**【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](#_Toc941254549)

[1.1课题来源](#_Toc1300661241)

[1.2国内现状](#_Toc977434674)

[1.3研究意义](#_Toc1642150015)

[2开发环境](#_Toc155470861)

[2.1 硬件](#_Toc1658646075)

[2.2 软件](#_Toc379360818)

[2技术路线](#_Toc38320183)

[2.1 技术栈](#_Toc1949705228)

[2.1 项目依赖](#_Toc242797423)

[2.1 项目结构](#_Toc477359061)

[2技术介绍](#_Toc2122316682)

[2.1 JAVA开发语言介绍](#_Toc73097704)

[2.1 Maven简介](#_Toc192465044)

[2.1 Spring简介](#_Toc649622126)

[2.1 MyBatis简介](#_Toc392210334)

[2.1 Tomcat简介与搭建](#_Toc1251770895)

[2.1 WebSocket简介](#_Toc1763626253)

[2.1 JavaWeb原理介绍](#_Toc1697138277)

[2.1 JavaBean简介](#_Toc925222085)

[2.1 AJAX简介](#_Toc278494668)

[3需求分析](#_Toc1293018263)

[3.1可行性分析](#_Toc1370922248)

[3.2安全性分析](#_Toc738173473)

[3.3性能需求分析](#_Toc468531992)

[3.4技术可行性](#_Toc1942139642)

[3.5功能需求分析](#_Toc1937012341)

[3.6网络原理图](#_Toc1661810314)

[4数据库设计](#_Toc2021118163)

[4.1用户](#_Toc36637295)

[4.1博客](#_Toc1582694023)

[4.1评论](#_Toc1605992819)

[4.1关注](#_Toc199349790)

[4.1实体关系图](#_Toc397431210)

[5项目实现](#_Toc952204300)

[5.1项目结构](#_Toc649532656)

[5.1数据库持久化](#_Toc1035971691)

[5.1日志](#_Toc1926284408)

[5系统实现](#_Toc1746066731)

[5.1系统总体架构及实现](#_Toc779511662)

[5.1用户登录](#_Toc1602256534)

[5.1修改密码](#_Toc1828117205)

[5.1粉丝关注](#_Toc1117326806)

[5.1用户管理](#_Toc1314619074)

[5.1微博系统](#_Toc1491016382)

[5.1评论系统](#_Toc525655431)

[5.1多人聊天](#_Toc2090588706)

[5.1站内搜索](#_Toc1544433175)

[6系统测试](#_Toc653530936)

[6.1测试的目的](#_Toc1663070594)

[6.1功能测试](#_Toc1727807653)

[6.4测试过程](#_Toc989349237)

[6.5正式上线测试](#_Toc26747538)

[7总结](#_Toc721788943)

[后 记](#_Toc2119126745)

[参考文献](#_Toc146917720)

# 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。

WIPO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短视频制作和消费平台，Twitter和微博故事于2017年4月开始发布。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

随着计算机技术的飞速发展，大数据时代已经到来，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开发环境

## 2.1 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。

## 2.2 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

# 2技术路线

## 2.1 技术栈

后端：

Java

Spring

SpringMVC

Tomcat

MySQL

log4J

MyBatis

Maven

Linux

WebSocket

HTTP

前端：

HTML5

CSS

JavaScript

Jsp

jquery

bootstrap

## 2.2 项目依赖

Pagehelper：MyBatis的分页助手插件，源自GitHub

jsqlparser ：MyBatis的分页助手插件的JSP解析器

junit ：Java语言的单元测试框架

mockito-all：用于模拟测试

commons-lang3：Apache的公共工具包

hamcrest-core：测试框架

hibernate-validator：ORM框架校验器

commons-fileupload：用于文件上传

commons-io ：Apache的IO公共工具包

commons-codec ：Apache的编码解码的公共工具包

thumbnailator：是一个图片开源工具，用于图片压缩

log4j Apache的控制日志信息的开源项目

metadata-extractor：用于读取图片信息

javaee-api ：JavaEE的核心API与组件

spring-web ：Apache Spring的Web相关的依赖，提供了表现层(springmvc)到业务层(spring)再到数据层的全套解决方案

c3p0：C3P0数据库连接池

spring-webmvc ：Apache Spring的WebMVC相关的依赖

spring-websocket ：Apache Spring的WebSocket相关的依赖

mysql-connector-java：MySQL的JDBC连接驱动依赖

jstl：JSP的JSTL标签依赖

taglibs：JSP的扩展标签库支持依赖

org.mybatis：MyBatis依赖：

mybatis-spring ：MyBatis整合Apache Spring的相关的依赖

spring-jdbc：Apache Spring的JDBC相关的依赖

spring-test：Apache Spring的单元测试相关的依赖

junit：Java语言的单元测试框架：

spring-core：Apache Spring的核心依赖：

javax.activation：打包时报错加的

## 2.3 项目结构

# 2技术介绍

## 2.1 JAVA开发语言介绍

Java是Sun于1995年推出的一种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。

4.鲁棒性

通过针对面向目标的异常的集成机制，Java程序集在组装过程中指出了可能的但未解决的异常，在汇编和运行程序时，Java会检查可能出现的问题以消除错误。它允许自动垃圾收集以进行内存管理，从而防止程序员犯下管理内存时可能出现的错误。这有助于程序员做出正确的选择以防止系统崩溃。在编辑数据时，Java可以记录与其类型相关的数据中的许多常见错误，从而避免了在动态操作期间出现不匹配的问题。

