

**本科毕业论文（设计）**

**（2020届本科毕业生）**



**题 目： 基于SSM微博系统的设计与实现**

**学生姓名： 施航程**

**学生学号：** **16008149**

**学院名称： 科信软件学院**

**专业名称： 软件工程**

**指导教师： 李晋**

**二零二零年五月**

**【摘 要】**微博系统是一个用于在网络上分享，记录生活点滴的社交应用，它是一种一种基于用户关系信息分享，传播以及获取的通过关注机制分享简短实时信息的广播式的社交媒体，网络平台，用户可以通过PC，手机等多种移动终端接入，以文字、图片、视频等多媒体形式，实现信息的即时分享，传播互动。系统基于SSM框架，采用JavaWeb技术，前台使用了JSP页面。用户可以在此微博系统上分享，记录有趣的生活内容，或者在闲暇之余看到自己的关注和粉丝的点滴，也可以在多人聊天时畅所欲言。对系统的功能，需求，以及设计模式和实现过程做了详细介绍。

**【关键词】**SSM，JSP/Servlet，Ajax，JAVA，MVC设计模式

Design and Implementation of Blog System

**【****Abstract】**Weibo system is a social application for sharing and recording daily life on the Internet. It is a broadcast-type social network based on user relationship information sharing, dissemination and acquisition, and sharing of short-term real-time information through attention mechanisms. Media, network platform, users can access through a variety of mobile terminals such as PCs, mobile phones, etc., in the form of text, pictures, videos and other multimedia to achieve the instant sharing of information, dissemination and interaction. The system is based on the SSM framework and uses JavaWeb technology. The front end uses JSP pages. Users can share and record interesting life content on this Weibo system, or see their followers and followers in their spare time, or speak freely when chatting with multiple people. The functions, requirements, design patterns and implementation process of the system are introduced in detail.   
**【Keywords】**SSM, JSP/Servlet, Ajax, JAVA, MVC design patterns

目 录

[1 绪论 1](#_Toc1544950743)

[1.1课题来源 1](#_Toc762361724)

[1.2国内现状 1](#_Toc1126057266)

[1.3研究意义 2](#_Toc2018572298)

[2 技术介绍 3](#_Toc197957180)

[2.1 Java简介 3](#_Toc614155057)

[2.2 Maven简介 4](#_Toc1297635517)

[2.3 Spring简介 4](#_Toc1663698934)

[2.4 MyBatis简介 5](#_Toc1550899798)

[2.5 Tomcat简介 6](#_Toc1963881347)

[2.6 WebSocket简介 6](#_Toc130144639)

[2.7 JavaWeb简介 7](#_Toc1202595027)

[2.8 AJAX简介 9](#_Toc2046016872)

[2.9 开发环境 9](#_Toc1897411940)

[3 需求分析 11](#_Toc1817801277)

[3.1性能需求分析 11](#_Toc1683700317)

[3.2技术可行性分析 12](#_Toc559211300)

[3.3网络原理图 13](#_Toc1275879828)

[3.4用户角色需求分析 14](#_Toc1088053901)

[3.4功能模块需求分析 14](#_Toc1098659902)

[4 系统设计 19](#_Toc1112576008)

[4.1数据库设计 19](#_Toc924852027)

[4.2功能模块设计 23](#_Toc501380803)

[5 系统实现 27](#_Toc2128808840)

[5.1系统总体架构及实现 27](#_Toc1812614860)

[5.2数据库持久化实现 29](#_Toc414935678)

[5.3项目部署实现 30](#_Toc944538337)

[5.4系统功能实现 30](#_Toc656711335)

[6 系统测试 57](#_Toc1428945412)

[6.1功能过程 57](#_Toc975915083)

[6.2测试过程 65](#_Toc1872187842)

[7 总结 67](#_Toc930664650)

[后 记 68](#_Toc1557371449)

[参考文献 69](#_Toc1211253707)

# 1 绪论

## 1.1 课题来源

随着互联网媒体的快速发展，人们的生活节奏越来越快，微博已逐渐最大的互联社交媒体平台之一。网站用户的稳定增长，内容生态上的多样性以及社交网络的影响突出表明了微博的老少咸宜，雅俗共赏，包括视频内容制作，和Internet上的大量数据的进入，它逐渐成为了新一代年轻人不可分割的娱乐方式和新的生活态度。

在新媒体的领导和粉丝文化的领导下，移动端的微博成为了当今现象级的互联网生态，微博作为一个拥有着大量流量的社交媒体平台，具备着信息发布，评论交流，互动，新闻，追星等等大量丰富的功能。

## 1.2 国内现状

随着国外的推特和Facebook的流行，国内群众曾经有一段时间苦于缺少此类适合互联网快节奏的社交平台，随着中国软件行业的发展与人们对互联网应用的接受和认可，大量类似抖音，快手，腾讯微视以及新浪微博等快节奏社交应用趁着这股互联网浪潮在当代人们之间快速走红，并占领市场，扎稳脚跟。

微博作为一个开放的社交媒体平台，取决于广泛用户的发展，而内容是其增长和关注的基础。与微博上的常规网站相比，主要文章是在Twitter和微博上写作的最新产品，内容比内容丰富得多，内容也更加详尽，并且在一定程度上满足了用户加深阅读需求的需求。近年来，通过内容格式的多样性，将重点放在微博上的内容构建上进一步增强了用户体验。

为了创建一个令人印象深刻的，近年来，微博站点上的活跃用户数量一直在稳定增长，好用户的比例有所增加，性别平衡发生了变化，区域覆盖率有所提高。庞大的数量使微博用户可以获得内容的“第一优势”。微博传播机制带给了公众，并传播了裂变式的基础，以支持事件。而Twitter故事鼓励普通用户创建和共享短视频。女性用户青睐微博上的故事写作和最低出版门槛。

活跃用户的数量和大量用户构成了内容生态的基础。微博关注用户需求，并在播放内容上投入更多资金，以吸引用户的注意力并赢得他们的关注。无论是在视频领域，还是进行垂直化，增强，机构MCN都对创建，交换信息和易用性的每个环节都予以肯定，为获取信息创造了适当的条件。随着用户需求的满足，用户数量增加，粘度增加。结果，价值增加了

## 1.3 研究意义

随着计算机和网络技术的快速发展和互联网时代的到来，越来越多的移动端应用开始瞄准了用户的碎片化时间（手机游戏，短视频应用等），微博的出现尤其影响了人们的日常社交方式。

因此，微博系统的重要性不言而喻，一个稳定，高效，安全的微博系统，方便了用户的社交，加深了用户的交流，保护了用户的隐私，还优化了从发布内容到浏览信息的所有细节，降低了用户的上手操作门槛，拉近了每一个用户之间的距离。

# 2 技术介绍

## 2.1 Java简介

1995年，Sun发布了一门面向对象的高级编程语言，多用于网络后端开发以及安卓应用开发。

它是一种简单的，完全面向对象的编程语言，它采用了先编译后运行的方式，拥有着编译行编程语言和解释性编程语言的特点，运行在JVM（Java虚拟机）之上使得它具有平台无关性，只要相应的平台安装了相应的JRE（Java运行环境），Java虚拟机的存在不光屏蔽了操作系统的差异性，而且还提供了大量的安全机制，是的Java的可移植性，安全性，以及健壮性大大提升，与此同时Java虚拟机还支持垃圾自动回收，多线程并发优化等功能，极大的简化了开发的难度。

目前网络上存在着大量支持Java的开源框架，尤其是阿帕奇的Spring全家桶，和阿里巴巴的许多优秀的消息中间件，这些开源框架经过阿帕奇和阿里巴巴以及网络上众多优秀的Java开发者们的共同努力下，版本经过了一次又一次的更迭，系统的架构与功能日渐完善和易用

1.简单性：对于Java，它是一种完全面向对象的语言，几乎万物皆为对象，它通过提供最简单的方式来执行分配给它的任务，在这些方式中，可以理解基本概念并针对每种情况定制应用程序。Java排除了模糊的概念，例如运算符重载和函数指针等，并通过GC大大简化了程序设计人员的内存管理。此外，Java适用于小型服务器，因为它的JRE（Java运行时环境）类库比较小，不会占用过多的系统空间和内存等性能。

2.面向对象：Java作为完全面向对象的高级编程语言，具体表现在：几乎万物皆为对象（除了八种基本类型），类的外面不允许有代码，不存在不属于任何类或者对象的全局变量，所有可执行代码不许出现在方法体的外面。而且面向对象的思想结合着面向抽象的设计模式，是得代码的可扩展性大大提升，耦合性降低，尤其是在使用框架的时候，面向抽象的原则使得大量的底层实现类被屏蔽，使相应框架的功能更加好用，易用，开发者的学习成本和理解成本更低。

3.安全性：Java相比于C++，缺少了函数指针，这可以防止运行到一个不希望被出现的指令跳转，通过实例对象才能正常访问。所以有人称Java为“C++--”，这样提升了Java程序的安全性，防止私有字段暴露给攻击者，也减少了Java开发者的编写难度，降低了犯错几率。

4.体系结构中立：Javac可以编译出平台无关的class文件（字节码文件），任何安装了JVM（Java虚拟机）的设备，不论是什么操作系统，都可以进行切换，部署，和运行。

5.多线程：并发运行机制允许并行执行应用程序，并发机制确保共享数据的正确运行。通过使用多个线程，程序设计人员可以使用不同的线程执行特定的行为，而无需全面的任务循环调度机制，这使得在Internet上实现实时交互变得容易。

6.动态性：Java设计模型意味着其适合不断变化的运行环境。新变量和新示例可以合并到组中，而不会影响用户操作的实现。而且Java支持多重继承，使其比各种严格的具体类型更加灵活和广泛。

## 2.2 Maven简介

1. 历史：Maven，开发者称之为POM（即项目对象模型），可以通过对信息的简短描述来管理涉及项目的开发构建的依赖，以及报告和文档编制的项目管理软件。

Maven不光有构建项目依赖的功能，还提供了管理项目的插件。由于Maven的缺省构建方式被众多Java项目所应用，复用性极强，因此通常使用两行或三行依赖坐标来构建简单的项目。得益于基于Maven项目的方法，许多大型项目都使用它来进行构建，并且在企业项目中使用Maven的比例也在增加。

Maven设计之初，是为了简化Jakarta Turbine项目的开发构建。在几个项目，每个项目包含了不同的Ant构建文件。JAR检查到CVS。Apache组织开发Maven可以建立多个项目，发布项目信息，项目部署，在几个项目中JAR文件提供团队合作和帮助。

Maven的经历了Maven -> Maven2 -> Maven3的发展

2. 国内外现状

国内外都有无数知名项目通过maven构建，并托管在GitHub平台

本系统的项目完全使用Maven构建和管理、部署，本系统使用到的所有依赖均由Maven从远程仓库（阿里巴巴国内镜像）拉取下载。

## 2.3 Spring简介

IOC：本章介绍了控制反转（IoC）原理的Spring框架实现。IoC也称为依赖注入（DI）。在此过程中，对象仅通过构造函数参数，工厂方法的参数或在构造或从工厂方法返回后在对象实例上设置的属性来定义其依赖项（即与它们一起使用的其他对象）。此过程从根本上讲是通过使用类的直接构造或诸如服务定位器模式之类的机制来控制其依赖项的实例化或位置的bean本身的逆过程（因此称为Control Inversion）。

