**潍坊学院**

**本科生毕业论文（设计）工作记录**

题 目： 开放社区论坛系统设计与实现

学 院： 计算机工程学院

专 业： 网络工程

姓 名： 付立威

学 号： 17026240040

指导教师： 薛莹

年 月 日

## 摘要

自中国改革开放以来，随着互联网的发展， 人与人之间的交流方式愈来愈多。网上聊天，网上视频早已不是什么稀奇的事，后来，为了人们在某一专业领域进行深入的的探讨与发言，出现了在线论坛(BBS)，大家可以就自己的学识经验进行提出问题和针对已有的问题发表自己的见解。BBS论坛系统已经成为人们获取信息、发表言论的重要场所。

在当今社会的很多居民社区中，大多数物业通知小区用户的方式还都是贴纸质告示在单元楼下，这样需要耗费很大人力而且用户容易错过一些重要的通知，这时候如果有一个社区专用的论坛，将会事半功倍，所以开发一个社区论坛系统还是很有必要的。开放社区论坛系统，该论坛使用多元化的社区服务模式，其主要目的就是为了丰富社区居民的生活方式。业主和物业等可以自行注册账号，在上面自由发布一些需求和通知，用户间可以互相交流，兴趣多样，话题讨论丰富，更好的方便大家的生活。

本系统是基于B/S结构的一个小型论坛，主要基于Windows平台，使用SpringBoot框架整合SSM框架和MySQL数据库进行开发的，还使用了很多中间件技术如Redis、ElasticSearch、Kafka等 ，通过这些技术，可以实现一个相对完整的论坛系统。

**[关键词]** 社区论坛、SpringBoot框架、SSM框架、MySQL数据库

## Abstract

## 目录

[1.前言 5](#_Toc70512960)

[2.系统总体分析 6](#_Toc70512961)

[2.1系统运行环境 6](#_Toc70512962)

[2.2系统所需技术 6](#_Toc70512963)

[3.系统总体设计 7](#_Toc70512964)

[3.1数据库设计 7](#_Toc70512965)

[4.系统总体功能实现 8](#_Toc70512966)

[4.1注册 8](#_Toc70512967)

[4.2登录 8](#_Toc70512968)

[5.测试与部署 9](#_Toc70512969)

[6.结论 10](#_Toc70512970)

[致谢 11](#_Toc70512971)

[参考文献 12](#_Toc70512972)

[附录 13](#_Toc70512973)

## 前言

BBS的英文全称是Bulletin Board System，中文的意思是“电子公告板”。随着中国网络的普及以及计算机及其外设的大幅降价，Internet这个概念已经存在于每个21世纪居民的心中。中国网民的数量呈几何级增长，BBS的普及数量也将近Email。每个网站几乎都拥有自己的BBS或者BBS连接。BBS可以通过 WEB浏览器访问，并且能实现诸私信评论等诸多功能。网民们逐步接收这种使用便捷，日渐强大的系统，在系统上网民可以张贴和发布各种各样的信息，讨论各式各样的话题。

论坛系统服务亦将是互联网站中一种极为常见的互动交流服务，论坛可以向网友提供开放性的分类专题讨论区服务，可以在此发表自己的某些观感、交流某些技术、经验乃至人生的感悟与忧欢，亦可以作为网友们之间的交流渠道，亦可以在此回答网友们提出的问题或发布的某些信息。通过论坛的方式，增强个人与个人，集体与集体，个人与集体之间的交流联系。在Internet上实现信息的传递，提高办事效率。因此论坛建设在Internet应用上的地位显而易见，他已经成为现代人沟通和获取信息的重要组成部分，而备受人们重视。

Java是现在主流的编程语言，SSM框架更是现在企业开发用的很多的框架组合，它们都得到了广泛的承认与支持，SMM框架可以很多的和Java技术结合在一起，创建一个更为稳定、高效、安全的运行环境，从而实现非常复杂的企业级应用系统。所以我选择使用Java语言来开发这个社区论坛系统，来作为我的毕业设计。

对于社区而言，该论坛可以从房地产开发的开始就吸引用户。 此外，所有者社区的独特“小圈子”特征使页面上的活动用户非常准确。比如在阿里巴巴旗下的产品“闲鱼”中，就有以小区为单位的“鱼塘”，所谓的鱼塘就是一个聚集地，其目的就是为了方便大家，在这种情况下，您可以参与多个鱼塘，并快速了解周围的人在卖什么，例如在1公里和2公里之内的朋友在卖什么。非常直观，更贴近生活。但是这种仅仅局限于在二手交易上，我觉得一个社区可以有很多丰富的多元化的东西，不仅仅只是如此。

本文主要实现了基于B/S模式的一种web论坛项目的设计与实现，主要功能是实现客户端与服务器的动态交互。BBS论坛最基本的功能就是发帖与回帖，此外，为了记录主题的发表者和主题的回复者，系统需要提供用户注册和登录的功能，只有在登录之后才能进行发表和回复，游客只能浏览不能发言。

写点什么呢。。

## 系统总体分析

### 2.1系统介绍

本系统是B/S结构的小型论坛，主要基于Windows操作系统 ，通过这个论坛，社区居民之间可以方便地在线上互相交流沟通。

### 2.2需求分析

#### 1.功能性需求

开发BBS论坛系统的目的是提供一个供用户交流的平台,为广大用户提供交流经验、探讨问题的社区。因此，BBS论坛系统最基本的功能首先是发表主题,其次是其他人员根据主题发表白己的看法。此外，为了记录主题的发表者和主题的加复者信息民，系统还需要提供用户注册和登录的功能。只有注册的用户登录后才能够发表和回复主题，浏览者(游客)只能浏览主题信息。根据用户的需求及以上的分析，BBS论坛需要其备前台功能和后台功能。

(1)用户权限：显示各论坛类别及版面、查看版面下所有根帖、查看精华帖子、查看自己发表的帖子、搜索帖子、查看根帖内容、用户注册、用户登录、发表帖子、回复帖子。

(2)管理员权限：进入后台、论坛类别管理、版面管理、用户管理、用户注册。

#### 2.使用需求

(1)操作平台：基于Windows操作系统，不需要安装其他软件。

(2)操作方式：所有操作都能基于浏览器进行。

### 2.3可行性分析？

### 2.4系统开发环境及简介

* 构建工具：Apache Maven
* 集成开发环境：IDEA
* 快速开发框架：SpringBoot整合Sring+SpringMVC+Mybatis
* 页面模板引擎：Thymleaf
* 数据库：MySQL
* 应用服务器：Apache Tomcat
* 版本控制工具：Git
* 中间件：Redis、ElasticSearch、Kafka、Caffeine
* 第三方工具： Wkhtmltopdf + kaptcha
* 权限框架：Spring Securtiy + Spring Actuator

Spring框架：  
Spring是一个轻量级的开源Java框架。 它是为解决企业级别应用程序开发的复杂性而创建的。 该框架的主要优点之一是其分层体系结构。 Spring的实现不限于服务器端开发，任何Java应用程序都将从Spring中获得增强和改善。它是一个分层架构，主要包含五大模块：Core Container，AOP，WEB，Data Access，Test。

Spring MVC框架：

SpringMVC是一个视图层框架，主要用于处理用户所接受到所看到的页面，早期SpringMVC还不流行的时候，市面上用的MVC框架都是Struts2，但是因为Struts2框架暴露出来一些安全问题，SpringMVC上手容易配置简单，所以它已经成为广大企业优先选择的MVC框架，这也是我在这个项目中为什么选择使用它来作为视图层框架的原因。SpringMVC共有六大组件：DisPatcherServlet、HandlerMapping、HandLer、HandlerAdapter、ViewResolver、View。

MyBatis框架：

MyBatis框架是一个独特的开源数据持久层框架。 他内部完成了JDBC的建立，使用JDBC访问数据库的一系列操作，支持常规的SQL查询、高级映射和存储过程，并且几乎消除了手动写JDBC连接代码的操作。 作为持久层框架的MyBatis，其目的就是为了让大量的SQL语句从程序中脱离出来，把它们写在配置文件中，这样可以灵活地配置SQL语句，不用每次都改代码。

SSM框架：

SSM框架就是Spring 框架+ Spring MVC框架 + MyBatis框架的一个整合。SSM框架是继SSH框架之后，现在很多公司都在使用的JavaEE企业开发框架，适用于各种大型公司来开发很复杂的企业级应用系统。

Redis：

Redis是一个内存型的非关系型数据库，由于它的数据存放于内存中，所以读写速度非常快，性能很高，不过也正是因为他的数据都是存在于内存中的，内存的空间相比于硬盘的空间来说可是稀少且珍贵的，故我们不能把所有的数据都存放于Redis中，大多数我们Redis来存储一些热点数据，也就是一些会被经常访问到的数据。