5.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。

6.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，都可以进行转换，并且正常运行，这就是Java的平台无关和操作系统无关。

10.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。

11.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

## 2.3 Spring简介

1. 历史

2002 年 10 月，罗宾·约翰逊编写了一本开发和设计企业级Java项目的书。本书由 Wrox出版，介绍了当时 Java 企业应用程序开发的情况，并指出了 Java EE 和 EJB 组件框架中的存在的一些主要缺陷。在这本书中，他提出了一个基于普通 Java 类和依赖注入的更简单的解决方案。

在本书发布后不久，开发者 Juergen Hoeller 和 Yann Caroff 说服 Rod Johnson 创建一个基于基础结构代码的开源项目。Rod，Juergen 和 Yann 于 2003 年 2 月左右开始合作开发该项目 。Yann 为新框架创造了“Spring”的名字。Yann Caroff 在早期离开了团队，Rod Johnson 在 2012 年离开，Juergen Hoeller 仍然是 Spring 开发团队的积极成员。

自 2004 年 1.0 版本发布以来，Spring 框架迅速发展。Spring 2.0 于 2006 年 10 月发布，到那时，Spring的下载量超过了 100 万。Spring 2.0 具有可扩展的 XML 配置功能，用于简化 XML 配置，支持 Java 5运行环境。

在 Rod 领导下管理 Interface21 项目于 2007 年 11 月更名为 SpringSource。同时发布了 Spring 2.5。Spring 2.5 中的主要新功能包括支持 Java 6 / Java EE 5，支持注释配置，classpath 中的组件自动检测和兼容 OSGi 的 bundle。

2007 年，SpringSource 从基准资本获得了 A 轮融资（1000万美元）。SpringSource 在此期间收购了多家公司，如Hyperic，G2One 等。2009年8月，SpringSource 以 4.2 亿美元被 VMWare 收购。SpringSource 在几周内收购了云代工厂，这是一家云 PaaS 提供商。2015 年，云代工厂转型成了非营利云代工厂。

2009 年 12 月，Spring 3.0 发布。Spring 3.0 具有许多重要特性，如重组模块系统，支持 Spring 表达式语言，基于 Java 的 bean 配置（JavaConfig），支持嵌入式数据库（如 HSQL，H2 和 Derby），模型验证/ REST 支持和对 Java EE 的支持。

2011 年和 2012 年发布了许多 3.x 系列的小版本。2012 年 7 月，Rod Johnson 离开了团队。2013 年 4月，VMware 和 EMC 通过 GE 投资创建了一家名为 Pivotal 的合资企业。所有的 Spring 应用项目都转移到了 Pivotal。

2013 年 12 月，Pivotal 宣布发布 Spring 框架 4.0。Spring 4.0 是 Spring 框架的一大进步，它包含了对Java 8 的全面支持，更高的第三方库依赖性（groovy 1.8+，ehcache 2.1+，hibernate 3.6+等），Java EE 7 支持，groovy DSL for bean 定义，对 websockets 的支持以及对泛型类型的支持作为注入 bean 的限定符。

2014 年至 2017 年期间发布了许多 Spring 框架 4.xx 系列版本。

Spring 5.0 GA版本于2017年9月28日发布。Spring 5.0开始支持JDK 8和Java EE 7，同时兼容JDK9。全面支持Servlet 3.1，还引入了一个全新的模块Spring WebFlux用于替代老话的 spring-webmvc；对Kotlin也有了更好的支持。

本项目中使用的版本为Spring 4

Spring 4.x新特性：

Spring 4.x全面支持Java 8.0，支持Lambda表达式的使用，提供了对@Scheduled和@PropertySource重复注解的支持，提供了空指针终结者Optional，对核心容器进行增加：支持泛型的依赖注入、Map的依赖注入、Lazy延迟依赖的注入、List注入、Condition条件注解注入、对CGLib动态代理类进行了增强。

Spring 4.x还支持了基于Groovy DSL的配置，提高Bean配置的灵活性。

Spring 4.x开始，Spring MVC兼容了Servlet 3.0规范，并且提供了Restful风格的接口规范支持，为了方便Restful接口的编写，加入了新的RestController注解，省去了大量的@ResponseBody注解，与此同时还增加异步无阻塞请求功能。

特征

[核心技术]依赖项注入，类型转换，资源，国际化，数据绑定，事件，Spring表达式，面向切面的验证方式等等。

[数据访问]事务，DAO支持，JDBC，ORM，封送XML。

[Spring MVC]和 [Spring WebFlux] Web框架。

[集成]远程处理，JMS，JCA，JMX，电子邮件，任务，调度，缓存。

[语言] Kotlin，Groovy，动态语言。

1. 国内外现状

国内外都有无数知名Java项目通过maven构建，并托管在GitHub平台

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） 。然后，容器在创建bean时注入那些依赖项。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）。

1.2 容器

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。

在大多数应用场景中，不需要显式的用户代码来实例化一个Spring IoC容器的一个或多个实例。例如，在Web应用程序场景中，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。

下图显示了Spring的工作原理的高级视图。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 AOP

Spring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。

SpringMVC

Spring Web MVC是基于Servlet API构建的原始Web框架，并且从一开始就已包含在Spring框架中。正式名称“Spring Web MVC”来自其源模块（[spring-webmvc]）的名称，但它通常被称为“ Spring MVC”。

与Spring Web MVC并行，Spring Framework 5.0引入了一个反应式堆栈Web框架，其名称“ Spring WebFlux”也基于其源模块（[spring-webflux]）。