ApplicationContextSpring提供了该接口的几种实现。在独立的应用程序中，通常会创建一个实例，代表着SpringApplicationContext（Spring应用上下文），一般是ClassPathXmlApplicationContext或者FileSystemXmlApplicationContext，开发中可以将项目的各个组件实例托管到Spring容器中，并可以随时通过SpringApplicationContext（Spring应用上下文）来管理，获取，或者操作他们，这些被Spring管理的组件实例就称之为Bean。尽管XML是定义配置元数据的传统格式，但是您可以通过提供少量XML配置来声明性地启用对这些其他元数据格式的支持，从而指示容器将Java注释或代码用作元数据格式。

在Web应用程序场景中，开发者一般会使用隐式创建Spring应用上下文的方式。通常使用应用程序文件中的简单八行（约）样板Web描述符XML web.xml通常就足够了。如果使用Spring Tool Suite（基于Eclipse的开发环境），则只需单击几次鼠标或击键即可轻松创建此样板配置。让您的应用程序类与配置元数据结合在一起，因此，在ApplicationContext创建和初始化之后，您将拥有一个完全配置且可执行的系统或应用程序。

AOP：面向方面的编程（AOP）通过提供另一种思考程序结构的方式来补充面向对象的编程（OOP）。OOP中模块化的关键单元是类，而在AOP中模块化是方面。方面使关注点（例如事务管理）的模块化跨越了多个类型和对象。（这种关注在AOP文献中通常被称为“跨领域”关注。）Spring的关键组件之一是AOP框架。虽然Spring IoC容器不依赖于AOP（这意味着您不需要使用AOP），但AOP是对Spring IoC的补充，以提供功能非常强大的中间件解决方案。具有AspectJ切入点的Spring AOPSpring通过使用基于模式的方法或@AspectJ批注样式，]提供了编写自定义方面的简单而强大的方法 。这两种样式都提供了完全类型化的建议，并使用了AspectJ切入点语言，同时仍然使用Spring AOP进行编织。本章讨论基于架构和基于@AspectJ的AOP支持。AOP在Spring Framework中的只要功能是在面向对象的程序设计中提供一种非侵入式的声明环绕增强，比如日志管理，性能统计，声明式事务等等。可以让用户自定义切点和切入方法。

## 2.4 MyBatis简介

MyBatis 是一流的持久性框架，支持自定义SQL语句，存储过程和ORM映射。MyBatis 封装了大量的 JDBC 代码，包括手动参数设置和结果集获取并把这些结果集封装成一个方便传递，使用，和操作的简单Java对象，即一个实体类对象。相比于其他ORM框架，MyBatis更为简单，而且相比于全自动的方式，这种半自动开发方式更加友好，方便开发者进行调试，查看绑定的SQL语句并优化，MyBatis可以使用简单的XML文件或注解进行配置，并将映射接口（MyBatis提供了面向接口的编程方式，只需扫描Mapper接口，即可自动生成代理，去除了大量重复和繁琐的数据层调用代码，大大简化了开发流程）和Java POJO映射到数据库记录。

MyBatis SQL映射器框架使将关系数据库与面向对象的应用程序一起使用变得更加容易。MyBatis使用XML描述符或注释将对象与存储过程或SQL语句耦合。相对于对象关系映射工具，简单性是MyBatis数据映射器的最大优势。

一般开发者使用以下方式来准备使用Mybatis：快速开始：要使用 MyBatis， 只需将心目中MyBatis所依赖的jar包文件置于类路径中即可。如果使用 Maven 来构建项目，则需将MyBatis官网提供的 dependency 代码（Maven中央仓库的坐标）置于 pom.xml 文件中，并在类路径下添加配置文件即可。

作为一个以数据为主题的微博系统，持久层的框架选择显得尤为重要，MyBatis最为使用最为广泛的ORM开源框架之一，半自动的SQL查询语句的配置和管理方式深受开发者喜爱，目前仍有无数企业的Java后台项目使用SpringBoot整合Spring、SpringMVC，以及MyBatis的方式作为基础来构建。

## 2.5 Tomcat简介

Apache Tomcat 是一个符合Servlet标准的，并且支持了Servlet，JSP，Java EL表达式和WebSocket技术的一个开源Web容器。它是Apache旗下的一个开源的轻量级Web应用服务器，由java语言编写，深受Java后端开发者的了解和喜爱，多用于轻量级访问量较小的个人系统，比如本微博系统，是Java Web开发初学者的必须了解的开源Web容器。

Tomcat主要组件：服务器Server，服务Service，连接器Connector、容器Container。连接器Connector和容器Container是Tomcat的核心。一个Container容器和一个或多个Connector组合在一起，加上其他一些支持的组件共同组成一个Service服务，有了Service服务便可以对外提供能力了，但是Service服务的生存需要一个环境，这个环境便是Server，Server组件为Service服务的正常使用提供了生存环境，Server组件可以同时管理一个或多个Service服务。

## 2.6 WebSocket简介

HTTP流是一种push机制，是不间断的HTTP连续相应，目前许多网站使用了 Ajax 轮询技术，为了是提高传输效率和减少页面的刷新次数，而传统的HTTP请求与相应技术，往往是一个请求对应一个相应，大部分Web服务器在处理完请求后终止它去处理其他请求，WebSocket是单独的一个协议，它基于TCP，保证了可靠的面向流和连接的连接的可靠性。这样一来，客户端和服务端只需要进行一个握手的动作，握手完成后，它们就建立了一个基于传输流的管道，双方可以方便地进行多次通信而连接不会断开，所以短时间内如果浏览器和用户需要多次相互传输信息，websocket会比http更加节省性能和时间，只要体现在握手方面。

客户端浏览器通常通过从向服务器获取HTTP的请求开始来创建WebSocket链接，这与包含附加原始信息的常规HTTP请求不同，客户端和服务器完成建立websocket连接，只要没有客户端或服务器的主动方从一侧或另一侧关闭连接，它们就可以自由地进行使用此链接，以发送消息，并且此连接将一直持续到为止，所以http面向请求和响应，而websocket面向管道或者流。

在本系统中，前台利用JavaScript与后端建立websocket连接，后台处理好多个客户端同时连接时的并发问题，这样就可以达到前后端（即客户端和服务端）的数据实时传输的效果，后台（服务端）的处理逻辑为：收到任意一个客户端的消息时，就将此消息广播给所有其他当前处于连接状态的客户端，这样即可简单实现一个网页多人实时聊天室的效果。

## 2.7 JavaWeb简介

JavaWeb是由JSP和Servlet两个技术组成的，所谓的Web端的开发，实际上就是JSP和Servlet的结合。在JSP和Servlet中，JSP负责实现前端的功能，而Servlet负责实现后台处理服务器的功能。在Web项目开发的过程中，无论选择任意框架结构，都需要依靠这两种技术的相互呼应和结合才能完成Web应用的开发。

Web普遍有三层架构，分别为视图（View）、控制器（Controller）和模型

（Model），也称界面层（UserInterfacelayer）、业务逻辑层（BusinessLogicLayer）、数据访问层（Dataaccesslayer）。通过高内聚低耦合的思想，把Web区分层次。分层式结构是最为常见且重要的软件体系架构。

JavaWeb技术简单来说就是用Java编程语言来解决Web中的问题，Web的特点是服务器与客户端分离，架构清晰，层次分明。基于这个特点，非常适合采用Java语言的进行Web的开发设计。

Java语言创世于1995年，是Sun公司研究院院士詹姆斯格林亲手设计的编程语言。Java语言将面向对象的思想发挥到了极致，允许程序员使用现实中的思路来进行编程思想的设计。与传统的面向过程的开发有着本质思维上的区别，其主要特点在于良好的封装性和继承性，在程序设计中，不断的提高各个模块的封装性或者独立性，减少各个模块之间的耦合性。这样的编程思想使得Java语言的应用程序可维护性和可拓展性大大提高。符合云存储系统中频繁的增减设备，设备互联等需求。

国内外各种云存储业务，很大程度上是依赖互联网来实现，这其中不可避免

的要使用很多Web方面的技术。Web技术当中的编程技在经历长期的发展演化后，与Java的编程思想产生了良好的化学反应。Java当中也专门的产生了很多Web方面的开发工具、开发思想以及开发脚本语言。开发工具如Tomcat，MySQL等，开发思想如Java中鼎鼎大名的MVC模式，脚本语言最著名是JSP和JavaScript。

JavaBean是Java技术中对数据进行连接的数据结构，通过JavaBean技术可以对数据进行连续的访问以及管理，对数据的访问提供相应的数据连接以及数据访问的控件，通过Javabean技术可以对数据的关联关系以及数据的提供方进行管理，有效的对系统的访问方式以及访问流程进行了梳理。在JavaBean技术中还包含了数据库的链接JDBC来对数据进行访问以及数据的管理，通过JDBC的数据连接以及数据的访问都是通过JDBC来进行数据连接和数据访问的。

JavaBean是使用Java语言描述的、易用的、与平台无关的软件组件模型，用于设计可重用的组件，有点类似于Microsoft的COM组件概念。Java模型允许用户通过JavaBean无上限的拓展Java程序的功能，并通过JavaBean的组合快速生成新的应用程序。它易于维护，并允许代码重用。这通常在JSP程序中用于关闭事务逻辑，数据库操作等。有助于分离前台的业务逻辑和程序（例如JSP文件），并且在封装逻辑和维护程序上也有很大优势，从而提高了系统的健壮性与灵活性。

实际上，JavaBean一就是一种使用符合某些命名方法和设计规范的Java类

JavaBean具有以下特性：

(1)JavaBean类必须有一个没有参数的构造函数，这个构造函数在使用<jsp:useBean>实例化JavaBean类时调用。如果JavaBean类内没有任何构造函数，系统会自动生成一个没有参数的构造函数。

(2)JavaBean内的属性都应该定义为私有的，那么这些属性只能通过JavaBean内的方法访问或者改变，这样可以较好的保护数据的完整性和封装性。

(3)属性通过setXxx和getXxx来操作。同时拥有setXxx()和getXxx()方法的可以使之具有读写属性，只有getXxx()方法使之具有只读属性，只有setXxx0方法使之具有只写属性。

## 2.8 AJAX简介

AJAX---Asynchronous JavaScript and XML,是一种无需编译的脚本语言，并没有加入新的技术，也不是单独的 WEB 服务模型或插件。它通过远程加载脚本来获取 JS， CSS 以及 DOM。与动态 HTML 类似，是一种使用现有的技术来实现的算法。AJAX 允许用户在与 WEB 网页交互的时候在后台与 WEB 服务器建立连接，这些连接可以被异步的创建，这样可以使得用户无需等待服务器的回应，继续进行其他操作，这些请求动作通常用 JS 代码编写，由浏览器在执行 JS 后进行处理。XML 常用于在客户端（浏览器）和服务器端之间传递数据。对于用户来说，不用完全加载整个网页而看到触发某事件的效果是很有用的，比如当用户在使用邮件时在输入框中输入邮件地址，可以通过 AJAX 在后台与 WEB 服务器建立一条连接用来检测此用户名是否存在，服务器可以返回相关信息并在页面上展现是否存在的说明。

AJAX 技术的出现带来了很多优势，比如在点击提交按钮的时候不用重新加载整个页面是很有必要的，这样可以提高用户与服务器之间交互的速度和效率，每一次的请求响应可以按照不同的请求展现不同的结果，用户执行某一事件即可获取更多相关信息。AJAX 不是一种解决所有 WEB 问题的通用方法，但是它在为用户提供友好应用方面确实是实用必要的方法，在服务端与用户进行交互的同时，也能够同时与数据库建立连接或者执行脚本。

AJAX 允许浏览器后台建立多条连接，大体上分为两种：同步和异步。出于交互性以及实时性考虑，通常应用异步请求，这样浏览器不必等待脚本的执行，页面也就无需完全重新加载。对于支持AJAX的浏览器，是通过内置的 AJAX 引擎实现异步发送和接受请求。