与Redis同样的内存型数据库还有memcached，但是memcached只能存储一种数据类型，那就是字符串。Redis可以存储五种不同的数据类型，有字符串、列表、集合、有序集合、hash散列表，这些丰富的数据类型使得Redis可以应用于诸多场景，而且Redis还支持本地方法，这些都是memcached所没有的，故这个系统选择了Redis作为内存型数据库。

Kafka：

Kafka是一个分布式的流处理平台。流处理是说的对连续产生的动态数据进行实时处理，该处理基于分布式内存，并具有高效、快速、低延迟的特特点。它同时具又队列模式和发布-订阅模型，使用者不必再纠结于怎样选择消息系统模型。

Kafka提供的关键功能包括：日志收集、用户行为跟踪、消息系统和流数据处理。

ElasticSearch：

ElasticSearch简称ES，它是一个开源的可分布式的分析与搜索引擎，底层基于lucene实现的搜索服务。当客户端调用ES的功能时，可以实现全文检索的各种操作，并且不需要关心lucene的底层代码，只要客户端支持http协议就可以了，非常的方便。

它的主要功能是：对海量数据进行检索和分析，适用的数据类型丰富，应用场景诸多，如网站搜索、应用程序搜索、日志的分析处理、业务分析、安全分析等。

Caffeine：

Caffeine是一个高性能的基于Java8的缓存库，它允许显式地去控制缓存的更新、删除和检索。它支持设置缓存大小、设置过期时间等。

SpringSecurity框架;

基于Spring框架，Spring Security为Web应用程序安全性提供了完整的解决方案。通常，Web应用程序安全性包括两个部分：用户身份验证和授权。用户身份验证是检查用户是否合法受制于系统，即用户是否有权访问系统。用户身份验证通常要求用户输入用户名和密码。系统通过验证用户名和密码来完成认证过程。用户身份验证是检查用户是否具有执行特定操作的权限。系统为每个用户具有不同的特权。例如，对于文件，某些用户只能读取该文件，而某些用户可以对其进行修改。通常，系统将不同的角色分配给不同的用户，并且每个角色对应一组特权。

Thymeleaf：

Thymeleaf是一个模板引擎，改引擎采用Java开发，它是独立环境下的服务器端Java模板引擎，面向Web ，可以处理HTML，XML，JavaScript，CSS乃至纯文本。它的优点是高度可维护，基于自然模板的概念，其逻辑被插入到模板文件中，而不会影响其用作设计原型。这样可以改善设计沟通，并缩小设计团队与开发团队之间的差距。故使用它可以完全代替JSP。

SpringActuator：

SpringActuator是基于SpringBoot的一个对程序的运行情况的生命周期的监视和管理程序。它可以自动提供一组端点来监视由Spring Boot构建的应用程序。 并且使用它所需的配置可以忽略不计。

SpringBoot框架：

SpringBoot框架其被设计出来的目的就是简化开发，秉承“约定优于配置”的设计理念，它内置许多默认配置，使得开发者近乎0配置，就可以让项目快速的跑起来。此外它还有一个重要的特点，它集成了非常多的第三方库的配置信息，近乎所有的常用框架，它都有相对应的组件的支持，它使得这些第三方库可以开箱即用，使得开发者不必花很多心思在配置组件上，而更专注于业务逻辑代码的开发。所以它的优点就显而易见了，这也是它这么流行的原因，本设计中，采用SpringBoot整合SSM框架和Redis、Kafka等中间件，非常方便高校。

Maven：

Maven是一个开源的项目的管理工具和构建工具，它的主要组成组件是项目对象模型、依赖管理系统、项目生命周期和各种插件。有了它，我们不需要自己管理导入依赖jar包，直接告诉Maven项目需要那些依赖jar包，它会帮助我们下载所有依赖项。

POM文件是 Maven工程的核心文件，它是一个XML文件，包含了描述项目如何构建、声明依赖插件、项目的基本信息等。

开发人员只需要配置pom.xml这个文件，就可以很方便的管理Maven项目，使得开发事半功倍。

## 系统总体设计

### 3.1 系统目标

1.实现帖子浏览分页查看详情功能

2.实现用户的注册登录及权限控制功能

3.实现帖子的置顶拉黑功能

4.实现用户点赞评论和私信功能

5.实现用户在线发布帖子功能

6.实现管理员管理普通用户功能

7.用户可以查看自己发布的帖子列表

8.保证系统化的易操作性和易维护性

9.保证系统运行安全稳定

### 3.2系统功能结构

未登录的游客可以浏览帖子，但不能点赞评论和私信帖子主人，游客注册登录之后，成为正式用户，可以解锁发布帖子、点赞、评论、私信等功能，用户可以查看编辑自己发布的帖子。社区管理员用户可以管理用户和所有帖子。

