**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， BBS社区论坛系统已经成为人们获取信息、发表言论的重要场所。

开放社区论坛系统，一个多元化的社区服务模式，业主和物业等可以在上面发布一些需求和通知，兴趣多样，话题讨论丰富。

本系统是基于B/S结构的一个小型论坛，主要基于Windows平台，通过这个论坛，用户间可以互相交流。

本系统使用SpringBoot整合SSM框架和MySQL数据库进行开发的，其主要目的就是为了丰富社区居民的生活方式，大家可以自行注册激活账号，并自由的发布话题，或者参与其他话题的讨论。更好的方便大家的生活。

该论坛为社区的居民网友提供了一个分享交流的平台，在这里注册的居民用户可以自地发表自己的意见，还可以快速处理其他社区居民提出的问题，查看对自己有用的帖子。此外该论坛还有一些特殊的功能，如强大的搜索功能等。

**[关键词]** 社区论坛、SpringBoot、SM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

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社区提供给用户的服务是非常全面且友好的。

对于一个社区来说，这个论坛可以从楼盘开始开发的时候，就着手来抓取用户。并且，由于业主社区先天性的“小圈子”特性，版面里活跃的用户，其精准性都是非常高的。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，这样的话你可以加入一些鱼塘，马上就能了解到你周边的人在出售一些什么，比方说1km以、2km以内的朋友在卖什么，非常直观，更贴近你的生活。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。

## 2.系统总体分析

### 2.1系统运行环境

准备工作就是一些工作环境的配置，IDE使用IntellijIDEA，具体的框架的版本与配置，以及mysql，redis，kafka，es的安装与配置。

### 2.2系统所需技术

熟悉快速开发框架：SpringBoot2.3.x 整合 SpringMVC + Mybatis

熟悉版本控制：Maven3.6.X + Git

数据库以及文件存储：MySQL + 文件存储阿里云OSS

熟悉页面模板引擎：Thymleaf3.x

第三方工具：网页长图生成工具Wkhtmltopdf + 验证码生成工具kaptcha

中间件：分布式缓存Redis + 全文检索ElasticSearch + Kafka + 本地缓存Caffeine

权限框架：Spring Securtiy + Spring Actuator

熟悉前端：Ajax + Vue + BootStrap + HTML + jQuery

* Spring Boot
* Spring、SpringMVC、MyBatis
* Redis、Kafka、Elasticsearch
* Spring Security、Spring Actuator

什么是Spring框架？

有很多模块组成，利用这些模块可以方便开发工作。这些模块是：核心容器（spring core)/数据访问和集成(Spring JDBC)/Web(Spring Web/MVC)/AOP(Spring Aop)/消息模块/测试模块(Spring Test)等。

Spring MVC是什么，是怎样的工作流程

服务器分为表现层/业务层/数据层，其中Spring MVC是工作在表现层，作用是接收/解析用户发送的请求，调用对应的业务类，根据业务类返回的结果（ModelAndView)，调用view进行视图渲染，并将渲染后的View返回给请求

什么是SSM框架？

包括Spring + Spring MVC(和Spring天生集成) + MyBatis（帮你你和数据库打交道的框架，简单的设置，你就可以像Java一样，操作数据库了）

每一次请求都是先开发数据访问层，在开发业务层，最后开发视图层（三层架构），但是每一次请求不一定要用到这三层

什么是Interceptor，在项目的哪里使用到了Interceptor?

Interceptor是SpringMVC的处理器（handler)拦截器，用于对处理器进行预处理和后处理。本项目中，每次请求都会检查request中的login\_ticket，把找到的user信息存放在协程中，并在完成处理后，自动释放。（方便的进行用户信息取用）

使用前缀树（字典树）存储敏感词，对text中的敏感词实现替换。

什么是Ajax，应用在项目哪些地方？

ajax指异步的json和xml技术，不是一门新的语言，而是使用现有技术的新方法。最大的特点是：不重新加载整个页面的基础上，可以与服务器交换数据，并更新部分网页数据。

项目中：帖子发布成功/失败的提示，使用到ajax

什么是事务，事务的四大特性。

定义：事务是逻辑上的一组操作，要么都执行，要么都不执行。

ACID：

A：原子性，事务是最小的执行单位，不允许被分割，事务的全部操作要么全部提交成功，要么全部失败回滚。

C：一致性，数据库在事务执行前后保持一致性状态，在一致性状态下，所有事务对同一个数据的读取结果相同。

I ：隔离性，一个事务所作的修改在最终提交前，对其他事务是不可见的。

D（Duability）：持久性，一旦事务提交，所做的修改将被永远保存到数据库中。即使系统发生崩溃，事务执行的结构也不能丢失。

在并发环境下，并发事务会出现哪些问题？

脏读：事务A读到了事务B修改但未提交的数据。

丢失修改：事务A和事务B同时读取了某一数据，并进行减１，此数据最终只会减一。

不可重复读：事务A多次读取某数据时，发现前后不一致（被事务B修改了）

幻读：和不可重复读类似，事务A读取到几行数据，事务B此时插入数据，随后的查询中，事务A发现了一些原本不存在的数据。（不是读取）

介绍SQL的四个隔离级别

READ-UNCOMMITED(读取未提交）:会出现脏读，不可重复读，幻读。

READ-COMMITED:会出现不可重复读，幻读

REPETABLE-READ:会出现幻读

SERIALIZABLE:都不会出现

怎么利用Spring实现事务管理

Spring管理事务忽略了底层数据库的结构，非常方便。有两种方式：注解（类型，传播方式）/编程式事务（override)。

是怎样实现统一捕获异常的？

在SpringBoot的项目某一路径下，加上对应的错误页面，发生错误时自动会跳转。服务器的三层结构中，错误会层层向上传递，所以只需要在表现层（controller)统一处理错误即可。