AJAX配合JavaScript支持无需刷新页面即可将HTTP请求对相应信息显示在网页上，因此AJAX经常在网站中被用于动态更新页面信息，在本系统中，AJAX主要用于在多人聊天功能模块实时更新聊天室的当前在线人数。

## 2.9 开发环境

1.硬件：

MacBook Pro (13-inch, 2018)

系统macOS Catalina 10.15.3

CPU 2.3 GHz 四核Intel Core i5

内存 8 GB 2133 MHz LPDDR3

磁盘 Macintosh HD

腾讯云主机 (标准型S2 1Mbps)

系统 Ubuntu 16.04.1 LTS

CPU 1核

内存 2GB

磁盘 高性能云硬盘。

2.软件：

JDK 1.8.0\_231

Tomcat 8.5.50

MySQL 5.7

IntelliJ IDEA 2019.3.2

Navicat Premium 15.0.8

Spring 4.2.23

MyBatis 3.3.0

# 3 需求分析

此微博系统将涉及和储存大量用户的个人信息，而且系统流程较为复杂，所以安全性显得尤为重要。尤其是用户名，密码，博客内容等私密信息，所以本系统必须通过强大的权限管理以及授权和认证来保证用户的隐私安全。对敏感信息，比如用户名和密码等进行加密传输，重要信息，比如博客内容等进行备份，防止丢失。

## 3.1 性能需求分析

为了保证此微博系统的高性能，是十分重要的，良好且高可用的性能是一个优秀的微博系统不可或缺的条件，试想一个拥有众多用户的社交系统，怎么可能会在使用时让用户感受到明显的卡顿和延迟呢。对于性能一词，人们往往觉得它过于抽象，而延迟和吞吐量，则是性能的具体表现。

Latency，中文译作延迟。Throughput，中文译作吞吐量。它们是衡量软件系统的最常见的两个指标。延迟一般包括单向延迟（One-way Latency）和往返延迟（Round Trip Latency），实际测量时一般取往返延迟。它的单位一般是ms、s、min、h等。

吞吐量一般指相当一段时间内测量出来的系统单位时间处理的任务数或事务数（TPS）。注意“相当一段时间”，不是几秒，而可能是十几分钟、半个小时、一天、几周甚至几月。它的单位一般是TPS、每单位时间写入磁盘的字节数等。

此微博系统中为了保证良好的性能，采用了包含但不限于以下方案。

1. 单例：Spring的Bean默认的作用域就是单例，使用该属性定义Bean时，IOC容器仅创建一个Bean实例，无论多少次请求获取该Bean，IOC容器每次返回的是同一个Bean实例。
2. WebSocket + 异步AJAX：WebSocket不依赖于HTTP协议，因此建立连接以后不需要一个Response对应一个Request，而是直接依赖于TCP协议解决客户端发起多个http请求到服务器资源浏览器必须要经过长时间的轮训问题而生的，他实现了多路复用，他是全双工通信。在WebSocket协议下客服端和浏览器可以同时发送信息。建立了WenSocket之后服务器不必在浏览器发送request请求之后才能发送信息到浏览器。这时的服务器已有主动权想什么时候发就可以发送信息到服务器。而且信息当中不必在带有head的部分信息了与http的长链接通信来说，这种方式，不仅能降低服务器的压力。而且信息当中也减少了部分多余的信息。
3. 并发容器：在多人聊天室设计中，为了保证并发性能，采用了JDK java.util.concurrent包下的并发容器ConcurrentHashMap，和保证原子性的int包装类AtomicInteger。ConcurrentHashMap在JDK1.8中其中抛弃了原有的Segment 分段锁，而采用了 CAS + synchronized 来保证并发安全性，有效地提升了性能。而AtomicInteger原子操作类利用了CAS在保证线程安全的同时，大大减少了线程因资源竞争而造成的性能消耗。
4. MyBatis缓存：MyBatis的一级缓存的命中策略是：同一个 SqlSession 对象， 在参数和 SQL 完全一样的情况先，只执行一次 SQL 语句（如果缓存没有过期）在同一个 SqlSession 中, Mybatis 会把执行的方法和参数通过算法生成缓存的键值，将键值和结果存放在一个 Map 中，如果后续的键值一样，则直接从 Map 中获取数据；不同的 SqlSession 之间的缓存是相互隔离的；用一个 SqlSession，可以通过配置使得在查询前清空缓存；任何的 UPDATE, INSERT, DELETE 语句都会清空缓存。

## 3.2 技术可行性分析

目前越来越多的Java项目都使用SSM作为基础框架。

此微博系统主要包含以下功能模块

1. 主要功能：发布微博、浏览微博、个人空间、粉丝关注、多人聊天

2. 用户登录功能：除了通过注册来添加用户，管理员也可以添加用户。登录后的用户可以进入并使用本系统。

3. 修改密码功能：用户和管理员都可以修改密码。

4. 粉丝关注功能：

用户可以关注自己喜欢的博主，并能够在关注动态看到他们新发的微博。

用户还能在关注列表里看到自己关注的所有博主，也可以随时取消关注。

博主也可以在个人空间看到有谁关注了自己。

5. 用户管理功能：

删除用户: 管理员可以删除普通用户。

禁言用户: 使普通用户（永久或期限）禁止在本系统发布任何内容。

查询用户: 通过用户名或ID查找用户，并查看他们的所有信息。

删除博客: 管理员可以删除任何微博。

管理员除了拥有以下功能以外，同时拥有普通用户的所有功能。

6. 微博系统功能：

发布微博: 用户登录后随时可以发布微博。

查看微博: 用户可以在首页看到微博，也可以在关注动态看到关注的博主发的微博。

个人空间: 用户可以进入自己或者别人的个人看见，并查看他的信息。

编辑微博: 用户只能编辑自己已经发出的微博。

删除微博: 普通用户只能删除自己发的微博，而管理员能删除任何微博。

7. 用户评论功能：

用户可以在博客下面发表评论，修改或删除自己发的评论。

8. 多人聊天功能：

用户可以在多人聊天室参与聊天，所有进入聊天室的用户都可以实时看到聊天内容。（聊天室属于公开场合，禁言不会影响用户进入聊天室和参与多人聊天）

9. 站内搜索功能：

用户可以通过关键词索用户或者微博内容。

本系统各个功能模块之间的关系如下图所示。

图3-1 功能模块关系图

## 3.3 网络原理图

本系统的网络原理图如图所示

用户

局域网或Internet

微博系统

系统服务器

后台数据库

图3-2 系统的网络原理图

## 3.4 用户角色需求分析

在本系统中的所有用户，一共分为两种角色：平台用户和管理员用户。其中，普通用户有着最基本的系统使用权限，包括注册登录、修改密码、发布博客、查看博客、删除博客、编辑博客、进入个人主页、在博客下面发布评论、编辑评论、删除评论以及进入多人聊天室和搜索站内信息。

管理员用户首先拥有普通用户的所有权限，除此之外，管理员用户还拥有系统管理权限，具体包括：管理员可以删除任何用户发布端博客，也可以删除任何用户发布的评论。除此之外，管理员用户还拥有查看用户、删除用户、恢复用户、禁言用户等高级权限，禁言的时候管理员可以选择禁言时间是永久还是期限。

## 3.5 功能模块需求分析

### 3.5.1 注册登录模块

所以用户只有在本系统中注册完一个专有账户，并且成功登录后，才可以正常进入并使用本系统。不管是登录还是注册，系统都会收集用户输入的用户名和密码，读取到这些信息后再进行逻辑判断，进行接下来的操作。

注册用户时，如果出现与原有用户重名的情况，或者用户名和密码中有任意一个不符合规范，则前台对应样式会相应地变红色，警告用户。

用户尝试登录时，如果用户名和密码中有任意一个不符合规范，或者出现密码错误的情况，前台对应样式也会相应地变红色，达到警告用户的目的。

而且用户登录成功后，上一位登录的用户残留信息（搜索记录，用户信息等）必须被删除，保证了用户信息的隐私安全。

图3-3 登录注册模块用例图

### 3.5.2 修改密码模块

本系统的所有用户都可以自行更换密码，为了保证安全性，用户在尝试修改密码时必须输入正确的原来的密码才可以修改成功。

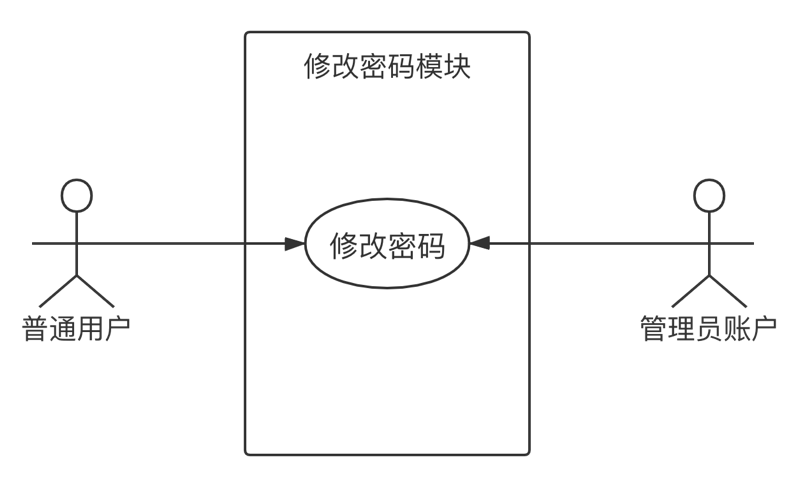
用户只有在登录状态下，才可以在自己的个人空间中修改自己的登录密码，如果验证的时候出现新指定登录密码不符合系统要求的情况（包含空白字符），也会相应地向用户发出警告。

图3-4 修改密码模块用例图

### 3.5.3 博客系统模块

1.发布博客：用户在成功登录本系统以后，可以在博客页面（即首页）正常发布博客，可以包含文字和图片，图片需要被保存在服务端，并做备份。

2.查看博客：其他用户都可以在首页成功看到这条博客（没有看到的需要刷新一下即可），按时间降序排序。当发布博客时，一些HTML相关的关键字符，比如<html>，在这些字符被传到服务端后台时，需要被自动转移义，保障了此微博系统的正常使用以及前端页面的正确渲染。

图3-5 博客系统模块用例图

3.删除博客：普通用户可以成功删除自己发布的博客，而管理员则可以删除所有博客。

4.编辑博客：普通用户可以成功编辑自己发布的博客，而管理员则可以编辑所有博客。而且编辑时可以正确显示和处理Windows系统和Unix系统下不同的回车和换行标识。

图3-5 博客系统模块用例图

### 3.5.4 个人主页模块

个人主页，即用户的个人空间，内有该用户发布的所有博客信息，其他用户可以通过访问此用户的个人空间来查看内容

用户在成功登录本系统以后，可以进入自己（通过点击个人主页或者自己的用户名）或者其他用户（通过点击此用户的用户名）的个人主页，并查看此人的关注数量和粉丝数量，以及他的所有博客内容，按时间降序排序。

### 3.5.5 粉丝关注模块

关注是一种用户与用户之间的关系。用户可以在其他用户的个人主页页面选择关注或者取消关注此人。在关注动态页面还可以看到所有关注的用户最近发布的所有博客。用户还可以在自己或者其他用户的个人主页中查看此人的关注数量和粉丝数量，点击关注数量后会进入关注列表，会显示此用户关注的所有用户，点击用户名还可以进入他的个人空间，点击粉丝数量后会进入粉丝列表，会显示关注此用户的所有用户，按时间降序排序，点击用户名还可以进入他的个人空间。