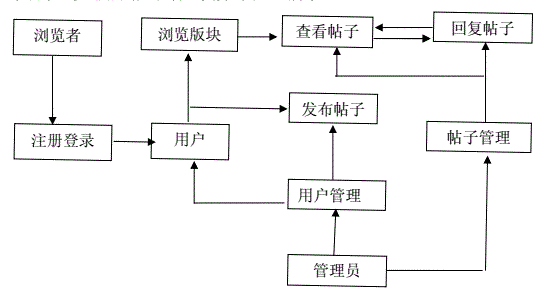


图3.1-1 系统运行流程总览

### 3.3系统流程

#### 3.3.1 注册流程

1．用户打开注册页面。

2．填写注册信息。

3．点击注册按钮，把注册的信息发送给服务器。

4．登录注册填写的邮箱中找到激活邮件。

5．打开邮件中的激活链接打开网页，完成注册。

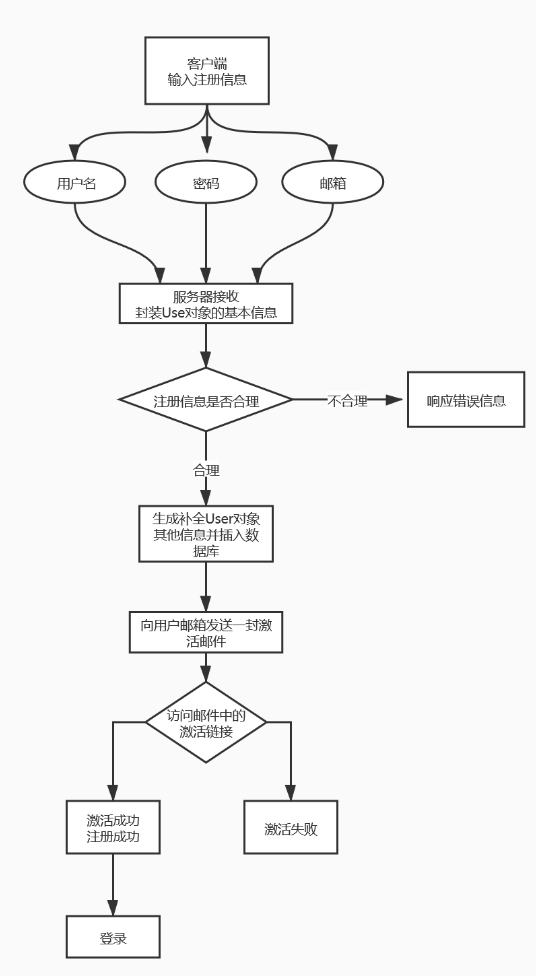


图3.3-1 用户注册流程总览

#### 3.3.2登录流程

1.客户端输入登录信息。

2.点击登录按钮将信息提交到服务器。

3.服务器收到登录信息进行有效性验证。

4.验证通过后生成一个登录凭证到Cookie中。

5.服务器将Cookie响应给客户端，之后用户再访问就会携带凭证Cookie，服务器就会将其判定为登录状态。

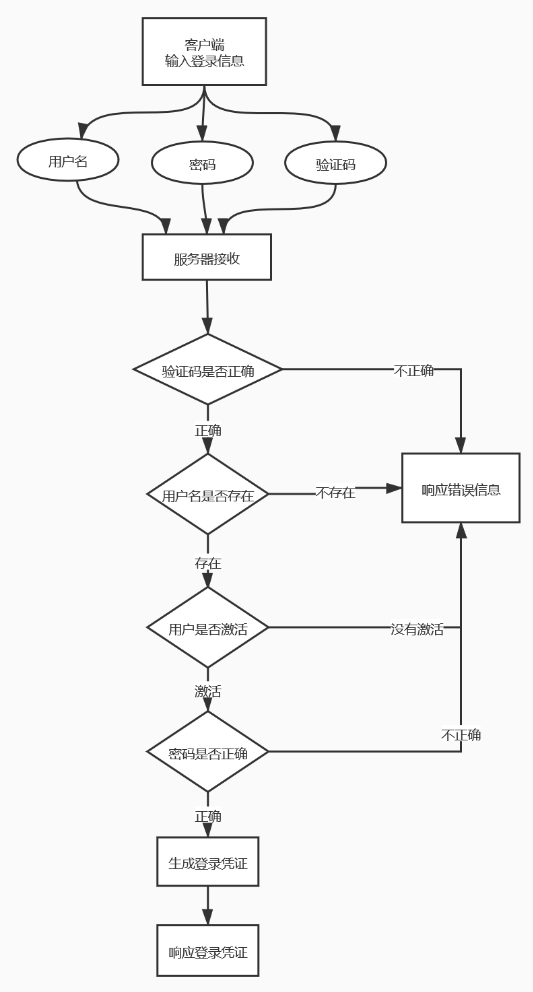


图3.3-2 用户登录流程总览

### 3.4数据库设计

#### 用户表 user

|  |  |  |
| --- | --- | --- |
| 字段 | 类型 | 备注 |
| id | int | 主键、自增 |
| username | varchar | 用户名 |
| password | varchar | 用户密码 |
| salt | varchar | 加密盐值 |
| email | varchar | 用户邮箱 |
| type | int | 用户类型 |
| status | int | 用户状态 |
| activation\_code | varchar | 激活码 |
| header\_url | varchar | 用户头像地址 |
| create\_time | timestamp | 注册时间 |

#### 评论表 comment

|  |  |  |
| --- | --- | --- |
| 字段 | 类型 | 备注 |
| id | int | 主键、自增 |
| user\_id | int | 评论的用户 id |
| entity\_id | int | 评论实体 id |
| entity\_type | int | 评论实体类型 |
| target\_id | int | 评论目标 id |
| content | text | 评论内容 |
| status | int | 评论状态 |
| create\_time | timestamp | 评论发表时间 |

#### 帖子表 discuss\_post

|  |  |  |
| --- | --- | --- |
| 字段 | 类型 | 备注 |
| id | int | 主键、自增 |
| user\_id | int | 发帖的用户 id |
| title | varchar | 帖子表标题 |
| content | text | 帖子内容 |
| type | int | 帖子类型 |
| comment\_count | int | 评论数量 |
| status | int | 帖子状态 |
| create\_time | timestamp | 评论发表时间 |

#### 用户登录凭证表 login\_ticket

|  |  |  |
| --- | --- | --- |
| 字段 | 类型 | 备注 |
| id | int | 主键、自增 |
| user\_id | int | 登录用户 id |
| ticket | varchar | 登录凭证，随机字符串 |
| status | int | 登录状态 |
| expired | timestamp | 过期时间 |

#### 消息表 message

|  |  |  |
| --- | --- | --- |
| 字段 | 类型 | 备注 |
| id | int | 主键、自增 |
| from\_id | int | 发消息的 id |
| to\_id | int | 收消息的 id |
| conversation\_id | varchar | 会话 id，由通信双方 id 拼接 |
| content | text | 消息内容 |
| status | int | 消息状态 |
| create\_time | timestamp | 消息发送时间 |

## 4.系统总体功能实现

### 4.1开发社区首页

用户访问首页的时候，要展示帖子列表，不能一次性全显示出来，要使用分页功能，创建一个分页JavaBean对象Page，包含当前页码，每页显示条目数，总条目数，查询路径这四个属性。

查询的数据库表是discuss\_post，排序规则是置顶精华类型优先展示，剩余帖子按热度进行排序。

由于还要展示用户的信息，所以还要查询user表。

**开发数据访问层：**

* 首先创建discuss\_post 表对应的 DisscussPost 实体类，作为查询数据库返回的接收对象
* 创建 DisscussPostMapper 接口，并标注 @Mapper 注解，让Spring扫描并加载到Bean容器中。
* 创建 disscusspost-mapper.xml文件，编写查询的SQL语句。
  + 注意：排除状态为拉黑的帖子，如果传入用户id字段，说明查询的是某用户帖子列表，需要添加条件。

**开发业务层：**

创建 DiscussPostService 类，在类上声明@Service注解，新增查询帖子详情方法，查询帖子数量方法。

创建 UserService 类，在类上声明@Service注解，新增根据用户id查询用户方法，展示帖子的时候不能展示用户id，要展示用户名。

**开发视图层：**

创建static目录， css、html、img、js 静态资源等静态资源。

创建template目录，存放mail、site、index.html 等文件。

创建 HomeController，声明@Controller注解，新增getIndexPage方法，将用户帖子信息封装到Model对象中，方法返回值为字符串，返回index主页路径。

修改 index.html文件，将里面需要加载到页面上的值用th标签动态替换。

**这里需要一张图片！**

### 4.2开发注册登录模块

#### 发送邮件

发送邮件的功能是必要的，因为需要给用户的邮箱发送激活邮件，在自己常用的邮箱设置中心开启SMTP服务，无论是哪种邮箱都会有的，记下来用户名和和访问SMTP服务的密码。

在pom中引入 spring-boot-starter-mail 依赖。

在application.properties文件中配置邮件服务器主机、端口、发件人邮箱、授权码等信息。

新建工具类：MailClient，封装使用JavaMailSender 组件发送邮件功能。

写一个邮件用的HTML模板，使用 thymeleaf 加载模板并替换相应的信息，调用 TemplateEngine 把注册用户的信息加载到 HTML 模板中。

#### 注册功能

新建工具类： CommunityUtil，在工具类新建产生随机字符串UUID和 传入字符串获取MD5 加密值的方法。

在 LoginController，新建 getRegisterPage 方法，返回register.html路径使其跳转到注册页面。

在 UserService 中新建 register 方法，供LoginController调用，用于判断传入的User对象包含的注册信息是否合理，如果合理生成密码盐和激活码插入数据库，并向用户邮箱发送激活邮件。

新建接口 CommunityConstant，定义激活码的成功、重复和失败三种状态，让 UserService 和 LoginController 实现此接口，

发送激活邮件成功后，用户登录自己的邮箱，点击邮件中的链接，访问到LoginController 的 activation 方法，如果激活链接中的激活码和数据库中设置的一样，就把对应用户的 status 改为 1，表示已经激活。

#### 生成验证码

在用户登录的时候，要求用户输入验证码，防止机器程序恶意操作。

首先在 pom.xml文件中导入 kaptcha 工具类的 jar 包。

新建配置类 KaptchaConfig，配置验证码的范围、长度、大小等。

在 LoginController 中新建 getKaptcha 方法获取生成的验证码图片。

在 login.html 中，将验证码img标签的src属性替换成LoginController中的getKaptcha的RequestMapping路径，将刷新验证码的超链接绑定单击事件 refresh\_kaptcha js方法，通过 Jquery的id 选择器获取 验证码img 组件，使其实现单击刷新验证码就会改变img的src属性访问后台 getKaptcha 方法生成新的验证码图片。

#### 登录退出功能

系统判定用户是否登录成功是在用户登录信息正确无误后响应给用户一个带着登录凭证的Cookie，要实现退出登录功能，只需将次Cookie的过期时间设置为立即过期，这样客户端浏览器再访问的时候，就不会携带有登录凭证的Cookie，这样就实现了退出登录的效果。

存放登录凭证的数据库表是login\_ticket，其中expire字段表示过期时间，status表示用户登录是否有效。

创建对应与login\_ticket 表对应的 LoginTicket 实体类，用于封装查询数据库的信息。

新建 LoginTicketMapper 接口，定义新增用户凭证、查询用户凭证和设置用户凭证状态的三个方法，分别通过 @Insert、@Select、@Update 注解来标注在方法之上，写入SQL语句来进行插入、查询、更新凭证。

在 UserServce中，新建login方法，改方法首先验证客户端传入的用户名密码是否正确，如果正确则将合规的用户的凭证信息系插入到数据库login\_ticket表。新建logout调用LoginTicketMapper更新凭证方法，将凭证状态置为失效状态。

在 LoginController中，新建login方法，访问UserServce的login方法，并接收返回值map对象，判断map中是否包含ticket键，如果包含，则代表登录成功，此时重定向到首页。若不包含则表示登录失败，此时回显错误信息到登录页面。