方法：在controller中加上advice包，并通过注解@ControllerAdvice和@ExceptionHandler，统一捕获异常。

是怎样实现统一记录日志的？

使用了AOP技术（面向切面编程），这里使用到的是SpringAOP。 AOP技术能够将哪些与业务，但是为业务模块共同调用的逻辑或责任（比如事务处理，日志记录，权限控制等），封装起来，便于减少系统的重复代码，降低模块间的耦合度，并有利于未来的扩展性和维护性。 SpringAOP本质上基于动态代理，当要代理的对象实现了某接口，会使用JDK动态代理，在运行时通过创建接口的代理实例，织入代码。当要代理的对象没有实现接口，则使用Cglib技术（编译时增强），通过子类代理织入代码。

什么是Redis，Redis有哪些优点？

概念：redis是一个非关系型数据库，数据存储在内存中，读写速度快。可以存储键和五种不同类型值的映射。只能以字符串为键，值支持：字符串，列表，无序集合，有序集合，hash散列表。

优点：由于数据存储在内存中，读写速度非常快，满足高性能，高并发的系统要求。与Java原生的map/guava相比，支持分布式缓存。与memcached相比，支持更丰富的数据类型，且支持数据持久化。

Redis分布式集群架构

典型的分片+复制

怎么往Spring框架中配置redis，介绍常见的redis操作

如何配置：

1，导入jar包

2，配置端口，以及配置类redisTemplate（注入连接工厂/设置序列化方式（json））

常见操作

Value类型：redisTemplate.opsForValue().set(redisKey, 1)，redisTemplate.opsForValue().get(redisKey)， redisTemplate.opsForValue().increment(redisKey)，

Hash类型：redisTemplate.opsForHash().put(redisKey, “id”, 1)， 还有get等操作

List类型：redisTemplate.opsForList().leftPush(redisKey, 101)， 还有size, index, range， leftPop等操作

Set类型：add, size, pop, members等操作

Zset类型：redisTemplate.opsForZSet().add(redisKey, “Linda”, 92), 有socre，rank，reverseRank， range等操作

操作key：可以delete，以及设置过期时间

同时支持绑定操作，支持事务（编程式事务，在事务中一般不包含查询）

为什么不包含查询：redis事务就是一系列命令的批量操作，批量操作在发送 EXEC 命令前被放入队列缓存，并不会被实际执行，也就不存在事务内的查询要看到事务里的更新，事务外查询不能看到。

怎样存储的点赞/关注/缓存用户数据

点赞使用set类型存储，key为点赞对象，set中保存点赞人的ID

关注使用zSet类型存储，key为被关注者，set保存关注者以及关注时间为score

缓存用户数据使用Value类型，key为用userID得到的key，value为user对象（设置过期时间，且数据修改时需要清除缓存）

什么是消息队列

消息队列是一个存放消息的容器，生产者把消息放在队列中，消费者从消息队列中取出数据。消息队列的主要功能（优点）在于：

1，解耦：生产者只负责把消息放在队列中，而不用关心谁去使用它。

2，异步：生产者把消息放在队列中后即可返回，而不用一个个的通知消费者去执行，消费者是异步的获取消息的。

3，限流：生产者一次性产生大量的数据时，不会给消费者造成压力，消费者可以根据自身的能力，去消息队列中取数据。

消息队列作为信息传递的中间件，需要注意哪些问题？

1，高可用：因为消息队列如果宕机，会导致整个系统不可用。（分布式/集群的现成支持）

2，数据持久化：防止数据丢失

3，如何取数据：消息队列主动通知或者消费者轮询。

Java中的blockingqueue，可以提供线程间的消息队列

BQ也是生产者与消费者模式，属于点对点式消息队列？（一个消息只会被消费一次）Blocking Queue构建了一个桥梁，能够解决生产速度/消费速度不匹配问题。阻塞的时候只是在那里等着，但是不会占用CPU资源，对性能不会有影响。

什么是Kafka，有哪些功能和应用场景？

Kafka为分布式流处理平台。流处理是指对不断产生的动态数据流实时处理，基于分布式内存，具有数据处理快速，高效，低延迟的特性。

Kafka主要提供的功能包括：消息系统，日志收集，用户行为跟踪，流式数据处理。

Kafka的消息模型，以及常见术语

消息模型：发布-订阅模型，消费者订阅了某一主题（topic）后，生产者采用类似广播的方式，将消息通过主题传递给所有的订阅者。

Topic：主题，类似于文件夹，用来存放不同的数据。

Partition：主题分区，同一主题的不同分区可以存放在不同的Broker上面，保证并发能力和负载均衡。

Offset：消息在Partition中的存放位置。

Broker：可以理解为kafka集群里面的一台或多台服务器，它本身是没有复制的，上面可能运行着topic1的leader， topic2的follower等等。

在项目哪里用到了Kafka

当有点赞，评论，关注请求时，会发送系统通知点赞，评论，关注的对象。在处理系统信息时，使用到了Kafka，具体来说，先定义了生产者类和消费者类，其中生产者被点赞/评论/关注功能对应的Controller使用，产生消息。而消费者负责消息（message）到来时，把消息存到数据库内。

什么是ElasticSearch，存储原理，功能，特点

概念：ES是一个基于lucene构建的，分布式的，RESTful的开源全文搜索引擎。

存储原理：数据按照Index – Type – Document – 字段四级存储，其中Index对应数据库，Type对应表，Document为搜索的原子单位，包含一个或多个容器，基于JSON表示。字段是指JSON中的每一项组成，类似于数据库中的行/列。Mapping是文档分析过滤后的结果，根据用户自定义，将某些文字过滤掉，类似于表结构定义DDL？？。同时ES也和分布式数据库一样，支持shard的replication