图3-6 粉丝关注模块用例图

### 3.5.6 用户评论模块

发布评论：用户可以在正常在自己或者其他用户发布的博客下面添加评论。

查看评论：所有用户都可以看到已发布的评论。

编辑评论：用户可以正常编辑自己发布发布的评论，而无法修改别人的评论。

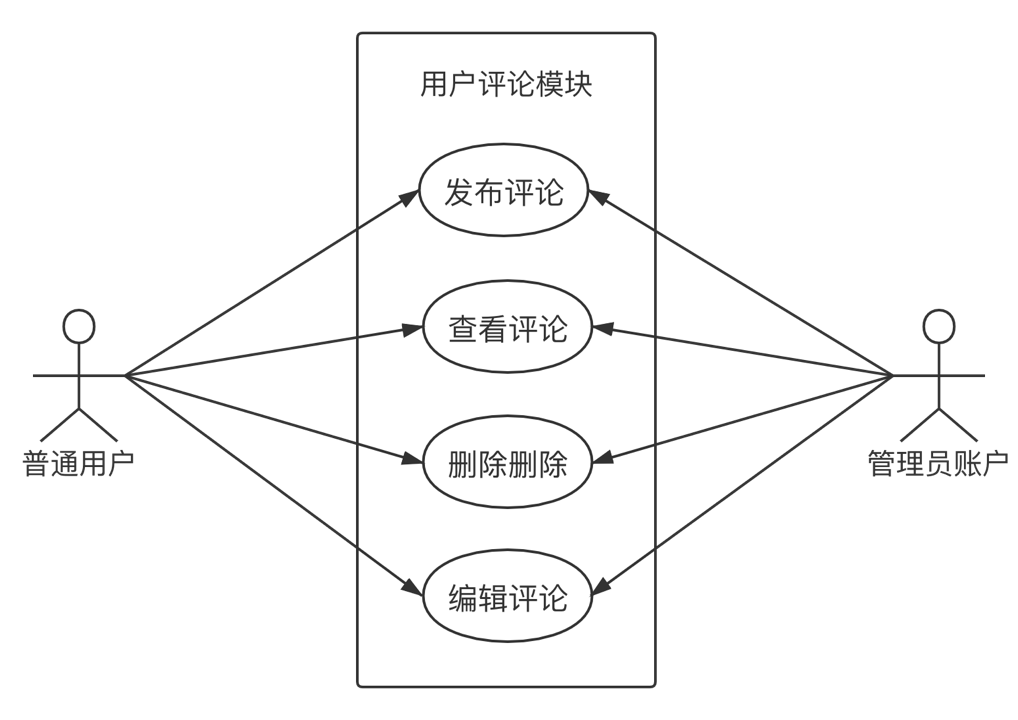
删除评论：用户可以正常删除自己发布的评论，或者自己发布的博客下面的评论。而管理员却可以删除任何评论。

图3-7 用户评论模块用例图

### 3.5.7 站内搜索模块

站内搜索模块的作用是为了方便因为用户可以正常搜索用户名，也可以正常搜索出博客内容，按时间降序排序。而且用户的搜索记录会被保留，直到用户退出登录或者有其他用户在这台电脑上登录。

### 3.5.8 多人聊天模块

用户进入聊天室和离开聊天室提示支持。

系统提供一个多人聊天时，支持所有用户在多人聊天室中实时发送信息和接收信息，任何用户发出的每一条信息都会被所有用户接收到。

人数同步显示正常。

用户发送消息和消息显示正常。

大量用户同时发送消息的高并发场景下，聊天室系统仍能正常工作。

### 3.5.9 后台管理模块

后台管理模块的存在是为了在管理员界面显示系统中所有用户的一个列表视图，让管理员用户可以更加方便地管理用户，而不用在系统的主页面中查看、筛选、查找所有用户信息，这样一来，管理员用户可以更好地行使禁言、删除用户等管理员特权，而且这个功能模块做到了权限的划分，它对于普通用户来说是完全不可见的。

管理员可以在后台管理界面正常看到所有用户的账号消息，按用户ID升序排序。管理员可以在此界面通过用户ID或者用户名来搜索用户，并且可以随时停用（删除）任意一个用户的账号，也可以永久或者期限禁言一个用户，在这段时间内，如果被禁言的用户尝试发布博客或者评论，系统会拦截用户的操作抛出异常，前台接收到这一信息警告用户被禁言，并告诉用户解禁时间，但是禁言不会影响用户在多人聊天时发言。

除此以外，管理员还拥有比普通用户更高级的权限，可以编辑和删除任何博客，也可以删除任何评论。

图3-8 用户管理模块用例图

# 4 系统设计

## 4.1 数据库设计

### 4.1.1 用户

属性：

用户ID：user\_id

用户名：user\_name

用户密码：user\_password

禁言截止日期：user\_date

启用状态：status

权限：admin

禁言状态：muted

图4-1 用户实体关系图

表4-1 用户信息表(user)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 名称 | 字段名称 | 数据类型 | 主键 | 非空 | 备注 |
| 用户ID | user\_id | int | yes | yes | 自增序列 |
| 用户名 | user\_name | varchar(30) | no | yes |  |
| 用户密码 | user\_password | varchar(30) | no | yes |  |
| 禁言截止日期 | user\_date | datetime | no | yes |  |
| 启用状态 | status | int | no | yes | 枚举 |
| 权限 | admin | int | no | yes | 枚举 |
| 禁言状态 | muted | int | no | yes | 枚举 |

### 4.1.2 博客

属性：

博客ID：blog\_id

博客内容：blog\_content

发布时间：blog\_time

发布者用户ID：user\_id

图片文件名：filenames

状态：status

图4-2 博客实体关系图

表4-2 博客信息表(user)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 名称 | 字段名称 | 数据类型 | 主键 | 非空 | 备注 |
| 博客ID | blog\_id | int | yes | yes | 自增序列 |
| 博客内容 | blog\_content | varchar(256) | no | yes |  |
| 发布时间 | blog\_time | datetime | no | yes |  |
| 发布者用户ID | user\_id | int | no | yes | 外键 |
| 图片文件名 | filenames | varchar(256) | no | No |  |
| 状态 | status | int | no | yes | 枚举 |

### 4.1.3 评论

属性：

评论ID：comment\_id

所属博客ID：blog\_id

评论者用户ID：user\_id

评论内容：comment\_content

发布评论时间：comment\_time

状态：status

图4-3 评论实体关系图

表4-3 评论信息表(user)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 名称 | 字段名称 | 数据类型 | 主键 | 非空 | 备注 |
| 评论ID | comment\_id | int | yes | yes | 自增序列 |
| 所属博客ID | blog\_id | int | no | yes | 外键 |
| 评论者用户ID | user\_id | int | no | yes | 外键 |
| 评论内容 | comment\_content | varchar(256) | no | yes |  |
| 发布评论时间 | comment\_time | datetime | no | yes |  |
| 状态 | status | int | no | yes | 枚举 |

### 4.1.4 关注

属性：

关注ID：attention\_id

博主用户ID：blogger\_id

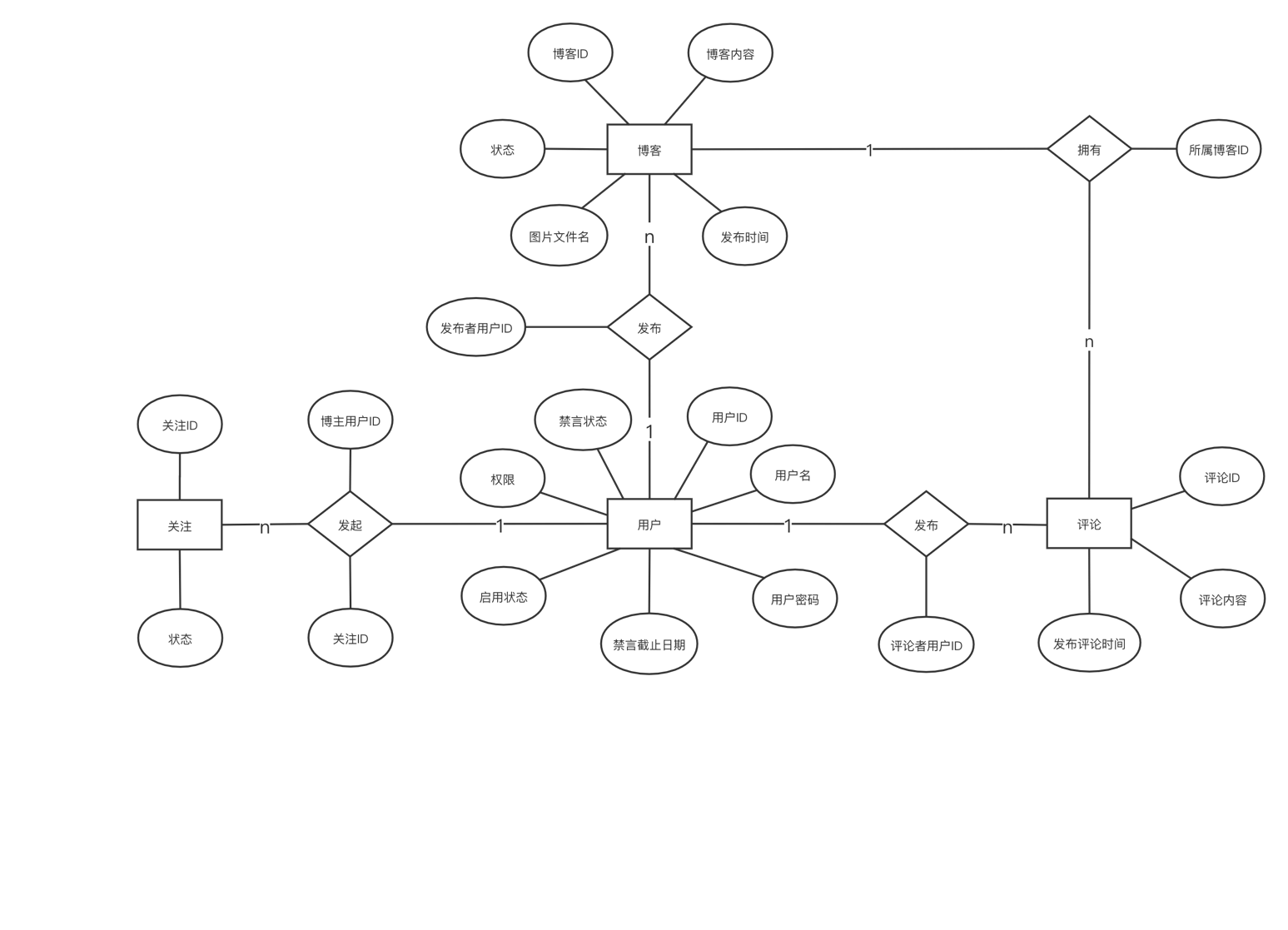
粉丝用户ID：fans\_id

状态：status

表设计：



图4-4 关注实体关系图

****表4-4 关注信息表(user)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 名称 | 字段名称 | 数据类型 | 主键 | 非空 | 备注 |
| 关注ID | attention\_id | int | yes | yes | 自增序列 |
| 博主用户ID | blogger\_id | int | no | yes | 外键 |
| 粉丝用户ID | fans\_id | int | no | yes | 外键 |
| 状态 | status | int | no | yes | 枚举 |

图4-5 总体功能实体关系图

## 4.2 功能模块设计

### 4.2.1 用户登录

用户请求登录时，后台验证账户和密码参数，它们都不为空，且验证正确是才回成功登录，清除上一位用户的搜索记录等残留信息，并在会话（HttpSession）中添加登录的用户信息，以便接下来使用和防止被登录拦截器拦截，如果登录失败，则重定向到登录页面，传递错误信息。

