springboot的校内社区系统，并实现了一个异步消息框架和一个敏感词过滤算法。

本文首先阐述了构建校内社区的意义，并简要分析了国内和的技术现状和各个社区系统的特点，从而确定了本文所写系统的大致设计要素。即该系统要有基本的安全方面的功能，拥有一定的并发能力并且对每个用户生成个性timeline。

在第三第四章，详细的解释了本系统各个模块的设计思路，并在后文实现了所有模块的功能，并对系统后续的发展展开讨论，最后在云服务上使用docker对本系统进行部署和压力测试。

【关键词】 社区系统，消息队列，SpringBoot，Java，Docker

Design and implementation of a Q&A community on campus

【Abstract】 build a school web community that conforms to the current mainstream architecture so that it can meet the daily communication needs of students, can support the school to issue notices, and can support the class-level traffic, which is a huge challenge for designers. This paper designs and implements a springboot-based campus community system, and implements an asynchronous message framework and a sensitive word filtering algorithm.

This paper first expounds the significance of building a community within the school, and briefly analyzes the current state of the art and the characteristics of each community system, thus determining the general design elements of the system written in this paper. That is, the system must have basic security functions, have certain concurrency capabilities, and generate personalized timelines for each user.

In the third and fourth chapters, the design ideas of each module of the system are explained in detail, and the functions of all modules are realized in the following. Finally, the system is deployed and stress tested by using docker on the cloud service.

【Key Words】 Community system,Message queue framework,SpringBoot,Java，Docker

目录

[第1章 绪论 1](#_Toc208196767)

[1.1 课题研究的背景与意义 1](#_Toc208196768)

[1.2 国内外研究现状与发展趋势 1](#_Toc208196771)

[1.3 本文研究的主要内容 3](#_Toc208196771)

[第2章 系统开发技术介绍 4](#_Toc208196774)

[2.1 相关开发技术介绍 4](#_Toc208196775)

[2.1.1 Spring Boot 4](#_Toc208196776)

[2.1.2 Thymeleaf 4](#_Toc208196777)

[2.1.3 Spring Data Jpa 4](#_Toc208196778)

[2.1.4 Mysql 4](#_Toc208196778)

[2.1.5 Redis,Jedis 5](#_Toc208196778)

[2.2 开发和部署环境介绍 5](#_Toc208196779)

[2.2.1 IntelliJ IDEA 5](#_Toc208196780)

[2.2.2 Docker 5](#_Toc208196781)

[2.2.3 Apache Tomcat 5](#_Toc208196781)

[2.2.4 Nginx 6](#_Toc208196781)

[2.3 本章小结 6](#_Toc208196787)

[第3章 系统总体设计 7](#_Toc208196782)

[3.1 系统功能设计 7](#_Toc208196783)

[3.1.1 用户模块 7](#_Toc208196784)

[3.1.2 讨论问答模块 8](#_Toc208196785)

[3.1.3 站内信模块 8](#_Toc208196786)

[3.1.4 粉丝关注模块 8](#_Toc208196786)

[3.2 数据库设计 9](#_Toc208196787)

[3.2.1 数据库总体设计 9](#_Toc208196788)

[3.2.2 数据库表设计 9](#_Toc208196789)

[第4章 系统分析与设计 1](#_Toc208196790)2

[4.1 系统关键框架/算法设计 1](#_Toc208196791)2

[4.1.1 消息队列框架设计 1](#_Toc208196792)2

[4.1.2 敏感词屏蔽算法设计 1](#_Toc208196793)3

[4.2 系统业务模块设计 1](#_Toc208196795)5

[4.2.1 用户模块设计 1](#_Toc208196796)5

[4.2.2 讨论问答模块设计 1](#_Toc208196797)5

[4.2.3 站内信模块设计 1](#_Toc208196797)8

[4.2.4 粉丝关注模块设计](#_Toc208196797) 20

[第5章 系统的开发与实现 2](#_Toc208196798)2

[5.1 系统关键框架的开发 2](#_Toc208196799)2

[5.1.1 异步消息队列 2](#_Toc208196800)2

[5.1.2 敏感词屏蔽算法 2](#_Toc208196801)5

[5.2 系统业务模块的开发与实现 2](#_Toc208196803)7

[5.2.1 用户模块 2](#_Toc208196804)8

[5.2.2 讨论问答模块 3](#_Toc208196805)2

[5.2.3 站内信模块 4](#_Toc208196805)0

[5.2.4 粉丝关注模块 4](#_Toc208196805)3

[5.2.5 技术讨论与发展方向 4](#_Toc208196805)4

[第6章 项目部署与性能测试 4](#_Toc208196806)8

[6.1 项目部署 4](#_Toc208196807)8

[6.2 性能测试 4](#_Toc208196811)9

[结论 5](#_Toc208196821)1

[参考文献 5](#_Toc208196822)2

[致谢 5](#_Toc208196824)4

图目录

[图3.1 系统模块图 7](#_Toc208196827)

[图4.1 消息队列设计图 1](#_Toc208196828)3

[图4.2 前缀树图 1](#_Toc208196828)4

[图4.3 用户登录流程图 1](#_Toc208196825)5

[图4.4 发布主题/评论流程图 1](#_Toc208196825)6

[图4.5 主题界面 1](#_Toc208196825)6

[图4.6 问题详情界面 1](#_Toc208196825)7

[图4.7 点赞操作流程图 1](#_Toc208196825)8

[图4.8 发送站内信流程图 1](#_Toc208196825)9

[图4.9 所有会话界面 1](#_Toc208196825)9

[图4.10 会话所有信息界面 2](#_Toc208196825)0

[图4.11 关注/取关流程图 2](#_Toc208196829)1

[图4.12 feed数据生成流程图 2](#_Toc208196829)1

[图5.1 推拉结合系统中发布feed流程 4](#_Toc208196830)5

[图5.2 推拉结合系统中读取feed流程 4](#_Toc208196830)6

[图5.3 半衰期算法函数曲线 4](#_Toc208196830)7

[图6.1 项目docker运行情况图 4](#_Toc208196830)8

表目录

[表3.1 entityType对应的实体类型表 9](#_Toc208196832)

[表3.2 用户表 9](#_Toc208196832)

[表3.3 token记录表 9](#_Toc208196832)

[表3.4 主题question表 1](#_Toc208196832)0

[表3.5 回答comment表 1](#_Toc208196832)0

[表3.6 站内信message表 1](#_Toc208196832)0

[表3.1 feed事件表 1](#_Toc208196832)0

[表3.1 key规则表 1](#_Toc208196832)1

[表6.1 运行服务器信息表 4](#_Toc208196835)8

[表6.2 apache bench测试结果表 5](#_Toc208196832)0

# 第1章 绪论

## 1.1 课题研究的背景与意义

网络社交社区占用了当代大学生越来越多的课余时间，而比起外网的公共社区，校内社区则有着许多优势。我们可以在内网更好的讨论校内热点，可以参与知识分享，甚至在校内社区的交友也会更加有保证。一个好的校内社区，对学生和老师都有着重要的意义。

很多著名的高校校内社区（如水木清华、北大未名、cc98）经过多次迭代，功能趋于完善，用户数量稳定，极大的方便了在校师生，做的好的甚至成为了一种代表高校文化符号。但是他们大多属于传统的BBS论坛，技术也较为落后，本课题意在使用较新的技术构建一个符合现在主流架构的校内web社区。

我校暂时还没有属于自己的校园社区，目前同学间主要的交流都在朋友圈，贴吧，微博等外网社区。朋友圈由于其封闭性，很多问题仅仅在一个个封闭的圈子中传播，缺少广泛性。而微博和贴吧一是难于管理，二是作为开放给全网的社区，有些相对敏感问题很难在上面讨论，并且也不会有学术性的问题在上面进行问答。拥有自己的校园社区的好处是显而易见的，首先是非常容易管理，如果运行在内网的话，用户的成分也会比较单纯；而且作为一个对全校开放的社区，所有学生老师都能第一时间参与到热点问题的讨论，没有朋友圈的滞后性；新同学的问题可能在社区上找到一年前的学长的回答；还能够根据需要做出各种各样的迭代。所以我们迫切的需要这样一个校园问答社区。

## 1.2 国内外研究现状与发展趋势

现今网络上有非常多不同类型的社区产品，如Facebook，instagram，微博，知乎，微信朋友圈等，分别从不同的侧重点出发，其技术特点也有较多不同。但无论他们的类型如何，都有一些特点可供我们参考：

1.Feed流系统：

2.高可用与高并发的网站架构设计

作为校园问答社区，有“知乎”这一非常成功的产品作为参考，“知乎”的问答方式为：题主发布问题，众多答主在该问题下阐述自己的回答，甚至很多回答会超越问题本身，这种模式下，我们可以在一个问题下看见非常多不同角度的优秀回答，对用户形成了非常强的吸引力。同时知乎还有“知乎专栏”（一种类似于博文的功能）对该模式进行补充。知乎主要展现给用户的有三种feed流，一种是根据用户关注的用户/问题生成的timeline（关注榜），一种是根据用户的阅读喜好生成的rank（推荐榜），还有一种是根据当前问题热度生成的rank（知乎热榜）。

随着云计算、大数据、物联网、移动互联网等新一代计算机技术生态的形成、发展和完善，大型互联网社区产品的架构也不断的发生演化。一个成熟的网站的系统架构并不是开始设计就具备完整的高性能，高可用，安全等特性的，从最初的单点型架构到集群架构，再到如今的集群加分布式架构，设计思想发生了很大的变化。为了支撑一天天增加的用户和海量的流量，如今留行的趋势是将越来越臃肿的应用程序进行业务拆分，每个业务应用负责相对独立的业务运作，业务直接通过消息进行通信或者共享数据库来实现。接着，把各个业务都会用到的一些基本业务服务，如用户服务，支付服务等基本要素抽取出来，利用分布式服务框架搭建分布式服务，大大提高了系统的可用性和拓展性[3]。