#### 显示登录信息

创建 CookieUtil 工具类，通过 name 查询对应 cookie 的 value。

在 UserService 中新增 findLoginTicket 方法，根据 ticket 查询 LoginTicket。

创建 HostHolder 类用来模拟 session 的功能，利用 ThreadLocal 实现，存储用户信息。

创建 LoginTicketInterceptor 拦截器，实现 HandlerInterceptor 接口。

* 在 preHandle 方法中通过 CookieUtil 的 getValue 方法查询是否有凭证 cookie，如果有则通过 UserService 的 findloginTicket 方法查询用户 ID，再通过用户 ID 查询用户。最后将用户放入 hostHolder 中。
* 在 postHandle 方法中通过 hostHolder 的 get 方法获取用户，并将其存入视图中。
* 在 afterCompletion 方法中清除 hostHolder 中存放的用户信息。

创建 WebMvcConfig 配置类，实现 WebMvcConfigurer接口，配置 LoginTicketInterceptor，拦截除了静态资源之外的所有路径。

#### 上传头像

在 UserService 新增 updateHeader 方法，更改指定用户的头像。

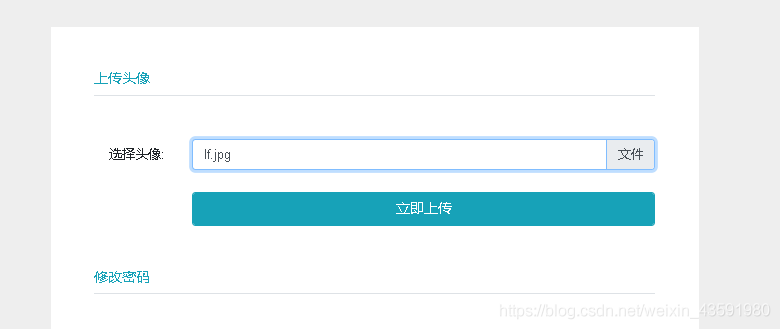
创建 UserController

* 新增 getSettingPage 方法访问账户设置 setting.html ，并在 index.html 的账号设置按钮关联该链接。
* 新增 uploadHeader 方法更新用户头像，如果上传出现错误将错误信息存在 Model 对象中。

如果没有错误，生成一个文件对象 dest，利用 MultipartFile 接口的 transferTo 方法将用户上传文件导入 dest，并从 hostHolder 中取出用户，更新用户的头像路径。

* 新增 getHeader 方法获取用户头像，利用文件输入流读取图片数据，利用 HttpServletResponse 的字节输出流再进行输出。

调整 setting.html 的 form 表单， method="post"，enctype="multipart/form-data"，并设置提交路径。



在这里插入图片描述

#### 修改密码

在 UserService 中新增 changePassword 方法，判断原密码是否正确，正确则修改密码并返回 1，否则返回 0。

在 UserController 中新增 changePassword 方法，根据 UserService 的 changePassword 方法的返回值判断原密码是否成功修改，封装为 JSON 数据并返回。

在 setting.html 中

* 首先在[前端](/jump/super-jump/word?word=%E5%89%8D%E7%AB%AF)判断两次输入的新密码是否一致，如果不一致不允许点击提交并显示错误信息。
* 利用 ajax 向 UserController 的 changePassword 方法发送 POST 请求，得到 JSON 数据并解析，如果状态码为 0 提示错误，如果状态码为 1 弹出修改成功提示。

【问题】js 的虚拟路径问题，需要加上 ../。

【问题】使用 ajax 请求时，表单按钮类型必须是 button，不能是 submit，否则 405 报错。

【问题】使用 ajax 请求时，Controller 中方法的返回值必须是 JSON 数据，并且需要加上 @ResponseBody。

【问题】使用 ajax 请求时，回调函数需要先对返回的 JSON 数据进行解析再使用。

#### 检查登录状态

利用拦截器，实现只处理带有自定义注解的方法，防止用户在未登录情况下通过 url 访问没有权限的页面。

创建 @LoginRequired 自定义注解，作用范围在方法上，有效期为运行时。

在 UserController 中需要在登录状态下调用的方法，访问设置页面、修改密码、上传头像等加上自定义注解。

创建 LoginRequiredInterceptor 拦截器，在 preHandle 方法中判断方法是否加了 @LoginRequired 注解，如果加了注解并且此时从 hostHolder 中获取不到用户则拒绝访问。

在 WebMvcConfig 配置类配置 LoginRequiredInterceptor，拦截除了静态资源之外的所有路径。

### 4.3开发核心功能

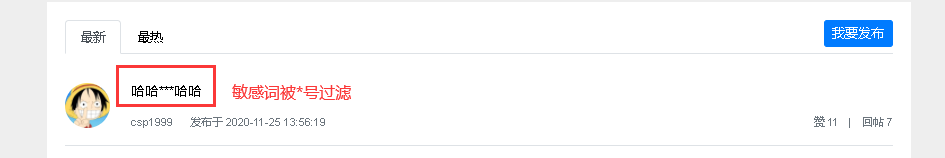
#### 敏感词过滤

利用字典树数据结构解决。

创建 SensitiveFilter 类

* 创建静态内部类 TrieNode ，通过 boolean 类型的结束符判断是否匹配到关键字尾部。
* 利用 @PostConstruct 注解，在构造方法执行后初始化字典树。
* 添加 filter 方法，利用双指针进行匹配，过滤敏感词。

【问题】判断子节点空时，直接添加了一个 new 的子节点，没有将对象赋值给子节点变量。



#### 发布帖子

引入 fastjson 依赖，在 CommunityUtil 中新增 getJSONString 方法封装 JSON 信息。

在 DisscussPostMapper 接口新增 insertDiscussPost 方法，并在 disscusspost-mapper.xml 配置 insert 语句。

在 DiscussPostService 新增 addDiscussPost 方法调用 DisscussPostMapper 的 insertDiscussPost 方法，其中需要进行对标题内容和发帖内容进行 HTML 转义以及过滤敏感词。

创建 DiscussPostController 类，新增 addDiscussPost 方法，调用 DiscussPostService 的 addDiscussPost 方法发帖。

在 index.html 中为发帖按钮绑定函数，利用 Ajax 向 DiscussPostController 的 addDiscussPost 方法发送 POST 请求。



#### 显示帖子内容

在 DisscussPostMapper 接口新增 selectDiscussPostById 方法，在 disscusspost-mapper.xml 配置 select 语句。

在 DiscussPostService 新增 findDiscussPostById 方法调用 DisscussPostMapper 的 selectDiscussPostById 方法。

在 DiscussPostController 新增 getDiscussPost 方法，调用 DiscussPostService 的 findDiscussPostById 方法查询帖子内容，将 DiscussPost 对象和 User 对象（通过 userId 查询，不在 DAO 层关联查询）数据存放到 Model 对象，返回模板 discuss-detail。

在 discuss-detail.html 取出 Model 对象存放的数据绑定到对应组件显示。

#### 显示评论（comment 表）

创建 comment 表对应的实体类 Comment。

创建 CommentMapper 接口

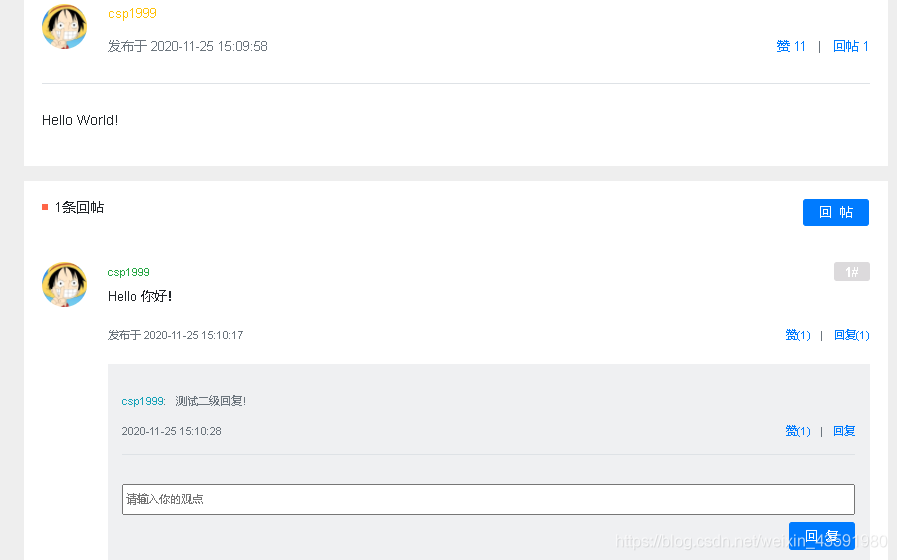
* 新增 selectCommentsByEntity 方法，根据实体查询一页的评论数据。
* 新增 selectCountByEntity 方法，根据实体查询评论的数量。
* 在 comment-mapper.xml 配置 select 语句。

