**武汉大学计算机学院**

**本科生课程设计报告**

Whisper匿名提问箱

总体设计与实现

专 业 名 称 ：计算机科学与技术

课 程 名 称 ：JavaEE架构程序设计

团 队 名 称 ：Whisper小组

指 导 教 师 ：刘峰 教授

团 队 成 员 一：马薪宇（2019302080309）

团 队 成 员 二：张嘉伟（2019302141095）

团 队 成 员 三：李嘉琦（2019302080330）

二○二一年十二月

**郑 重 声 明**

本团队呈交的设计报告，是在指导老师的指导下，独立进行实验工作所取得的成果，所有数据、图片资料真实可靠。尽我所知，除文中已经注明引用的内容外，本设计报告不包含他人享有著作权的内容。对本设计报告做出贡献的其他个人和集体，均已在文中以明确的方式标明。本设计报告的知识产权归属于培养单位。

团队成员签名： 马薪宇、张嘉伟、李嘉琦 日期： 2021.12.27

摘 要

本次实验的实验目的是编写一个Whisper匿名提问箱的网站，功能与手机TapeApp的功能类似，实现用户发布某个主题的Whisper提问箱，其他用户可以进行匿名提问，当发布人回答问题后，匿名问题和回答就会公布，用户也可对Whisper提问箱的主题进行管理，实现增删查等基本功能。

实验设计主要基于SSM框架，包括Spring，Spring MVC和MyBatis，其中Spring MVC负责请求的转发和视图管理；Spring实现业务对象管理；MyBatis作为数据对象的持久化引擎，整个项目分为四层：View层即表现层，Controller层，Service层，Dao层。基于框架的分层开发简化了项目的开发流程，使整个开发过程更清晰易懂。

实验内容主要包括：利用MySQL建立数据库表，包括用户表格，主题表格，评论表格等，在Dao层定义实体类，编写接口函数，通过Mybatis的配置文件与数据库交互进行耦合，基本的数据库操作包括对表格进行增删改查等操作；在Service层可以装载Bean，利用控制反转就不需要我们显式地“new”对象了，对象间的依赖关系交给Spring控制；Controller层调用业务逻辑处理，与前端灵活地进行交互。

实验结果是编写出设想的Whisper匿名提问箱网站，可以实现用户注册、登录、发布提问箱、提问、回答等操作。

**关键词：**Spring；MyBatis；Spring MVC；MySQL数据库；匿名提问

目 录

[1. 实验目的和意义](#_Toc26942)

[1.1实验目的 5](#_Toc7008)

[1.2实验意义 5](#_Toc10089)

[1.3需求分析 6](#_Toc18162)

[2. 实验设计](#_Toc23701)

[2.1概述 6](#_Toc16738)

[2.2小组分工 7](#_Toc23177)

[2.3 E-R图 7](#_Toc20062)

[2.4用例图 7](#_Toc13105)

[2.5实验原理 8](#_Toc2353)

[2.5.1 Maven配置 8](#_Toc26999)

[2.5.2建立数据库表 11](#_Toc4799)

[2.5.3 Spring整合Mybtis和Spring MVC 12](#_Toc9674)

[2.5.4 数据持久层 16](#_Toc14448)

[2.5.5 业务逻辑层 21](#_Toc29659)

[2.5.6控制逻辑层 23](#_Toc9072)

[2.5.7视图层 26](#_Toc31587)

[3. 实验结论](#_Toc23273)

[3.1实验结果 33](#_Toc29671)

[3.1.1用户登录 33](#_Toc31018)

[3.1.2用户查看提问箱 34](#_Toc13009)

[3.1.3匿名提问流程 36](#_Toc2160)

[3.2实验总结 38](#_Toc189)

[3.3项目特点 39](#_Toc32333)

[3.4需要完善的方面 40](#_Toc32231)

[4. 实验心得](#_Toc21290)

[5. 教师评语评分](#_Toc24671)

**1. 实验目的和意义**

**1.1实验目的**

本实验的目的是编写一个Whisper匿名提问箱的网站，功能与手机上的TapeApp的功能类似，具体实现如下：

**1.1.1目的一：用户注册与登录**

新用户进入网站时可以进行用户的注册，之后便可使用用户信息进行用户登录，网站在取得用户输入的信息后与数据库中的信息进行对比，如果信息无误便可进入网站首页进行网站的使用。

**1.1.2目的二：用户发布提问箱**

用户进入网站后可以发布某个主题的Whisper提问箱，其他用户可以进行匿名提问，同时当用户回答了提问箱中的问题后，匿名用户问题和发布提问箱的用户的回答就会公布，发布此提问箱的用户也可对Whisper提问箱的主题进行管理，实现增删查等基本功能。