另一方面参考现在各种各样的社区产品（Facebook，instagram，微博，知乎，微信朋友圈等），这些产品通过将内容分发与算法结合，越来越多的根据用户的浏览喜好给用户呈显出他们喜欢看的内容，大大增加了用户粘性，并且通过用户的喜好来构建出准确的用户画像，方便精准广告推送等功能的实现。但也造成了一些问题，比如基于贝叶斯的算法会让用户越来越难以发现自己喜好之外的惊喜，部分社区的改版（微博）出现水土不服的情况等，需要更好的算法设计来克服这样的现象。

## 1.3 本文研究的主要内容

使用较新的技术构建一个符合现在主流架构的校内web社区。能够满足同学日常交流使用需要，可以支撑校方发布通知，并能支持选课级别的流量。

该系统的特点是支持问答和讨论，这一特点可以模仿知乎的这种成功模式，即一个问题多个回答，浏览者可以对问题进行关注，可以对回答进行踩/赞，评论，对评论本身也可以进行赞和评论，这样就能在一个个问答中形成浓厚的讨论气氛。

同时我会参考知乎的feed流生成来实现自己的feed流系统。由于根据用户喜好生成rank难度较大，我将参考“关注榜”（根据用户关注的用户/问题生成的timeline）来实现本次的feed流系统。

系统也应该拥有XSS过滤[9]，敏感词过滤，防止cookie泄露，防撞库等基本的安全功能。其中XSS过滤主要靠的是对各种输入输出进行转义，敏感词过滤我将设计实现一个敏感词过滤算法，通过加盐处理来防止泄露和撞库。

同时本系统的主要目标为支撑一个校内社区的完善有效的后端系统，前端页面仅作为展示后端成果的平台，所以对工作量稍作取舍，在目前的计划中将不会投入大量工作去学习使用现在非常流行的React、Vue和Angular等框架，主要通过thymeleaf、freemarks等模板类配合jquery等js工具达到基本的展示目标即可。

# 第2章 系统开发技术介绍

## 2.1 相关开发技术介绍

### 2.1.1 Spring Boot

Spring Boot是由Pivotal团队提供的全新框架，其设计目的是用来简化新Spring应用的初始搭建以及开发过程。该框架使用了特定的方式来进行配置，从而使开发人员不再需要定义样板化的配置[3]。

### 2.1.2 Thymeleaf

Thymeleaf是用来渲染XML/XHTML/HTML5内容的模板引擎。类似于JSP，Velocity，FreeMaker等引擎，它也能方便的与Spring 等Web框架集成作为Web应用的模板。与其它模板引擎对比，Thymeleaf最显著的特点是可以直接在浏览器中打开并正确显示模板页面，而不需要启动整个Web应用。

### 2.1.3 Spring Data Jpa

spirng data jpa是spring提供的一套简化JPA开发的框架，按照约定好的方法命名规则写dao层接口，就可以在不写接口实现的情况下，实现对数据库的访问和操作。同时提供了很多除了CRUD之外的功能，如分页、排序、复杂查询等等。

### 2.1.4 Mysql

MySQL是一种开放源代码的关系型数据库管理系统（RDBMS），MySQL数据库系统使用最常用的数据库管理语言--结构化查询语言（SQL）进行数据库管理。

### 2.1.5 Redis,Jedis

Redis是一个开源的使用ANSI C语言编写、支持网络、可基于内存亦可持久化的日志型、Key-Value数据库，并提供多种语言的API[5]。

Jedis是是Redis官方推荐的Java客户端开发包，用于处理redis服务上的缓存数据。

## 2.2 开发和部署环境介绍

### 2.2.1 IntelliJ IDEA

IntelliJ IDEA，是java编程语言开发的集成环境。IntelliJ在业界被公认为最好的java开发工具之一，尤其在智能代码助手、代码自动提示、重构、J2EE支持、各类版本工具(git、svn等)、JUnit、CVS整合、代码分析、 创新的GUI设计等方面的功能可以说是超常的。

### 2.2.2 Docker

Docker是一个开放源代码软件项目，让应用程序部署在软件货柜下的工作可以自动化进行，借此在Linux操作系统上，提供一个额外的软件抽象层，以及操作系统层虚拟化的自动管理机制。 Docker利用Linux核心中的资源分离机制，例如cgroups，以及Linux核心名字空间，来创建独立的容器。docker不像传统的软件交付方式那样，只把代码以及说明文档之类的给你就完了，而是直接给你一个像集装箱那样的标准docker货件，这个标准件不仅包括了应用代码本身，还包括了代码运行需要的OS等整体依赖环境，这个特点使得服务部署的工作量大大降低。[8]

### 2.2.3 Apache Tomcat

Tomcat是由Apache软件基金会属下Jakarta项目开发的Servlet容器，按照Sun Microsystems提供的技术规范，实现了对Servlet和JavaServer Page的支持，并提供了作为Web服务器的一些特有功能，如Tomcat管理和控制平台、安全局管理和Tomcat阀等。

### 2.2.4 Nginx

Nginx是异步框架的Web服务器，也可以用作反向代理，负载平衡器 和 HTTP缓存。在正常情况下，单次请求会得到更快的响应；另一方面，在高峰期（如有数以万计的并发请求），Nginx可以比其他Web服务器更快地响应请求，同时，本项目通过docker部署，需要将服务器映射出的多个服务（如tomcat，数据库管理组件phpmyadmin等）的端口反向代理到80端口，apache的反向代理功能比较粗糙，所以选用Nginx作为本项目的Web服务器。

## 2.3 **本章小结**

本章介绍了论文所述系统用到的主要技术，从开发环境IDEA到前后端交互模板引擎，再到数据库，以及主要用到的Java框架。最后是部署系统的环境。为整个系统的设计开发流程提供了可靠的保障。

# 第3章 系统总体设计

## 3.1 系统功能设计

此系统包含五个模块，分别为用户模块，社区讨论模块，站内信模块，关注粉丝模块，站内搜索模块，具体功能结构如下图3.1所示。

A

B

C

D

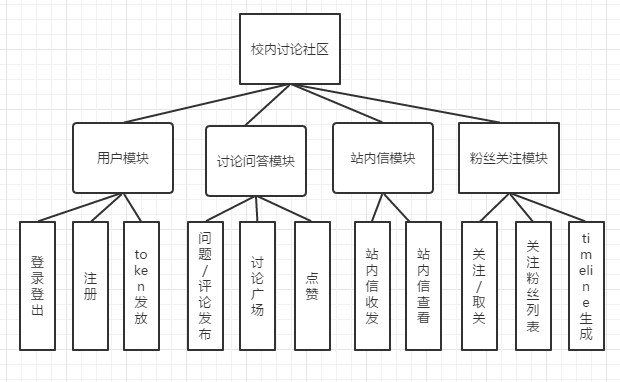


图3.1 系统模块图

### 3.1.1 用户模块

本系统中的用户有两种，普通用户和系统用户。系统用户由系统所有，负责发送通知和站内信，普通用户则可以使用本系统中的所有模块内容，未登录用户可以使用部分内容。

用户在注册和登录时的密码应该加密传输，且用户应该能在个人页面查看自己的相关信息。

### 3.1.2 讨论问答模块

主题发布功能，已登录的用户可以发布主题，也就是提出自己的问题，提出问题后，任何已登录用户都能参与问题的讨论。

在讨论广场中，展示了近期发布的主题，各个主题的发布人，关注人数，评论人数，问题描述等，在具体的问题页面中，展示了问题的提出人，关注该问题的人，问题描述，其他用户的回答，每个回答的点赞数等信息。每个用户都能查看讨论广场展示的信息。

已登录用户可以对问题添加回答（评论）。同时，系统能过滤掉问题标题，描述和用户回答中的敏感词和恶意代码。

已登录用户可以给每个回答点赞/踩，系统可以记录每个用户的点赞情况，并防止重复点赞。

未登录用户在进行操作时，系统将重定向至登录界面。

### 3.1.3 站内信模块

已登录用户可以给其他用户发送站内信，系统用户会在用户被关注/用户回答被点赞的时候给用户发送站内信通知。站内信的通知发送将进行异步处理，防止高耗时操作阻塞系统进程。

已登录用户可以查看已收到站内信。

未登录用户无法使用站内信。

### 3.1.4 粉丝关注模块

只有登录用户可以使用该模块。

可以关注/取关用户和问题。系统将记录每个用户的关注情况，并生成每个用户/问题的关注列表。展示给用户。

同时，系统会根据用户的关注情况推送feed流给不同的用户，生成每个用户独有的timeline。Timeline中的元素有统一的格式。

## 3.2 数据库设计

### 3.2.1 数据库总体设计

本系统的数据库有两部分组成，用户点赞，关注信息存放在redis内存数据库中，用户登录，站内信，问题，评论信息存放在mysql关系型数据库中。

其中，评论，关注功能用entityId,和entityType字段来标识具体对象，entity对应的实体见表3.1

表3.1 entityType对应的实体类型表

|  |  |  |
| --- | --- | --- |
| 类型 | entityType | 注释 |
| ENTITY\_QUESTION | 1 | 问题实体 |
| ENTITY\_COMMENT | 2 | 评论实体 |
| ENTITY\_USER | 3 | 用户实体 |

### 3.2.2 数据库表设计

（1）用户注册登录以及token存放的表格设计如表3.2，表3.3所示

表3.2 用户表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 字段 | 类型 | 空 | 默认 | 注释 |
| id (主键) | int(11) | 否 |  | 用户编号 |
| head\_url | varchar(256) | 否 |  | 用户头像地址 |
| name | varchar(64) | 否 |  | 用户名 |
| password | varchar(128) | 否 |  | 密码 |
| salt | varchar(32) | 否 |  | 盐 |