功能：

1，分布式的搜索引擎和数据分析引擎

2，全文检索，结构化检索，数据分析。

3，对海量数据进行近实时的处理

特点：

1，可以作为分布式集群处理PB级别的数据，也可单机使用。

2，不是特有技术，而是将分布式+全文搜索（lucene) + 数据分析合并在一起。

3，操作简单，作为传统数据库的补充，提供了数据库所不具备的很多功能。

项目中哪里使用到了ES,如何使用

在进行帖子搜索时，使用到了ES。可用Repository和Template两种方式，由于Repository搜索到的结果（直接返回的post类，方便）没有高亮标签（why），所以使用了template方式重写了mapResults函数，获得了带有高亮标签的post。

使用消息队列（kafka）的方式，实现发帖/删帖后ES数据库的自动更新。

搜索：定义SearchQuery，确定搜素内容，排序方式，高亮等。接着使用elasticTemplate.queryForPage方法，需要重写mapResults函数，得到高亮数据。

项目中使用到了SpringSecurity在哪些地方？

重构了用户权限控制（之前用的拦截器）

怎样统计网站UA和DAU

使用Redis的高级数据结构：

HyperLogLog：超级日志，统计独立整数个数。统计UA（独立访问）时，以日期为ｒｅｄｉｓｋｅｙ，将客户端ｉｐ　ａｄｄ到HyperLogLog中（redisTemplate.opsForHyperLogLog().add(redisKey, i);）

Bitmap：位图，比如３６５天的签到，只需要３６５／８个字节的大小。统计DAU（日活跃用户）时，以日期为ｒedis　ｋｅｙ，以用户ID作为位（在数据中的位置），用ｏｒ操作，既可以方便的统计一段时间内的注册用户访问人数。

什么是Quartz，特点，专业术语，项目应用

概念：quartz是一个开源项目，完全基于java实现。是一个优秀的开源调度框架。

特点：

1，强大的调度功能，例如支持丰富多样的调度方法

2，灵活的应用方式，例如支持任务和调度的多种组合方式

3，分布式和集群能力

专业术语：

scheduler：任务调度器 ， scheduler是一个计划调度器容器，容器里面有众多的JobDetail和trigger，当容器启动后，里面的每个JobDetail都会根据trigger按部就班自动去执行

trigger：触发器，用于定义任务调度时间规则

job：任务，即被调度的任务， 主要有两种类型的 job：无状态的（stateless）和有状态的（stateful）。一个 job 可以被多个 trigger 关联，但是一个 trigger 只能关联一个 job

misfire：本来应该被执行但实际没有被执行的任务调度

项目应用：定时的统计帖子分数（如何设置定时任务和trigger）

什么是Caffeine，如何缓存，项目应用

概念：Caffeine 是一个基于Java 8的高性能本地缓存框架

初始化cache：缓存保存的对象，使用Caffeine.newBuilder()创建，创建时设置缓存大小，过期时间，缓存未命中时的加载方式。

为什么只缓存热度帖子？不会经常变。

### 2.3所需开发环境

* 构建工具：Apache Maven
* 集成开发环境：IDEA
* 数据库：MySQL、Redis
* 应用服务器：Apache、Tomcat
* 版本控制工具：Git

### 2.4 开发环境的搭建

Apache Maven

* 可以帮助我们构建项目、管理项目中的 jar 包
* Maven 仓库：存放构件位置
  + 本地仓库：默认是 ~/.m2/repostity
  + 远程仓库：中央仓库【官网】、镜像仓库【第三方，优选】、私服仓库【公司搭建的】
* 示例：安装、配置、常用命令
* 官网：<http://maven.apache.org>
* jar 包查询网站：<https://mvnrepository.com/>

Spring Initializr

* 创建 Spring Boot 项目的引导工具
* 示例：创建 “牛客社区” 项目
* 官网：<https://start.spring.io>
* 创建完项目，解压用IDEA打开，记得把 pom.xml 文件加载到 Maven 里面

### 2.5 用户功能分析

1. 用户可以注册登录
2. 注册后可以发布或回复帖子

管理员功能分析：

1. 管理所有用户的信息，包括增删改查。
2. 管理所有用户。

注册管理：能够对用户名和密码进行验证，校验用户输入的邮箱的有效性，防止恶意批量刷注册，。

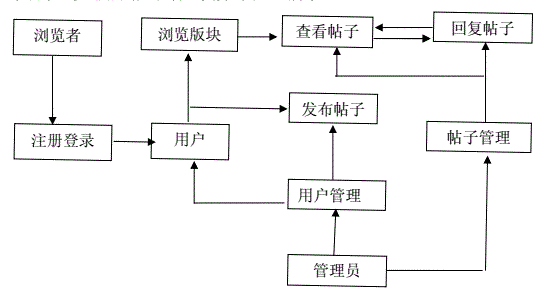
浏览帖子；可以根据贴主或者帖子的内容进行搜索，管理员可以设置置顶的帖子，浏览帖子时用户可以不用登录，但回复和发帖时用户必须登录。

发表和回复帖子：只有登录且用户状态正常才能进行发表和回复，且帖子的字数有所限制。

用户个人信息管理：每个用户注册之后都会随机一个头像，用户后续可以自行修改。可以查看自己发布的帖子列表，并对其进行编辑和删除。

用户管理：只有管理员用户才能有此项权限，如果有用户恶意发帖，管理员可以设置用户的状态码，限制用户的行为。

## 3. 系统总体设计



### 3.1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，创建索引 |
| password | varchar | 用户密码 |
| salt | varchar | 加密盐值 |
| email | varchar | 用户邮箱，创建索引 |
| type | int | 用户类型：0 普通、1 管理员、2 版主 |
| status | int | 用户状态：0 未激活、1 已激活 |
| 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，创建索引 |
| entity\_id | int | 评论实体 id，创建索引 |
| entity\_type | int | 评论实体类型：1 帖子评论、2 评论回复 |
| target\_id | int | 评论目标 id |
| content | text | 评论内容 |
| status | int | 评论状态：0 有效、1 无效 |
| 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，创建索引 |
| title | varchar | 帖子表标题 |
| content | text | 帖子内容 |
| type | int | 帖子类型：0 普通、1 置顶 |
| comment\_count | int | 评论数量 |
| status | int | 帖子状态：0 普通、1 精华、2 拉黑 |
| 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：0 有效、1 无效 |
| 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，创建索引 |
| to\_id | int | 收消息的 id，创建索引 |
| conversation\_id | varchar | 会话 id，由通信双方 id 拼接，创建索引 |
| content | text | 消息内容 |
| status | int | 消息状态：0 未读、1 已读、2 删除 |
| create\_time | timestamp | 消息发送时间 |

