摘要

如今社会高速发展，随着科技水平的提高，网络越来越发达，随之出现了越来越多的网络社交平台网站。而社交网站众多的情况下，用户们更倾向于方便快捷的社交网站。作为其代表的微博客充分满足了现代快节奏高压力社会下人们急剧上升的个人表达与倾诉沟通的需求，并因其拥有良好的界面交互，所以微博客系统更受现代网民的喜爱和青睐，我由此而开发出本系统给广大用户使用。

本系统采用 SpringBoot 和 MyBatis 作为系统基础架构，同时采用使用 MySQL 数据库进行数据的存储，在前端的 html 页面上使用了大量的 JavaScript 插件和 JQuery 框架等让系统界面更加美观友好，并是由 Thymeleaf 模板引擎进行页面的数据渲染，让页面与用户的交互性更强，同时也让用户拥有更好的使用体验。在系统通知功能中引入 WebSocket 协议，让用户之间的通信更加顺畅便捷，保证了信息的准确传输。而引入 Redis 数据库，利用其数据结构的特点，能够很好地将数据保存在 Redis 数据中并实行自动排序，能过很容易实现类似排行榜这样的功能。在最为关心的系统安全方面，使用 Shiro 安全框架，将大大增加系统的安全性、健壮性。总的来说，本微博客系统基本实现了博客系统的功能，系统安全性和界面美观性也同时具备，相信用户在使用到本博客系统时，会感受到一个功能丰富，界面友好的系统。

关键词： 社交平台 微博客系统 Springboot MyBatis MySQL WebSocket

Abstract

Nowadays, with the rapid development of society, with the improvement of technology level, the network is more and more developed, and more and more network social platform websites appear. In the case of a large number of social networking sites, users tend to be more convenient and efficient social networking sites. As a representative of the micro blog, it fully meets the needs of the rapid rise of personal expression and communication under the modern fast-paced and high pressure society, and because of its good interface interaction, the micro blog system is more popular and favored by modern Internet users, so I developed this system for the majority of users.

The system uses SpringBoot and MyBatis as the system infrastructure, uses MySQL database for data storage, uses a large number of JavaScript plug-ins and jQuery on the front-end HTML page to make the system interface more beautiful and friendly, and uses the Thymeleaf template engine to render the page data, making the page more interactive with the user, and at the same time, making the user have better user's experience. WebSocket protocol is introduced into the system notification function, which makes the communication between users more smooth and convenient, and ensures the accurate transmission of information. The introduction of Redis database, using the characteristics of its data structure, can well save the data in Redis data and implement automatic sorting, and can easily achieve such functions as leaderboards. In the most concerned system security, using Shiro security framework will greatly increase the security and robustness of the system. In general, this micro blog system basically realizes the functions of the blog system, and the system security and interface aesthetics are also available. I believe that when users use this blog system, they will feel a system with rich functions and friendly interface.

Key words： social platform　Microblog system　SpringBoot　MyBatis　MySQL　WebSocket

目 录

第一章 绪论

1.1 研究背景与现状

以前在网络不发达的时候，我们的社交方式为BB机、手机、书信等方式，但是这种方式都有一个缺点：慢。而随着现代互联网技术的高速发展，尤其是在2004年Facebook社交软件的出现，开启了一个网络社交的新时代。而现在，新浪博客、脸书、推特等成为目前最主流的网络社交平台。目前。脸书（Facebook）是当前世界上使用人数最多的一款网络社交网站，截止到2018年，它每个月的活跃用户有22亿。而2006年建立的推特发展至2014年，也已经拥有大约3亿的月活跃用户。至于国内的网络社交发展虽然比国外的晚一些，但由于我国社会经济的高速发展，我们网络社交的发展也更为迅速。尤其是最近几年来，国内网络社交迅速普及，使用用户的数量迅猛增长。据《中国互联网发展报告 2018》，到 2017 年 9 月为止，微信日均登录人数已达到了 9. 02 亿，是中国当前使用人数最多的社交网络平台。从用户构成来说，主要是以年轻人群体，尤其是“90后”为主。新浪财报显示，微博 2017 年度营收超 77 亿元，月活跃用户平均达 3.92 亿。而腾讯公布的 2018 年第一季度财报则宣布微信月活跃用户已突破 10 亿。跟国外相比，国内网络社交的发展尽管起步略晚，但发展更迅速，用户数量和范围更大[23]。

但是却也有不少人在感受到网络社交带来方便的同时，在生活中却感到越来越孤独。人们不但感受不到与别人交流的充实和喜悦，反而觉得自己的内心深处从满了孤独和失落。其背后的原因并不是因为当今社会的社交发生变化，而是网络社交作为一种新的社交方式确实与以往有所不同。社交网络平台的便利性是人们喜欢使用它作为交流工具的重要因素，即时化的沟通提高了人们的互动频率，同时也会让人们习惯与即时性的答复。网络社交最大的缺点是浅层化、碎片化和低效率。所谓的浅层化是指在网络上社交会很少有深入的思考和沟通。特别是在群聊里，很少有人会花费足够的文字来表达自己深邃的思考，相对来说人们更多的是发送表情、语音等来表达自己，使社交变得极其空洞，没有点感情在里面。碎片化是指在网络社交中传递的信息往往不是完整的，没有一套完整体系和流程的，通常来说百字以上已属长文，能够传递和表达的信息非常有限，而在经过多人解读转发后，信息的原意更加支离破碎。在碎片化互动过程中，参与双方对真实意思的理解可能存在比较大的差异，或许有人能感觉到，本来当面十分钟能够说清楚的事情，在使用网络社交工具文字或者语音沟通时反而需要花费更长的时间，甚至不得不约个时间当面锣对面鼓地说清楚。当然，大多数使用网络社交的

人对低效率是习以为常的，因为很多时候并不是真的有社交的需求，而是填补闲暇时间空白的一种方式，也因此形成了大量为社交而社交的行为[24]。

1.1 研究意义

网络社交平台是时代进步和计算机网络高速发展而催生的产物，它能够很好地体现出现代人们的社交方式与过去社交活动的巨大区别。而博客系统作为社交平台最常见的一种系统，它能被用户广泛使用，自有其独特的原因。社交博客系统能够让用户自由地发言，并能浏览其他用户的博客，如果对其博客感兴趣或对其产生不同的观点时，能够可以对其进行评论，发表自己的观点、见解。同时对某条博客认同、喜爱时，可以点赞，甚至转发这条博客给更多的人了解。当然用户之间也可以通过私信来私下交流，也可以关注对方。对于被多人点赞、转发的博客，我们将它放在热门榜单上，从这里我们就可以知道当前最热门的话题、博客是什么了，能够第一时间了解当前的热点。这就是为什么用户都喜欢使用社交博客的原因。当代网络社交博客系统能够让人民在工作下班后，或者娱乐休息时，能够刷下博客，消遣下时间，同时又能了解网上发生的大大小小事情，使其能够在最短的时间内得到更多的信息。通过对当代流行的社交平台进行研究分析，综合其优缺点，设计一款更加合适、贴切人们日常生活的社交博客系统，让人们能够更加合理地使用现在的网络社交平台，能够更好地体现目前网络带来的便利。