创建 CommentService 类

* 新增 findCommentByEntity 方法，调用 CommentMapper 的 selectCommentByEntity 方法。
* 新增 findCommentCount 方法，调用 CommentMapper 的 selectCountByEntity 方法。

在 DiscussPostController 的 getDiscussPost 方法中增加查询帖子评论和回复的逻辑，将结果存储在 Model 对象。

【问题】sql 的 xml 文件中绑定参数时，应传入实体类属性名，拼错成数据库字段名（entityId 写成 entity\_id）。



#### 添加评论

在 CommentMapper 接口新增 insertComment 方法，添加评论数据，在 comment-mapper 配置对应 sql。

在 DiscussPostMapper 接口新增 updateCommentCount 方法，增加评论数量，在 discusspost-mapper 配置对应 sql。

在 DiscussPostService 类新增 updateCommentCount 方法，调用 DiscussPostMapper 的 updateCommentCount 方法。

在 CommentService 类新增 addComment 方法，调用 CommentMapper 的 insertComment 新增评论，并调用 DiscussPostService 的 updateCommentCount 更新评论数量，使用 @Transactional 注解保证事务。

创建 CommentController 类，新增 addComment 方法，从 hostHolder 获取用户信息，然后调用 CommentService 的 addComment 方法添加评论。

【问题】sql 的 xml 文件中绑定参数时，应传入实体类属性名，拼错成数据库字段名（entityId 写成 entity\_id）。

#### 显示私信列表 （message 表）

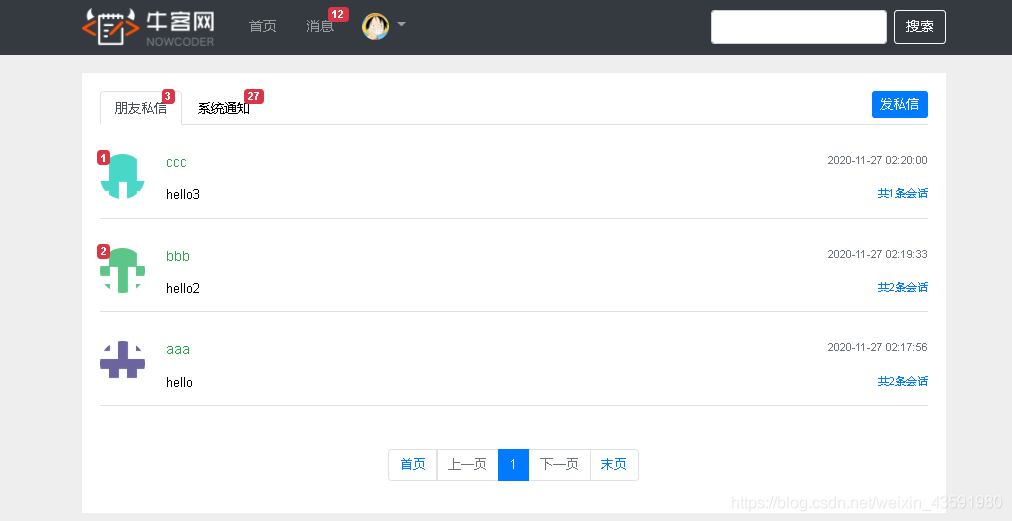
创建对应 message 表的实体类 Message。

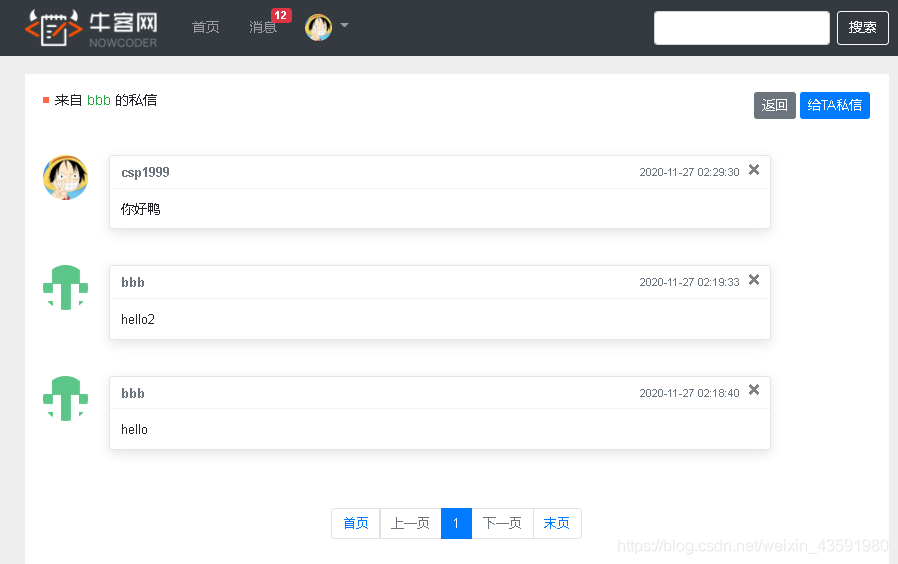
创建 MessageMapper 接口，增加查询会话列表、会话数量、私信列表、私信数量、未读私信数量等方法，在 message-mapper.xml 中配置对应的 sql。

创建 MessageService，调用 MessageMapper 中的方法。

创建 MessgaeController

* 新增 getLetterList 方法，将会话列表信息存储到 Model 对象，返回 letter 视图。
* 新增 getLetterDetail 方法，将每个会话具体的私信信息存储到 Model 对象，返回 letter-datail 视图。





#### 发送私信

在 MessageMapper

* 新增 insertMessage 方法插入私信记录，在 message-mapper.xml 配置 insert 语句。
* 新增 updateMessgae 方法修改私信状态，在 message-mapper.xml 配置 update 语句，利用 foreach 动态 sql。

在 MessageService

* 新增 addMessage 发送私信方法，过滤敏感词后，调用 MessageMapper 的 insertMessage 。
* 新增 readMessage 方法读取信息，调用MessageMapper 的 updateMessgae 更新私信的状态为 1。

在 MessageController

* 新增 getLetterIds 方法，将私信集合中未读私信的 id 添加到 List 集合并返回，在 getLetterDetail 方法调用该方法设置已读。
* 新增 sendLetter 发送私信方法，设置私信信息后调用 MessageService 的 addMessage 发送。



#### 统一异常处理

@ControllerAdvice

用于修饰类，表示该类是Controller 的全局适配类。

在此类中，可以对Controller 进行如下三种全局配置：

🚡 异常处理方案

👶 绑定数据方案

🐤 绑定参数方案

@ExceptionHandler(我们实例中使用则个注解修饰方法)

用于修饰方法，该方法会在Controller 出现异常后被调用，用于处理捕获到的异常。

@ModelAttribute

用于修饰方法，该方法回在Controller 方法执行前被调用，用于为Model 对象绑定参数。

@DataBinder

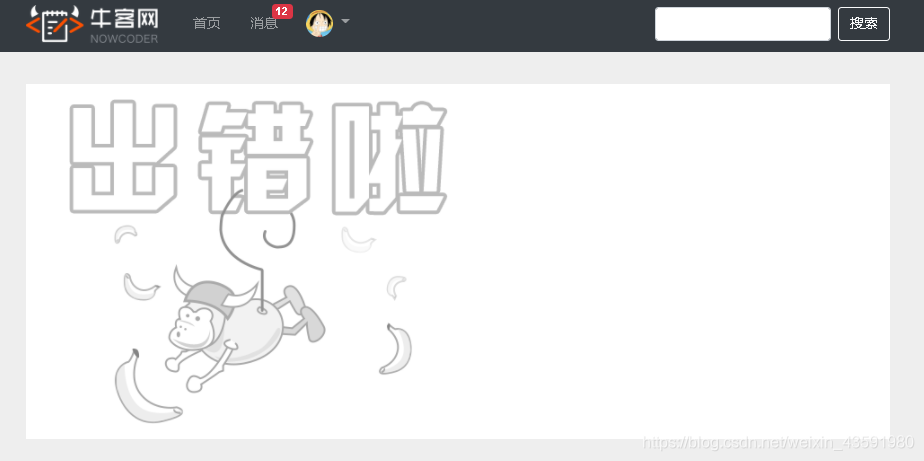
用于修饰方法，该方法回在Controller 方法执行前被调用，用于绑定参数的转换器。