## 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

## 2.5 Tomcat简介与搭建

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。

搭建

本项目使用的版本是Tomcat 8.5.50（https://tomcat.apache.org/download-80.cgi）

目录结构如下所示

tomcat

├── BUILDING.txt

├── CONTRIBUTING.md

├── LICENSE

├── NOTICE

├── README.md

├── RELEASE-NOTES

├── RUNNING.txt

├── bin

│   ├── attachments

│   ├── bootstrap.jar

│   ├── catalina-tasks.xml

│   ├── catalina.sh

│   ├── ciphers.sh

│   ├── commons-daemon-native.tar.gz

│   ├── commons-daemon.jar

│   ├── configtest.sh

│   ├── daemon.sh

│   ├── digest.sh

│   ├── hs\_err\_pid69993.log

│   ├── logs

│   │   └── logback

│   │   ├── error-2020-01-06.log

│   │   ├── smile\_2020-1-6.log

│   │   └── spring\_2020-1-6.log

│   ├── logs.log

│   ├── logs.log\_2020-01-19.log\ \

│   ├── logs.log\_2020-02-02.log\ \

│   ├── setclasspath.sh

│   ├── shutdown.sh

│   ├── startup.sh

│   ├── tomcat-juli.jar

│   ├── tomcat-native.tar.gz

│   ├── tool-wrapper.sh

│   └── version.sh

├── conf

│   ├── Catalina

│   │   └── localhost

│   ├── catalina.policy

│   ├── catalina.properties

│   ├── context.xml

│   ├── jaspic-providers.xml

│   ├── jaspic-providers.xsd

│   ├── logging.properties

│   ├── server.xml

│   ├── tomcat-users.xml

│   ├── tomcat-users.xsd

│   └── web.xml

├── lib

├── logs

├── temp

├── webapps

│   ├── ROOT

│   ├── blog

│   ├── blog.war

│   ├── docs

│   ├── examples

│   ├── host-manager

│   ├── manager

└── work

└── Catalina

└── localhost

├── ROOT

├── blog

├── docs

├── examples

├── host-manager

└── manager

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。

## 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。

Linux严格来说是单指操作系统的内核，因操作系统中包含了许多[用户图形接口](https://zh.wikipedia.org/wiki/GUI)和其他实用工具。

Linux的发行版中也有许多以桌面作为用户交互逻辑的，比如Ubuntu，Ubuntu由[Canonical公司](https://zh.wikipedia.org/wiki/Canonical%E5%85%AC%E5%8F%B8)发布，他们提供商业支持。Ubuntu的开发由[英国](https://zh.wikipedia.org/wiki/%E8%8B%B1%E5%9B%BD)Canonical有限公司主导，[南非](https://zh.wikipedia.org/wiki/%E5%8D%97%E9%9D%9E)企业家[Mark Shuttleworth](https://zh.wikipedia.org/wiki/Mark_Shuttleworth)所创立。Canonical通过销售与Ubuntu相关的技术支持和其他服务来产生收益。[[11]](https://zh.wikipedia.org/wiki/Ubuntu#cite_note-11)Ubuntu项目公开承诺开源软件开发的原则；鼓励人们使用自由软件，研究它的运作原理，改进和分发。

Ubuntu是著名的Linux发行版之一，它也是当前最多用户的Linux版本，用户数超过10亿人(含服务器、手机与其分支版本)。

# 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。

## 3.3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默认开启一级缓存，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。

## 3.4技术可行性

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

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

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

用户和管理员都可以修改密码。

4. 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。

5. 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。

6. 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）。

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

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

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

用户可以通过关键词索用户或者微博内容。

## 3.6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

系统的网络原理图

# 4数据库设计

依据系统功能实现需求，数据库设计如下：

## 4.1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

表设计：

实体属性图：

## 4.2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

实体属性图：

## 4.3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

实体属性图：

## 4.4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：

实体属性图：

## 4.5实体关系图

# 5项目实现

## 5.1项目结构

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，还控制了业务的逻辑，但是，这里的控制业务逻辑不是业务逻辑的实现，而仅仅是一个大的模块，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。

controller

├── admin

│   └── AdminController.java

├── blog

│   ├── BlogsController.java

│   └── CommentController.java

├── error

│   └── ErrorController.java

├── fun

│   ├── PageController.java

│   ├── RredirectController.java

│   └── SearchController.java

├── other

│   └── DownloadController.java

└── user

├── AttentionController.java

├── LoginController.java

└── RegisteController.java

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。

service

├── config

│   ├── GetHttpSessionConfigurator.java

│   ├── RootConfig.java

│   └── WebSocket.java

├── dao

│   ├── BlogDao.java

│   ├── CommentDao.java

│   └── UserDao.java

└── service

├── BlogService.java

├── CommentService.java

├── UserService.java

└── impl

├── BlogServiceImpl.java

├── CommentServiceImpl.java

└── UserServiceImpl.java

## 5.2数据库持久化

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。下面是来自[维基百科的Unicode字符平面映射](https://zh.wikipedia.org/wiki/Unicode字符平面映射)，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日志

1. Log4j介绍

Apache Log4j 2是Log4j的升级版，对Log4j的前身Log4j 1.x进行了重大改进，并提供了Logback中可用的许多改进，同时解决了Logback体系结构中的一些固有问题。

Log4j的API与实现是分开的，从而使应用程序开发人员可以清楚地了解他们可以使用哪些类和方法，同时确保向前的兼容性。这允许Log4j团队以兼容的方式安全地改进实施。