表3.3 token记录表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 字段 | 类型 | 空 | 默认 | 注释 |
| id (主键) | int(11) | 否 |  | token编号 |
| expired | datetime | 否 |  | 过期时间 |
| status | int(11) | 是 | NULL | token状态 |
| ticket | varchar(45) | 否 |  | token内容 |
| user\_id | int(11) | 否 |  | 用户id |

（2）讨论问答模块的主题question表设计如表3.4所示，回答comment表如表3.5所示。

表3.4 主题question表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 字段 | 类型 | 空 | 默认 | 注释 |
| id (主键) | int(11) | 否 |  | 问题编号 |
| comment\_count | int(11) | 否 |  | 评论数量 |
| content | varchar(255) | 是 | NULL | 问题描述 |
| created\_date | datetime | 否 |  | 提问时间 |
| title | varchar(255) | 否 |  | 问题标题 |
| user\_id | int(11) | 否 |  | 提问者 |

表3.5 回答comment表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 字段 | 类型 | 空 | 默认 | 注释 |
| id (主键) | int(11) | 否 |  | 评论编号 |
| content | varchar(5000) | 是 | NULL | 评论内容 |
| created\_date | datetime | 是 | NULL | 创建时间 |
| entity\_id | int(11) | 是 | NULL | 对应的实体编号 |
| entity\_type | int(11) | 是 | NULL | 对应的实体类型 |
| status | int(11) | 是 | NULL | 状态 |
| user\_id | int(11) | 是 | NULL | 评论创作者 |

（3）站内信模块的储存站内信表格如表3.6所示。

表3.6 站内信message表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 字段 | 类型 | 空 | 默认 | 注释 |
| id (主键) | int(11) | 否 |  | 站内信编号 |
| content | varchar(255) | 是 | NULL | 内容 |
| conversation\_id | varchar(45) | 是 | NULL | 会话编号 |
| created\_date | datetime | 是 | NULL | 时间 |
| form\_id | int(11) | 是 | NULL | 发送人 |
| has\_read | int(11) | 是 | NULL | 阅读状态 |

1. Feed流的feed事件储存见表3.7

表3.7 feed事件表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 字段 | 类型 | 空 | 默认 | 注释 |
| id (主键) | int(11) | 否 |  | 事件编号 |
| type | int(11) | 是 | NULL | 事件类型 |
| user\_id | int(11) | 是 | NULL | 用户id |
| create\_date | datetime | 是 | NULL | 创建时间 |
| data | tinytext | 是 | NULL | 内容 |

1. 关注和点赞则存储在redis中，key的规则如表3.8所示：

表3.8 key规则表

|  |  |  |
| --- | --- | --- |
| key | 类型 | 注释 |
| LIKE:entityType:entityId | set | 对某个实体点赞的集合 |
| DISLIKE:entityType:entityId | set | 对某个实体点踩的集合 |
| FOLLOWER:entityType:entityId | sorted set | 关注某个实体的集合 |
| FOLLOWEE:userId:entityType | sorted set | 某个用户的关注列表 |

# 第4章 系统分析与设计

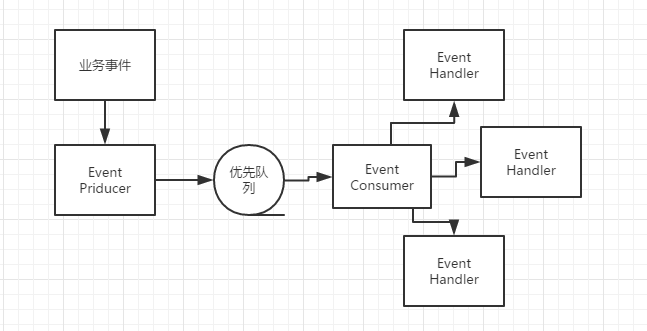
## 4.1 系统关键框架/算法设计

### 4.1.1 消息队列框架设计

该系统内存在很多耗时操作如：系统给某个被点赞用户发送站内信通知-对于用户来说，点赞功能实际只需要第一步，只要服务端将他的点赞存到数据库中他便可以去做别的事情了。至于其他的事情，不需要用户花时间等待处理，所以不需要在这一次请求中全部完成，实际当第一步做完后，服务端就可以把其他的操作放入对应的消息队列中然后马上返回用户结果，由消息队列异步的进行这些操作。

目前在生产环境，使用较多的消息队列有ActiveMQ、RabbitMQ、ZeroMQ、Kafka、MetaMQ、RocketMQ等，由于Redis有优先队列的数据结构，加上消息队列需要保证拓展性，可以方便的将任何操作改造为该队列的事件，所以本系统使用Redis的优先队列设计一个消息队列框架。

消息队列使用生产者与消费者模型。消息队列的设计见图4.1：



A

B

C

D

图4.1 消息队列设计图

事件的类型为：

*LIKE*(0),  
*COMMENT*(1),  
*LOGIN*(2),  
*MAIL*(3),  
*FOLLOW*(4),  
*UNFOLLOW*(5);

每个handler可以处理某些事件，例：某个用户关注了另一个用户，此事件为FOLLOW(4)，该事件类型可以被messagehanler和feedhanler响应，分别给被关注用户推送站内信通知和加入feed事件入库。

### 4.1.2 敏感词屏蔽算法设计

使用前缀树构建敏感词树，每个节点包含一个hashmap，每个 Key 对应该节点下的一个子节点字符，value 则指向相应的后一个状态。图4.9是一个标准前缀树：

A

B

C

D

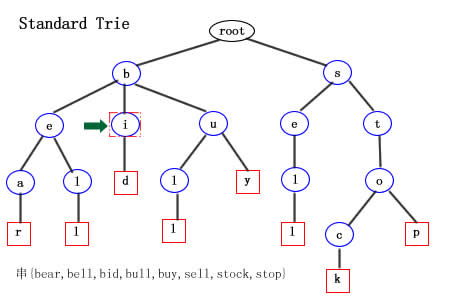


图4.2 前缀树图

构建完敏感词的前缀树后，即可从前缀树中检索文本中的敏感词，对每个敏感词有查找过程F：

1. 从前缀树的根节点开始，一个个比较是否存在对应位置的字符，找到对应的字符后，从该节点的孩子中查找下一个位置的字符。
2. 若孩子中不存在对应位置的字符，则跳出查找
3. 若所有字符全部查找完且全部比对成功，判断下一个节点是否为标志位，是，则为敏感词。否，则跳出查找。

对文本，指定两个指针，对指针间的字符串实行过程F，若查找跳出，则前指针后移一位，后指针回到前指针位置。若查找成功。则标记/替换敏感词，前指针指向后指针位置。知道指针指向文本的最后一个字符即为过滤结束。

将敏感词算法封装为一个工具类，后续的业务中，只需要调用该类中的方法即可。

## 

## 4.2 系统业务模块设计

### 4.2.1 用户模块设计

功能：

1. 用户注册登录退出。
2. 拦截未登录用户访问某些页面。
3. 将用户信息读取，供后续业务使用

用户登录流程图如图4.3所示

A

B

C

D

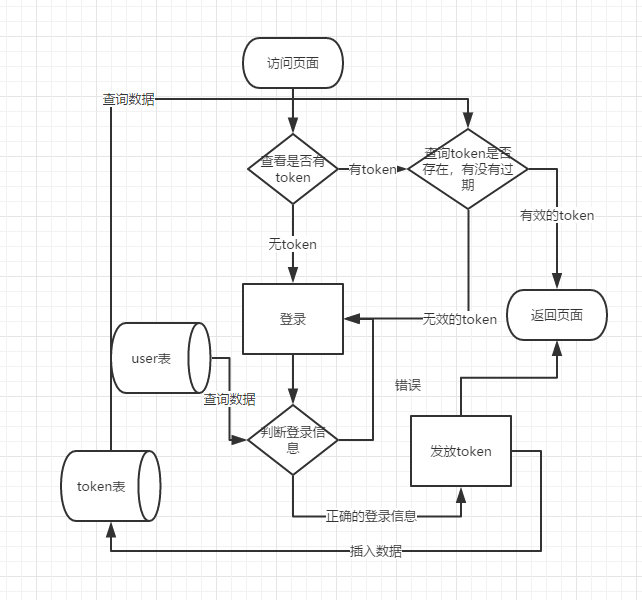


图4.3 用户登录流程图

### 4.2.2 讨论问答模块设计

（1）发布主题/评论

在发布主题/评论前，需要验证用户权限，过滤问题以及其描述中的敏感词，发布后，异步推送feed事件。

同时，考虑到评论的对象既可以是主题，也可以是评论，所以本模块的将用entityType+entityID作为评论的参数，具体的entityType对应的实体表可以查看表3.1，如entityType=1，entityId=7即代表了评论的对象为id=7的主题（question），此种设计将简化service层的函数编写，也更有拓展性，以后若添加了新的评论对象，只需增加新的entityType即可，不需要添加新的Service层方法。