在 HomeController 中增加 getErrorPage 方法，返回错误页面。

创建 ExceptionAdvice 类

* 加上 @ControllerAdvice 注解，表示该类是 Controller 的全局配置类。
* 创建 handleException 方法，加上 @ExceptionHandler 注解，该方法在 Controller 出现异常后调用，处理捕获异常。如果是异步请求返回一个 JSON 数据，否则重定向至 HomeController 的 getErrorPage 方法。



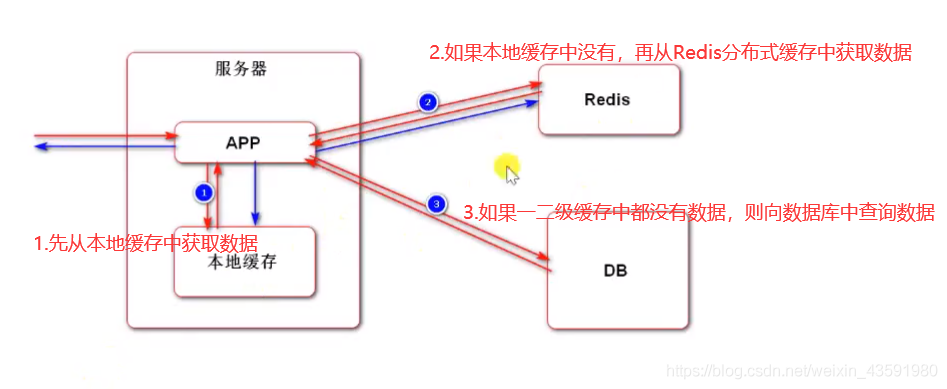


#### 统一日志处理

在 pom.xml 引入 aspectj 的依赖。

创建 ServiceLogAspect 类，添加 @Aspect 切面注解，配置切入点表达式，拦截所有 service 包下的方法，利用 @Before 记录日志。

### Redis



#### 点赞

创建 RedisKeyUtil 工具类

* 定义分隔符 : 以及实体获得赞的 key 前缀常量 like:entity。
* 新增 getEntityLikeKey(int entityType,int entityId) 方法，通过实体类型和实体 id 生成对应实体获得赞的 key。

创建业务层的 LikeService 类

* 注入 RedisTemplate 实例。
* 新增 like 点赞方法，首先通过 RedisKeyUtil 工具类的 getEntityLikeKey 方法获得实体点赞的 key，然后通过 RedisTemplate 对象对 set 集合的 isMember 方法查询 userId 是否存在于对应 key 的 set 集合中，如果存在则移除出点赞的用户集合，如果不存在则添加到点赞的用户集合。
* 新增 findEntityLikeCount 方法查询实体的点赞数量，通过调用 set 集合的 size 方法查询元素个数。
* 新增 findEntityLikeStatus 方法查询某用户对某实体的点赞状态，逻辑如 like 方法，通过 set 集合的 isMember 方法实现。

创建表现层的 LikeController 类

* 注入 LikeService 和 HostHolder 实例。
* 新增 like 点赞方法，调用业务层的 like 方法进行点赞、调用 findEntityLikeCount 和 findEntityLikeStatus 查询点赞数量和点赞状态，封装到 map 集合，然后通过工具类封装成 JSON 数据返回。

（更新首页帖子点赞数量）在表现层的 HomeController 类

* 注入 LikeService 实例。
* 在 getIndexPage 方法在通过 LikeService 类的方法获得点赞数量，存储到 map 集合。

#### 收到的赞

对点赞功能进行重构

在 RedisUnitl 工具类

* 新增用户获得赞 key 的前缀常量 like:user
* 新增 getUserLikeKey(int userId) 方法，通过用户 id 生成对应用户获得赞的 key。

在 LikeService 中

* 重构 like 方法，在参数列表中加入 entityUserId 表示被点赞用户的 id，用来更新用户的被点赞数量。
  + 通过 RedisTemplate 对象的 execute 方法实现事务，保证被点赞用户点和点赞用户的数据更新一致。通过 isMember 方法查询用户的点赞状态，之后通过 mutli 方法开启事务。
  + 当用户已点赞时，调用 remove 方法将当前用户从点赞用户的集合中移除，调用 decrement 方法将被点赞用户的被点赞数减 1；当用户未点赞时，调用 add 方法将当前用户添加到点赞用户的集合，调用 increment 方法将被点赞用户的被点赞数加 1。
* 增加 findUserLikeCount 方法，以用户 id 作为 key，调用 get 方法查询用户所获得的点赞数。

在 LikeController 中给 like 方法增加 entityUserId 参数即可。

#### 关注

在 RedisUnitl 工具类

* 新增用户关注实体（帖子、评论、用户等）和粉丝（用户）的前缀常量 followee 和 follower
* 新增 getFolloweeKey(int userId, int entityType) 方法，通过用户 id 和实体类型生成用户关注实体的 key。
* 新增 getFollowerKey(int entityType, int entityId) 方法，通过实体类型和实体 id 生成实体用户粉丝的 key。

创建业务层的 FollowService 类

* 新增 follow 方法，当用户关注某实体时，
  + 调用 add 方法将当前实体 id 和时间作为 value 和 score加入用户的关注集合。
  + 调用 add 方法将当前用户 id 和时间作为 value 和 score 加入实体的粉丝集合。
* 新增 unfollow 方法，当用户取消关注某实体时，
  + 调用 remove 方法将当前实体从用户的关注集合移除。
  + 调用 remove 方法将用户从实体的粉丝集合移除。

#### 个人主页

在业务层的 FollowService 类

* 新增 findFolloweeCount 方法，调用 zset 的 zcard 方法查询某用户关注的实体数量。
* 新增 findFollowerCount 方法，调用 zset 的 zcard 方法查询某实体的粉丝数量。
* 新增 hasFollowed 方法，根据 zset 的 zscore 方法返回值查询当前用户是否关注某实体。

在 UserController 中新增 getProfilePage 方法获取个人主页。

* 调用 LikeService 的 findUserLikeCount 查询用户获赞数，并添加到 Model 中。
* 调用 FollowService 的findFolloweeCount、findFollowerCount 、hasFollowed 方法分别查询关注数量、粉丝数量、用户是否关注三项信息并添加到 Model 对象中存储。

#### 关注列表和粉丝列表

在业务层的 FollowService 类

* 新增 findFollowees 方法，查询用户关注列表，主要通过 zset 的 reverseRange 获取 value 即关注用户的 userId，再查询出其 user，之后通过 score 获取关注时间，存入 map 集合，将 map 添加到 list 列表返回。
* 新增 findFollowers 方法，查询用户粉丝列表，主要通过 zset 的 reverseRange 获取 value 即粉丝的 userId，再查询出其 user，之后通过 score 获取关注时间，存入 map 集合，将 map 添加到 list 列表返回。

在表现层的 FollowController 类

* 新增 getFollowees 方法，获取关注列表，存入 Model 对象。
* 新增 getFollowers 方法，获取粉丝列表，存入 Model 对象。

#### 优化登录模块

**存储验证码**

在 RedisUntil 工具类

* 新增验证码前缀常量 kaptcha
* 新增 getKaptchaKey 方法，通过一个用户凭证（由于未登录，利用 cookie 实现）获得对应验证码的 key 值（利用 string 存储验证码）。

在表现层的 LoginController 类

* 重构 getKaptcha 方法，将验证码存入 [redis](/jump/super-jump/word?word=redis)，key 值是当前随机生成的一个字符串，同时将该字符串存入 cookie。
* 重构 login 方法，从 cookie 中获得随机字符串，生成验证码的 key 值，然后获取对应的 value 值即验证码。

**存储登录凭证**

在 RedisUntil 工具类

* 新增登录凭证前缀常量 ticket
* 新增 getTicketKey 方法，通过字符串获得登录凭证的对应 key 值（利用 string 存储）。

在业务层的 UserService 类

* 重构 login 方法，将登录凭证存入 [redis](/jump/super-jump/word?word=redis) 中。
* 重构 logout 方法，先从 [redis](/jump/super-jump/word?word=redis) 中获取登录凭证对象，将状态设为无效再重新存储进 [redis](/jump/super-jump/word?word=redis)。
* 重构 findLoginTicket 方法，根据 ticket 字符串获得对应登录凭证的 key，然后从 [redis](/jump/super-jump/word?word=redis) 查询登录凭证。