Log4j API是一个日志外观，可以与Log4j实现一起使用，但是也可以在其他日志实现（例如Logback）之前使用。与SLF4J相比，Log4j API具有多个优点：1. Log4j API支持记录[消息]而不只是字符串。2. Log4j API支持lambda表达式。3.与SLF4J相比，Log4j API提供了更多的日志记录方法。4.除了SLF4J支持的“参数化日志记录”格式外，Log4j API还支持使用java.text.MessageFormat语法以及printf样式消息的事件。5. Log4j API提供了LogManager.shutdown（）方法。基础的日志记录实现必须实现Terminable接口才能使该方法生效。6.完全支持标记，日志级别和ThreadContext（aka MDC）之类的其他构造。

Log4j 2包含基于LMAX Disruptor库的下一代异步记录器。在多线程方案中，与Log4j 1.x和Logback相比，异步Logger的吞吐量高18倍，延迟降低了几个数量级。。否则，Log4j 2明显优于Log4j 1.x，Logback和java.util.logging，尤其是在多线程应用程序中。Log4j 2提供对Log4j 1.2，SLF4J，Commons Logging和java.util.logging（JUL）API的支持。

编码为Log4j 2 API的应用程序始终可以选择使用任何SLF4J兼容库作为其Log4j-to-slf4j适配器的记录器实现。与Logback一样，Log4j 2可以在修改后自动重新加载其配置。与Logback不同，它在进行重新配置时不会丢失日志事件。Log4j 2支持基于上下文数据，标记，正则表达式和Log事件中的其他组件进行过滤。可以指定过滤，以将其应用于所有事件，然后再传递给Logger或当事件通过Appender时。此外，过滤器也可以与Loggers关联。与Logback不同，您可以在任何这些情况下使用通用的Filter类。

在Log4j 2中，可以通过代码或配置轻松定义[自定义日志级别]。无需子类化。除了使用Log4j API中的许多日志方法之一之外，还可以使用构建器来构造日志事件。

Log4j 2 在独立应用程序中是无垃圾的，而在Web应用程序中是低垃圾的。这样可以减少垃圾收集器上的压力，并可以提供更好的响应时间性能。

在创建log4j.properties，并从官网参考一份log4j.properties的模板，放到项目的类路径（classpath）上，根据实际要求修改loggers, appenders 和layouts，然后再测试日志在控制台和log文件中是否能够正常输出，磨刀不误砍柴工，日志的配置将大大提升接下来的开发效率。

Logger: 日志记录器，日志记录的核心类，用于输出不同日志级别的消息

Appender: 日志输出目标，指定日志的输出位置，如控制台、文件等等。

Layout: 日志格式化器，用于指定日志按照什么格式输出，是日志输出的格式化器。

# 5系统实现

## 5.1系统总体架构及实现



## 5.2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。

/\*\*

\* 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){ }

}

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。

/\*\*

\* 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:/";

}

相关SQL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;

## 5.3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。

/\*\*

\* 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";

}

相关SQL语句

UPDATE user SET user\_password = '123456'