A

B

C

D

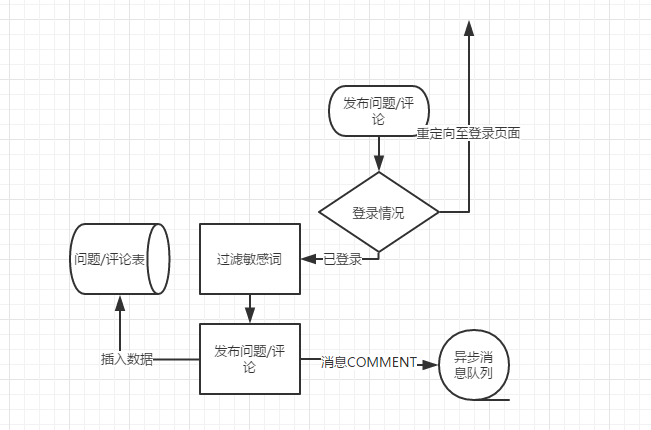


图4.4 发布主题/评论流程图

1. 展示问题，评论

根据用户信息前往数据库选出对应的数据，通过模板展示在前端页面即可。

A

B

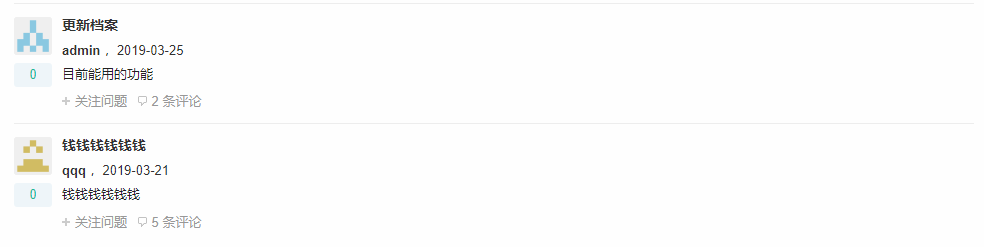


图4.5 主题界面

A

B

C

D



图4.6 问题详情界面

1. 点赞/点踩

登录用户才可点赞/踩，数据存入redis集合中即可自动去重，操作后，异步推送站内信和feed事件。

虽然点赞的对象只能是回答（comment）,但是基于拓展性的考虑，本系统也用entityType+entityID作为点赞的参数，方便以后加入新对象。点赞操作流程图如图4.7所示：

A

B

C

D

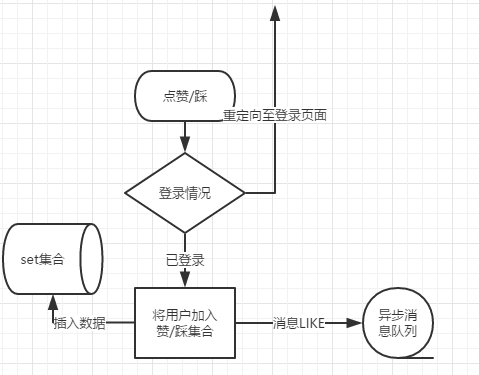


图4.7 点赞操作流程图

### 4.2.3 站内信模块设计

（1）发送站内信

登录用户才能发送站内信，站内信必须发给存在的用户，发送成功后将站内信内容存入数据库，站内信通过conversationId来区分会话，用户a发送给用户b的站内信和用户b发送给用户a的站内信的conversationId是一样的，用fromId来标记发送者。

A

B

C

D

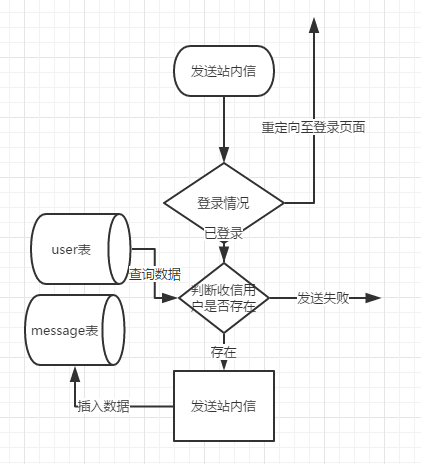


图4.8 发送站内信流程图

1. 查看站内信

查看站内信分为查看所有会话和查看某个会话里所有信息。两人间的信件往来为一个会话，将这些信息查询出来后在前端展示即可。

A

B

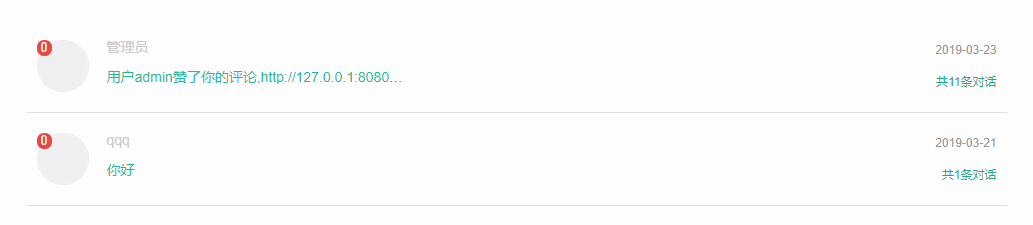


图4.9 所有会话界面

A

B

C

D



图4.10 会话所有信息界面

### 4.2.4 粉丝关注模块设计

1. 关注/取关

已登录用户可以使用关注/取关功能。验证登录信息后，将用户信息加入该关注实体的列表中（follower），同时把被关关注实体的信息加入用户的关注列表中（followee），关注结束后，如果关注对象为用户，则给用户发送一条站内信。

与评论类似，关注的对象可以为用户（user），也可以为主题（question），所以也用entityType+entityID作为关注的参数，如entityType=3，entityId=12即代表了关注的对象为id=12的用户（user）

A

B

C

D

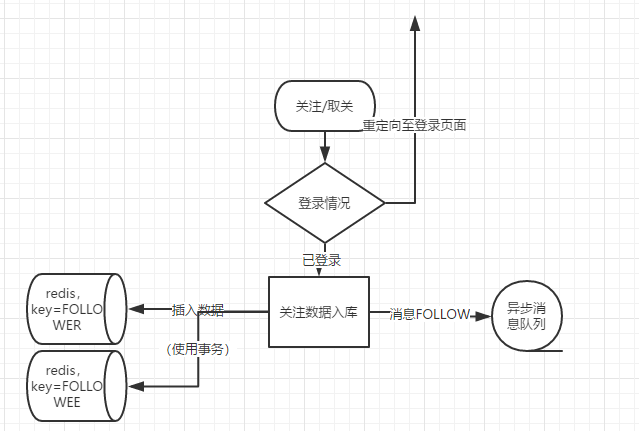


图4.11 关注/取关流程图

1. feed数据生成。

Feed数据采用享元模式，feedhandler可以响应消息队列中的COMMENT事件和FOLLOW事件生成feed数据。

A

B

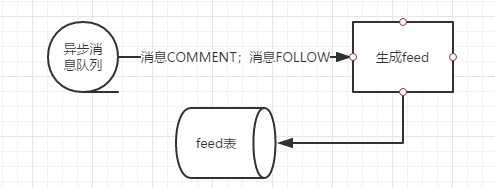


图4.12 feed数据生成流程图

1. timeline生成

使用拉算法生成timeline，即根据用户的关注列表从数据库中拉取feed事件

# 第5章 系统的开发与实现

## 5.1 系统关键框架的开发

### 5.1.1 异步消息队列

异步消息队列由四部分组成：

1. 调用EventModel的set函数构造一个标准的消息事件

EventModel中的每个Set的函数的返回值都是EventModel，这样就可以通过链式调用构造标准消息事件，使代码更加简化，增加代码易读性。同时，在使用回调技术时也可以使取值器方法。

1. 通过调用EventProducer的fireEvent()函数将消息推入redis队列

关键代码如下

public boolean fireEvent(EventModel eventModel) {  
 try {  
 String json = JSONObject.*toJSONString*(eventModel);  
 String key = RedisKeyUtil.*getEventQueueKey*();  
 jedisAdapter.lpush(key, json); //推入redis队列  
 return true;  
 } catch (Exception e) {  
 return false;  
 }  
}

1. 事件被EventConsumer接收，分配给对应的Handler

关键代码如下：

@Override  
public void afterPropertiesSet() throws Exception {  
 Map<String, EventHandler> beans = applicationContext.getBeansOfType(EventHandler.class);  
 if (beans != null) {  
 for (Map.Entry<String, EventHandler> entry : beans.entrySet()) {  
 List<EventType> eventTypes = entry.getValue().getSupportEventType();  
  
 for (EventType type : eventTypes) {  
 if (!config.containsKey(type)) {  
 config.put(type, new ArrayList<EventHandler>());  
 }  
 config.get(type).add(entry.getValue());  
 }  
 }  
 }  
  
 Thread thread=new Thread(new Runnable() {  
 @Override  
 public void run() {  
 while (true) {  
 String key = RedisKeyUtil.*getEventQueueKey*();  
 List<String> events = jedisAdapter.brpop(0, key); //从redis队列中获取事件，如果队列为空，则阻塞  
 for (String message : events) {  
 if (message.equals(key)) {  
 continue;  
 }  
 EventModel eventModel = JSONObject.*parseObject*(message, EventModel.class);  
 if (!config.containsKey(eventModel.getEventType())) {  
 *logger*.error("不能识别事件");  
 continue;  
 }  
 for (EventHandler handler : config.get(eventModel.getEventType())) { //分配给handler运行  
 handler.doHandle(eventModel);   
 }  
 }  
 }  
 }  
 });  
 thread.start();  
}

EventConsumer实现了ApplicationContextAware接口和InitializingBean接口。Spring发现某个Bean实现了ApplicationContextAware接口，Spring容器会在创建该Bean之后，自动调用该Bean的setApplicationContextAware()方法，调用该方法时，会将容器本身ApplicationContext对象作为参数传给该方法。

若该Bean实现InitializingBean接口，则会自动调用afterPropertiesSet()方法，在这个方法中，我定义了一个线程使用brpop()方法将redis队列中存在的消息分配给各个Handler，当消息队列为空时，此方法会阻塞线程。在业务膨胀后，可以使用多线程来提高消息队列的吞吐效率。