1.2 论文章节安排

第一章为绪论，介绍了微博客系统的研究背景、现状以及研究意义，简述了本文将要设计实现的内容。

第二章介绍了需求分析，包括系统的功能和安全的需求分析以及系统的可行性分析。

第三章介绍了开发上的设计，从系统的设计到数据库的设计，简述了系统设计的整个流程。

第四章介绍了系统是如何实现的，详细介绍了各个功能的实现过程以及所用的技术。

第五章主要介绍了系统在测试方面的具体情况。

第六章对本文进行总结。

第二章 需求分析

2.1 系统需求分析

2.1.1 功能需求分析

在一个博客系统中，用户的基本操作流程可以分析为通过注册登录进入一个博客系统中，在这个博客系统中要能够查看别人发送的博客，博客展示是展示最近发送的博客，这样用户就能浏览到最新的信息。如果看着某条博客，觉得符合自己的观点或者觉得有趣生动时，用户可以对这条博客点赞，或者在其下方评论。而既然用户能评论对方的博客，那么对方也能对用户的评论进行点评回复进行互动交流。如果想让这条博客的内容分享给更多的别人阅读时，可以对其转发。如果此博客对于用户具有收藏价值，值得多次查看时，可以将其收藏起来，方便下次阅读。光是看博客肯定是不满足用户的需求的，用户自己也能够发送高质量博客，来让更多人了解自己的生活、知识、观点等。发送博客也不仅仅只是文字，有时候一张图片、一首音乐、一段视频可能会更加适合表达用户此时此刻的心情状态。并且目前的博客系统，内容都是多姿多彩的，仅仅是文字的话，那么这个社交平台就显得有点枯燥空洞了；如果用户发送的内容及其多且繁琐的话，那么就会影响到其他用户的观感，为此我们需要限制用户发送内容的字数，使其真正成为“微”博客，能让用户能够阅读更多精简的信息。而用户查看除了按最新发送的顺序来查看，也能搜索某条微博客进行查看阅读，同时也能搜索某名用户，关注他，查看他发送过的博客。除了发送和查看微博客外，同时用户也能修改自己的个人信息，如个人头像、个人说明等。当用户对某条微博客进行评论、点赞等操作时，为了能让这条博客的博主能够收到更好的信息反馈，应该使用消息来提醒此条博客的博主，让他能够及时得到反馈，以便下次发表更好的博客。用户之间，应该是可以互相聊天通信的，这样这个博客系统就不仅仅是发送查看博客，同时也能进行及时聊天互动，使系统的功能更加多样性。同时人们都是有好奇心的，通过将那些最多互动的博客进行排名，让用户知道现在最多人查看的热点博客是什么，能够第一时间掌握当前热点消息，这大大地满足了“吃瓜群众”们的好奇心。而人们在平时工作学习后，正是通过在社交网站上活动使其能够较为快捷地释放自己的压力，他们只要打开手机、电脑就能随时随地地浏览新鲜、有趣的内容了。

2.1.2 安全性能需求分析

互联网快速发展的今日，安全问题已经成为每个网站系统不可忽视的重要一部分，我们在编程时，应当遵守代码规范，注意代码安全漏洞，而本系统也对用户的密码进行了存储加密，即把表里的用户密码使用了 MD5 加盐的方式进一步进

行加密，这种加密方式可以防止被数据库被拖库的时候黑客无法使用社工库对密码进行快速的破解，甚至在很大程度上可以防止密码的破解，从而给用户争取到足够的时间去修改密码 **Error! Reference source not found.**。而用户在没有登录时，是无法进入本博客系统的，这样将限制游客进入本系统，大大加大系统的安全性，避免不是本系统的用户也能进入系统对系统进行一些恶意操作。

2.2 系统可行性分析

对于一个博客系统的开发，首先可以肯定其开发是可以实现的。毕竟目前网上拥有大量的博客系统可以参考；其次，开发一个博客系统，使用 SpringBoot 框架来开发的话很容易搭建起来，但是对于其众多的功能需求，大部分基本功能都是可以开发出来的，但有些功能，如：第三方登录、分布式构建、消息的订阅和发布等，这些功能都是需要大量的技术和软、硬件设备支撑。鉴此考虑，本系统没有引入这些功能。而前端的页面上，对于交互性极强的页面或者区域，通过引入已经封装好的 JavaScript 插件，也是能够达到同样的效果。因此综合外在因数和内在因数，开发出这个博客系统并实现基本功能是可行的。

2.3 本章总结

在本章主要介绍了系统在开发前对于功能、安全性上的需求分析，并分析本系统能否实现，分析了系统实际可行性，这将给系统的设计带了极大的帮助。

第三章 系统设计

3.1 MVC 分层模式

MVC 模式就是分层架构模式的一种，它适合用于应用开发，也适用于其他领域范围的设计工作。MVC 是三个英文单词首字母缩写而拼接成的，分别为 Model 模型层、View 视图层、Controller 控制层。

从开发结构上，MVC 认为可以分为三层：

（1）最直观的一层，是直接面向用户的客户端，也就是视图层 View。它提供了用户看到的程序的界面，是程序给用户的外在形象。

（2）最里面的一层，是数据层 Model，用于对数据的操作，以及和数据库的交互都在这里进行。

（3）中间的一层，是控制层 Controller，它负责处理用户从客户端发送的请求，然后根据需要去调用数据层 Model，来完成用户的请求，在这里主要进行业务逻辑运算。

MVC 这三层是关系密切的，但又互不干扰对方，换句话说，每一层的内部变化对于其它层来说都是透明的。每一层都只会暴露接口供外部调用。这样一来，各个层就实现了解耦操作，每个层也各司其职，当需要修改的时候可能只需要修改某一层就够了，为后期的维护和迭代提供了很大的便利性。而在本系统中，view 层则是我们所写的前端，我们前端用的是 html 页面和 Thymeleaf 模板引擎和 JQuery 前端框架，所以我们的 view 视图层可以用这三个技术来表示。model 层则是我们所写的一个个对应数据表字段的实体类，这个实体类里面只有 private 域的属性，以及 get、set 方法和被重写的 toString 方法。Controller 层实际还可以分为 Controller 层和 Service 层，Controller 层主要是接收前端发送过来的请求，调用 service 层的代码来进行逻辑运算，并返回结果给前端页面；而 Service 层主要就是进行逻辑运算，通过调用 dao 层对数据进行持久化操作或者进程查询等。

本系统不仅使用传统的 MVC 模式，同时将其进行细分，总体架构为：页面 - Controller - Service - Dao。在当今的互联网架构模式下，传统的 MVC 设计模式已经无法满足人们日益高涨的开发需求，于是便出现了 DAO 层来直接操作数据库，出现了 service 层来作为 Controller 层间接调用 DAO 层的中间层，而我们将逻辑都写进 service 层当中，在 Controller 层我们尽量的少些业务逻辑。这么做的好处是我们将每个层分割开来各司其职，能够让我们写代码的人和看代码的人都能做到对每个分层作用都一目了然，Controller 层就更加关注前端的调用，而 service 层则更关注业务逻辑的实现，DAO 层则更加的关注与数据库操作

相关的实现。同时在 service 层中引入 Redis，将大大缓解 Dao 层对数据库的查询压力。整个程序的设计，也变成了针对服务进行设计。而开发人员的分配，理论上真的可以按层次划分了，从整体上看，service 层和 DAO 层的引入，更加清晰的定义了每一层的工作和边界。这对于系统后期的维护和迭代是非常有利的。架构模式图如下图 3-1 所示：

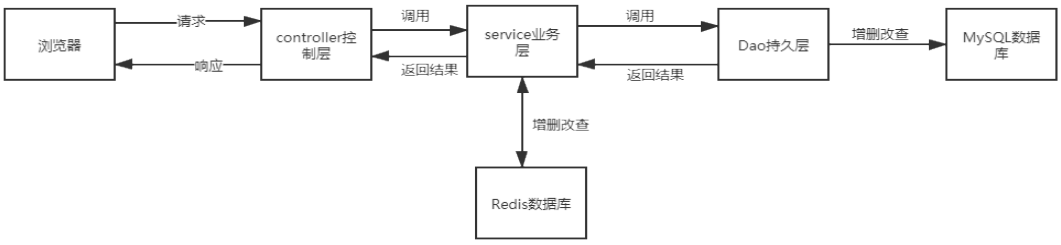

图 3-1 模式架构图

3.2 系统架构设计

本博客系统的架构总体上由前端（浏览器和网页）和后端（服务器）组成，客户端（浏览器）面向的是使用本博客系统的用户，系统的所有功能点几乎都由客户端和服务端以及数据库进行交互的产生结果。前端页面中引入大量的 JavaScript 插件和 jQuery 框架进行渲染美化，而后端中引入多个框架对其进行扩展，本博客系统的总体架构如下图 3-2 所示：