**缓存用户信息**

在 RedisUntil 工具类

* 新增用户前缀常量 user
* 新增 getUserKey 方法，通过用户 id 获得用户的对应 key 值（利用 string 存储）。

在业务层的 UserService 类

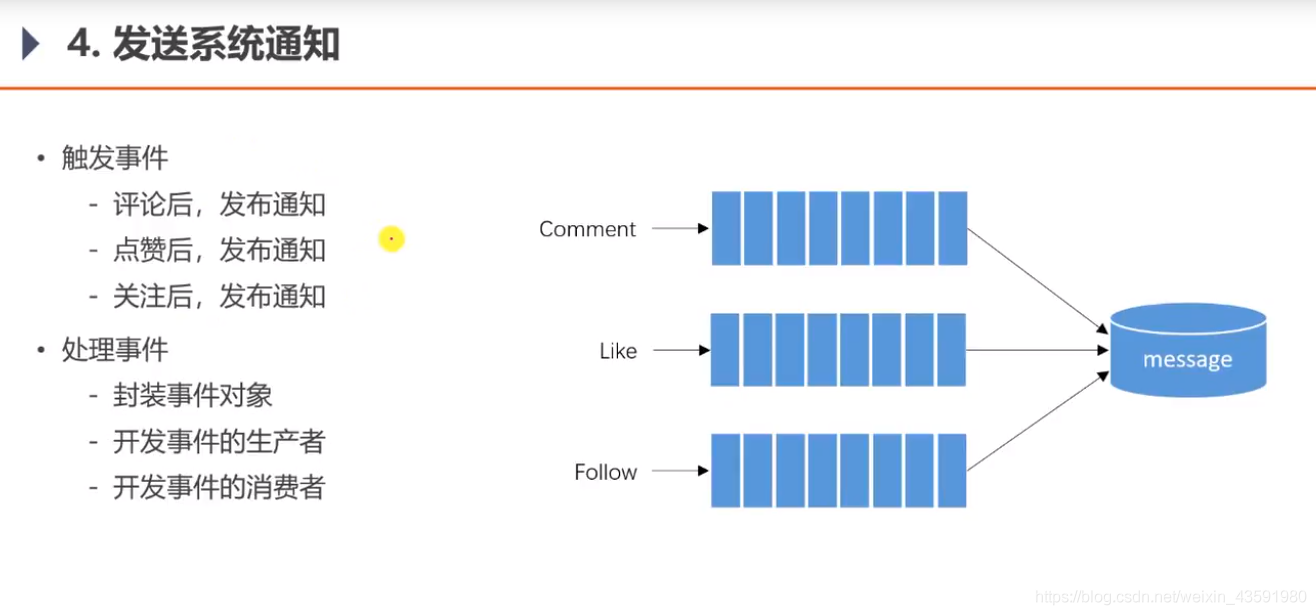
* 新增 getCache，从缓存获取用户信息。
* 新增 initCache，从 MySQL 查询用户信息并存入 [redis](/jump/super-jump/word?word=redis)。
* 新增 clearCache，用户信息变更（更新头像，激活）时清除缓存。
* 重构 findUserById 方法，首先调用 getCache从缓存获取用户信息，如果获取为 null 则调用 initCache。

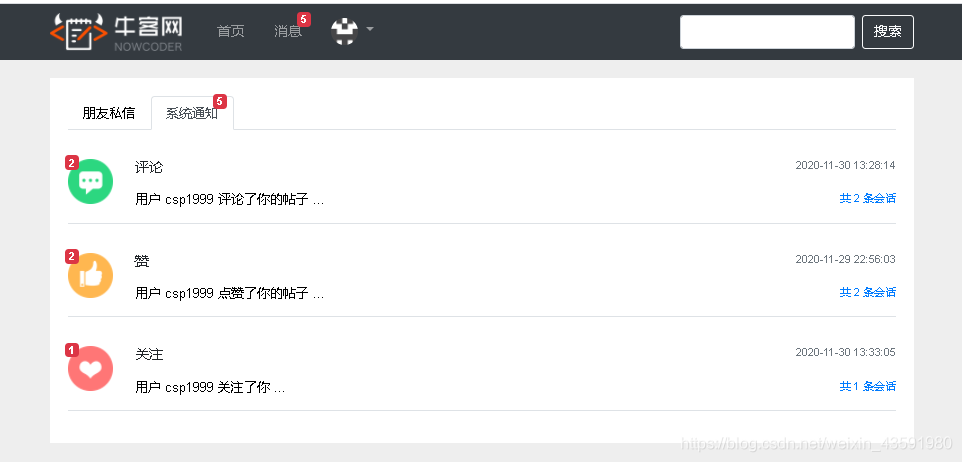
### Kafka

#### 发送系统通知

在 CommunityConstant 接口中新增三个常量，代表三个主题：评论、点赞、关注。

创建 Event 类，封装事件对象，包括主题、用户 id、实体类型、实体 id、实体用户 id 以及一个 map 集合存放其它信息。





#### 触发事件

创建 EventProducer 事件生产者，新增 fireEvent(Event event) 方法，通过 Event 获取事件类型，并将其封装成 JSON 数据，然后调用注入的 KafkaTemplate 实例的 send 方法发送。

在 CommentController、LikeControler、FollowController 中注入 EventProducer 实例，分别重构 addComment 方法、like 方法、follow 方法，封装 Event 对象，然后调用 EventProducer 的fireEvent 方法发布通知。

#### 消费事件

创建 EventConsumer 事件消费者，消费者是被动触发的。

* 注入 MessageService 实例。
* 增加 handleCommentMessage(ConsumerRecord record) 方法，通过 @KafkaListener 注解，topic 包括了评论、点赞和关注。从 recored 中获取信息，封装成 Message 对象然后调用 addMessage 方法插入数据库。

【问题】没有向数据库插入系统通知记录，原因是 ServiceLogAspect 类进行日志处理时要获取 ServletRequestAttributes 请求对象，Kafka 的消费事件是自动触发的，没有进行新的请求，产生了请求对象的空指针异常。

### 显示系统通知

#### 通知列表

在 MessageMapper 接口中

* 新增 selectLatestNotice(int userId, String topic) 方法，查询某主题最新的通知。
* 新增 selectNoticeCount(int userId, String topic) 方法，查询某主题通知的数量。
* 新增 selectNoticeUnreadCount(int userId, String topic) 方法，查询未读通知的数量。
* 在 message-mapper.xml 配置三个方法的 sql 语句，其中查询未读通知时使用 if 动态语句，如果没有传入 topic 就查询未读总量。

在业务层的 MessageService 中

* 新增 findLatestNotice 方法，调用 selectLatestNotice 方法查询最新通知。
* 新增 findNoticeCount 方法，调用 selectNoticeCount 方法查询某主题通知的数量。
* 新增 findNoticeUnreadCount 方法，调用 selectNoticeUnreadCount 方法查询未读通知的数量。

在表现层的 MessageController 中新增 getNoticeList 方法，获取通知列表

* 调用业务层 MessageService 的方法查询评论、点赞、关注的通知，将其封装在一个 HashMap 集合中然后添加到 Model 对象里。
* 调用业务层 MessageService 的方法查询私信和通知的总未读数量，添加到 Model 对象里。
* 返回 notice.html 页面。

#### 显示通知详情

在 MessageMapper 接口新增 selectNotices 方法，查询某个主题的通知列表，在 message-mapper.xml 配置 SQL。

在业务层的 MessageService 中新增 findNotices 方法，调用 selectNotices 方法。

在表现层的 MessageController 中新增 getNoticeDetail 方法

* 调用 findNotices 方法获取通知列表详情，封装到 List 集合并存入 Model 对象。
* 从通知集合中获取 id 集合，调用 readMessage 方法将消息设为已读。
* 返回 notice-detail.html 页面。

#### 显示未读通知总数

创建 MessageInterceptor 拦截器

* 注入 MessageService 实例和 HostHolder 实例。
* 重写 postHandle 方法，查询私信和通知的未读数量和，然后添加到 ModelAndView 对象。

在 WebConfig 中注入 MessageInterceptor 实例，并在 addInterceptors 方法中添加该拦截器。

**1）发表问题+敏感词过滤**

    发布问题时检查标题和内容，防止xss注入，并且过滤敏感词。防止xss注入直接使用HTMLutils的方法即可实现。过滤敏感词首先需要建立一个字典树，并且读取一份保存敏感词的文本文件，然后初始化字典树。最后将过滤器作为一个服务，让需要过滤敏感词的服务进行调用即可。

**2）发表评论和站内信**

     首先建立表comment和message分别代表评论和站内信。评论的逻辑是每一个问题下面都有评论，显示评论数量，具体内容，评论人等信息。消息的逻辑是，两个用户之间发送一条消息，有一个唯一的会话id，这个会话里可以有多条这两个用户的交互信息。通过一个用户id获取该用户的会话列表，再根据会话id再获取具体的会话内的多条消息。逻辑清楚之后，再加上一些附加功能，比如显示未读消息数量，根据时间顺序排列会话和消息。本节内容基本就是业务逻辑的开发，没有新增什么技术点，主要是前后端交互的逻辑比较复杂，前端的开发量也比较大。