1. Handler处理消息。

每一个需要异步的事件都有对应的结构相似的handler，以下代码为点赞后系统发送站内信给用户的handler的关键代码：

@Override  
public void doHandle(EventModel model) {  
 MessageEntity message = new MessageEntity();  
 message.setFormId(ZhiHuUtil.*SYSTEM\_USERID*);  
 message.setToId(model.getEntityOwnerId());  
 message.setCreatedDate(new Date());  
 UserEntity user = userService.getUser(model.getActorId());  
 message.setContent("用户" + user.getName()  
 + "赞了你的评论,http://tongcangzhen.xin/question/" + model.getExt("questionId"));  
  
 messageService.addMessage(message);   
}  
  
@Override  
public List<EventType> getSupportEventType() {  
 return Arrays.*asList*(EventType.*LIKE*);  
}

### 5.1.2 敏感词屏蔽算法

构建敏感词前缀树的结构为：

private class TrieNode {  
 private boolean end = false;  
 private Map<Character, TrieNode> subNodes = new HashMap<Character, TrieNode>();  
  
 public void addSubNode(Character key, TrieNode node) {  
 subNodes.put(key, node);  
 }  
  
 TrieNode getSubNode(Character key) {  
 return subNodes.get(key);  
 }  
  
 boolean isEnd() {  
 return end;  
 }  
  
 void setEnd(boolean end) {  
 this.end = end;  
 }

敏感词树构建的关键代码为：

private void addWord(String lineText) {  
 TrieNode tempNode = rootNode;  
 for (int i = 0; i < lineText.length(); i++) {  
 Character c = lineText.charAt(i);  
 if (isSymbol(c)) {  
 continue;  
 }  
 TrieNode node = tempNode.getSubNode(c);  
 if (node == null) {  
 node = new TrieNode();  
 tempNode.addSubNode(c, node);  
 }  
 tempNode = node;  
 if (i == lineText.length() - 1) {  
 tempNode.setEnd(true);  
 }  
 }  
  
}

敏感词过滤的关键代码为：

public String filter(String text) {  
 if (StringUtils.*isEmpty*(text)) {  
 return text;  
 }  
 StringBuilder result = new StringBuilder();  
 String replacement = "\*\*\*";  
 TrieNode tempNode = rootNode;  
 int begin = 0;  
 int position = 0;  
 while (position < text.length()) {  
 Character c = text.charAt(position);  
 if (isSymbol(c)) {  
 if (tempNode == rootNode) {  
 result.append(c);  
 begin++;  
 }  
 position++;  
 continue;  
 }  
 tempNode = tempNode.getSubNode(c);  
 if (tempNode == null) {  
 result.append(text.charAt(begin));  
 position=begin+1;  
 begin = position;  
 tempNode = rootNode;  
 } else if (tempNode.isEnd()) {  
 result.append(replacement);  
 position++;  
 begin = position;  
 } else {  
 position++;  
 }  
 }  
 return result.toString();  
}

## 5.2 系统业务模块的开发与实现

本章详细介绍了本系统各个模块的实现方法，由于dao层和service层的代码较为简单且重复度较高。而且根据其函数名就能直观的看出这两层的函数作用。考虑到篇幅，本章仅在5.2.2 讨论问答模块详细介绍这两层的实现方法，其余模块仅展示controller层的业务关键代码。同样，前端的模板展示仅在5.2.1用户模块详细介绍。

### 5.2.1 用户模块

用户登录注册部分的业务较为常规。为了保证安全，在用户注册时，使用加盐的方式来储存密码，即储存的密码为用户输入的密码加上随机生成的盐的md5码，在用户登录时，查询出用户的盐，再将用户输入的密码加上盐的md5码与储存的密码比对。没有明文储存的密码，一定程度上保证了用户的信息安全。

在用户登录时，会发放token给用户，可以指定过期时间，在用户退出时，需要将用户的token状态设为失效。

Token的验证以及用户信息的获取则使用了aop来实现：

Token的dao层：

public interface LoginTicketDao extends CrudRepository<LoginTicketEntity, Long> {  
 LoginTicketEntity findByTicket(String ticket);  
}

Service层关键代码：

public String addLoginTicket(int userId) {  
 LoginTicketEntity loginTicketEntity = new LoginTicketEntity();  
 loginTicketEntity.setUserId(userId);  
 Date now = new Date();  
 now.setTime(3600 \* 1000 \*1 + now.getTime());  
 loginTicketEntity.setExpired(now);  
 loginTicketEntity.setStatus(0);  
 loginTicketEntity.setTicket(UUID.*randomUUID*().toString().replaceAll("-", ""));  
 loginTicketDao.save(loginTicketEntity);  
 return loginTicketEntity.getTicket();  
}

当访问任意页面，token拦截器-passportInterceptor捕获请求，若cookie中存在合法toke，则将用户信息读取出来存放在临时的hosthoder中，供后续操作调用，若不存在合法token，则为未登录用户。passportInterceptor实现了接口HandlerInterceptor，代码为

@Component  
public class PassportInterceptor implements HandlerInterceptor {  
  
 @Autowired  
 LoginTicketDao loginTicketDao;  
  
 @Autowired  
 UserDao userDao;  
  
 @Autowired  
 HostHolder hostHolder;  
  
 @Override  
 public boolean preHandle(HttpServletRequest request, HttpServletResponse response, Object handler) throws Exception {  
 String ticket = null;  
 if (request.getCookies() != null) {  
 for (Cookie cookie : request.getCookies()) {  
 if (cookie.getName().equals("ticket")) {  
 ticket = cookie.getValue();  
 break;  
 }  
 }  
 }  
  
 if (ticket != null) {  
 LoginTicketEntity loginTicketEntity = loginTicketDao.findByTicket(ticket);  
 if (loginTicketEntity == null || loginTicketEntity.getExpired().before(new Date()) || loginTicketEntity.getStatus() != 0) {  
 return true;  
 }  
  
 UserEntity userEntity = userDao.findById(loginTicketEntity.getUserId());  
 hostHolder.setUsers(userEntity);  
 }  
 return true;  
 }  
  
 @Override  
 public void postHandle(HttpServletRequest request, HttpServletResponse response, Object handler, ModelAndView modelAndView) throws Exception {  
 if (modelAndView != null) {  
 modelAndView.addObject("user", hostHolder.getUser());//将用户信息传送给前端  
 }  
 }  
  
 @Override  
 public void afterCompletion(HttpServletRequest request, HttpServletResponse response, Object handler, Exception ex) throws Exception {  
 hostHolder.clear();  
 }

其中，重载方法preHandle()在请求处理之前调用，Spring框架中的Interceptor是链式调用的，在一个请求中可以同时调用多个Interceptor，调用顺序会根据Interceptor的注册声明顺序依次执行，注册Interceptor的代码会在后文给出。在这个方法中，取出cookie中的ticker，并判断合法性，用合法的ticket查询出用户信息。

重载方法postHandle()在preHandle（）方法返回true后，在DispatcherServlet进行试图返回渲染之前被调用，所以在这个方法中，对modelAndView对象进行操作。在有多个Interceptor的情况下此方法的调用顺序与preHandle()相反。

重载方法afterCompletion()在preHandle（）方法返回true后，在DispatcherServlet进行试图返回渲染之后被调用，用来进行资源清理。

modelAndView对象被操作后，前端页面展示信息的关键代码为：

<div class="top-nav-profile" >  
 <ul class="topnav-noauth clearfix" th:unless="${user}">  
 <li>  
 <a href="/reglogin">注册/登陆</a>  
 </li>  
 </ul>  
 <span th:if="${user}">  
 <a th:href="@{'/user/'+${user.id}}" class="zu-top-nav-userinfo " id=":0" role="button" aria-haspopup="true" aria-activedescendant="">  
 <span class="name" th:text="${user.name}"></span>  
 <img class="Avatar" th:src="@{${user.headUrl}}" srcset="https://pic1.zhimg.com/da8e974dc\_xs.jpg 2x" alt="huzhi">  
 <span id="zh-top-nav-new-pm" class="zg-noti-number zu-top-nav-pm-count" style="visibility:hidden" data-count="0">  
 </span>  
 </a>  
 <ul class="top-nav-dropdown" id="top-nav-profile-dropdown" aria-labelledby=":0">  
 <li>  
 <a th:href="@{'/user/'+${user.id}}" tabindex="-1" id=":1">  
 <i class="zg-icon zg-icon-dd-home"></i>我的主页  
 </a>  
 </li>

当访问登录用户可查看页面，login拦截器-LogintInterceptor捕获请求，查看hosthoder中是否有用户信息，若没有，则重定向到登录界面。LogintInterceptor和passportInterceptor类似，在重定向时添加next参数，使用户登录后可以回到原来的页面，关键代码为：

public boolean preHandle(HttpServletRequest request, HttpServletResponse response, Object handler) throws Exception {  
 if (hostHolder.getUser() == null) {  
 response.sendRedirect("/reglogin?next=" +request.getRequestURI());  
 }  
 return true;  
}

注册拦截器的关键代码为：

@Override  
public void addInterceptors(InterceptorRegistry registry) {  
 super.addInterceptors(registry);  
 registry.addInterceptor(passportInterceptor);  
 registry.addInterceptor(loginStatusInterceptor).addPathPatterns("/user/\*");  
}

### 5.2.2 讨论问答模块

（1）Dao层：

QuestionDao，函数作用可由函数名得知，其中pageable为分页参数，作用类似sql语句中的offset，limit。

public interface QuestionDao extends CrudRepository<QuestionEntity, Long> {  
   
 QuestionEntity findById(int id);  
   
 List<QuestionEntity> findAllByUserId(int userId, Pageable pageable);  
  
 List<QuestionEntity> findAllBy(Pageable pageable);

CommentDao与QuestionDao类似，多了两个count的函数用于之后的业务。

public interface CommentDao extends CrudRepository<CommentEntity, Long> {  
  
 List<CommentEntity> findAllByEntityIdAndEntityType(int entity\_id, int entity\_type);  
  
 int countByEntityIdAndEntityType(int entity\_id, int entity\_type);  
  