### 3.2注册流程

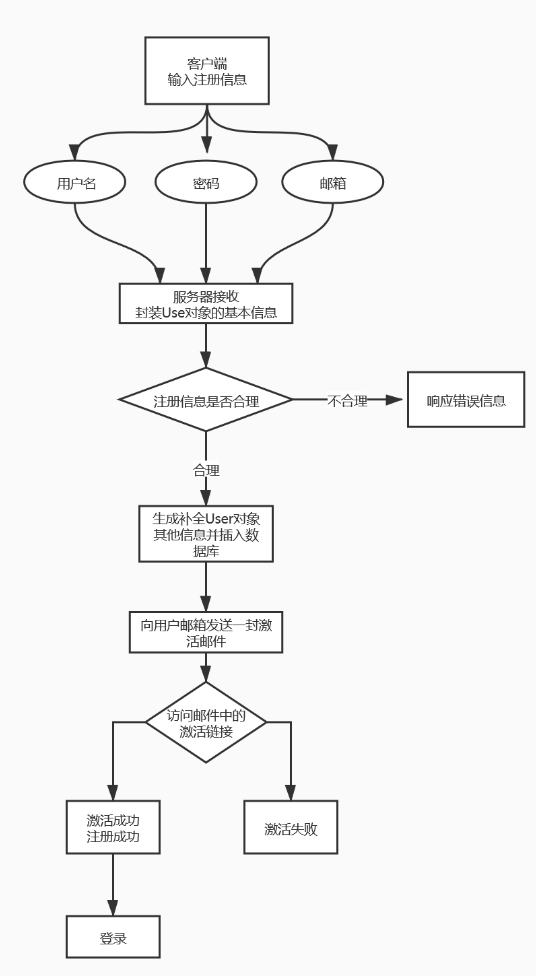
根据请求来拆解功能

1，打开注册网页

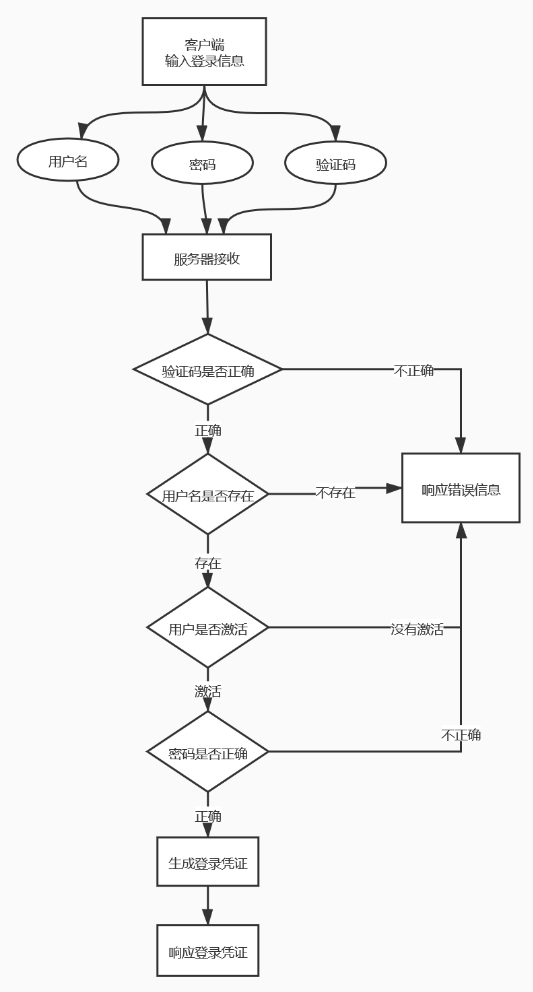
2，把注册的信息发送给服务器（点注册）

3，把激活邮件发送给邮箱

4，利用激活链接打开网页



### 3.3登录流程



## 4.系统总体功能实现

### 4.1开发社区首页（discuss\_post 表）

功能拆分：开发社区首页，显示前 10 个帖子。开发分页组件，分页显示所有帖子。

用到的表是 discuss\_post 数据库表，包括帖子 id、发帖人 id、标题、内容、类型、状态、发帖时间、评论数量（为了提高效率，避免关联查询，因此冗余存储）、分数（用于进行热度排名）。

#### 开发数据层

帖子相关操作：

* 创建对应 discuss\_post 表的 DisscussPost 实体类。
* 创建 DisscussPostMapper 接口，使用 @Mapper 注解。
  + 分页查询中用户 id 是可选参数，通过动态 SQL 选择，如果为 0 就不使用，在开发用户个人主页查询用户发帖记录时需要使用。
  + 如果只有一个参数，并且在动态 SQL 的 <if> 里使用，必须使用 @Param 加别名。
* 创建 disscusspost-mapper.xml。
  + where status != 2 拉黑的帖子不展现。
  + <if test="userId!=0"> userID 为 0 时不使用，按照类型，发帖时间[排序](/jump/super-jump/word?word=%E6%8E%92%E5%BA%8F)。