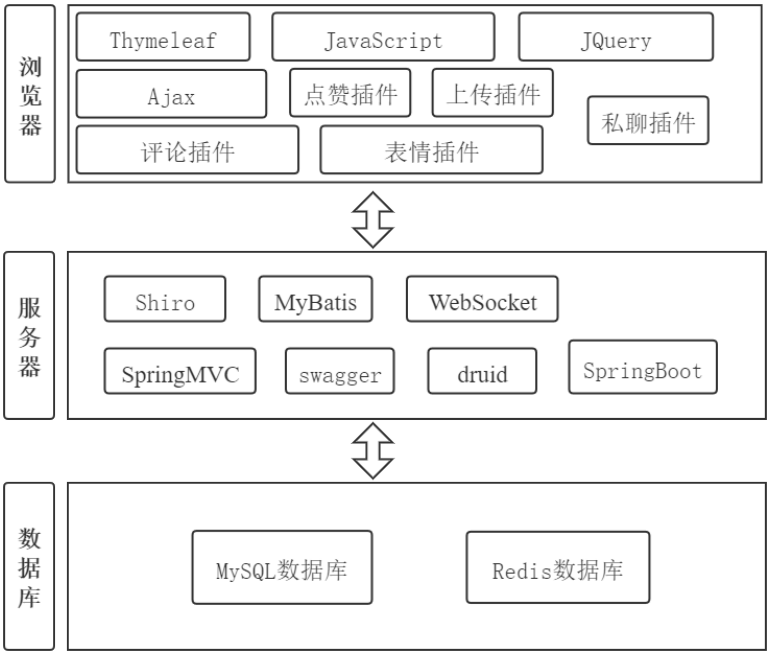

图 3-2 系统架构图

3.3 总体功能设计

根据上面系统的功能需求分析，设计并画出本博客系统的系统功能模块，如下图 3-3 所示。

（1）博客模块包括了发送博客、和博客操作两个功能，而发送博客又细分为发送文字、表情、图片、音乐、视频；博客操作细分为点赞、评论、转发、收藏博客。

（2）个人中心包括个人信息、个人收藏、个人博客，个人信息功能能够查看和修改个人的信息，比如头像、个人签名等；个人收藏功能可以查看自己收藏的博客；个人博客功能可以查看自己发送过的博客。