 CommentEntity findAllById(int id);  
  
 int countAllByUserId(int userId);  
}

点赞功能直接调用jedis的函数即可，不需要自行编写dao。

（2）Service层：

QuestionService，写了增加问题，获取问题，和更新问题等函数供controller层调用，其中增加问题需要调用敏感词屏蔽函数：

@Service  
public class QuestionService {  
 @Autowired  
 QuestionDao questionDao;  
 @Autowired  
 SensitiveService sensitiveService;  
  
 public int addQuestion(QuestionEntity questionEntity) {  
 questionEntity.setContent(HtmlUtils.*htmlEscape*(questionEntity.getContent()));  
 questionEntity.setTitle(HtmlUtils.*htmlEscape*(questionEntity.getTitle()));  
 questionEntity.setTitle(sensitiveService.filter(questionEntity.getTitle()));  
 questionEntity.setContent(sensitiveService.filter(questionEntity.getContent()));  
  
 return questionDao.save(questionEntity) != null ? questionEntity.getId() : 0;  
 }  
  
 public List<QuestionEntity> getLatestQuestion(int userId, int offert, int limit) {  
 if (userId != 0) {  
 return questionDao.findAllByUserId(userId, new PageRequest(offert, limit, Sort.Direction.*DESC*, "id"));  
 } else {  
 return questionDao.findAllBy(new PageRequest(offert, limit, Sort.Direction.*DESC*, "id"));  
 }  
  
 }  
  
 public int updateComment(int id, int commentCount) {  
 QuestionEntity questionEntity = questionDao.findById(id);  
 questionEntity.setCommentCount(commentCount);  
 return questionDao.save(questionEntity) != null ? questionEntity.getId() : 0 ;  
 }  
  
 public QuestionEntity getQuestionById(int id) {  
 return questionDao.findById(id);  
 }  
   
  
}

CommentService，写了获取评论，增加评论以及统计评论数量的函数供后续调用，其中增加评论函数需要调用敏感词过滤函数。

@Service  
public class CommentService {  
 @Autowired  
 CommentDao commentDao;  
  
 @Autowired  
 SensitiveService sensitiveService;  
  
 public List<CommentEntity> getCommentsByEntity(int entityId, int entityType) {  
 return commentDao.findAllByEntityIdAndEntityType(entityId, entityType);  
 }  
  
 public int addComment(CommentEntity commentEntity) {  
 commentEntity.setContent(sensitiveService.filter(commentEntity.getContent()));  
 commentEntity.setContent(HtmlUtils.*htmlEscape*(commentEntity.getContent()));  
 return commentDao.save(commentEntity) != null ? commentEntity.getId() : 0;  
 }  
  
 public int getCommentCount(int entityId, int entityType) {  
 return commentDao.countByEntityIdAndEntityType(entityId, entityType);  
 }  
  
 public CommentEntity getCommentById(int id) {  
 return commentDao.findAllById(id);  
 }  
  
 public int getUserCommentCount(int userId) {  
 return commentDao.countAllByUserId(userId);  
 }  
  
  
}

LikeService，写了获取点赞数量，获取点赞状态和踩/赞函数，调用jedis中各种对set操作的函数即可：

@Service  
public class LikeService {  
 @Autowired  
 JedisAdapter jedisAdapter;  
  
 public long getLikeCount(int entityType, int entityId) {  
 String likeKey = RedisKeyUtil.*getLikeKey*(entityType, entityId);  
 return jedisAdapter.scard(likeKey);  
 }  
  
 public int getLikeStatus(int userId, int entityType, int entityId) {  
 String likeKey = RedisKeyUtil.*getLikeKey*(entityType, entityId);  
 if (jedisAdapter.sismember(likeKey, String.*valueOf*(userId))) {  
 return 1;  
 }  
 String disLikeKey = RedisKeyUtil.*getDisLikeKey*(entityType, entityId);  
 return jedisAdapter.sismember(disLikeKey, String.*valueOf*(userId)) ? -1 : 0;  
 }  
  
 public long like(int userId, int entityType, int entityId) {  
 String likeKey = RedisKeyUtil.*getLikeKey*(entityType, entityId);  
 jedisAdapter.sadd(likeKey, String.*valueOf*(userId));  
  
 String disLikeKey = RedisKeyUtil.*getDisLikeKey*(entityType, entityId);  
 jedisAdapter.srem(disLikeKey, String.*valueOf*(userId));  
  
 return jedisAdapter.scard(likeKey);  
 }  
  
 public long disLike(int userId, int entityType, int entityId) {  
  
 String disLikeKey = RedisKeyUtil.*getDisLikeKey*(entityType, entityId);  
 jedisAdapter.sadd(disLikeKey, String.*valueOf*(userId));  
  
 String likeKey = RedisKeyUtil.*getLikeKey*(entityType, entityId);  
 jedisAdapter.srem(likeKey, String.*valueOf*(userId));  
  
  
  
 return jedisAdapter.scard(likeKey);  
 }

在调用Jedis的方法之前，使用一个类对Jedis方法进行简单封装，将try catch写在封装好的方法中，这样使得服务层的代码更简洁更易读。

1. controller层

增加问题，需要验证用户是否登录和数据是否正确入库，成功则返回0，失败则返回1并记录失败原因到日志。关键代码如下：

@RequestMapping(value = "/question/add", method = {RequestMethod.*POST*})  
@ResponseBody  
public String addQuestion(@RequestParam("title") String title,  
 @RequestParam("content") String content) {  
 try {  
 QuestionEntity questionEntity = new QuestionEntity();  
 questionEntity.setTitle(title);  
 questionEntity.setContent(content);  
 questionEntity.setCreatedDate(new Date());  
 questionEntity.setCommentCount(0);  
 if (hostHolder.getUser() == null) {  
 throw new Exception("用户未登录");  
 } else {  
 questionEntity.setUserId(hostHolder.getUser().getId());  
 }  
 if (questionService.addQuestion(questionEntity) > 0) {  
 return ZhiHuUtil.*getJSONString*(0);  
 }  
 } catch (Exception e) {  
 *logger*.error("提问失败:" + e.getMessage());  
 }  
 return ZhiHuUtil.*getJSONString*(1, "失败");  
}

增加评论和增加问题类似，需要指定entityType为Question,同时增加评论成功后，需要更新对应question中的评论数量。更新question的中的评论数量操作可以放到消息队列中异步处理。关键代码如下：

@RequestMapping(path = {"/addComment"}, method = {RequestMethod.*POST*})  
public String addComment(@RequestParam("questionId") int qusetionId,  
 @RequestParam("content") String content) {  
 try {  
 CommentEntity commentEntity = new CommentEntity();  
 commentEntity.setEntityType(EntityType.*ENTITY\_QUESTION*);  
 commentEntity.setEntityId(qusetionId);  
 commentEntity.setCreatedDate(new Date());  
 if (hostHolder.getUser() == null) {  
 return "redirect:/";  
 } else {  
 commentEntity.setUserId(hostHolder.getUser().getId());  
 }  
 commentEntity.setStatus(0);  
 commentEntity.setContent(content);  
 commentService.addComment(commentEntity);  
  
 //修改评论数量  
 // *TODO: 2019/3/21 放入消息队列异步处理；* int count = commentService.getCommentCount(commentEntity.getEntityId(), commentEntity.getEntityType());  
 questionService.updateComment(commentEntity.getEntityId(), count);  
 } catch (Exception e) {  
 *logger*.error("增加评论失败" + e.getMessage());  
 }  
 return "redirect:/question/" + qusetionId;  
}

展示具体问题则通过service层函数查询该问题以及其评论的具体信息，包括每个评论的点赞情况。关键代码如下：

@RequestMapping(value = "/question/{qid}", method = {RequestMethod.*GET*})  
public String questionDetail(Model model, @PathVariable("qid") int qid) {  
 QuestionEntity questionEntity = questionService.getQuestionById(qid);  
 model.addAttribute("question", questionEntity);  
 List<CommentEntity> commentList = commentService.getCommentsByEntity(qid, EntityType.*ENTITY\_QUESTION*);  
 List<ViewObject> vos = new ArrayList<>();  
 for (CommentEntity comment : commentList) {  
 ViewObject vo = new ViewObject();  
 vo.set("likeCount", likeService.getLikeCount(EntityType.*ENTITY\_COMMENT*, comment.getId()));  
 if (hostHolder.getUser() == null) {  
 vo.set("liked", 0);  
 } else {  
 vo.set("liked", likeService.getLikeStatus(hostHolder.getUser().getId(), EntityType.*ENTITY\_COMMENT*, comment.getId()));  
 }  
 vo.set("comment", comment);  
 vo.set("user", userService.getUser(comment.getUserId()));  
 vos.add(vo);  
 }  
 model.addAttribute("comments", vos);

点赞则需要在点赞的给消息队列推送一个发站内信的消息，让系统通过站内信业务异步的通知被点赞的用户。代码如下：

@RequestMapping(path = {"/like"}, method = {RequestMethod.*POST*})  
@ResponseBody  
public String like(@RequestParam("commentId") int commentId) {  
 if (hostHolder.getUser() == null) {  
 return ZhiHuUtil.*getJSONString*(999);  
 }  
  
 CommentEntity commentEntity=commentService.getCommentById(commentId);  
 eventProducer.fireEvent(new EventModel(EventType.*LIKE*)  
 .setActorId(hostHolder.getUser().getId()).setEntityId(commentId).setEntityOwnerId(commentEntity.getUserId())  
 .setEntityType(EntityType.*ENTITY\_COMMENT*).setExt("questionId",String.*valueOf*(commentEntity.getEntityId())));  
 long likeCount = likeService.like(hostHolder.getUser().getId(), EntityType.*ENTITY\_COMMENT*, commentId);  
 return ZhiHuUtil.*getJSONString*(0, String.*valueOf*(likeCount));  
}