#### 开发业务层

创建 DiscussPostService 类，可以分页查询帖子和帖子数量。

创建 UserService 类，实现根据 id 查询用户功能，因为显示帖子时不显示用户 id，而是显示用户名。

#### 开发视图层

把静态资源 css、html、img、js 放到 static 目录下。

把模板 mail、site、index.html 放到 template 目录下。

创建 HomeController，getIndexPage 方法，用 map 集合把帖子和用户封装到一起。

修改 index.html，使用 <th:text="${map.xxx.xxx}" 动态替换。

【问题】使用帖子关联查询用户时，给查询用户的 findUserById 方法传入了帖子的 getId 方法，应该是 getUserId 方法。

#### 开发分页组件

创建 Page 实体类，封装分页信息，包括当前页码、显示限制、帖子总数、查询路径等。显示的起始页不能小于 1，最大页不能超过 total。

在 index.html 中，当 page.rows > 0 时显示分页信息。

如果 page.current 等于 1 或 page.total，代表是首页或末页，此时不能点击上一页和下一页，用 disabled 属性实现。

### 4.2开发注册登录模块

#### 发送邮件

在[新浪](/jump/super-jump/word?word=%E6%96%B0%E6%B5%AA)邮箱打开 SMTP 服务。

引入 spring-boot-starter-mail 依赖。

在配置文件配置主机、端口、发送邮箱、授权码等。

创建 MailClient 类，调用 JavaMailSender 发送邮件。

使用 thymeleaf 发送 HTML 邮件，调用 TemplateEngine 把信息封装到 HTML 模板。

【问题】发送邮件成功但没接收到，在垃圾箱中可找到。

#### 注册功能

怎么实现注册功能的？

根据请求来拆解功能

1，打开注册网页

2，把注册的信息发送给服务器（点注册）

3，把激活邮件发送给邮箱

4，利用激活链接打开网页

把 register.html 地址关联到首页的注册 href 属性。

设置域名、创建 CommunityUtil 工具类，在工具类创建生产随机字符串和 MD5 加密方法。

创建 LoginController，创建 getRegisterPage 方法，跳转注册页面。

在 UserService 中创建 register 方法，判断注册信息合规后插入数据库，发送激活邮件。

在 LoginController 创建 register 方法，调用 UserService 的 register 方法。

创建接口 CommunityConstant，定义激活码的三种状态，成功、重复、失败，让 UserService 和 LoginController 实现该接口。

点击激活邮件的 url 【本地服务器的url】后，服务器通过 LoginController 的 activation 方法查询数据库用户，如果 url 中的激活码和设置的一样，就把用户 status 改为 1。

#### 生成验证码

在 pom.xml 导入 kaptcha 的 jar 包。

创建配置类 KaptchaConfig，设置验证码的大小、范围、长度等。

在 LoginController 类新增 getKaptcha 方法生成验证码图片。

在 login.html 中，将刷新验证码的链接绑定 refresh\_kaptcha 方法，通过 id 选择器获取 img 组件，重新访问 getKaptcha 方法生成验证码图片。

【问题】由于访问同一个生成验证码路径，需要在 url 参数加上一个随机数字，保证会重新请求获取新图片。

#### 登录退出功能（login\_ticket 表）

登录成功时，需要生成一个登录凭证发送给[客户端](/jump/super-jump/word?word=%E5%AE%A2%E6%88%B7%E7%AB%AF)。凭证可以在多个业务中连续地验证用户的登陆状态，凭证信息存储在 login\_ticket 数据库表中，status 的 0 和 1 表示有效和无序，expire 表示过期时间。

创建对应 login\_ticket 表的 LoginTicket 实体类，对应 login\_ticket 数据库表。

创建 LoginTicketMapper 接口，通过 @Insert、@Select、@Update 注解来插入、查询、更新凭证。

在 UserServce

* 创建 login 方法，验证账户合规后将凭证信息插入数据库，添加登录凭证到 map 中。
* 创建 logout 方法，将对应凭证设为无效。

在 LoginController

* 创建 login 方法，判断验证码正确后调用 UserServce 的 login 方法，如果 map 包含 ticket 代表登录成功，重定向跳转首页，否则添加错误信息并跳回登录页。
* 创建 logout 方法，判断验证码正确后调用 UserServce 的logout 方法，跳转至登录页。