（3）搜索模块包括查找用户和查找博客。通过输入关键字进行查找，如果能查找到就会显示出来，不存在就不会显示。

（4）热门排行模块主要是将多人评论、转发和点赞的博客列为热门博客并将其放入排行榜进行排行。

（5）消息通知模块主要是将各种消息发送给用户，以便提醒用户。例如某个人私信了 A 用户，系统会发送一些通知给 A 用户，来提醒 A 用户有人私信了他。

(6) 登录注册模块主要包括登录和注册两大功能。登录本系统时首先要有账号才能登录。而账号通过注册产生，所以两个功能相辅相成。

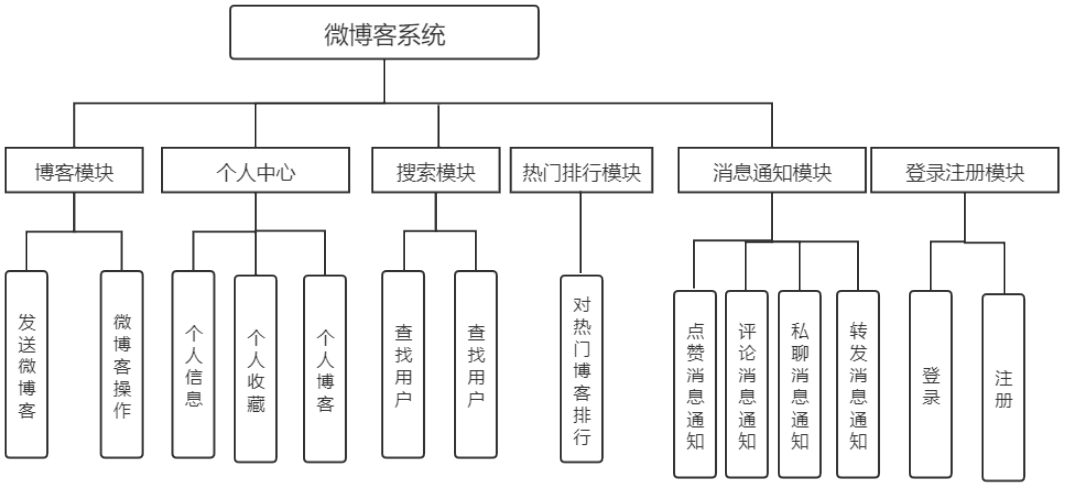

图 3-3 总体功能图

3.4 数据库设计

数据库的各个实体一定要设计合理，每个实体的属性都要与功能业务结合起来，同时属性的类型以及与其他实体的属性关联都要谨慎考虑，避免反复更改。

结合功能的相关需求，设计出的各个实体模型以及他们之间的实体关系用E-R图表示如图3-4.1 和 3-4.7所示。

用户与博客之间的关系，可以用图3-4.1表示。

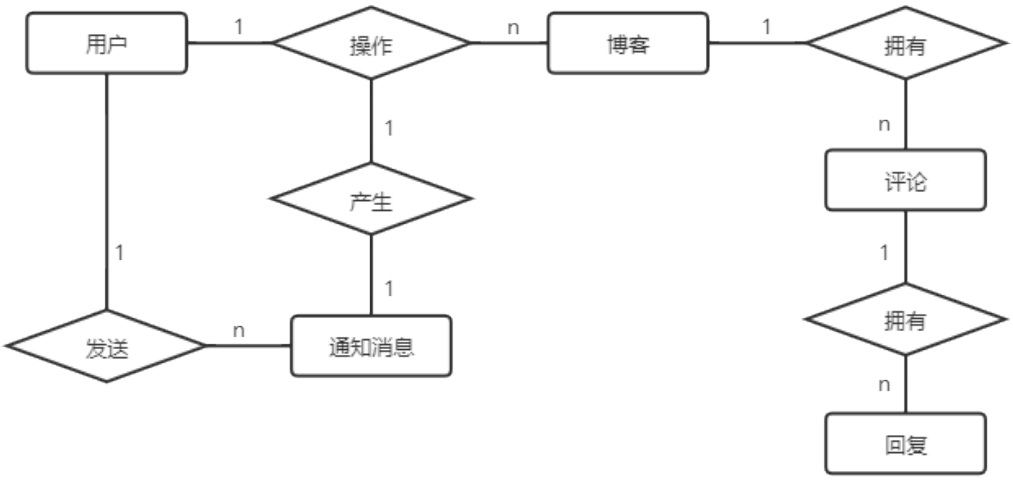

图3-4.1　用户与博客E-R关系图

用户与用户之间的关系、用户与聊天记录的关系，可以用图3-4.2表示。

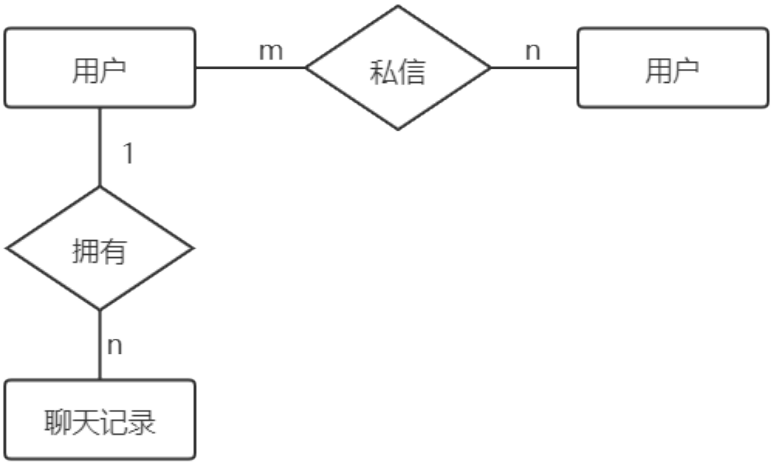

图3-4.2　用户与用户、用户与聊天记录E-R关系图

（1）用户实体：姓名、密码、用户名、收到的通知消息数量、个性签名、性别、头像名。

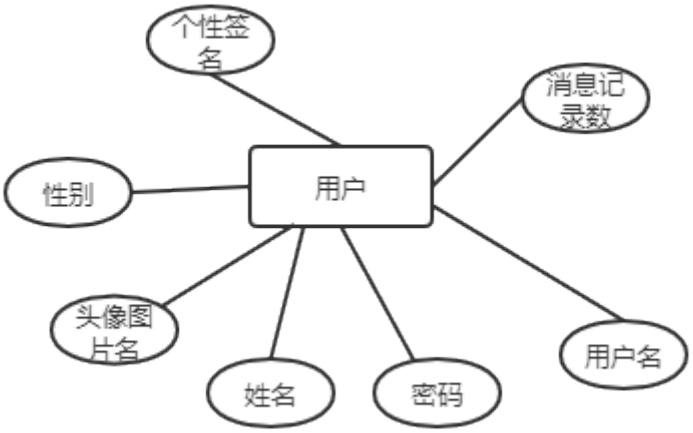

图 3-4.3 用户实体图

（2）聊天记录：会话 id、消息发送者、消息接收者、发送消息的内容、发送时间。

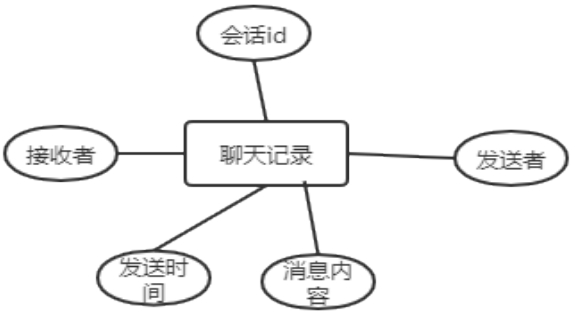

图 3-4.4 聊天记录实体图

（3）博客实体：图片、博客文字内容、音乐、视频、作者 id、发送时间、点赞数。

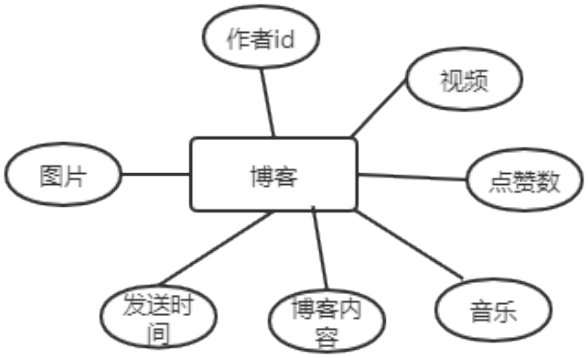

图 3-4.5 博客实体图

（4）通知消息实体：通知人、被通知人、消息内容、产生时间。

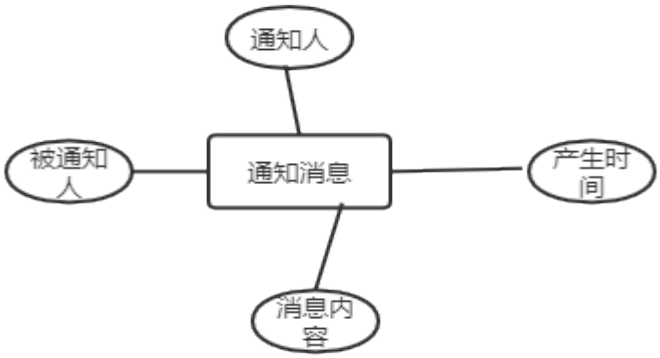

图 3-4.6 通知消息实体图

（5）回复消息实体：回复者、被回复者、回复内容、回复时间、在哪条评论下回复。

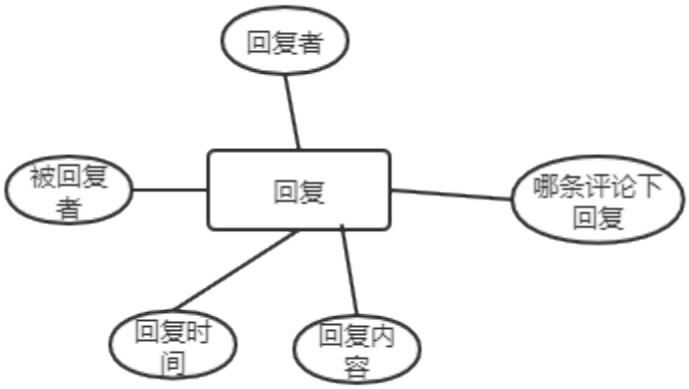

图 3-4.7 回复消息实体图

（6）评论消息实体：评论者、被评论者、评论内容、评论时间、在哪条博客下评论。

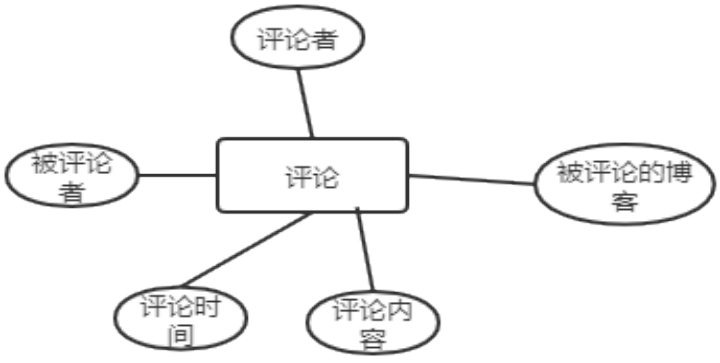

图 3-4.8 评论消息实体图

3.5 前端设计

一个网站的外观直接影响到用户的访问数量,用户都喜欢使用一个外观好看、页面炫酷的网站,所以在本系统的前端页面设计上,也是下了大量的时间和精力。首先,用户第一次访问本系统是通过登录页面作为系统入口,所以一定要给用户一个良好的“第一印象”。登录页面和注册页面都是以深夜星空作为网页背景,其次在页面的按钮上增长按钮的长度;而输入框中如果输入错误,应该能给用户友好的提示,让用户能够根据提示做出修改。

当用户登陆成功进入首页,用户应该能够立刻看到其他用户发布的博客,并且也能立刻发布博客,所以在首页应该展示用户的博客列表,和发布博客输入区域。而发送博客时,发送的内容不应只有文字,还应支持表情、图片、视频等格式,所以这里引入了 JavaScript 插件,比如:表情插件、上传插件等。发送按钮也应该要色彩斑斓,给用户一个良好的视觉体验。并且用户查看微博客时,能够按照图片、音乐、视频等进行分类查看。对于微博操作,为了能让用户有更好的体验性,给予用户更好的交互性,这里也是使用了 JavaScript 插件,如点赞插件、评论插件、转发插件。点赞插件能够使用户点击某个按钮会改变按钮样式,并且让按钮内的数字加 1,而再次点击则减一并回复原样。评论功能引入插件好,使用户拥有极好的体验性。评论框采用层级模式,并且在评论里再内嵌一层回复框,这样的设计真正地让用户能够自由地评论和回复了。