WHERE user\_id = 2;

```

## 5.4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。

/\*\*

\* 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";

}

}

相关SQL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}

</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>

## 5.5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。

/\*\*

\* 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){ }

}

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。

/\*\*

\* 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;

}

}

相关SQL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>

## 5.6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。

/\*\*

\* 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";

}

相关SQL语句

<insert id="addBlog">

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

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

</insert>

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。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，

/\*\*

\* 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";

}

相关SQL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>

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。

/\*\*

\* 个人主页相关

\*/

@Controller

public class PageController {

@Resource

private BlogService blogService;

@Resource

private UserService userService;

/\*\*

\* 进入我的个人主页（默认第一页）

\*/

@GetMapping("user/page.html")

public String seeMe(Model model, HttpSession session) {

return seeWithPage(0, model, "1", session);

}

/\*\*

\* 进入某人个人主页（默认第一页）

\*/

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

public String see(@PathVariable int userID, Model model, HttpSession session) {

return seeWithPage(userID, model, "1", session);

}

/\*\*

\* 进入某人个人主页（指定页号）

\*/

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

public String seeWithPage(@PathVariable int userID, Model model, @PathVariable String str, HttpSession session) {

/\* 确认用户\*/

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

User someone = userService.findUserById(userID == 0 ? me.getId() : userID);

/\*确认页号,开始查询\*/

PageInfo<Blog> blogPage = blogService.selectBlogsByUserId(someone.getId(), Integer.parseInt(str));

/\*返回信息\*/

model.addAttribute("blogPage", blogPage);

model.addAttribute("someone", someone);

boolean hoster = me.equals(someone);

model.addAttribute("hoster", hoster);

if (!hoster)

model.addAttribute("isAttention", userService.isAttention(me.getId(), someone.getId()));

return "page";

}

}

相关SQL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 blog\_id, blog\_content, blog\_time, user\_id, filenames

FROM blogs

WHERE user\_id = 2 AND status = 1 ORDER BY blog\_id DESC

LIMIT 15;

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。

/\*\*

\* 编辑博客

\*

\* @return 成功则转到个人主页，否则转到错误提示页面

\*/

@PostMapping("user/editBlog")

public String editBlog(Blog blog, HttpSession httpSession) {

/\*如果有编辑权限则删除\*/

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

userService.checkMuted(me);

Blog oldBlog = blogService.blogDetail(blog.getId());

User someone = oldBlog.getUser();

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

blog.setContent(blog.getContent().replaceAll("\\\\r\\\\n","\r\n"));

blogService.editBlog(blog);