在 login.html 绑定登录链接，index.html 绑定退出登录链接。

【问题】登录成功后，创建了凭证，但忘记将凭证信息插入数据库。

#### 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。

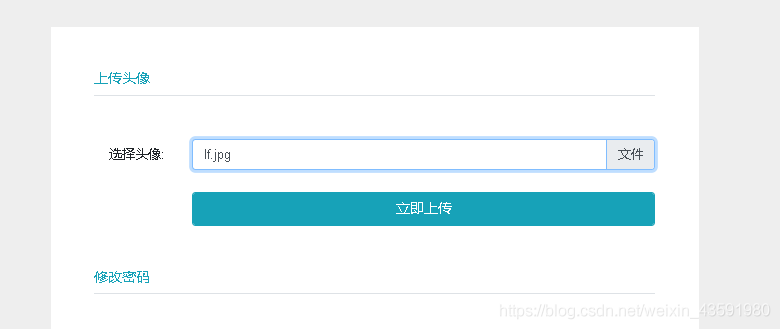
创建 UserController

* 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

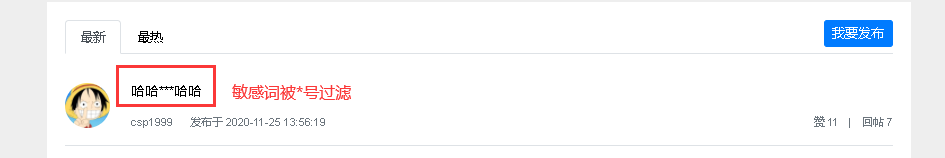
#### 敏感词过滤

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

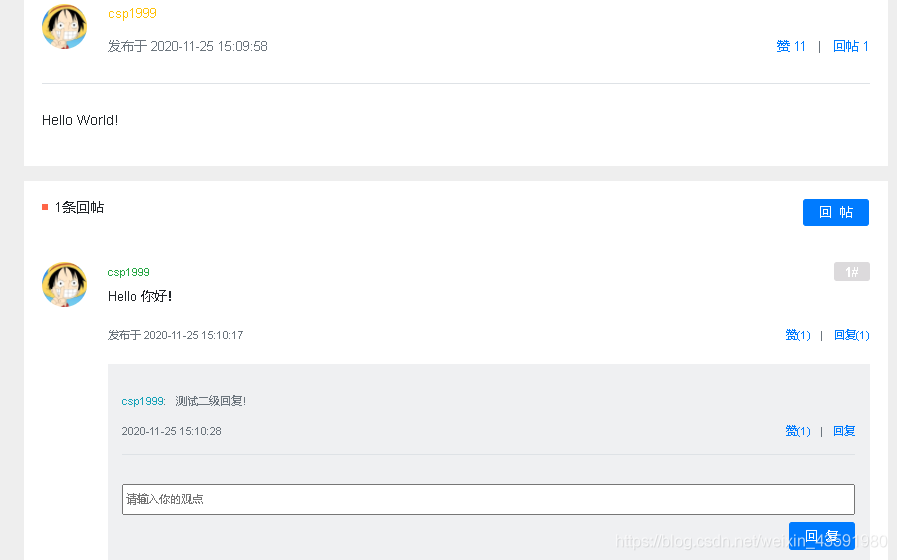
* 新增 selectCommentsByEntity 方法，根据实体查询一页的评论数据。
* 新增 selectCountByEntity 方法，根据实体查询评论的数量。
* 在 comment-mapper.xml 配置 select 语句。

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）

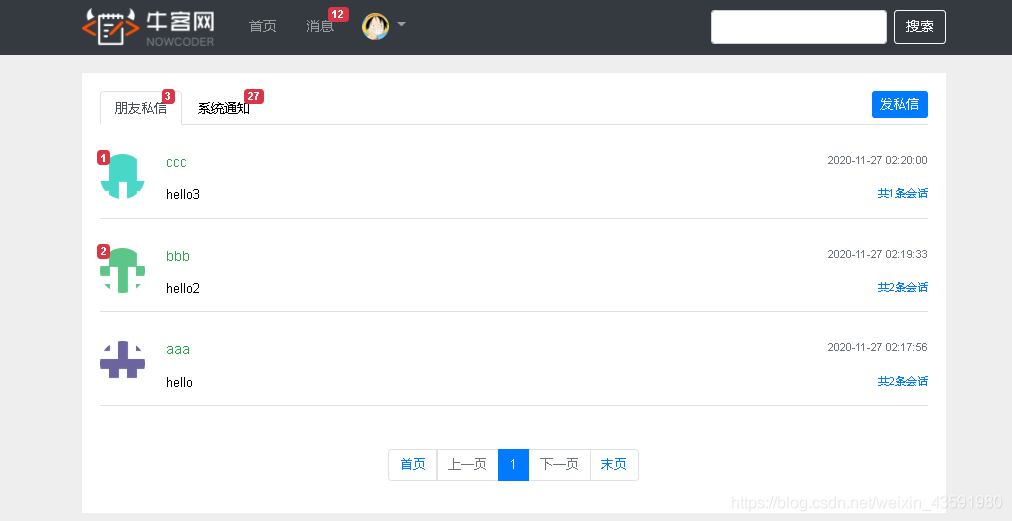
创建对应 message 表的实体类 Message。

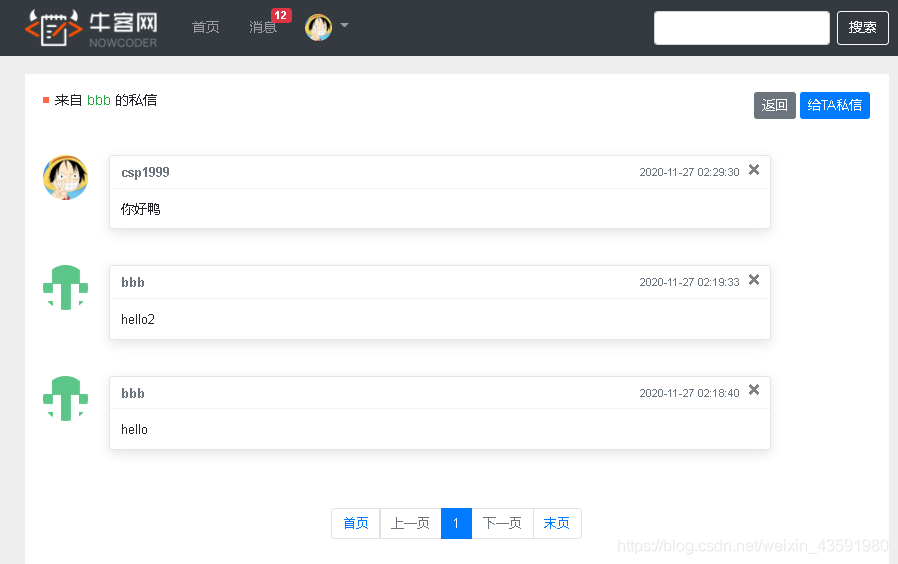
创建 MessageMapper 接口，增加查询会话列表、会话数量、私信列表、私信数量、未读私信数量等方法，在 message-mapper.xml 中配置对应的 sql。