### 5.2.3 站内信模块

获取站内信列表需要验证用户登录情况，通过已登录用户信息查询站内信，关键代码如下：

@RequestMapping(path = {"/msg/list"}, method = {RequestMethod.*GET*})  
public String getConversationList(Model model) {  
 if (hostHolder.getUser() == null) {  
 return "redirect:/reglogin";  
 }  
 int localUserId = hostHolder.getUser().getId();  
 List<MessageEntity> conversationList = messageService.getConversationList(localUserId, 0, 10);  
 List<ViewObject> conversations = new ArrayList<ViewObject>();  
 for (MessageEntity message : conversationList) {  
 message.setId(messageService.getConversationCount(message.getConversationId()));  
 ViewObject vo = new ViewObject();  
 vo.set("message", message);  
 int targetId = message.getFormId() == localUserId ? message.getToId() : message.getFormId();  
 vo.set("user", userService.getUser(targetId));  
 vo.set("unread", messageService.getConversationUnreadCount(localUserId, message.getConversationId()));  
 conversations.add(vo);  
 }  
 model.addAttribute("conversations", conversations);  
 return "letter";  
}

其中getConversationList()函数调用的dao中的findList函数的查询较为复杂，很难用接口CrudRepository中的写法表示，所以此处使用了jpa中的sql源语句语句为：

@Query(value = "SELECT \* ,*count*(id) as id" + " FROM (SELECT \* FROM message WHERE form\_id=:userId or to\_id= :userId order by created\_date desc )"  
 + "tt group by conversation\_id order BY created\_date desc ",  
 nativeQuery = true)  
List<MessageEntity> findlist(@Param("userId") int userId,  
 Pageable pageable);

获取某个会话的所有问题则相对比较简单，只需要通过会话id查询即可：

@RequestMapping(path = {"/msg/detail"}, method = {RequestMethod.*GET*})  
public String getConversationDetail(Model model, @RequestParam("conversationId") String conversationId) {  
 try {  
 List<MessageEntity> messageList = messageService.getConversationDetail(conversationId, 0, 10);  
 List<ViewObject> messages = new ArrayList<ViewObject>();  
 for (MessageEntity message : messageList) {  
 ViewObject vo = new ViewObject();  
 vo.set("message", message);  
 vo.set("user", userService.getUser(message.getFormId()));  
 messages.add(vo);  
 }  
 model.addAttribute("messages", messages);  
 } catch (Exception e) {  
 *logger*.error("获取详情失败" + e.getMessage());  
 }  
 return "letterDetail";  
}

发送站内信则需要先后验证用户是否为登录用户以及收信用户是否存在，详细代码如下：

@RequestMapping(path = {"/msg/addMessage"}, method = {RequestMethod.*POST*})  
@ResponseBody  
public String addMessage(@RequestParam("toName") String toName,  
 @RequestParam("content") String content) {  
 try {  
 if (hostHolder.getUser() == null) {  
 return ZhiHuUtil.*getJSONString*(999, "未登录");  
 }  
  
 UserEntity user = userService.getUserByName(toName);  
 if (user == null) {  
 return ZhiHuUtil.*getJSONString*(1, "用户不存在");  
 }  
  
 MessageEntity message = new MessageEntity();  
 message.setCreatedDate(new Date());  
 message.setFormId(hostHolder.getUser().getId());  
 message.setToId(user.getId());  
 message.setContent(content);  
 messageService.addMessage(message);  
 return ZhiHuUtil.*getJSONString*(0);  
  
 } catch (Exception e) {  
 *logger*.error("发送消息失败" + e.getMessage());  
 return ZhiHuUtil.*getJSONString*(1, "发信失败");  
 }  
}

### 5.2.4 粉丝关注模块

关注需要先验证用户身份，在关注者加入被关注者的排序集合后向被关注者异步发送站内信。用entityType区分关注的是问题还是人。

关注问题的关键代码：

@RequestMapping(path = {"/followQuestion"}, method = {RequestMethod.*POST*})  
@ResponseBody  
public String followQiestion(@RequestParam("questionId") int questionId) {  
 if (hostHolder.getUser() == null) {  
 return ZhiHuUtil.*getJSONString*(999);  
 }  
 QuestionEntity questionEntity = questionService.getQuestionById(questionId);  
 if (questionEntity == null) {  
 return ZhiHuUtil.*getJSONString*(1, "不存在的问题");  
 }  
 boolean res = followService.follow(hostHolder.getUser().getId(), EntityType.*ENTITY\_QUESTION*, questionId);  
 eventProducer.fireEvent(new EventModel(EventType.*FOLLOW*)  
 .setActorId(hostHolder.getUser().getId()).setEntityId(questionId)  
 .setEntityType(EntityType.*ENTITY\_QUESTION*).setEntityOwnerId(questionEntity.getUserId()));

关注人的关键代码：

@RequestMapping(path = {"/followUser"}, method = {RequestMethod.*POST*})  
@ResponseBody  
public String follow(@RequestParam("userId") int userId) {  
 if (hostHolder.getUser() == null) {  
 return ZhiHuUtil.*getJSONString*(999);  
 }  
 boolean res = followService.follow(hostHolder.getUser().getId(), EntityType.*ENTITY\_USER*, userId);  
 eventProducer.fireEvent(new EventModel(EventType.*FOLLOW*)  
 .setActorId(hostHolder.getUser().getId()).setEntityId(userId)  
 .setEntityType(EntityType.*ENTITY\_USER*).setEntityOwnerId(userId));  
 return ZhiHuUtil.*getJSONString*(res ? 0 : 1, String.*valueOf*(followService.getFolloweeCount( hostHolder.getUser().getId(),EntityType.*ENTITY\_USER*)));  
  
}

在编写FollowService的函数时，因为需要同时更新关注列表和粉丝列表，所以需要使用Redis事务，Jedis使用事务较为简单，直接调用方法即可。

取关与关注类似，只需要改动函数名即可，取消关注则不需要给被取消关注着发送站内信。

生成timeline的关键代码：

@RequestMapping(path = {"/pullfeeds"}, method = {RequestMethod.*GET*, RequestMethod.*POST*})  
private String getPullFeeds(Model model) {  
 int localUserId = hostHolder.getUser() != null ? hostHolder.getUser().getId() : 0;  
 List<Integer> followees = new ArrayList<>();  
 if (localUserId != 0) {  
 // 关注的人  
 followees = followService.getFollowees(localUserId, EntityType.*ENTITY\_USER*, Integer.*MAX\_VALUE*);  
 }  
 List<Feed> feeds = feedService.getUserFeeds(Integer.*MAX\_VALUE*, followees, 10);  
 model.addAttribute("feeds", feeds);  
 return "feeds";  
}

### 5.2.5 技术讨论与发展方向

本系统实现timeline的方法即为简单的pull模式，此方法也称为读扩散，此方法的优点是实现简单，节省储存空间。我们知道，在实际使用feed流产品中，对大多数用户而言，读取feed流的时候的延迟敏感度远远大于发布是的延迟敏感度，而pull模式恰恰在读取时会造成最大的系统资源开销（数据库查询），所以此模式虽然实现简单，但显然不是最佳的feed流方案。

在数据量增大后，势必遇到性能瓶颈，目前主流的feed流系统推送模式为写扩散（push模式），即feed消息进入队列后，获取粉丝列表，将自己的feed消息写给自己的所有粉丝，这样，用户在读取feed时，直接读取自己的feed消息表，此时就消灭了读取的高延迟。此方法非常适合使用在类似朋友圈这样的用户关系较为平均的社区中。然而在类似微博这样的存在大V（拥有巨量粉丝的用户）的社区中，大V的存在会让数据膨胀的非常夸张，同样会造成储存瓶颈和性能瓶颈。

所以更多的社区中，feed系统采用推拉结合的方式，区分大v和普通用户，此时，发布feed的流程图5.1所示：

A

B

C

D

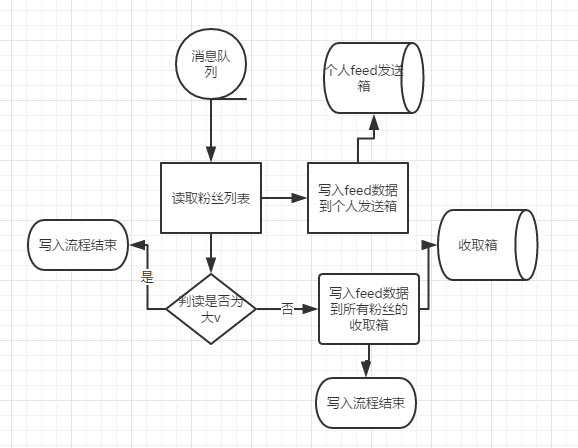


图5.1 推拉结合系统中发布feed流程

读取feed流的流程如图5.2所示：

A

B

C

D

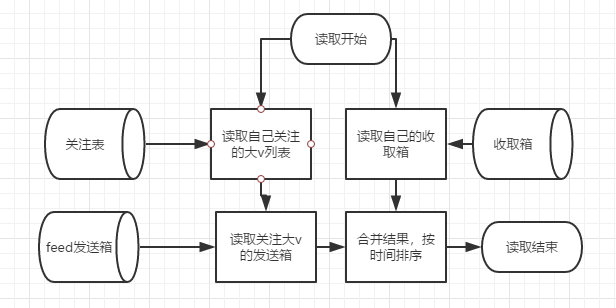


图5.2 推拉结合系统中读取feed流程

还有通过是否活跃，距离上次上线时间的长短来分类用户，对于活跃用户，使用推模式，对于不活跃用户，使用拉模式。不管是什么方法，将大V，活跃用户id集合储存在Redis这样的内存数据库中，有助于提高查找效率。