而点击首页头像或者首页最上方的用户名时,用户可以进入自己的个人主页,个人主页里用户能够查看自己发送的博客和收藏的博客。同时也能对个人头像进行修改等操作。而点击博客左上角作者的头像时,能够进入对方的个人主页并且鼠标放在对方的头像时,会弹出个小卡片显示这个用户的小部分信息。这样的效果会让用户更加喜爱,便于用户了解其他人。

在进入对方的主页后,我们应该能够点击私聊按钮和此用户进行聊天。而聊天界面的设计包括聊天用户列表的展示以及聊天内容的显示。这里聊天的界面采用类似于微信的聊天框,这能让用户使用得更加有亲切感,在这个页面配合 websocket 技术,用户能够实时收到别人发送过来的消息,能看到他发送的内容,发送时间和他的个人头像。这样的设计就像一个网页版的微信聊天一样,给用户带来极大的使用体验。同时当有消息过来时,聊天对象模块应该出现一块红色区域,里面的数字表示此用户发送了多少条消息过来,当用户点击聊天对象查看消息时,红色区域消除。通过这样的设计来表示消息通知,让用户能够更直观的收到消息通知。

在搜索页面中,用户要能按分类来查找博客并显示博客列表。同时加入用户

查找，找出用户时，应该要展示用户的头像、用户名、和一些简短的介绍，便于用户能够更加直观的了解查找的人。这里还新增了一个小细节，就是搜索框的搜索关键字随着点击不同的查找方式而不会改变，也就是说如果搜索“666”，点击用户查找，那么就会跳转到用户查找页面，并且搜索输入框的内容不会清空，下次再次点击别的搜索方式时，还是会按照这个输入框的内容进行查找，这给用户带来了极大的使用体验，用户不再用担心跳转到一个页面后就重新输入一次搜索内容的情况。部分页面的设计如下图 3-5.1 至 3-5.3 所示：

登录页面，用户登录成功会进入主页，失败则跳回登录页面。

图 3-5.1 登录页面设计图

登录成功后，会跳转到主页，主页上有一个输入框用于发送内容，输入框下面的几个按钮为发送表情、图片、视频、音乐按钮，点击不同按钮会选择使用对应的功能，同时也会触发页面对应的 JavaScript 函数来对页面进行改变。输入框右侧为个人头像和用户名，用户点击个人头像可以进入到个人主页。而输入框下方则是其他用户和本用户发送的博客列表。

图 3-5.2 主页页面设计图

私信页面，页面展示聊天对象列表和聊天框，聊天框上方为聊天对象的用户名，聊天框中间为聊天的内容。当用户发送消息过来时，会在聊天对象列表中对应的对象区域显示一个红色圆圈，圆圈里面的数字表示该用户发送了多少条消息，当用户点击查看时，红圈消失，里面的数字也会归零。

图 3-5.3 私信页面设计图

3.6 本章总结

本章中主要介绍了系统方方面面的设计，从系统整体到数据库方面，再到前端页面上，从头到尾地进行设计和分析。

第四章 系统实现

4.1 技术选用

4.1.1 SpringBoot 框架

SpringBoot 是一种非常强大的框架，它简化 Spring 应用的初期搭建以及整个应用开发过程。该框架采用约定优于配置的设计，从而使开发人员不需要再自己完成样板化的配置。SpringBoot 提供了一种全新的便捷的编程范式，可以更容易地开发 Spring 相关的项目，在开发过程当中可以专注于应用本身的业务功能开发，而无需在 Spring 配置上花费太多的人力物力和时间 **Error! Reference source not found.**。

SpringBoot 基于 Sring4 进行设计，保留了 Spring 框架原有的好处和优点。实际上，SpringBoot 不是一个框架，更像是一些类库的集合。使用 maven 或者 gradle 项目时，导入相应依赖即可使用 SpringBoot，而无需自行管理这些类库的版本 **Error! Reference source not found.**。

4.1.2 MyBatis 持久层框架

Mybatis 是一款优秀好用的持久层框架，非常适合与 Springboot 或 Spring 框架结合在一起使用，Mybatis 支持定制化 SQL 和高级映射等特性，Mybatis 与 Springboot 结合使用时，只需要在 Springboot 配置文件上写好相关的配置项即可完成环境搭建，能够极其方便地操作数据库数据，并且框架比较简单，容易入门和便于上手使用。

4.1.3 MySQL 数据库

MySQL 数据库由于其免费，并且性能是非常高的，使用 MySQL 作为本系统的数据库是极其合适的。并且 MySQL 数据库的操作和安装简单方便，便于系统的开发。

4.1.4 Redis 存储

由于 Redis 数据库拥有其独特的数据存储结构，所以使用 Redis 的 Zset 结构来存储热点微博是最合适不过了，也正是因为 Redis 支持排行榜功能，所以选择了 Redis 数据库作为排行榜功能的开发工具。

4.1.5 WebSocket 协议

WebSocket 是 HTML5 开始提供的一种在浏览器和服务器间进行全双工通信的协议。目前很多没有使用 WebSocket 进行客户端服务端实时通信的 web 应用，大多使用设置规则时间的轮询，或者使用长轮询较多来处理消息的实时推送。这样势必会较大程度浪费服务器和带宽资源，而 WebSocket 正是来解决该问题而出现，

使得 B/S 架构的应用拥有 C/S 架构一样的实时通信能力[15]。WebSocket 正是由于

其对系统负担较小，所以选用它，但由于其技术难度还是有点高，所以在使用方面并没有涉及得很全面，而且需要花费大量时间去研究和测试改正。

4.2 功能实现

4.2.1 “登录注册”功能的实现

注册流程图，如图 4-2.1 所示，用户输入账号、密码时，前端会进行 JavaScript 校验，如果输入不符合规则，则不能点击注册按钮并出现提示信息。如果输入格式正确时，会把数据提交到后台检查此用户名是否已被别人使用，如果此用户名不可用则返回提示信息给前端，可用的话就会注册成功，并且密码以 MD5 并加盐方式加密，以密文形式保存到数据库，最后注册成功会跳转到登录页面。