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

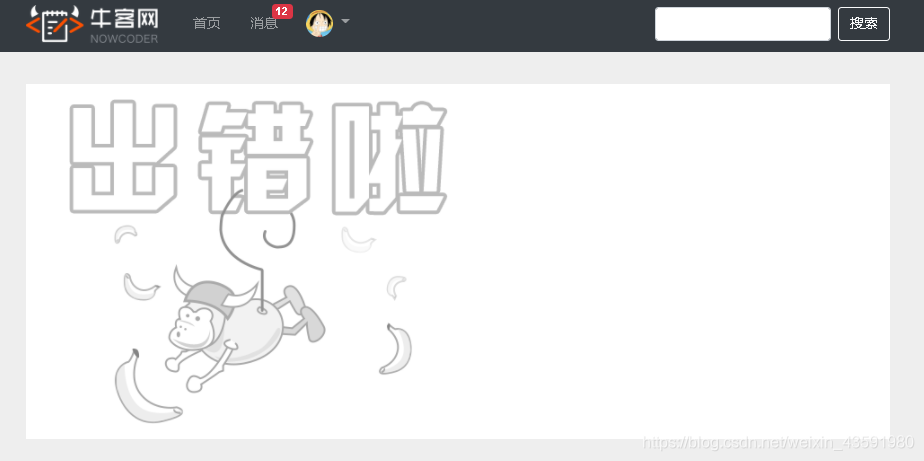
用于修饰方法，该方法回在Controller 方法执行前被调用，用于绑定参数的转换器。

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。



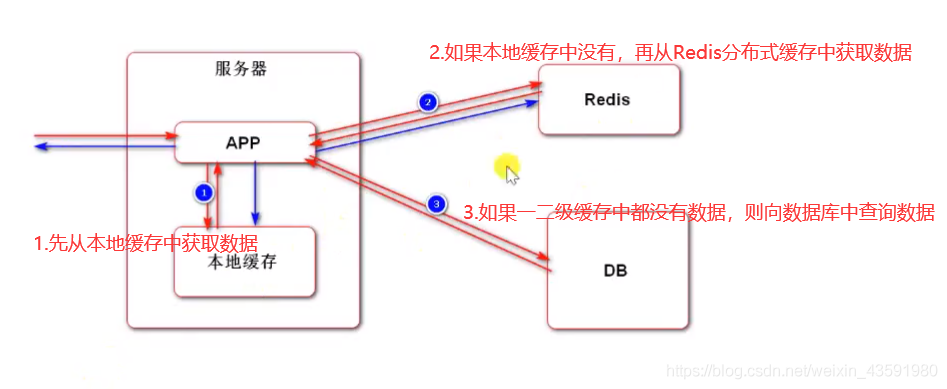


#### 统一日志处理

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

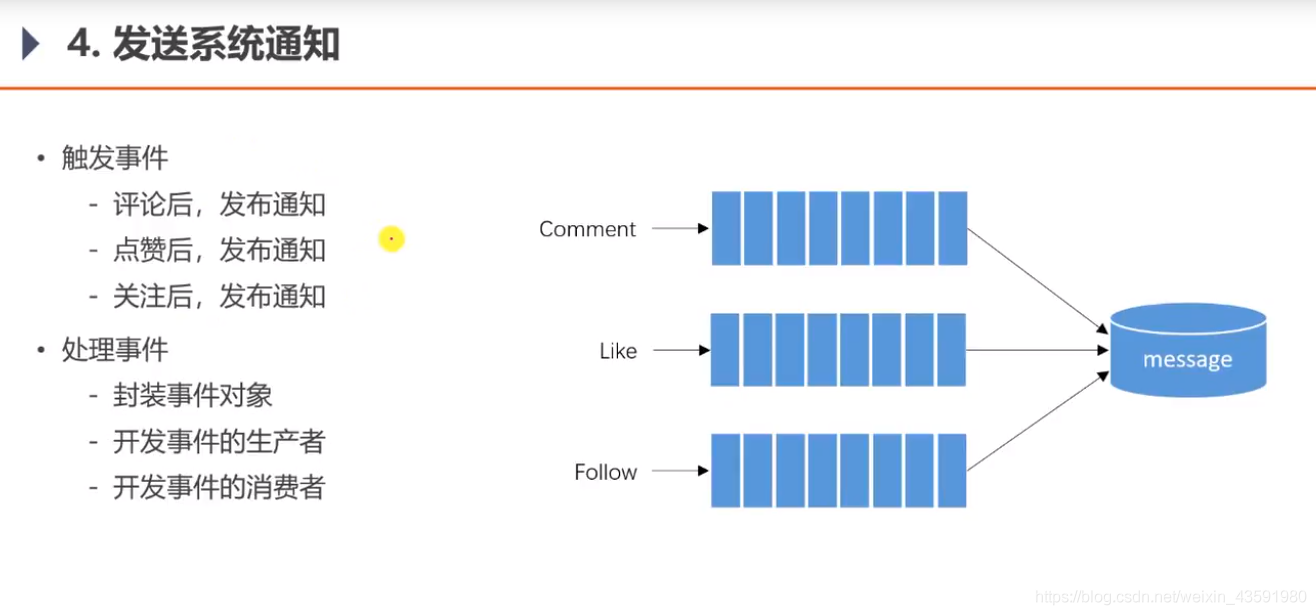
* 新增 getCache，从缓存获取用户信息。
* 新增 initCache，从 MySQL 查询用户信息并存入 [redis](/jump/super-jump/word?word=redis)。
* 新增 clearCache，用户信息变更（更新头像，激活）时清除缓存。
* 重构 findUserById 方法，首先调用 getCache从缓存获取用户信息，如果获取为 null 则调用 initCache。