在未来的发展中，按照需求本社区还可能更新热度榜功能已经引入问答评价算法。在此可稍作讨论。

热度榜是对问题（主题）的排序，将每个主题的回答数，关注主题的用户数量，该主题下的所有回答的点赞/点踩，各个回答的浏览量等因素加权，获得一个热度值，通过这个热度值，对整个问题列表进行排序，得到热度榜。同时，由于热度的时效性，可以引入半衰期算法，图5.3展示了一个半衰期算法的函数曲线：

A

C

D

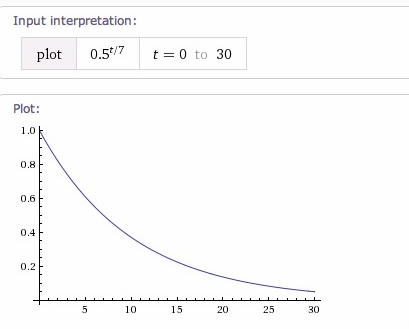


图5.3 半衰期算法函数曲线

可以通过设定不同的半衰期来区分不同类型的话题的热度时间，比如关于选课，突发事件这样的短时间热门话题，半衰期可以设置为一周，对于学术讨论，学校政策讨论这类长时效的问题，可以设定较长的半衰期，这样就能保证热度榜上的项目大多数为新鲜的问题。

问答评价算法则作用于问题下的回答，好的回答需要有更高的权重，得到更多的展示机会，评价算法通过点赞数将每个回答获取一个基本的rank值，然后根据回答的篇幅，图片，公式，表格的引用数量来修正每个回答的rank值，力求将更用心的回答展示给更多的用户。甚至可以开放老师作为一种新的用户角色，使老师拥有推荐回答的权限，被推荐的回答将拥有更高的rank值。

系统的分布式部署也是后续需要考虑的问题，本系统在单服务器上使用docker部署，很容易通过Kubernetes这样的跨主机集群的自动部署、扩展以及运行应用程序容器的平台改造为分布式服务，分布式部署后只需要更多的机器就能容易的提升服务器的性能。在分布式部署后，对数据库的分库分表改造，对业务进行拆分等操作，又将进一步提升系统的性能。

# 第6章 项目部署与性能测试

## 6.1 项目部署

本项目所有组件通过docker部署在阿里云ecs中，基本信息见表格6.1

表6.1 运行服务器信息

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| vCPU | cpu型号 | 主频 | 带宽 | 内存 | 操作系统 |
| 1 vCPU | Intel Xeon E5-2682v4 | 2.5 GHz | 0.5 Gbps | 2 GiB | CentOs 7.3 64位 |

各个组件均有官方镜像，所以直接使用官方镜像部署即可。

Redis部署：

docker run -p 46379:6379 -v /docker/redis/:/data -d redis:3.2 redis-server --appendonly yes

Tomcat部署：

docker pull tomcat

docker run -p 58080:8080 tomcat

Mysql部署：

docker run --name mysql -d --restart always -p 43306:3306 -e MYSQL\_ROOT\_PASSWORD= -v /docker/mysql\_data:/var/lib/mysql mysql/mysql-server

Docker运行情况见图6.1

A

B

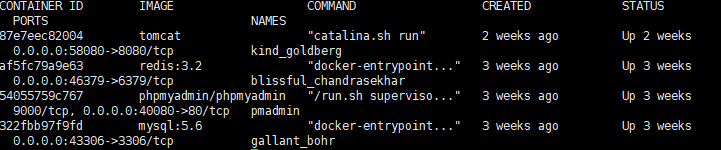


图6.1 项目docker运行情况

通过nginx将tomcat反向代理，nginx代理tomcat关键配置为：

server {

listen 80 default\_server;

listen [::]:80 default\_server;

server\_name \_;

root /usr/share/nginx/html;

# Load configuration files for the default server block.

include /etc/nginx/default.d/\*.conf;

location /pma/ {

proxy\_pass http://localhost:40080/;

proxy\_redirect default;

}

location / {

proxy\_pass http://localhost:58080/zhihufake-0.0.1-SNAPSHOT/;

}

location /zhihufake-0.0.1-SNAPSHOT/{

proxy\_pass http://localhost:58080/zhihufake-0.0.1-SNAPSHOT/;

}

同时也使用Nginx反向代理了phpmyadmin等服务供开发使用，此处省略这些配置。

## 6.2 性能测试

使用apache bench对项目进行简单的压力测试。ab命令会创建很多的并发访问线程，模拟多个访问者同时对某一URL地址进行访问。它的测试目标是基于URL的，因此，既可以用来测试Apache的负载压力，也可以测试nginx、lighthttp、tomcat、IIS等其它Web服务器的压力。

ab命令对发出负载的计算机要求很低，既不会占用很高CPU，也不会占用很多内存，但却会给目标服务器造成巨大的负载，其原理类似CC攻击。自己测试使用也须注意，否则一次上太多的负载，可能造成目标服务器因资源耗完，严重时甚至导致死机。

测试结果见表6.2：

表6.2 apache bench测试结果

|  |  |  |  |
| --- | --- | --- | --- |
|  | 测试1 | 测试2 | 测试3 |
| 并发请求数 | 10 | 10 | 1 |
| 整个测试持续的时间/s | 1.982 | 26.988 | 26.081 |
| 完成的请求数 | 100 | 1000 | 1000 |
| 失败的请求数 | 0 | 0 | 0 |
| 整个场景中的网络传输量/bytes | 2573900 | 25739000 | 25739000 |
| 吞吐率[#/sec] | 50.45 | 37.05 | 38.34 |
| 用户平均请求等待时间/ms | 198.233 | 269.881 | 26.081 |
| 服务器平均请求处理时间/ms | 19.823 | 26.988 | 26.081 |

由于本系统部署的服务器是个人使用的云服务，内存和cpu数都较低，与实际部署的服务器有一定的差别，根据结果估算，本系统性能大致达到设计需求。

# 结论

此问答社区系统的开发旨在使用较新的技术构建一个符合现在主流架构的校内web社区。能够满足同学日常交流使用需要，可以支撑校方发布通知，并能支持选课级别的流量。

这套系统通过四个模块完成了一个社区系统的基本功能，功能可靠稳定，符合设计规范。具有快速，高可用，可拓展，可控性高的特点。

设计了基于redis的消息队列异步处理框架，框架拓展性强，使用便捷。

设计了基于前缀树的敏感词过滤算法，过滤效果符合预期要求。

设计了基于拉取的timeline生成，在网站流量不大的情况下符合要求。

当然，本系统也有很多不足之处，例如在功能上缺少对用户身份验证的功能，导致用户无法找回密码；对发布的回答的结构有限制，仅仅能使用文字，无法贴图，缺少对各种格式的支持；用户个性化自定义的功能较少；缺少管理员的角色；没有做问题、评论的修改与删除功能，在技术方面缺少缓存的支持；没有使用效果更好的feed流推送方式等，离真正的可以使用的社区还有一定的距离。

当然一个完善的系统是需要在实际使用中不断改进的，所以我将会在后续的开发中继续完善该系统。

# 参考文献

1. 余君泽. 深入分布式缓存从原理到实践[M].北京:机械工业出版社,2017.
2. Qifan Pu. FairRide: Near-Optimal, Fair Cache Sharing [J]. NSDI.2016,16:19.
3. 李家智. Spring Boot 2精髓：从构建小系统到架构分布式大系统 [M].北京：机械工业出版社，2017.
4. Felix Gessert. Quaestor:Query Web Caching for Database-as-a-Service Providers[D].New York:Proceedings of the VLDB Endowment ,2017.
5. 彭灿华. Redis-在高速缓存系统中的序列化算法研究 [J]. 现代电子技术.2017,10(22):19.
6. 王伟,魏乐,刘文清,舒红平.基于ElasticSearch的分布式全文搜索系统 [J]. 电子科技.2018,8(2):17.
7. Pautasso, Wilde, Erik, Alarcon, Rosa. REST: Advanced Research Topics and Practical Applications [M]. New York :Springer-Verlag ,2014.
8. Stubbs, Joe Moreira, Walter Dooley. Distributed Systems of Microservices Using Docker and Serfnode[C]. Rion. International Workshop on Science Gateways conference.New York: :, 2016.
9. 邓一贵,伍玉英. 基于文本内容的敏感词决策树信息过滤算法[J]. 计算机工程.2014,9(1):127-131.
10. 王丹,刘立家,林九川,赵文兵,杜晓林.基于DOM状态转换检测XSS漏洞[J]. 北京工业大学学报.2018, 9(4):10.
11. 潘伟,仲倩黎,付晓东等.基于 Spring 框架的工作流引擎设计与实现[J].东北师大学报（自然科学版）,2007,39(3):27-31.DOI:10.3321/j.issn:1000-1832.2007.03.006.
12. Xiaofeng Wu,Shengtao Luo.Design and Implementation of College OA System Based on UML and J2EE[C].//Sensors, measurement and intelligent materials II : Part 2 /.2014:1195-1200.
13. 杨曼,徐东平.基于 Web 服务的动态电子商务的研究[J].计算机工程与设计.2006,27(5):82282-4.
14. 许国艳,李晓芳.J2EE 平台上基于 Web 服务的动态电子商务架构研究[J].微电子学与计算机.2006,23（增）:18-19.
15. 刘斌,王最.Struts,Spring,Hibernate 框架在开发中的应用[J].计算机技术与发展,2010,20(1):151-154,162.DOI:10.3969/j.issn.1673-629X.2010.01.040.

# 致谢

感谢彭彬导师在定题上的帮助，在论文和毕设过程中的指导和建议，感谢家人同学一直以来的支持。