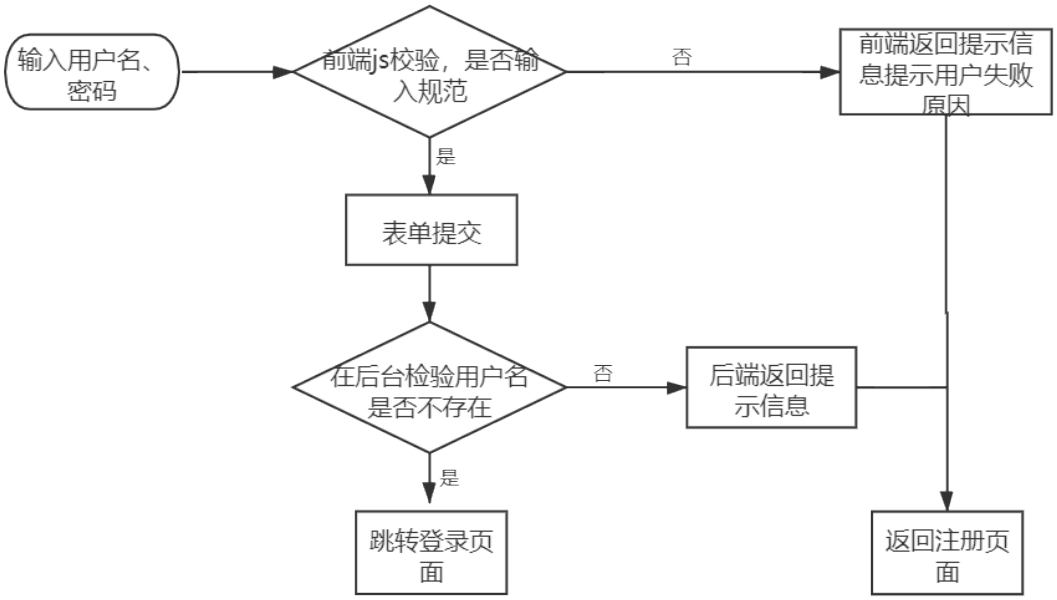

图 4-2.1 注册流程图

而前端的表单校验，最为关键是表单的 onsubmit 事件，只有触发这个事件，才会判断是否会把表单提交到后台。如果 function 的结果为 false，则不能提交表单，同时出现提示信息；结果为 true 时才能提交表单。提交到后台时会校验用户名是否存在，如果存在则返回提示信息，用户名不存在时则注册成功，并把密码加密。校验核心代码如下图 4-2.1.2 所示：

```
window.onload=function () {
  document.getElementById("from").onsubmit = function () {
    return checkname() &&checkpassword() && checkRepassword();
  }
  //    绑定离焦事件：
  document.getElementById("username").onblur = checkname;
  document.getElementById("password").onblur = checkpassword;
  document.getElementById("Repassword").onblur = checkRepassword;

}

function checkname() {
  var name = document.getElementById("username").value;
  var reg_username = /^[a-zA-Z]{1}([a-zA-Z0-9]|[._]){4,19}$/;
  var flag = reg_username.test(name);
  var s_name = document.getElementById("s_username");
  if(flag){
    s_name.innerHTML="用户名可用";
    s_name.style.color = 'green';
  }else{
    s_name.innerHTML="用户名格式不规范，请重新输入！";
    s_name.style.color = 'red';
  }
```

图 4-2.1.2 前端校验代码

其中登录流程如图 4-2.1.3 所示。其中登录时在后台的数据检验交给了 shiro 安全框架来执行，shiro 框架会对登录的密码进行加密，然后与数据库中加密的密码进行比对，如果匹配成功，那么就能直接跳转到主页。匹配失败则会抛出异常返回给前端页面。

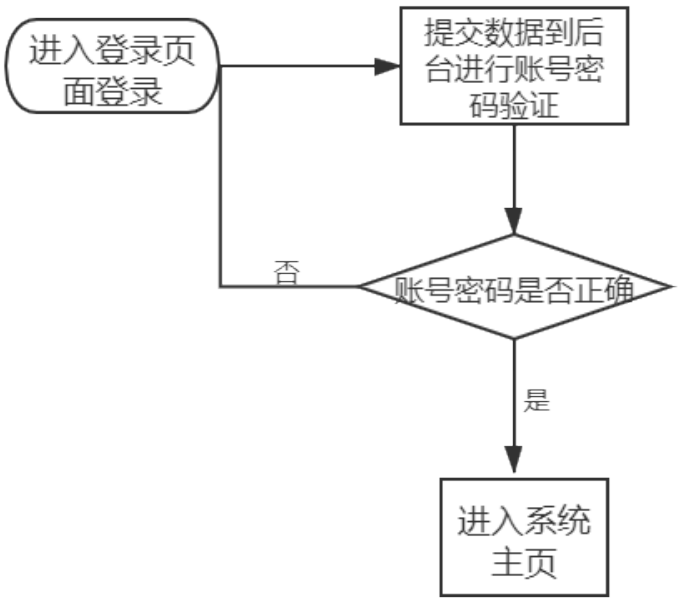

图 4-2.1.3 登录流程图

登录时系统会自动调用 shiro 框架的校验方法 login()，将用户名密码（token）传给 shiro 框架。而 shiro 框架在调用 login()方法时，会调用 shiro 内部的认证方法来进行身份认证，我们只需自己写一些业务代码来把 token 拿来和数据库查出的密码进行对比就行了，对比成功则能登录成功，对比失败就会抛出异常信息传给前端，前端以信息提示的方式来提醒用户。Shiro 核心代码如图 4-2.1.4 至 4-2.1.5 所示：

```
@RequestMapping(value="/login")
public String userLogin(Model model, User user, HttpServletResponse response){
    if(user==null){
        return "/login/login";
    }
    String username=user.getUsername();
    String password=user.getPassword();
    UsernamePasswordToken token = new UsernamePasswordToken(username,password, rememberMe: false);
    Subject currentUser = SecurityUtils.getSubject();
    try {
        //此步将 调用realm的认证方法
        currentUser.login(token);
    } catch(IncorrectCredentialsException e){
        //这最好把 所有的 异常类型都背会
        model.addAttribute( attributeName: "msg", attributeValue: "账号或者密码错误!");
        return "/login/login";
    } catch (AuthenticationException e) {
        model.addAttribute( attributeName: "msg", attributeValue: "登录失败!");
        return "/login/login";
```

图 4-2.1.4 登录部分核心代码

当调用 Shiro 的 login（）方法后，shiro 将会调用它内部的认证方法 doGetAuthenticationInfo 来对传进来的 token 进行验证，当验证失败是会立即抛出异常并拦截回登录页面，而验证成功则会放行。

```
protected AuthenticationInfo doGetAuthenticationInfo(AuthenticationToken authcToken) throws AuthenticationException {

    System.out.println("---------------认证 -------------");
    //获取基于用户名和密码的令牌
    //实际上这个authcToken是从LoginController里面currentUser.login(token)传过来的
    //        这个username是shiro自动把登录的username传入给token带进来的
    UsernamePasswordToken token = (UsernamePasswordToken) authcToken;
    String account = token.getUsername();
    User user = userService.findUser(account);//根据登陆名account从库中查询user对象
    if(user==null){throw new AuthenticationException("用户不存在");}

    //进行认证，将正确数据给shiro处理
    //密码不用自己比对，AuthenticationInfo认证信息对象，一个接口，new他的实现类对象SimpleAuthenticationInfo
    /* 第一个参数随便放，可以放user对象，程序可在任意位置获取 放入的对象
     * 第二个参数必须放密码，
     * 第三个参数放 当前realm的名字，因为可能有多个realm*/
    AuthenticationInfo authcInfo=new SimpleAuthenticationInfo(user, user.getPassword(), ByteSource.Util.bytes(user.getSalt()), this.getName());
    //AuthenticationInfo authcInfo=new SimpleAuthenticationInfo(user,user.getPassword(),new MySimpleByteSource(account), this.getName());

    //清之前的授权信息
    super.clearCachedAuthorizationInfo(authcInfo.getPrincipals());
    SecurityUtils.getSubject().getSession().setAttribute( o: "user", user);
    return authcInfo;//返回给安全管理器，securityManager，由securityManager比对数据库查询出的密码和页面提交的密码
```

图 4-2.1.5 shiro 核心代码

4.2.2“微博客”功能的实现

发送博客步骤为：

用户在输入框中输入发布的博客内容，点击输入框右下角的发布按钮，将输入的内容发送出去，然后就能在页面上看到自己发送的博客了。其中此功能的实现过程中，在前端使用了大量的 JavaScript 插件使页面更加美观，交互性更好。而博客的发布，一定要在输入框输入文字才能够发布博客，否则发布按钮将不可选用。而发布成功后，页面会重新加载并能显示出自己发送的博客与其他用户发送的博客，而博客则会以最新时间的顺序进行排序显示出来。