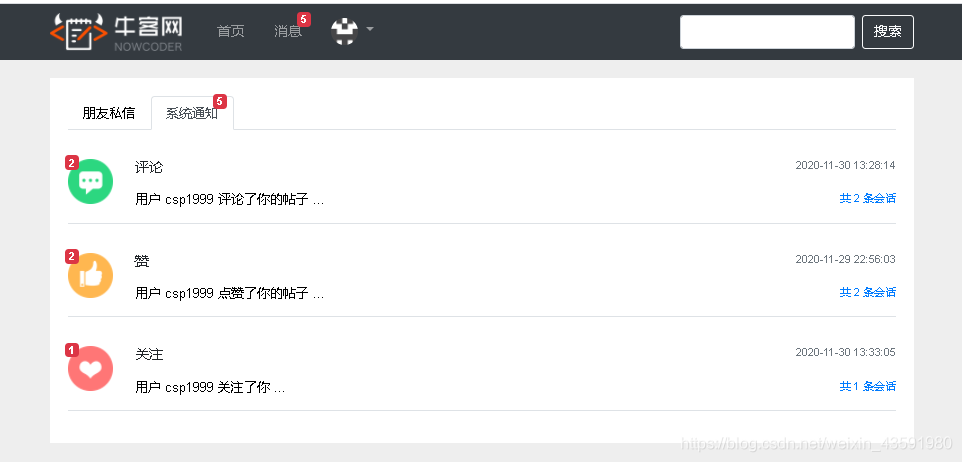
### Kafka

#### 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。

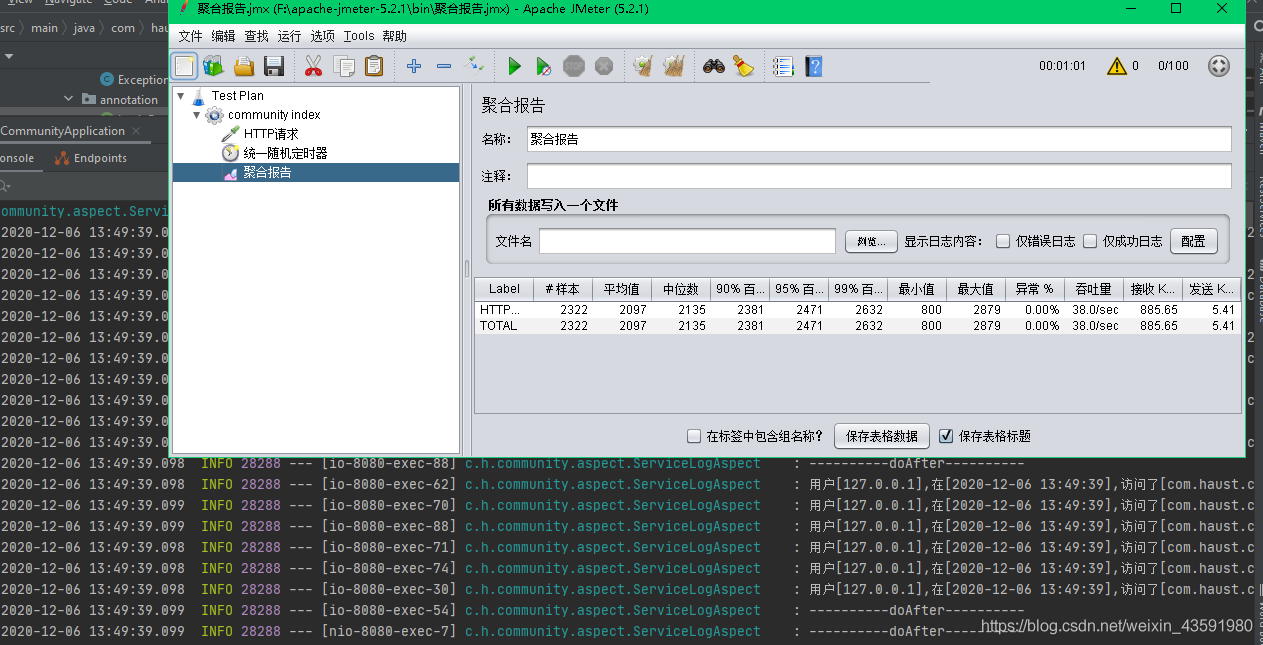
**6）使用爬虫爬取数据，对网站进行数据填充**

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

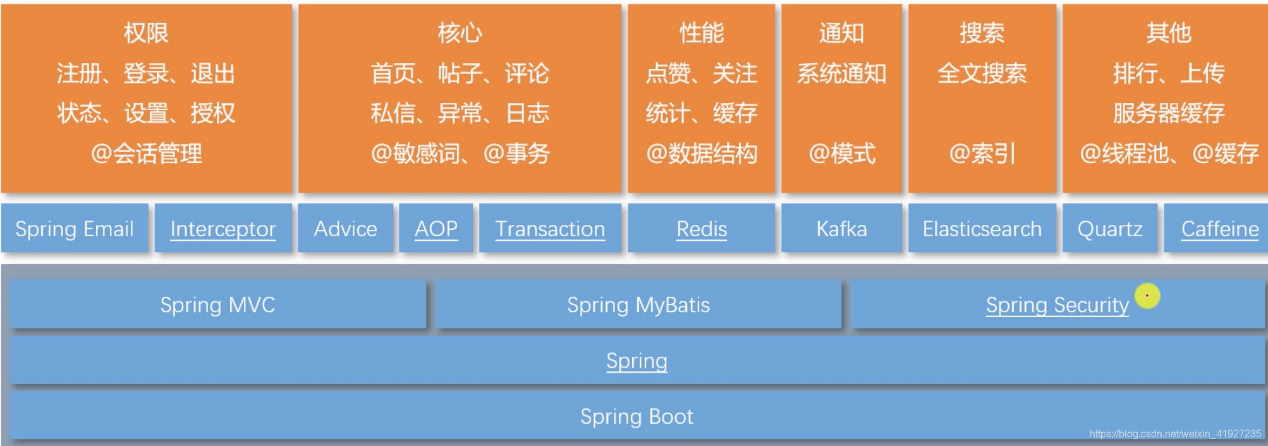
Wktmltopdf

Nginx

Tomcat

JDK8

项目压缩包



## 6.结论

## 致谢

## 参考文献

## 附录