}

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

}

相关SQL语句

<update id="editBlog" parameterType="com.arvinclub.model.entity.Blog">

update blogs

set blog\_content = #{content}

where blog\_id = #{id}

</update>

update blogs set blog\_content = '测试'

WHERE blog\_id = 223;

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。

/\*\*

\* 删除博客

\*

\* @param blogId blog\_id

\* @return 成功则转到个人主页，否则转到错误提示页面

\*/

@GetMapping("user/delBlog/{blogId}")

public String delBlog(@PathVariable int blogId, HttpSession httpSession) {

/\*如果有编辑权限则删除\*/

Blog blog = blogService.blogDetail(blogId);

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

User someone = blog.getUser();

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

blogService.delBlog(blogId);

return "redirect:/user/page.html/" + blog.getUser().getId();

}

相关SQL语句

删除博客实际上是改变此博客的标志，而非真正删除所有数据

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

UPDATE blogs

SET status = 0

WHERE blog\_id = #{id}

</update>

UPDATE blogs SET status = 0

WHERE blog\_id = 223;

## 5.7评论系统

- 用户可以在博客下面发表评论，修改或删除自己发的评论。

/\*\*

\* 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();

}

}

相关SQL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>

## 5.8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。

/\*\*

\* 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();

}

}

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。

/\*\*

\* 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);

}

}

## 5.9站内搜索

用户可以通过关键词索用户和微博内容。

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

/\*\*