表4-1 用户登录模块HIPO表

|  |  |  |  |
| --- | --- | --- | --- |
| 功能 | 输入项 | 处 理 | 输出项 |
| 用户登录 | 用户名和密码。 | 读取用户名和密码，与后台数据库匹配。 | 如果匹配成功，则进入网站博客页面，匹配失败则停留在登录页面，并提示错误。 |

图 4-6 用户登录流程图

### 4.2.2 修改密码

用户在已经登录的状态下可以在个人空间修改自己的密码，修改之前会校验密码是否符合规范（任意长度的非空字符串），不符合会抛出异常，警告用户。

表4-2 修改密码模块HIPO表

|  |  |  |  |
| --- | --- | --- | --- |
| 功能 | 输入项 | 处 理 | 输出项 |
| 用户登录 | 用户在登录状态下，输入新密码。 | 后台用新密码替换掉原始密码。 | 提示修改成功。 |

### 4.2.3 粉丝关注

用户可以关注自己喜欢的博主，并能够在关注动态看到他们新发的微博。

用户还能在关注列表里看到自己关注的所有博主，也可以随时取消关注。

博主也可以在个人空间看到有谁关注了自己。

表4-3 粉丝关注模块HIPO表

|  |  |  |  |
| --- | --- | --- | --- |
| 功能 | 输入项 | 处 理 | 输出项 |
| 添加关注 | 用户在登录状态下，输入添加的用户ID。 | 后台为用户关注指定用户。 | 提示添加关注成功。 |
| 取消关注 | 用户在登录状态下，输入取消的用户ID。 | 后台为用户取消关注指定用户。 | 提示取消关注成功。 |
| 查看关注 | 用户在登录状态下，查看关注。 | 后台查询出所有关注的用户 | 输出关注的所有用户。 |

### 4.2.4 用户管理

删除用户: 管理员可以删除普通用户。

禁言用户: 使普通用户（永久或期限）禁止在本系统发布任何内容。

查询用户: 通过用户名或ID查找用户，并查看他们的所有信息。

删除博客: 管理员可以删除任何微博。

管理员除了拥有以下功能以外，同时拥有普通用户的所有功能。

非管理员用户尝试符合/admin/\*\*的请求是，会被AdminFilter拦截，重定向到主页面，做到权限校验拦截器的功能。

用户请求登录时，后台会在会话（HttpSession）中添加登录的用户信息时，加入用户的管理员权限信息，以便接下来使用和被登录拦截器验证通过或者拦截，如果验证失败，则重定向到主页面。

### 4.2.5 微博系统

发布微博: 用户登录后随时可以发布微博（限制140字，9张图片）。

1. 从会话（Http Session）中获取当前用户信息，包括用户ID、用户名。

2. 为传过来的博客实体添加图片文件信息（如果用户发了图片的话）。

3. MyBatis自动生成的SQL语句，并执行。

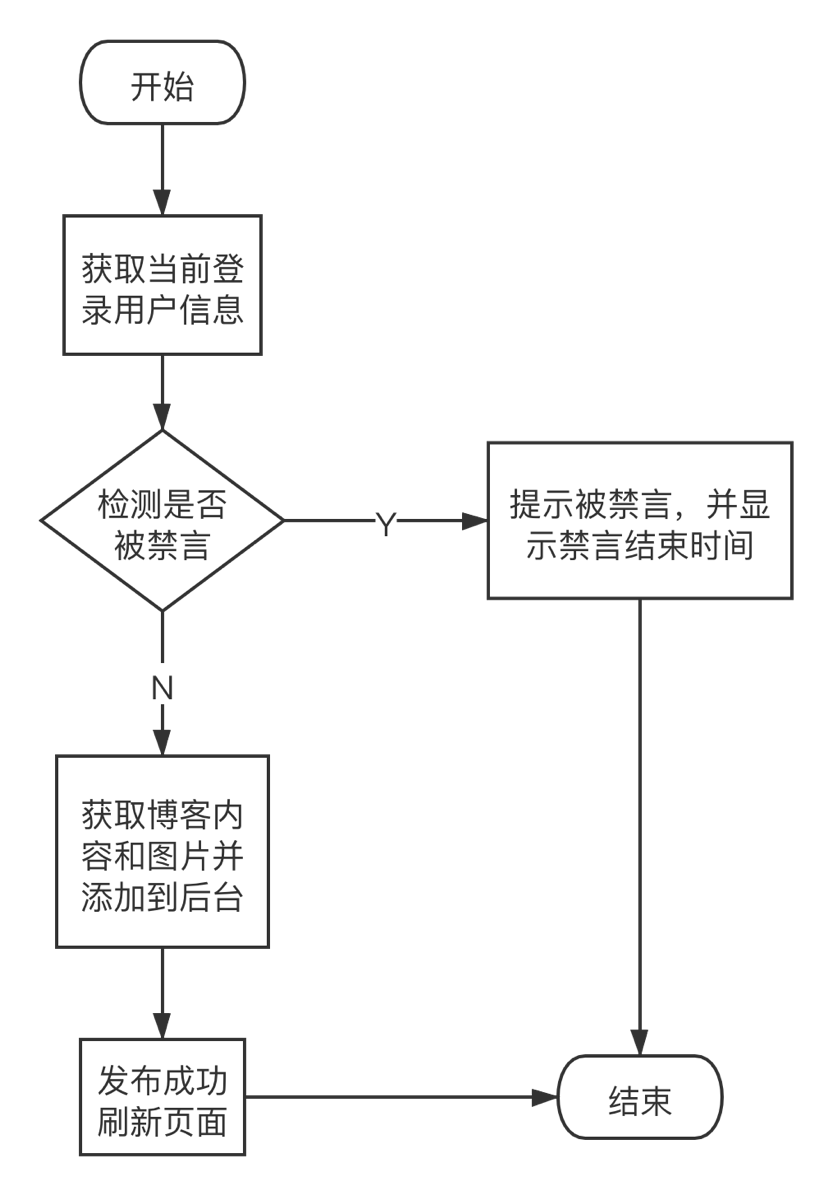
4. 重定向到博客浏览页面，用户就能看到自己刚刚发布的博客了。

图 4-7 发布博客流程图

编辑微博: 用户只能编辑自己已经发出的微博。

1. 获取当前登录的用户。

2. 通过传过来的参数：博客ID确定尝试修改的博客。

3. 获取该博客发布者的用户。

4. 确定当前登录的用户是否有权限更改此博客（当前登录的用户就是发布者，或者权限高于发布者则被判断为有权限）。

5. 无论是否修改成功，都重定向到此博客的详情页面。

删除微博: 普通用户只能删除自己发的微博，而管理员能删除任何微博。

1. 获取当前登录的用户。

2. 通过传过来的参数：博客ID确定尝试的博客。

3. 获取该博客发布者的用户。

4. 确定当前登录的用户是否有权限删除此博客（当前登录的用户就是发布者，或者权限高于发布者则被判断为有权限）。

5. 无论是否删除成功，都重定向到此博客的详情页面。

图 4-8 删除博客流程图

### 4.2.6 个人主页

个人主页: 用户可以查看其他用户的个人主页，会显示他的所有博客。通过参数中的用户ID，查找某个用户的博客，如果该用户不存在，则抛出异常。通过页号，查询出一部分信息，并返回，如果未指定页号则默认第一页。

### 4.2.7 评论系统

普通用户都可以在博客下面发表评论，修改或删除自己发的评论。管理员用户由于拥有额外权限，可以删除任何博客下面的任何评论。

### 4.2.8 多人聊天

用户可以在多人聊天室参与聊天，所有进入聊天室的用户都可以实时看到聊天内容。

利用了java.util.concurrent包下的并发容器ConcurrentHashMap，和保证原子性的int包装类AtomicInteger，每有一个用户连接，则在在线用户集合：clients:ConcurrentHashMap<String, WebSocket>中添加一个用户信息，包含了用户名与会话。同时同步更新在线人数：onlineCount:AtomicInteger

当用户发送消息时，服务端会接收到消息，并转发给当前登录的所有用户，期间对每个用户单独加锁，既保证了每个用户的会话的同步，又保证了并发性能。用户断开连接时，在线用户集合：clients:ConcurrentHashMap<String, WebSocket>和在线人数：onlineCount:AtomicInteger也会同步更新。

为了保证系统的高可用，在某一个连接抛出异常是，也会对它做注销操作，而其他连接不会受任何影响，与此同时，系统记录了用户的登录状态，当某一用户出现重复登录的状况时，系统能正确给用户发出警告，并强制下线上次一次的登录。

为了保证服务端可以拿到用户名，需要在握手时获取HTTP会话：HttpSession（不是同一个会话），并把HTTP会话保存到WebSocket上下文的用户配置参数 UserProperties中，以便后续获取。

### 4.2.9 站内搜索

所有用户都可以通过关键词索用户和微博内容，方便在系统内查找相应的信息。

用HTTP会话（HttpSessiom）保存用户的搜索记录，以便用户下次再进入搜索页面时，可以重现上一次搜索的结果

# 5 系统实现

## 5.1 系统总体架构及实现

1. 设计maven的模块结构和依赖关系

项目分为三个模块：

blog-model 负责数据、实体、模型

blog-service 封装可复用的、高可用的业务操作

blog-web 负责解析视图，进行页面的渲染（SpringMVC直接与这一层交互）

2. 完成 SpringMVC（基于注解）的搭建

本项目基于 Web 3.0 抛弃了基于繁琐的 web.xml 的形式的旧版，使用更加方便，快捷，现代 SpringMVC` 类库中提供的现有类来配置 DispacherServlet

扩展AbstractAnnotationConfigDispatcherServletInitializer类会自动配置DispatcherServlet和Spring应用上下文

/\*\*

\* 继承AbstractAnnotationConfigDispatcherServletInitializer来初始化WEB

\* 扩展AbstractAnnotationConfigDispatcherServletInitializer的任意类都会自动配置DispatcherServlet和Spring应用上下文

\*/

public class BlogWebAppInitializer extends AbstractAnnotationConfigDispatcherServletInitializer {

protected String[] getServletMappings() {

return new String[]{"/"};

}

protected Class<?>[] getRootConfigClasses() {

return new Class<?>[]{RootConfig.class};

}

protected Class<?>[] getServletConfigClasses() {

return new Class<?>[]{WebConfig.class};

}

/\*\*

\* 文件上传的临时路径

\*/

@Override

protected void customizeRegistration(ServletRegistration.Dynamic registration) {

registration.setMultipartConfig(new MultipartConfigElement("/arvin/file/images/"));

}

/\*\*

\* 这项配置使得当handlerMapping（处理器适配）无法找到合适的handler（处理器时）

\* 会抛出异常，方便我们后序的捕获处理

\*/

@Override

protected FrameworkServlet createDispatcherServlet(WebApplicationContext servletAppContext) {

final DispatcherServlet dispatcherServlet = (DispatcherServlet) super.createDispatcherServlet(servletAppContext);

dispatcherServlet.setThrowExceptionIfNoHandlerFound(true);

return dispatcherServlet;

}

}

3. 完成三层架构的搭建

Controller

1. DispacherServlet收到发送的HttpServletRequest，调用HandlerMapping处理器映射器。

2. 处理器映射器帮助DispacherServlet找到对应的处理器，找到后发送给DispacherServlet。

3. DispatcherServlet调用HandlerAdapter处理器适配器。

4. HandlerAdapter经过适配调用具体的处理器(Controller，也叫后端控制器)。

5. Controller执行完成返回ModelAndView。

业务模块流程。我经常喜欢用控制视图的跳转来简单形容，但是这个是不全面的，因为他除了控制视图的转换之外，还控制了业务模块的逻辑，你看到之后，知道它实现了这个业务逻辑，但是怎么实现的，不需要关心，仅仅需要调用service层里的一个方法即可，这样使controller层看起来更加清晰。