部分前端 JavaScript 插件代码如下图 4-2.2.1 所示，其中 js 文件、css 文件的引入以及他们之间的加载顺序可能会影响他们的正常显示，所以为了能让各个插件间能够互相协调共处，需要花费较多时间去对 js、css 文件的加载进行合理排序。

```
<script>
    $('#music').fileinput({
    // <!--uploadUrl: '#',      // you must set a valid URL here else you will get an error-->
    uploadAsync : true,           //默认异步上传
    showUpload : false,           //是否显示上传按钮,跟随文本框的那个
    allowedFileExtensions : ['mp3', 'mpeg','flac'],
    overwriteInitial: false,
    maxFileSize: 1000000,       //单位为kb，如果为0表示不限制文件大小
    msgFilesTooMany: '选择上传的音乐数量 超过允许的最大数值（1首）！',
    maxFileCount: 1,     //表示允许同时上传的最大文件个数
    showCaption: false,
    dropZoneEnabled: false
```

图 4-2.2.1　前端 JavaScript 插件代码

发送微博客功能主要把着重点放在前端页面上，主要是为了让用户有个良好的使用体验，所以在前端使用了许多 JQuery 函数、JavaScript 插件等让页面的元素随着用户的操作而发生相应的变化。另外前端发送数据给后台服务器时，是通过 Ajax 方式发送，使用 Ajax 异步请求来发送数据给服务器，能够实现页面的异步更新，这意味着可以在不重新加载整个网页的情况下，可以对网页的某部分进行更新，不再因等待服务器的响应而进入等待状态。而后端在接收到前端传过来的文字、文件、图片等内容后，会将其汇总在一起保存在一个微博对象中并将此对象存入数据库对应的表中进行数据的持久化，这样就能方便系统在显示博客时能从数据库中再次拿取数据出来了。后台部分代码如下图 4-2.2.2 所示：

```
@RequestMapping(value = "/postWB")
@ApiOperation(value = "发送微博")
public String upload(@RequestParam("images[]") MultipartFile[] images,@RequestParam("videoFile") MultipartFile videoFile,@RequestParam("musicFile")
    接收图片！！！！！！！！！！
    String[] a = new String[6];
    boolean flag = images[0].getOriginalFilename().equals("");
    if (images != null && images.length > 0 && !flag) {
        for (int i = 0; i < images.length; i++) {
            String Filename =  images[i].getOriginalFilename();
            a[i] = Filename;
            System.out.println("上传的文件名为：" + Filename);
            // 获取文件的后缀名
            String suffixName = Filename.substring(Filename.lastIndexOf( str: "."));
            System.out.println("文件的后缀名为：" + suffixName);
            // 设置文件存储路径
            String filePath = "E:/lidea编程项目/毕业设计项目（微博）/src/main/resources/static/imgUpload/";
            String path = filePath + Filename;
            File dest = new File(path);
```

图 4-2.2.2　发送博客的代码

微博操作模块实现：

微博操作模块包括点赞、评论、回复评论、转发、收藏等操作。

（1）点赞：点赞的时候，点击点赞按钮，前端页面会进行样式改变使其按钮变为已点赞，同时将一些数据传入到后台，将这个用户对这条微博点赞这个行为记录到数据库中，并且会记录这个微博被点赞的次数，每点击一次会自动+1，取消点赞就-1。当用户下一次登录时，会查出对这条微博是否点过赞，从而在点赞按钮的样式也会相应改变为 点赞/已点赞，同时也会查询出对这条微博的点赞次数。部分核心代码如图 4-2.2.3 和 4-2.2.4 所示：

```
public String like(String zCount, String username, String weiboid){
    love love = new love();
    User user = userService.findUser(username);
    love.setDzid(UUID.randomUUID().toString());
    love.setUserid(user.getId());
    love.setWbid(weiboid);
    likeService.insert(love);

    weibo weibo = new weibo();
    weibo .setWeiboId(weiboid);
    weibo.setZan(Integer.parseInt(zCount));
    weibo weibo1 = weiboService.queryWeiboByID(weiboid);
    weibo.setPostTime(weibo1.getPostTime());
    weiboService.updateZanByPrimaryKey(weibo);
```

图 4-2.2.3　部分点赞代码

在显示博客时，会检验当前登录用户是否对显示的博客进行过点赞，如果点过赞，那么这条博客的点赞按钮会显示为已点赞，而以下代码就是判断用户是否已点赞这条博客。

```
//          是否对这一条微博点赞：
love.setWbid(weibo.getWeiboId());
love love1 = likeService.ifLike(love);

if(love1 == null){
  weibo.setIflike(false);
}else{
  weibo.setIflike(true);
}
```

图 4-2.2.4　判断是否点赞代码

（2）评论：对某条微博进行评论，评论完时前端的 JavaScript 评论插件会立刻生成一个评论模块在微博下方，同时把这条评论数据存进数据库使其持久化，在往后再次登录时，仍然能查看到评论。评论功能的关键点是要把评论关联好哪一条微博，这样在查询微博的时候，只需查出和这条微博关联的所有评论即可。微博和评论是一对多的关系，一条微博带着多条评论。部分核心代码如图 4-2.2.5 所示：

```
@GetMapping("/pl")
@ApiOperation(value = "评论")
public String pl(String plContent,String time,String username,String weiboid) throws ParseException {

    weibo weibo = new weibo();
    weibo .setWeiboId(weiboid);
    weibo weibo1 = weiboService.queryWeiboByID(weiboid);

    SimpleDateFormat sdf = new SimpleDateFormat( pattern: "yyyy-MM-dd HH:mm:ss");

    User user = userService.findUser(username);

    plList plList = new plList();
    plList.setId(UUID.randomUUID().toString());
    plList.setUserid(user.getId());
    plList.setUsername(username);
    plList.setPlcontent(plContent);
    Date date = sdf.parse(time);
```

图 4-2.2.5　部分评论代码

（3）回复评论：在评论时，我们也可以对别人的评论进行回复，设计思路和评论功能相同，一条评论带着多条回复，当点击回复时，前端会立即创建回复内容模块，同时我们将回复内容的数据保存到数据库中进行持久化，下次登录时还能查看到回复。

其中在查询时，微博只需关注要查出他带有的评论就行了，而评论也只需查出他自己带有的回复，就是查询只需查出与他直接关联的下一层，对于下层则不用关心。这样查询的话就能避免进行多表联查，使数据库的查询性能降低，使查询速度大大提高。核心逻辑代码如下图 4-2.2.6 所示：

```
    List<plList> plList =  plListService.selectByWeiboId(weibo.getWeiboId());
    for (plList plL : plList) {
      List<hfplList> hfplLists = hfplListService.selectByPlId(plL.getId());
      for (hfplList hfpl : hfplLists) {
        hfpl.setUserHeadImg(userService.findUser(hfpl.getUsername()).getHeadImgName());
      }
      plL.setHfplLists(hfplLists);
      plL.setUserHeadImg(userService.findUser(plL.getUsername()).getHeadImgName());
    }

    weibo.setPlLists(plList);
  }
  return weibos;
}
```

图 4-2.2.6　核心逻辑代码

（4）转发：转发的本质其实也就是发送一条博客，只是我们的博客内容是另外一条博客，所以我们在转发后，只需在一条微博里面再嵌套一条博客就可以。核心代码如图 4-2.2.8 所示，这里通过在博客实体类里面设置转发的那条博客 id，而在显示博客的时候，服务器会专门查看这个转发的博客 id 属性是否为空，如果不为空的话则表示这条博客是属于转发博客，是一条转发别人博客的博客，那么在页面显示时就会以一条博客里面再嵌套一条博客的形式来显示。

```
@GetMapping(value = "/zfWB")
@ApiOperation(value = "转发所有微博（实时）")
public String upload(String zfContent,String username, String weiboid) throws Exception {
    weibo weibo = new weibo();
    //        微博id(用uuid 取代id )
    weibo.setWeiboId(UUID.randomUUID().toString());
    weibo.setContent(zfContent);

    User user = userService.findUser(username);
    // 用户id
    weibo.setUserId(user.getId());

    // 发送时间 （格式转换）
    Date postTime = new java.sql.Date(new java.util.Date().getTime());
    weibo.setPostTime(postTime);
    weibo.setZfwbid(weiboid);
    weiboService.post(weibo);
```