**3）Redis实现点赞和点踩功能**

首先了解一下redis的基础知识，数据结构，jedis使用等。编写list，string，hashmap，set，sortset的测试用例，熟悉jedis api。开发点踩和点赞功能，在此之前根据业务封装好jedis的增删改查操作，放在util包中。根据需求确定key字段，格式是 like：实体类型：实体id 和 dislike：实体类型：实体id 这样可以将喜欢一条新闻的人存在一个集合，不喜欢的存在另一个集合。通过统计数量可以获得点赞和点踩数。一般点赞点踩操作是先修改redis的值并获取返回值，然后再异步修改mysql数据库的likecount数值。这样既可以保证点赞操作快速完成，也可保证数据一致性。

**4）异步消息机制**

在之前的功能中有一些不需要实时执行的操作或者任务，我们可以把它们改造成异步消息来进行发送。具体操作就是使用redis来实现异步消息队列。代码中使用事件event来包装一个事件，事件需要记录事件实体的各种信息：一个异步工具类（事件生产者+事件消费者+eventHandler接口），让以后各种事件的实现类来实现这个接口。事件生产者一般作为一个服务，由Controller中的业务逻辑调用并产生一个事件，将事件序列化存入redis队列中，事件消费者则通过一个线程循环获取队列里的事件，并且寻找对应的handler进行处理。整个异步事件的框架开发完成，后面新加入的登录，点赞等事件都可以这么实现。

**5）关注+粉丝列表的实现**

新增关注功能，使用redis实现每一个关注对象的粉丝列表以及每一个用户的关注对象列表。通过该列表的crud操作可以对应获取粉丝列表和关注列表，并且实现关注和取关功能。由于关注成功和添加粉丝成功时同一个事务里的两个操作，可以使用redis的事务multi来包装事务并进行提交。除此之外，关注成功或者被关注还可以通过事件机制来生成发送邮件的事件，由异步的队列处理器来完成事件响应，同样是根据redis来实现。对于粉丝列表，除了显示粉丝的基本信息之外，还要显示当前用户是否关注了这个粉丝，以便前端显示。对于关注列表来说，如果被关注对象是用户的话，除了显示用户的基本信息之外，还要显示当前用户是被这个用户关注，以便前端显示。

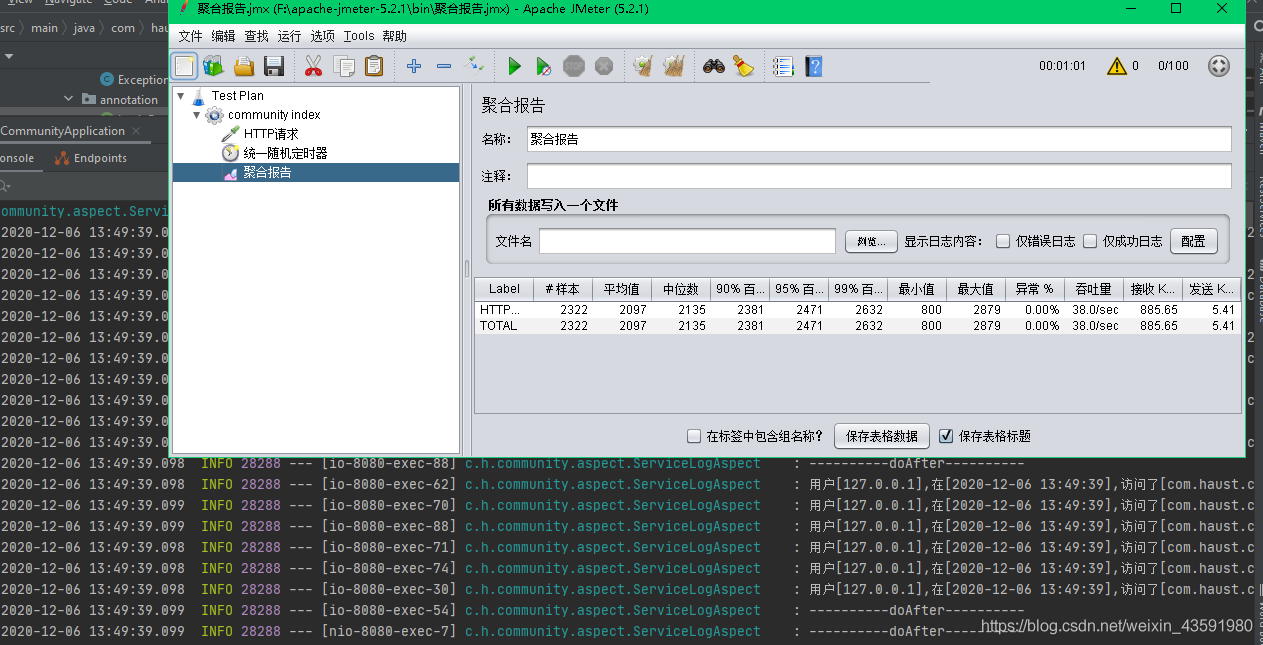
**6）使用爬虫爬取数据，对网站进行数据填充**

安装python3.x并且配置环境变量。同时安装pycharm,安装pip。安装好以后，先熟悉python的语法，写一些例子，比如数据类型，操作符，方法调用，以及面向对象的技术。因为数据是要导入数据库的，所以这里安装MySQLdb的一个库，并且写一下连接数据库的代码，写一下简单的crud进行测试。使用requests库作为解析http请求的工具，使用beautifulsoup作为解析html代码的工具，请求之后直接使用css选择器匹配。即可获得内容。当然现在我们有更方便的工具pyspider，可以方便解析请求并且可以设置代理，伪装身份等，直接传入url并且写好多级的解析函数，程序便会迭代执行，直到把所有页面的内容解析出来。这里我们直接启动pyspider的web应用并且写好python代码，就可以执行爬虫了。知乎：先找到问题，再把问题下所有的回答进行爬取，最后把问题和评论一起处理。

## 5.测试与部署

jemeter 压力测试

我们使用测试工具，测试缓存是否生效，访问首页的热门帖子：



我们模仿100个线程访问http://localhost:8080/community/index?orderMode=1 接口，可以看到控制台只有第一次访问的时候打印sql（从DB中查询数据），其他时候都是走本地缓存获取数据！这样就能提高热点页面访问速度！

20. 项目部署

20.1 服务器配置要求(我用的阿里云服务器)

2核4G(或者2个1核1G)

CentOS 7.X

20.2 需要部署的内容

MySQL

Redis

Kafka

ElasticSearch

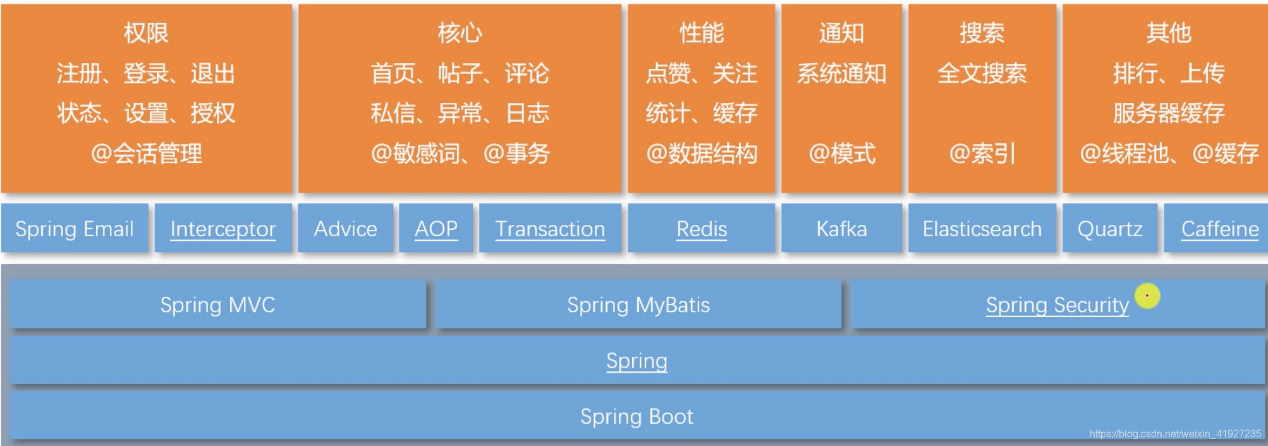
Wktmltopdf

Nginx

Tomcat

JDK8

项目压缩包



## 6.结论

## 致谢

## 参考文献

## 附录