Service封装了一系列的可复用的业务操作功能，这种多层调用有效地防止了Controller中代码过多，逻辑过于复杂，大量重复代码导致复用率过低等一系列问题。

## 5.2 数据库持久化实现

1. Mysql的安装（https://dev.mysql.com/downloads/mysql/）

本项目安装了MySQL5.7，这个版本比较稳定，而且安全性方面有所增强，同时InnoDB首次支持了全文索引

2. 设置mysql，最大连接时间，编码格式等

配置MySQL的用户名，这里用的root账户，配置好密码，并设置最长连接时间为8小时。并调整时区为北京时间（与操作系统一致），根据机器性能设置最大并发数量，并把所有编码格式都调整为utf-8mb4（Mysql的utf8编码并不是真正的UTF-8编码，Mysql的utf8最多只支持3个字节，而emoji表情、一些特殊的中文字符则需要4个字节才能存储，因此才会报错。其中UTF-8编码是U+2528D，属于CJK Unified Ideographs Extension B（中日韩统一表意文字扩充B）字符集的字符，处于第二辅助平面（SIP，表意文字补充平面），最多支持4个字节。而Mysql的utf8编码则属于常见的基本多文种平面（BMP，即Unicode编码范围在0000-FFFF之内）的字符，最多支持3个字节。）

3. 设置默认引擎，缓存配置，以及最大并发量

4. 表设计：设计主要表的第一个版本：用户，博客，评论，关注

5. 添加测试数据：手动添加测试数据到数据库

6. 数据库移动到服务器：用MySQL的dump命令导出blog.sql文件，在Linux云主机上导入初始测试数据

7. 构建MyBatis测试：写一个简单的MyBatis测试，测试其久化功能是否正常，性能是否足够，其中MyBatis一级缓存默认打开，由于二级缓存是以namespace为单位的，不同namespace下的操作互不影响。会造成缓存失效的DML（数据操纵语言）操作会刷新所在namespace下的缓存。所以在符合Cache使用时的注意事项的要求时，并没有什么危害。但是其他情况就会有很多危害了：针对一个表的某些操作不在他独立的namespace下进行。更危险的情况是在XXXMapper.xml做了insert,update,delete操作时，会导致UserMapper.xml中的各种操作充满未知和风险，因此关闭MyBatis的二级缓存。

## 5.3 项目部署实现

本项目使用的版本是Tomcat 8.5.50（https://tomcat.apache.org/download-80.cgi）先把Tomcat的目录的读写权限设置好，方便接下来的设置与部署。在conf/server.xml中配置端口号，初始堆大小，最大堆大小，以及自动热部署功能，并把本项目设置为Tomcat的默认应用，最后把本项目编译好的blog.war拷贝到wepapps下，启动Tomcat：startup.sh，Tomcat会启动JVM，自动扫描wepapps下的项目并解压，把类加载到堆内存中。

## 5.4 系统功能实现

### 5.4.1 用户登录

输入网址后自动进入登录页面，未登录用户尝试符合/user/\*\*的请求是，会被UserFilter拦截，重定向到登录页面，做到登录校验拦截器的功能。

用户登录界面如下图所示。

图5-1用户登录界面图

用户登录功能关键代码：

/\*\*

\* 未登录拦截

\*/

@WebFilter(filterName = "userFilter",urlPatterns = "/user/\*")

public class UserFilter implements Filter

{

public void destroy(){}

public void doFilter(ServletRequest req, ServletResponse resp, FilterChain chain) throws ServletException, IOException

{

if (((HttpServletRequest) req).getSession().getAttribute("user") != null)

chain.doFilter(req, resp);

else

((HttpServletResponse)resp).sendRedirect("/");

}

public void init(FilterConfig config){ }

}

/\*\*

\* 登录功能

\*/

@PostMapping({"/", "login.html"})

public String login(@Valid User user, Errors errors, HttpSession session, RedirectAttributes model) {

/\*检查用户名和密码是否为空\*/

if (errors.hasErrors())

return "login";

/\*检查用户名和密码是否正确\*/

User realUser = userService.checkUser(user);

if (realUser != null) {

session.setAttribute("user", realUser);

/\*清除搜索的记录\*/

session.removeAttribute("blogList");

session.removeAttribute("keyword");

session.removeAttribute("page");

return "redirect:user/blogs.html";

}

/\*通知是否有错误\*/

model.addFlashAttribute("msg", "err");

return "redirect:/";

}

用户登录功能关键查询语句：

SELECT user\_id, user\_name, status, admin, user\_date, muted, user\_password

FROM user WHERE user\_name = '施航程' AND user\_password = '123' and status > 0;

### 5.4.2 修改密码

在我的主页界面，点击修改密码按钮，前台会弹出一个输入框，并提示输入新密码，在输入完新密码后点击确认，密码即修改完成，修改密码界面如下图所示。

图5-2 用户修改密码界面图

修改密码功能关键代码：

/\*\*

\* 修改密码

\*/

@PostMapping("user/changePassword")

public String changePassword(HttpSession session, String password) {

if (!Pattern.matches("[\\S]+", password))

throw new IllegalArgumentException("密码格式错误!");

User user = (User) session.getAttribute("user");

user.setPassword(password);

userService.changePassword(user);

return "redirect:/user/page.html";

}

修改密码功能关键查询语句：

UPDATE user SET user\_password = '123456'

WHERE user\_id = 2;

### 5.4.3 粉丝关注

所有用户都可以随时在其他用户的个人主页界面选择关注或者取消关注此人，自己已关注的用户数量和列表以及粉丝的数量和列表都可以在自己的个人主页中查看，也可以通过点击他们的用户名来进入他们的个人主页，在关注动态页面可以查看所有关注的用户发布的博客。粉丝关注界面如下图所示。

图5-3 粉丝关注界面图

粉丝关注功能关键代码：

/\*\*

\* 用户关注相关

\*/

@Controller

public class AttentionController {

@Resource

private UserService userService;

/\*\*

\* 关注

\*/

@PostMapping("user/addAttention/{userId}/{page}")

public String addAttention(HttpSession session, @PathVariable int userId, @PathVariable String page) {

User me = (User) session.getAttribute("user");

int myId = me.getId();

if (myId != userId && !userService.isAttention(me.getId(), userId))

userService.addAttention(myId, userId);

return "redirect:/user/page.html/" + userId + "/" + page;

}

/\*\*

\* 取关

\*/

@PostMapping("user/delAttention/{userId}/{page}")

public String delAttention(HttpSession session, @PathVariable int userId, @PathVariable String page) {

User me = (User) session.getAttribute("user");

userService.delAttention(me.getId(), userId);

return "redirect:/user/page.html/" + userId + "/" + page;

}

/\*\*

\* 关注列表

\*/

@GetMapping("user/attentionList/{userId}")

public String attentionList(@PathVariable int userId, Model model) {

PageInfo<User> listPage = userService.attentionList(userId);

model.addAttribute("listPage", listPage);

model.addAttribute("host", userService.findUserById(userId));

model.addAttribute("msg", "的关注");

return "list";

}

/\*\*

\* 粉丝列表

\*/

@GetMapping("user/fansList/{userId}")

public String fansList(@PathVariable int userId, Model model) {

PageInfo<User> listPage = userService.fansList(userId);

model.addAttribute("listPage", listPage);

model.addAttribute("host", userService.findUserById(userId));

model.addAttribute("msg", "的粉丝");

return "list";

}

}

粉丝关注功能关键查询语句：

<insert id="attention">

INSERT INTO attention(blogger\_id, fans\_id)

VALUES (#{blogger}, #{fans})

</insert>

<delete id="delAttention">

DELETE

FROM attention

WHERE blogger\_id = #{blogger}

AND fans\_id = #{fans}

关键查询语句：

</delete>

<select id="attentionList" resultMap="userMap" parameterType="int">

select u.user\_id, u.user\_name, u.status, u.admin, u.user\_date, u.muted

from user u

inner join attention a on u.user\_id = a.blogger\_id

where a.fans\_id = #{id}

</select>

<select id="fansList" resultMap="userMap">

select u.user\_id, u.user\_name, u.status, u.admin, u.user\_date, u.muted

from user u

inner join attention a on u.user\_id = a.fans\_id

where a.blogger\_id = #{id}

</select>

### 5.4.4 用户管理

用户管理功能是管理员用户的专属功能，管理员用户在登录后，可以从自己的个人主页中点击管理员按钮进入用户管理界面，这个界面方便管理员执行查询用户、删除用户、禁言用户、恢复用户等管理员权限，用户管理修改密码界面如下图所示。

图5-4 用户管理界面图

用户管理功能关键代码：

/\*\*

\* 未登录拦截

\*/

@WebFilter(filterName = "adminFilter",urlPatterns = "/admin/\*")

public class AdminFilter implements Filter

{

public void destroy(){}

public void doFilter(ServletRequest req, ServletResponse resp, FilterChain chain) throws ServletException, IOException

{

User user = (User) ((HttpServletRequest) req).getSession().getAttribute("user");

if (user != null && user.getAdmin() > 0)

chain.doFilter(req, resp);

else

((HttpServletResponse)resp).sendRedirect("/user/blogs.html");

}

public void init(FilterConfig config){ }

}

/\*\*

\* 管理员相关

\*/

@Controller

public class AdminController {

@Resource

private UserService userService;

/\*\*

\* 管理员页面,默认第一页

\*/

@GetMapping("admin/manager.html")

public String admin(Model model, @RequestParam(required = false) String username) {

return admin(model, 1, username);

}

/\*\*

\* 管理员页面,分页

\*/

@GetMapping("admin/manager.html/{page}")

public String admin(Model model, @PathVariable int page, @RequestParam(required = false) String username) {

/\*根据页号获取数据\*/

username = StringUtils.isEmpty(username) ? null : username.trim();

PageInfo<User> userPage = userService.selectAllUser(page, username);

model.addAttribute("userPage", userPage);

return "admin/list";

}

/\*\*

\* 删除用户

\*

\* @param userId userId

\* @param page page

\*/

@PostMapping("admin/delUser/{userId}/{page}")

public String delUser(@PathVariable int userId, @PathVariable String page) {

userService.delUser(userId);

return "redirect:/admin/manager.html/" + page;

}

/\*\*

\* 恢复用户

\*

\* @param userId userId

\* @param pages page

\*/

@PostMapping("admin/reuseUser/{userId}/{pages}")

public String reuseUser(@PathVariable int userId, @PathVariable String pages) {

userService.reuseUser(userId);

return "redirect:/admin/manager.html/" + pages;

}

/\*\*

\* 禁言用户

\*

\* @param userId userId

\* @param days 天数

\* @param pages pages

\*/

@PostMapping("admin/mute/{userId}/{days}/{pages}")

public String mute(@PathVariable int userId, @PathVariable int days, @PathVariable String pages) {

userService.mute(userId, days);

return "redirect:/admin/manager.html/" + pages;

}

}

用户管理功能关键查询语句：

<select id="selectAllUser" resultMap="userMap" parameterType="string">

SELECT user\_id, user\_name, status, admin, user\_password, user\_date, muted

FROM user

<where>

<if test="\_parameter != null">

user\_id = #{key}

OR user\_name LIKE &quot;%&quot;#{key}&quot;%&quot;