图 4-2.2.8　核心代码

（5）收藏：收藏功能时为了让用户能够对自己喜欢的博客进行收藏，方便

下次查看。点击收藏后，前端按钮样式会改变为相应的样式，同时会将数据保存到收藏表中，下次登录时会查看是否对这条微博进行收藏，从而使前端页面显示出对应的按钮样式，而取消收藏后，会将这条收藏记录删除。同时收藏后可以在自己的个人主页中查看收藏的微博记录。核心代码如图 4-2.2.9 所示：

```
@GetMapping("/collect")
public String collect(String state,String username,String weiboid){
    collect collect = new collect();

    User user = userService.findUser(username);

    collect.setCollectid(UUID.randomUUID().toString());
    collect.setUserid(user.getId());
    collect.setWbid(weiboid);
    collectService.insert(collect);
```

图 4-2.2.9　收藏核心代码

4.2.3“消息通知”和“用户私信”的功能实现

4.2.3.1 生成消息通知的步骤及消息通知的实现

消息通知产生的步骤过程为用户对某条博客进行了点赞、评论或转发等操作时，以及用户发送私信给其他用户时，都会产生消息通知并发送给对方，对方如果此时在线的话，会立即受到消息提醒并可以查看消息通知的内容；而如果对方此时不在线，那么他就不能够立刻受到消息通知，只有当他再次上线时才能收到消息提醒。

消息通知模块实现思路:通过在网上查询消息通知模块的资料，发现实现的方案主要有两种：1.通过 ajax 长轮询，通过在 ajax 一直往后台发送请求查看是否有关于“我”的消息通知。2.通过 websocket 的方式实现消息的在线发送。我之所以选择 websocket 这种方式，是因为通过 ajax 长轮询这种方式会大大增加系统内存资源的消耗,这不利于以后系统的扩展以及优化,而 webSocket 是 html5 一种新的协议，实现了浏览器与服务器之间的全双工通信，能很好的节省服务器资源与带宽，并在服务器端与浏览器端实现实时通行，他建立在 TCP 之上， 同 http 一样，通过 tcp 来传输数据。并且它有以下特点：

（1）只需要一次 HTTP 握手，所以说整个通讯过程是建立在一次连接/状态中，服务器端会知道连接的信息，知道客户端关闭请求，同时由服务器主动推送，当有信息需要发送时，直接发送。

（2）客户端的连接通过 session 对象存储，能够实现实时推送，与 Ajax 沦陷和 long poll 相比有明显的优势。

但是使用 websocket 来实现消息通知却有一个缺点：当目标用户不在线时，会发送消息失败，造成用户上线后收不到消息通知。所以我在 websocket 的基础上进行了改进，在通过 websocket 发送消息的同时，将这条通知数据保存到数据库，当用户上线加入 websocket 后，会立即将数据库的消息查询出来然后加入到 websocket 的 onOpen()方法中，这样的话当成功加入 websocket 后会自动调用到 onOpen()方法将消息通过 websocket 发送给用户，这样一来用户在一上线时就能收到消息通知了，产生消息的代码如下图 4-2.3.2 所示：

```
//消息通知：----------------------------------------------------------------------------
//      本条微博作者的用户名
    String userName = weiboService.getUsernameByWeiboID(weiboid);
//      获取对该条微博点赞的实时总数量（当用户点击消息提示时清0）
int likeCount = userService.getLikeCount(userName);
int plCount = userService.getPlCount(userName);
int zfCount = userService.getZfCount(userName);
int chatMessage = userService.getChatMessage(username);
//    点赞后点赞通知数量+1 并保存到数据库
likeCount++;
userService.updateLikeCount(userName);
//        通过websocket发送通知给本条微博者：
webSocketServer.sendInfo(userName, message: likeCount+","+plCount+","+zfCount+","+ chatMessage );

//   添加一条点赞通知（将通知保存到数据库）
    likemessage message = new likemessage();
    message.setDzweibo(weiboid);
```

图 4-2.3.2　消息通知部分代码

当用户登录成功后，或者说连接上服务器后，用户会马上进行 websocket 的连接。而通过在 websocket 连接成功后会执行的 onOpen()方法里加入查询数据库的代码，这样就能保证用户一旦登录系统，服务器就会先去数据库中查询关于这个用户用户的消息通知，并通过 websocket 发送给用户。而由于 websocket 一次只能发送一条消息，但是本系统的消息通知有 4 种，那么可以直接将这四种信息以字符串的形式拼接起来，同时每种消息之间以一个特殊符号间隔开，这样直接拼接成一个字符串传给前端，由前端页面对这个字符串按照特殊符号进行分解解析，这样就能将这个字符串分解成 4 种不同的信息了，就能够把这四种不同的消息通知放在相对应的页面区域进行显示了。连接 websocket 成功后会执行的代码如下图 4-2.3.3 所示：

```java
/**
 * 连接建立成功调用
 * @param session 客户端与socket建立的会话
 * @param userName 客户端的userName
 */
@OnOpen
public void onOpen(Session session, @PathParam(value = "username") String userName){
//    不能通过@Autowrite 自动注入，只能通过 这种方式来或者service服务。
    UserService userService = applicationContext.getBean(UserService.class);
    sessionPools.put(userName, session);
    addOnlineCount();
    System.out.println(userName + "加入webSocket！当前人数为" + online);

//    当建立连接时立即获取到该用户的通知数并显示到前端
    //        获取对该条微博点赞的实时总数量（当用户点击消息提示时清0）
    int likeCount = userService.getLikeCount(userName);
    int plCount = userService.getPlCount(userName);
    int zfCount = userService.getZfCount(userName);
    int chatNum = userService.getChatMessage(userName);

    //        通过websocket发送通知给本条微博者：
    sendInfo(userName,  message: likeCount+","+plCount+","+zfCount+","+chatNum);
```

图 4-2.3.3　连接成功时执行的代码

4.2.3.2 用户之间私信流程及其实现：

用户之间的进行私信时所执行的流程为：用户通过点击其他用户的头像进入到他人的个人首页，通过点击私聊按钮建立起会话连接，一旦建立起会话连接，那么双方就会强行关联起来，双方都把对方加入到自己的聊天对象列表中。其中一方发送消息时，如果对方在线，那么 websocket 能直接发送聊天消息和通知消息给对方，同时将消息记录保持到数据库中。而如果对方不在线，那么 websocket 发送聊天消息和通知消息失败，但是会把数据存到数据库中。对方上线后会立即加入到 websocket 中，那么 websocket 连接成功时会发送一条消息给用户，那么此时可以将保存到数据库中的消息发送给用户，使用户能够一上线就能收到私信消息提醒，同时如果用户点击消息提示，能够直接进入聊天页面，并查询数据库获得之前别人给我发送的消息内容。通过将消息保存到数据库，能使 websocket 发送失败的消息保存起来，避免消息丢失，这样就能解决对方不在线的情况下，消息发送出去而对方接收失败的问题。如果不将消息存储到数据库的话，一旦用户不在线，消息会发送失败，这样就非常影响用户的使用体验。用户只想发送一次消息，对方就一定能够收到，不必要等到对方在线后才能发送信息。这样的话，对于线上、线下都能够保证消息的完整性了，不用再担心因为不在线而造成发送不了消息或发送的消息丢失的问题。

而正是因为 websocket 能够在线发送信息，所以很多在线聊天网页都会使用此技术来实现在线聊天。在本系统中也正是因为如此而引入了这种优秀的技术来实现私聊功能。