\* 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";

}

}

相关SQL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;

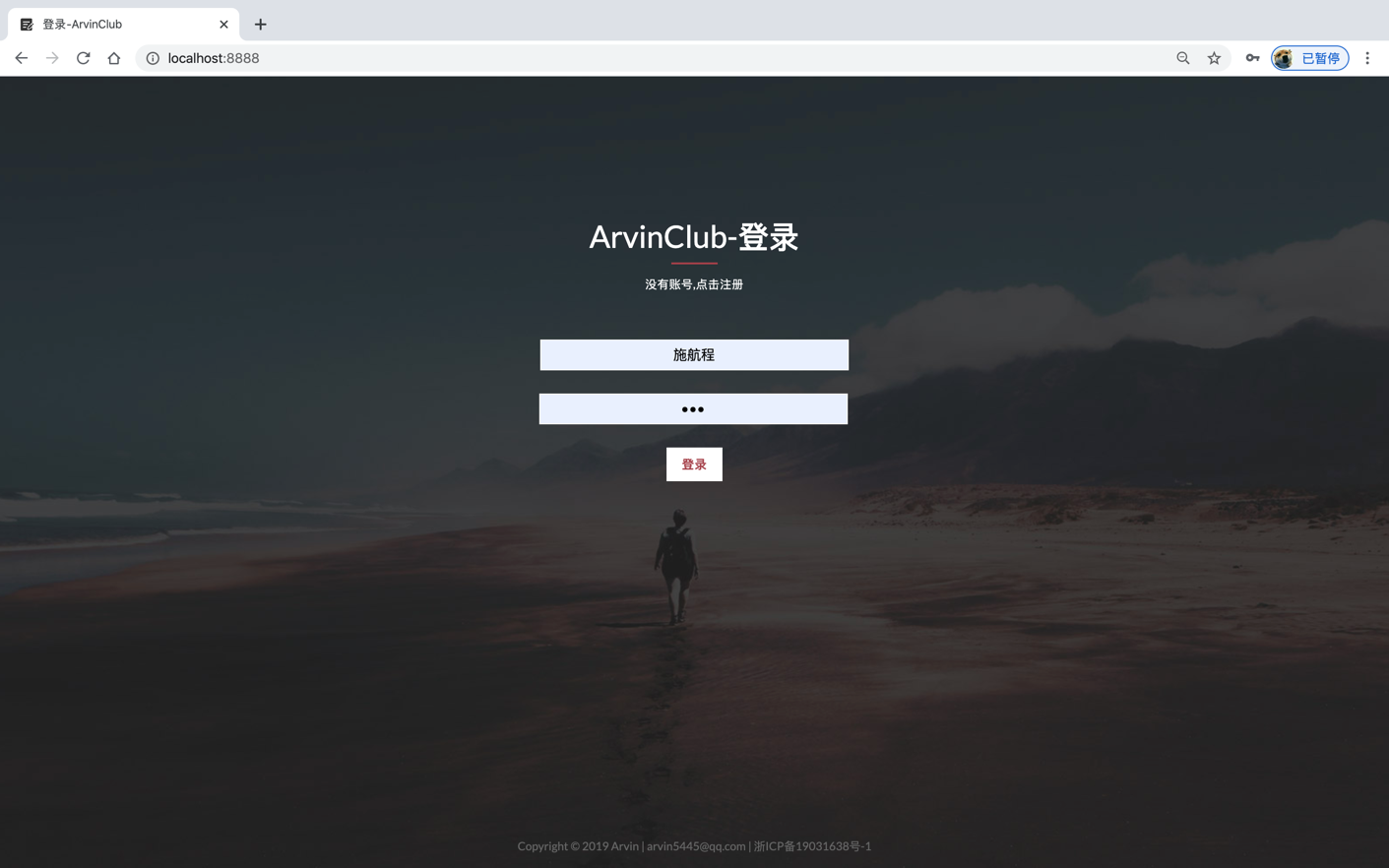
# 6系统测试

## 6.1测试的目的

### 6.2.1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。

### 6.2.2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。



### 6.2.3博客系统测试

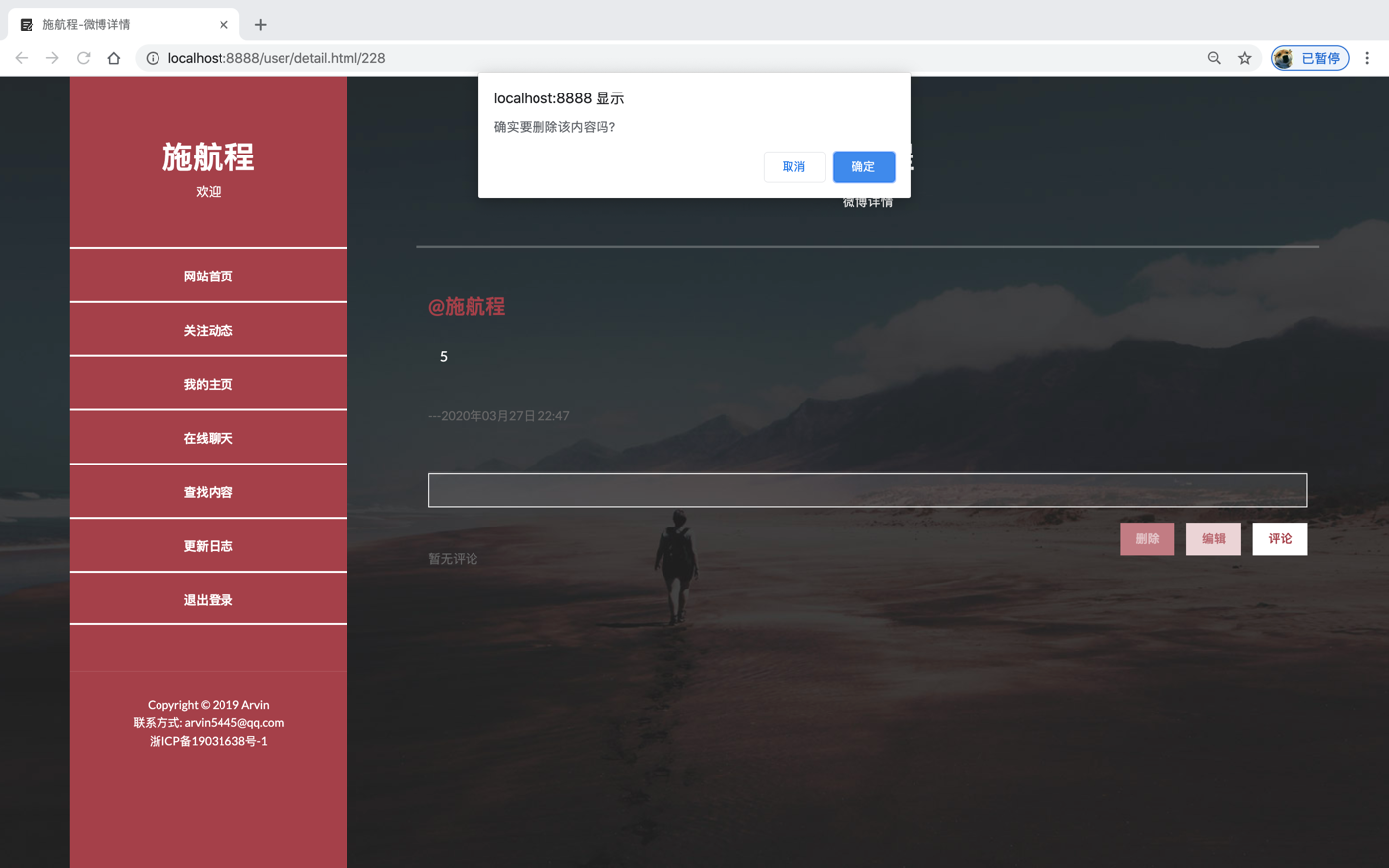
发布：

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。

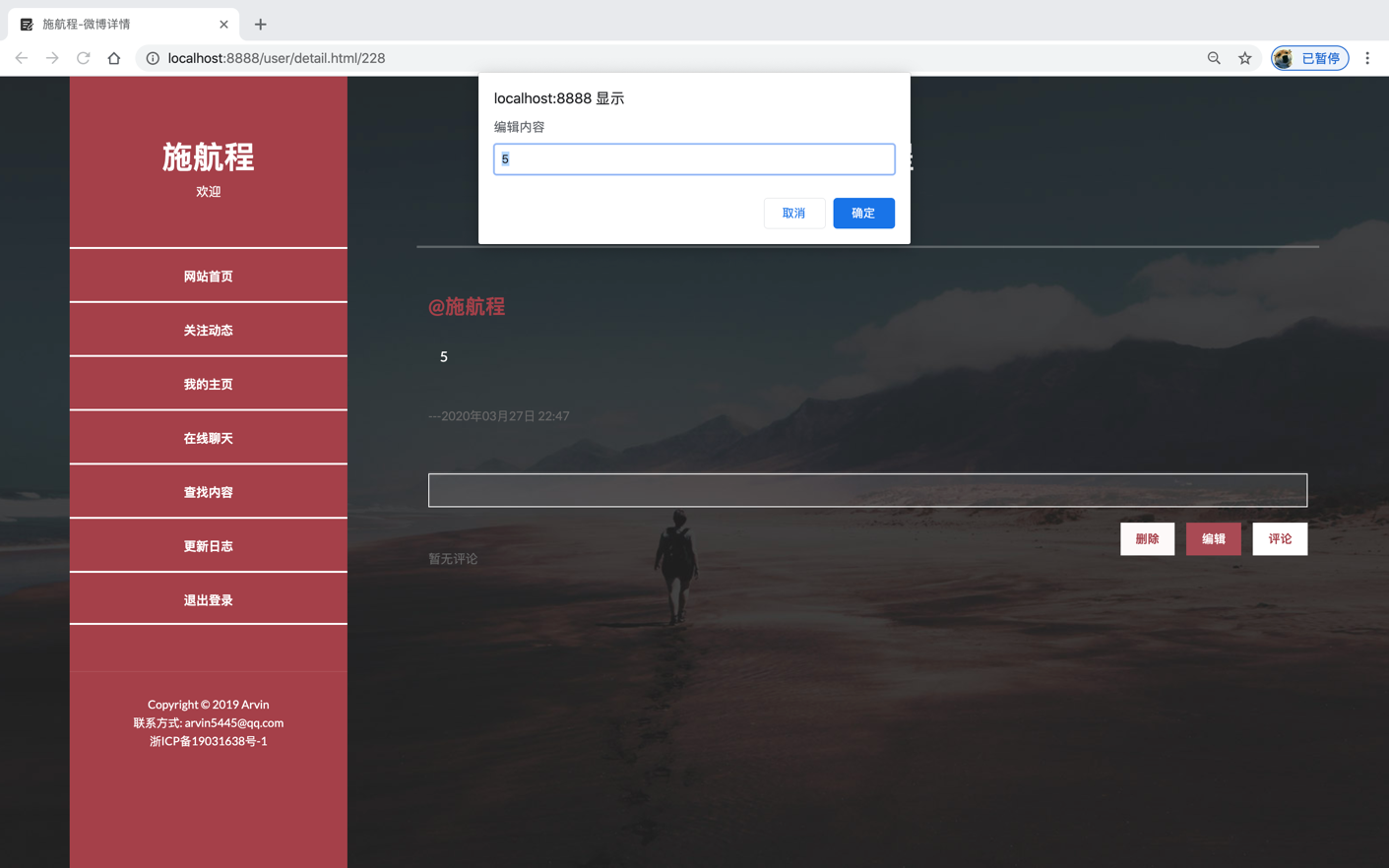
查看

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。

### 6.2.4个人主页测试

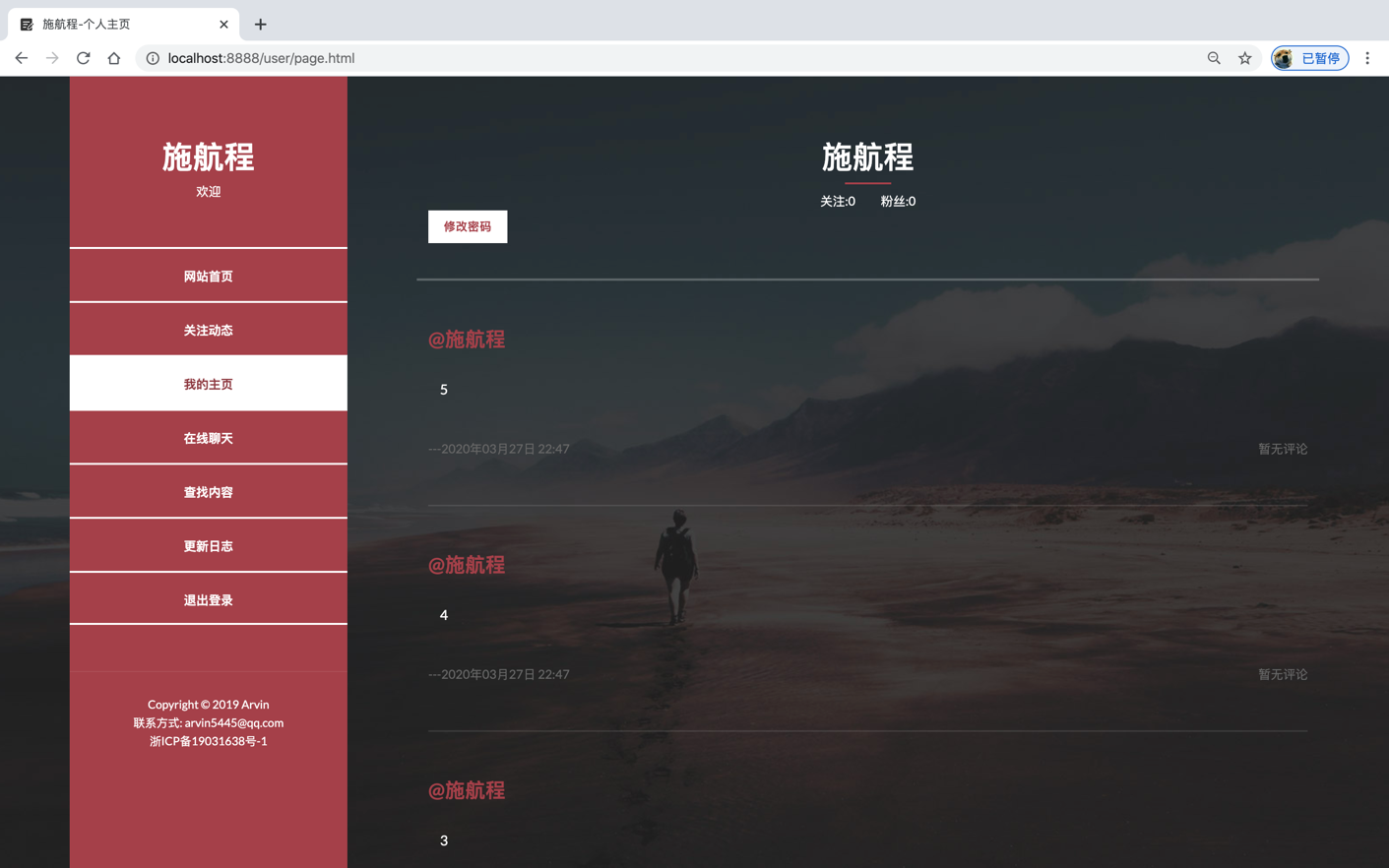
用户在成功登录本系统以后，可以进入自己（通过点击个人主页或者自己的用户名）或者其他用户（通过点击此用户的用户名）的个人主页，并查看此人的关注数量和粉丝数量，以及他的所有博客内容，按时间降序排序。

### 6.2.5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

### 6.2.6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。

### 6.2.7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。

### 6.2.8多人聊天测试

用户进入聊天室和离开聊天室提示支持。

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。

### 6.2.9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



### 6.2.10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。

## 6.3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

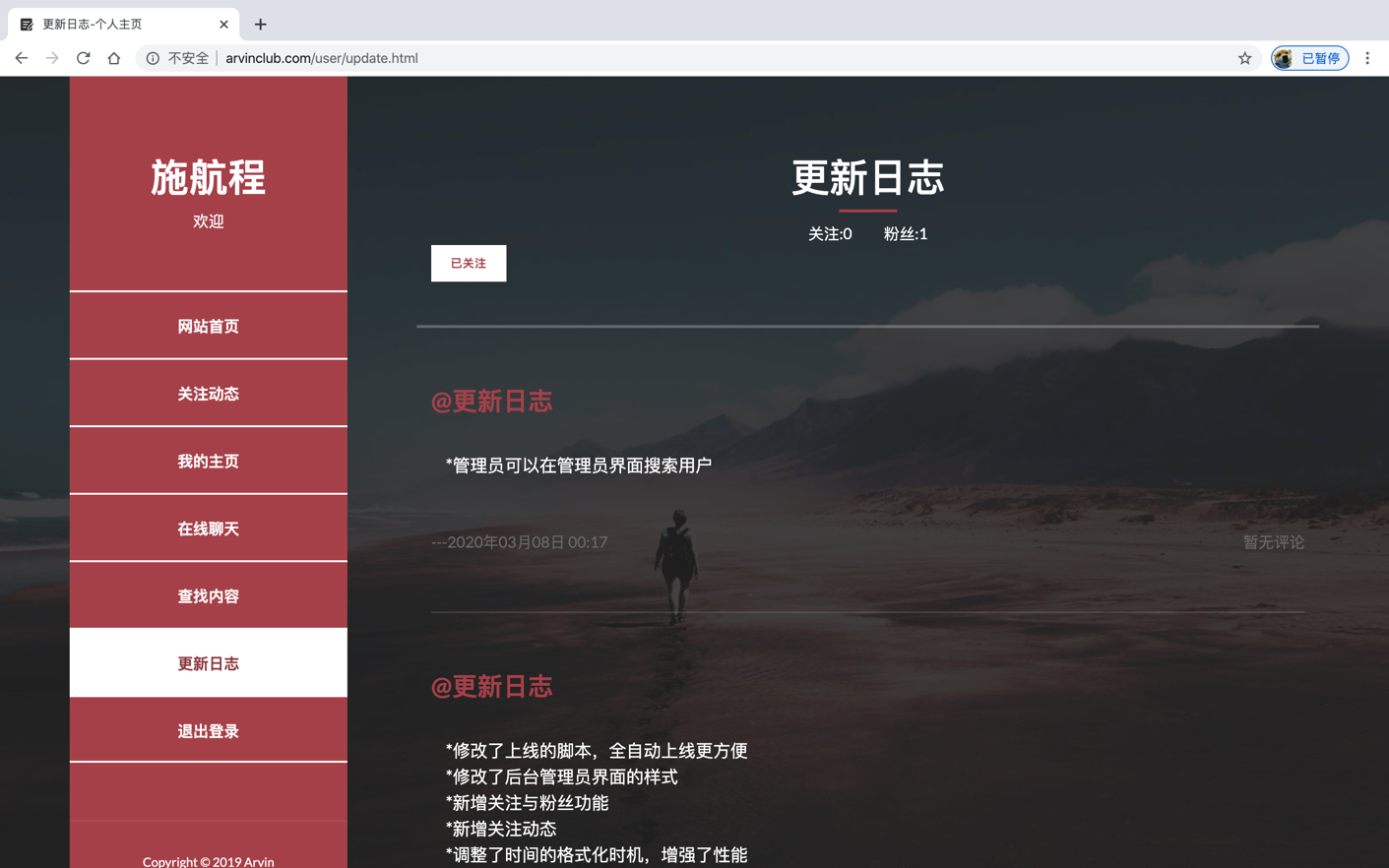
1)alpha阶段:在软件开发过程中由最终用户对软件进行检查

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

### 6.3.1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。

### 6.3.2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。域名：[www.arvinclub.com](http://www.arvinclub.com)。

# 7总结

\*\*\*\*\*\*\*

# 后 记

在本次毕业设计过程中，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，和指导老师，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；