</if>

</where>

ORDER BY user\_id

</select>

<update id="delUser" parameterType="int">

UPDATE user

SET status = 0

WHERE user\_id = #{id}

</update>

<update id="reuseUser" parameterType="int">

UPDATE user

SET status = 1

WHERE user\_id = #{id}

</update>

<update id="mute" parameterType="com.arvinclub.model.entity.User">

UPDATE user

SET user\_date = #{muteDeadline},

muted = #{muted}

WHERE user\_id = #{id};

</update>

### 5.4.5 微博系统

用户登录系统后默认直接进入的就是博客系统的页面，在上面的输入框中可以键入博客内容，也可以选择要发布的图片，点击发表按钮即可发布成功，随后在下面即可看到刚刚发布的博客了。微博系统界面如下图所示。

图5-5 微博系统界面图

发布博客功能关键代码：

/\*\*

\* 发布博客

\*

\* @param blog 博客

\* @param files 图片文件

\* @return 成功则转到社区主页，否则转到错误提示页面

\*/

@PostMapping("user/addBlog")

public String addBlog(Blog blog, HttpSession httpSession, @RequestParam MultipartFile[] files) throws Exception {

User me = (User) httpSession.getAttribute("user");

userService.checkMuted(me);

/\*给blog添加filenames（面向数据库的图片文件名）\*/

blog.setFilenames(ToolsUtil.saveIMG(files, CLASSPATH));

blog.setUser(me);

/\*添加博客\*/

blogService.addBlog(blog);

return "redirect:/user/blogs.html";

}

发布博客功能关键查询语句：

<insert id="addBlog">

INSERT INTO blogs (blog\_content, blog\_time, user\_id, filenames)

VALUES (#{content}, #{time}, #{id}, #{filenames})

</insert>

查看博客功能关键代码：

/\*\*

\* 查看博客（默认第一页）

\*/

@GetMapping("user/blogs.html")

public String showBlogs(Model model) {

return showBlogs(model, 1);

}

/\*\*

\* 查看博客（指定页号）

\*

\* @param page 页号

\*/

@GetMapping("user/blogs.html/{page}")

public String showBlogs(Model model, @PathVariable int page) {

/\*根据页号获取数据\*/

PageInfo<Blog> blogPage = blogService.selectAllBlog(page);

model.addAttribute("blogPage", blogPage);

return "blogs";

}

/\*\*

\* 查看关注动态（默认第一页）

\*/

@GetMapping("user/attention.html")

public String attention(Model model,HttpSession session) {

return attention(model, 1, session);

}

/\*\*

\* 查看关注动态（指定页号）

\*

\* @param page 页号

\*/

@GetMapping("user/attention.html/{page}")

public String attention(Model model, @PathVariable int page,HttpSession session) {

/\*根据页号获取数据\*/

User me = (User) session.getAttribute("user");

PageInfo<Blog> blogPage = blogService.attentionBlogs(me.getId(), page);

model.addAttribute("blogPage", blogPage);

return "attention";

}

查看博客功能关键查询语句：

<select id="selectBlogsByUserId" resultMap="blogMap" parameterType="int">

SELECT blog\_id, blog\_content, blog\_time, user\_id, filenames

FROM blogs

<where>

<if test="\_parameter > 0">

user\_id = #{id}

</if>

AND status = 1

</where>

ORDER BY blog\_id DESC

</select>

<select id="attentionBlogs" resultMap="blogMap" parameterType="int">

SELECT blog\_id, blog\_content, blog\_time, user\_id, filenames

FROM blogs

WHERE status = 1

AND user\_id IN (SELECT blogger\_id FROM attention WHERE fans\_id = #{id} and status = 1)

ORDER BY blog\_id DESC

</select>

### 5.4.6 评论系统

用户在点击任何一篇博客后，会跳转至博客详情页面，此页面显示了博客内容、发布者、发布时间、以及所有用户对它的评论。评论系统界面如下图所示。

图5-6用户评论界面图

评论系统功能关键代码：

/\*\*

\* 评论相关

\*/

@Controller

public class CommentController {

@Resource

private CommentService commentService;

@Resource

private UserService userService;

/\*\*

\* 发布评论

\*

\* @return 成功则转到微博详情，否则转到错误提示页面

\*/

@PostMapping("user/addComment/{blogId}")

public String addComment(Comment comment, HttpSession httpSession, @PathVariable int blogId) {

if (StringUtils.isNotBlank(comment.getContent())) {

/\*构建评论\*/

User me = (User) httpSession.getAttribute("user");

userService.checkMuted(me);

comment.setUser(me);

comment.setTime(ToolsUtil.getNowTimeString());

Blog blog = new Blog();

blog.setId(blogId);

comment.setBlog(blog);

commentService.addComment(comment);

}

/\*添加评论\*/

return "redirect:/user/detail.html/" + blogId;

}

/\*\*

\* 删除评论

\*

\* @return 成功则转到微博详情，否则转到错误提示页面

\*/

@GetMapping("user/delComment/{commentId}")

public String delComment(HttpSession httpSession, @PathVariable int commentId) {

Comment comment = commentService.backCommentById(commentId);

User me = (User) httpSession.getAttribute("user");

User someone = comment.getUser();

User blogger = comment.getBlog().getUser();

if (me.equals(someone) || me.getAdmin() > someone.getAdmin() ||

me.equals(blogger) || me.getAdmin() > blogger.getAdmin()) {

commentService.delComment(commentId);

}

return "redirect:/user/detail.html/" + comment.getBlog().getId();

}

/\*\*

\* 编辑评论

\*

\* @return 成功则转到微博详情，否则转到错误提示页面

\*/

@PostMapping("user/editComment")

public String editComment(Comment comment, HttpSession httpSession) {

Comment oldComment = commentService.backCommentById(comment.getId());

User me = (User) httpSession.getAttribute("user");

userService.checkMuted(me);

if (StringUtils.isNotEmpty(comment.getContent()) && me.equals(oldComment.getUser())) {

commentService.updateComment(comment);

}

return "redirect:/user/detail.html/" + oldComment.getBlog().getId();

}

}

评论系统功能关键查询语句：

<update id="updateComment" parameterType="com.arvinclub.model.entity.Comment" useGeneratedKeys="true" keyProperty="id">

update comment

set comment\_content = #{content}

where comment\_id = #{id}

</update>

<delete id="delComment" parameterType="int">

DELETE FROM comment

WHERE comment\_id = #{id}

</delete>

<select id="backCommentById" resultMap="commentMap" parameterType="int">

SELECT comment\_id, comment\_content, comment\_time, user\_id, status, blog\_id

FROM comment

WHERE comment\_id = #{id}

AND status = 1

ORDER BY comment\_id

</select>

<select id="backCommentByBlogId" resultMap="commentMap" parameterType="int">

SELECT comment\_id, comment\_content, comment\_time, user\_id, status, blog\_id

FROM comment

WHERE blog\_id = #{id}

AND status = 1

ORDER BY comment\_id

</select>

<insert id="addComment">

INSERT INTO comment(blog\_id, user\_id, comment\_content, comment\_time)

VALUES (#{blog}, #{user}, #{content}, #{time})

</insert>

<resultMap id="commentMap" type="com.arvinclub.model.entity.Comment">

<id column="comment\_id" property="id" jdbcType="INTEGER"/>

<result column="comment\_content" property="content" jdbcType="VARCHAR" javaType="string"/>

<result column="comment\_time" property="date" jdbcType="TIMESTAMP" javaType="java.util.Date"/>

<result column="status" property="status" jdbcType="INTEGER" javaType="int"/>

<association property="user" select="com.arvinclub.service.dao.UserDao.backUserById" column="user\_id"/>

<association property="blog" select="com.arvinclub.service.dao.BlogDao.backBlogsById" column="blog\_id"/>

</resultMap>

### 5.4.7 多人聊天

微博系统界面如下图所示。

图5-7多人聊天界面图

多人聊天功能关键代码：

/\*\*

\* 基于WebSocket的多人聊天室

\*/

@ServerEndpoint(value = "/webSocketByTomcat", configurator = GetHttpSessionConfigurator.class)

public class WebSocket {

/\*\*

\* 聊天室人数

\*/

private static final AtomicInteger onlineCount = new AtomicInteger(0);

/\*\*

\* 当前聊天室成员

\*/

private static final Map<String, WebSocket> clients = new ConcurrentHashMap<>();

/\*\*

\* 用户信息

\*/

private Session session;

private String username;

private final Object lock = new Object();

/\*\*

\* 新建连接

\*/

@OnOpen

public void onOpen(Session session, EndpointConfig config) throws Exception {

//获取HttpSession，并得到用户名

HttpSession httpSession = (HttpSession) config.getUserProperties().get(HttpSession.class.getName());

username = ((User) httpSession.getAttribute("user")).getName();

//登录冲突提醒

if (clients.containsKey(username)) {

//language=HTML

clients.get(username).sendMessageToMe("<span style=\"color:blue\">你已在其他位置登陆</span>");

clients.get(username).session.close();

}

//该用户进入聊天室

this.session = session;

clients.put(username, this);

//聊天室人数调整，并通知其他用户

onlineCount.set(clients.size());

//language=HTML

sendMessageAll("<span style=\"color:green\">[" + username + "] 进入聊天室, 当前人数: " + getOnlineCount() + "</span>");

}

/\*\*

\* 关闭连接

\*/

@OnClose

public void onClose() throws Exception {

clients.remove(username);

onlineCount.set(clients.size());

//language=HTML

sendMessageAll("<span style=\"color:red\">[" + username + "] 离开聊天室, 当前人数: " + getOnlineCount() + "</span>");

if (session != null && session.isOpen()) {

session.close();

}

}

/\*\*

\* 处理收到的消息

\*/

@OnMessage

public void onMessage(String message) throws IOException {

if (message.equalsIgnoreCase("clear!!!"))

for (WebSocket webSocket : clients.values())

webSocket.session.close();

else if (message.equalsIgnoreCase("num!!!")) {

//language=HTML

sendMessageToMe("<span style=\"color:blue\">当前人数: " + getOnlineCount() + "</span>");

for (String u : clients.keySet())

//language=HTML

sendMessageToMe("<span style=\"color:blue\">[" + u + "] </span>");

} else

sendMessageAll("[" + username + "]: " + message);

}

/\*\*

\* 出错也要保证正常关闭和注销

\*/

@OnError

public void onError(Throwable error) throws Exception {

if (session != null && session.isOpen()) {

session.close();

}

}

/\*\*

\* 广播信息

\*/

private void sendMessageAll(String message) throws IOException {

for (WebSocket item : clients.values())

synchronized (item.lock) {

item.session.getBasicRemote().sendText(message);

}

}

/\*\*

\* 打印出提醒（单人）

\*/

private void sendMessageToMe(String message) throws IOException {

synchronized (lock) {

session.getBasicRemote().sendText(message);

}

}

/\*\*

\* 获取当前人数

\*/

public static int getOnlineCount() {

for (String str : clients.keySet())

if (!clients.get(str).session.isOpen())

clients.remove(str);

return onlineCount.get();

}

}

/\*\*

\* 用于WebSocket连接时获取HttpSession

\*/

public class GetHttpSessionConfigurator extends Configurator {

/\*\*

\* 获取HttpSession并放到UserProperties中

\*/

@Override

public void modifyHandshake(ServerEndpointConfig sec, HandshakeRequest request, HandshakeResponse response) {

HttpSession httpSession = (HttpSession) request.getHttpSession();

sec.getUserProperties().put(HttpSession.class.getName(), httpSession);

}

}

### 5.4.8 站内搜索

****站内搜索界面如下图所示。

图5-8站内搜索界面图

站内搜索功能关键代码：

/\*\*

\* 查找内容

\*/

@Controller

public class SearchController {

@Resource

private BlogService blogService;

/\*\*

\* 进入查找内容页面

\*/

@GetMapping("user/search.html")

public String search() {

return "search";

}

/\*\*

\* 按关键词查找内容（默认第一页）

\*/

@PostMapping("user/search.html")

public String search(@RequestParam String keyword, HttpSession session) {

session.setAttribute("keyword", keyword);

return searchByPage(1, session);

}

/\*\*

\* 按关键词查找内容（指定页号）

\*/

@GetMapping("user/search.html/{page}")

public String searchByPage(@PathVariable int page, HttpSession session) {

/\*确认关键词\*/

String keyword = (String) session.getAttribute("keyword");

if (StringUtils.isBlank(keyword))

return "redirect:/user/search.html";

/\*开始查询,返回信息\*/

session.setAttribute("blogPage", blogService.selectBlogsByKey(keyword, page));

return "search";

}

}

站内搜索功能关键查询语句：

在SQL查询语句的拼接上，占位符使用了#{}而不是${}，#{}使用了预编译SQL语句的形式，不光有效地提升了SQL语句在重复查询上的性能，还能防止被SQL注入。

<select id="selectBlogsByKey" resultMap="blogMap" parameterType="string">

SELECT blog\_id, blog\_content, blog\_time, user\_id, filenames

FROM blogs a INNER JOIN user b USING(user\_id)

<where>

a.status = 1

AND blog\_content LIKE &quot;%&quot;#{key}&quot;%&quot;

OR user\_name LIKE &quot;%&quot;#{key}&quot;%&quot;

</where>

ORDER BY blog\_id DESC

</select>

SELECT blog\_id, blog\_content, blog\_time, user\_id, filenames

FROM blogs a

INNER JOIN user b USING(user\_id)

WHERE a.status = 1 AND blog\_content LIKE "%"'test'"%" OR user\_name LIKE "%"'test'"%" ORDER BY blog\_id DESC

LIMIT 15;

# 6 系统测试

## 6.1 功能过程

### 6.1.1 注册登录测试

注册用户时，如果出现与原有用户重名的情况，或者用户名和密码中有任意一个不符合规范，则前台对应样式会相应地变红色，警告用户。

用户尝试登录时，如果用户名和密码中有任意一个不符合规范，或者出现密码错误的情况，前台对应样式也会相应地变红色，达到警告用户的目的。

而且用户登录成功后，上一位登录的用户残留信息（搜索记录，用户信息等）会被删除，保证了用户信息的隐私安全。

### 6.1.2 修改密码测试

用户只有在登录状态下，才可以在自己的个人空间中修改自己的登录密码，如果验证的时候出现新指定登录密码不符合系统要求的情况（包含空白字符），则会抛出一个异常，前台也会相应地向用户发出警告。

### 6.1.3 博客系统测试

发布：用户在成功登录本系统以后，可以在博客页面（即首页）正常发布博客，可以包含文字和图片，图片会被保存在服务端，并做备份。

查看：其他用户都可以在首页成功看到这条博客（没有看到的需要刷新一下即可），按时间降序排序。当发布博客时，一些HTML相关的字符，比如<html>，在这些字符被传到服务端后台时，会被自动转移义，保障了此微博系统的正常使用。

删除：普通用户可以成功删除自己发布的博客，而管理员则可以删除所有博客。

编辑：普通用户可以成功编辑自己发布的博客，而管理员则可以编辑所有博客。而且编辑时可以正确显示和处理Windows系统和Unix系统下不同的回车和换行标识。

### 6.1.4 个人主页测试

用户在成功登录本系统以后，可以进入自己（通过点击个人主页或者自己的用户名）或者其他用户（通过点击此用户的用户名）的个人主页，并查看此人的关注数量和粉丝数量，以及他的所有博客内容，按时间降序排序。

### 6.1.5 粉丝关注测试

用户可以在其他用户的个人主页页面选择关注或者取消关注此人。在关注动态页面还可以看到所有关注的用户最近发布的所有博客。用户还可以在自己或者其他用户的个人主页中查看此人的关注数量和粉丝数量，点击关注数量后会进入关注列表，会显示此用户关注的所有用户，点击用户名还可以进入他的个人空间，点击粉丝数量后会进入粉丝列表，会显示关注此用户的所有用户，按时间降序排序，点击用户名还可以进入他的个人空间

### 6.1.6 用户评论测试

发布：用户可以在正常在自己或者其他用户发布的博客下面添加评论。

查看：所有用户都可以看到已发布的评论。

编辑：用户可以正常编辑自己发布发布的评论，而无法修改别人的评论。

删除：用户可以正常删除自己发布的评论，或者自己发布的博客下面的评论。而管理员却可以删除任何评论。

### 6.1.7 站内搜索测试

用户可以正常搜索用户名，也可以正常搜索出博客内容，按时间降序排序。而且用户的搜索记录会被保留，直到用户退出登录或者有其他用户在这台电脑上登录。

### 6.1.8 多人聊天测试

1.用户进入聊天室和离开聊天室提示支持。

2.人数同步显示正常。

3.用户发送消息和消息显示正常。

4.大量用户同时发送消息的高并发场景下，聊天室系统仍能正常工作。

### 6.1.9 后台管理测试

管理员可以在后台管理界面正常看到所有用户的账号消息，按用户ID升序排序。管理员可以在此界面通过用户ID或者用户名来搜索用户，并且可以随时停用（删除）任意一个用户的账号，也可以永久或者期限禁言一个用户，在这段时间内，如果被禁言的用户尝试发布博客或者评论，系统会拦截用户的操作抛出异常，前台接收到这一信息警告用户被禁言，并告诉用户解禁时间，但是禁言不会影响用户在多人聊天时发言。

除此以外，管理员拥有管理权限，可以编辑和删除任何博客，也可以删除任何评论

### 6.1.10 权限管理测试

用户只有登录后才能正常使用本微博系统，而且只有登录了管理员账号才能进入后台管理员页面，使用管理员权利。

## 6.2 测试过程

软件测试分为四个阶段: 单元测试阶段、集成测试阶段，系统测试阶段，验收测试阶段。

1、单元测试阶段：单元测试是以最小单位的测试、也是最初期的测试阶段、一般是以一个函数方法窗口、一个功能模块、都可以看做是一个单元，主要依据的是详细设计文档。主要以白盒为主，一般有开发人员完成

2、集成测试阶段:  集成测试又称组装测试，在单元测试的基础上把软件逐渐组装起来一起继续测试的过程。逐渐组装的过程中会出现很多临时版本(迭代测试)。集成测试主要以黑盒为主(当然接口测试也在这阶段进行)

3、系统测试阶段:   整个功能全部完成后对集成了硬件和软件的完整系统进行模拟真实的环境模拟、测试重点主要在于

1)整个系统能否正常运行

2)真个系统的兼容性测试

4、验收测试阶段:   由用户参与完成的过程

1)alpha阶段:在软件开发过程中由最终用户对软件进行检查

2)beta阶段:在最终用户的实际环境中由最终用户对软件进行检查

### 6.2.1 整体测试

综合上面的测试，各个页面的功能正常，性能正常。并且对各个页面添加返回的链接。测试结果：用户页面美观，简洁，易于使用。基本满足了一个微博系统需要的功能。系统运行稳定性较好，每一处功能都有相关的权限校验，暂未发现任何安全问题。

### 6.2.2 正式上线测试

将此项目编译打包成blog.war，拷贝到Linux云主机上部署，并配置好环境，项目已经可以成功运行并能够被用户正常访问到，且长时间内正常运行未出现进程异常终止或内存溢出等现象。域名：[www.arvinclub.com](http://www.arvinclub.com)。

# 7 总结

本次毕业设计：《基于SSM微博系统的设计与实现》是沈阳师范大学在校学生施航程在指导老师李晋的指导下完成课题研究，这是我第一次独自完全设计一个拥有完整功能的大型系统，在实习过程中，一边工作，一边学习，一边进行着我的毕业设计编码、实现以及毕业论文的编写工作，以下是我对本次系统设计实践经历的总结。

首先我选用了Java作为微博系统项目的后台编程语言，首先是因为Java简单易用，在学习了李晋老师的课程后，我已经数据它的编程语法、基础知识、以及各种高级特性，在将来一段时间内Java会作为我的主要学习方向。

Web容器方面，我选择了Java开发者最熟悉的Tomcat，持久化方面我选择了简单易用的MySQL，原因是在我们大学的课堂里，MySQL是作为经典的数据库课程出现在我们的课堂和教材中，我们在学习阶段都是用它来做为学习和使用数据库的工具。

转眼大学四年时光就结束了，本次项目设计虽然历经坎坷，但是还算圆满成功，感谢所有期间帮助过我的老师和同学，以及支持我的同事和朋友。

# 后 记

在本次毕业设计过程中，有幸得到了李晋老师细心的指导与支持，帮助我顺利完这次毕业设计起到了非常关键性的作用。在此深表敬意!

时光荏苒，日月如梭，大学四年时光如白驹过隙，这次毕业论文不仅是一个任务，更是我将知识用于实践并锻炼我动手能力的一个契机，一个将理论与实际相结合的机会

通过本次微博系统的设计，我极大地提改了问题解决能力，而不是像平时一样只局限于课堂上的理论知识，从一开始的手忙脚乱和到处查阅资料，到现在编写完整个系统，从当初拟定题目到写完毕业论文手工，也经历了很长时间，期间我逐渐认识到自己能力的不足和实战经验的欠缺，并在完成任务的过程中一步一步积累经验和知识，吸取整合了平时的积累，所以我才说这个任务是我的一次契机。

在这里我要感谢的有：我的学校、专业、还有指导老师，还有期间对我有重大帮助的同学，同事，以及理解我的家人，在学习的过程中，离不开它们的支持和建议，他们在我的本科大学生活中留下了难忘的回忆，再次衷心老师在百忙之中关心我的毕业设计的进度和我的就业情况。本次毕业设计给我留下了深刻的印象，它的结束也意味着我与我的的大学思念时光的告别。

# 参考文献

[1]Cay S.Horstmann,《Java核心技术 卷一 基础知识》.原书第十版，机械工业出版社，2016.8；

[2]Cay S.Horstmann,《Java核心技术 卷二 高级特性》.原书第十版，机械工业出版社，2016.8；

[3] R.J(Bud)Bates. GPRS:General Packet Radio Service .McGraw-Hill, 2003;

[4] Cameron,J.R. JSP & JSD the Jackson approach to software development .Silver Spring MD: IEEE Computer Society Press, 1983;

[5] M.J.King,J.P.Pardoe. Program design using JSP a practical introduction .Basingstoke Hampshire: Macmillan, 1992;

[6] James Goodwill. Pure Java Server Pages .Indianapolis Ind: Sams, 2000;

[7] Gwenael Le Bodic. Mobile messaging technologies and services SMS EMS and MMS .New York: J.Wiley, 2005;

[8] Henry Labordre,Vincent Jonack. SMS and MMS interworking in mobile networks .Boston: Artech House, 2004;

[9] 张黎明，龚琪琳. 基于MVC模式的Java Web应用设计. 计算机与现代化, 2007.02:P22-24;

[10] 刘佳音.我国人力资源信息网站的现状与发展——基于人才招聘网站的调查分析.情报科学,2004.06:24-25;

[11] 方小明.基于Struts的Java Web应用开发(一).软件报, 2006.04.10;

[12] 蒋曹清. Java安全体系结构研究.罗海鹏.广西计算机学会2006年年会,中国广西，2006 年:P246-249;