**1.1.3目的三：用户对个人信息进行管理**

用户可对自己的个人信息进行修改，修改后后台数据库中用户的个人信息会同步进行更新。

**1.2实验意义**

新媒体背景下,人们的社会交往呈现出虚拟化加剧的现象,例如微博、微信、QQ等社交软件在人们日常生活中的运用,相比于这种熟人社交平台,匿名社交软件在人际传播过程中产生了新的特点社交网络的高速发展衍生出众多细分需求的空白，匿名社交应运而生。

以综合社交为主的社交媒体市场已经进入成熟阶段，许多较小的细分需求衍生出来，社交行业不同细分领域的需求一直存在，有已经被发掘的职业、婚恋和匿名，每个行业的发展速度都非常快.而匿名社交发展是顺应用户心智而为，当用户心里的压力在微博、微信等圈子里得不到释放时，基于通讯录的熟人匿名社交粉墨登场，作为一个树洞，满足了用户的猎奇心理;而在移动应用爆炸式发展的年代，仅仅满足用户需求是远远不够的，只有占领用户的心智，真正触到用户的痛点才能成功。

网络聊天的出现是为了方便人们沟通，但是渐渐的，这个“方便”也只是缩短了时间和距离，而并没有实现更加坦白的沟通，社交媒体依然没有真正实现“人人皆可说“的幻境，它只是给了每个人一支关掉的话筒，怎么能打开，还得靠自己。

Whisper匿名提问信箱提供了一种简便真诚的社交方式，就是为了能够打破这些束缚，实现提问双方的轻松社交。

**1.3需求分析**

用户可以注册账号，注册好的账号会存入数据库中；

用户可以输入账号密码进行登录，与数据库信息进行对比确定是否登陆成功；

用户可以修改自己的个人信息，更新到数据库中；

用户可以选择模糊搜索某个主题的提问箱；

用户可以进入到提问箱中匿名进行提问，问题只会被提问箱的主人看到；

当提问箱的主人回答了此问题后，问题和回答会被公开所有人可见；

用户可以对自己的提问箱和其中的问题进行删除；

1. **实验设计**

**2.1概述**

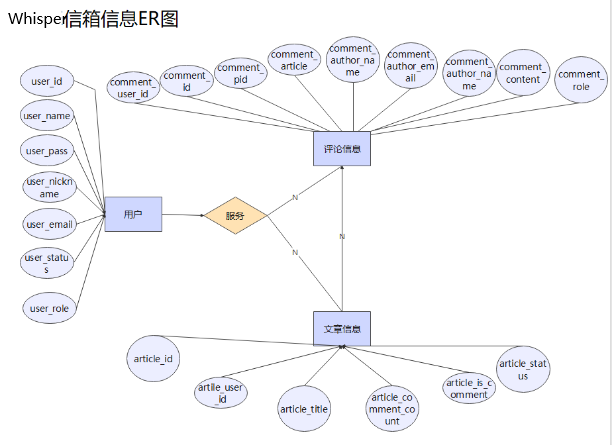
实验设计主要是基于SSM框架，包括Spring，Spring MVC和MyBatis，同时利用MySQL建立数据库表格，MyBatis作为数据对象的持久化引擎，Spring实现业务对象管理，Spring MVC负责请求的转发和视图管理，基于SSM框架的分层开发简化了项目的开发流程，使整个开发过程更清晰易懂。

先写底层的实体类，再在Dao中写函数接口，在Resource/Mapper/\*.xml中与数据库连接，相关配置均在spring-mybatis.xml中配置约束；在Service层中调用Dao层的函数写业务层函数接口；最后在Controller层调用业务层逻辑函数，实现与前端交互转发页面等功能，相关配置在spring-mvc.xml中配置约束，前端的相关配置在web.xml中。

**2.2小组分工**

|  |  |
| --- | --- |
| 小组成员 | 主要负责任务 |
| 马薪宇 | 负责创建数据持久层的实体类、映射文件的编写、业务逻辑层的编写、Spring整合Mybatis的配置和实验报告的撰写 |
| 张嘉伟 | 负责控制逻辑层、业务逻辑层的编写、View层与控制器的交互、Spring整合Spring MVC的配置和实验报告的撰写 |
| 李嘉琦 | 负责数据库的设计构建、映射文件的编写、PPT制作和实验报告的撰写 |

**2.3 E-R图**



**2.4用例图**

根据项目的需求分析，我们得到如下的用例图：

****

**2.5实验原理**

**2.5.1 Maven配置**

全局配置编译约束，加载/\*.xml文件，使Dao层对应的映射文件能在配置文件中被加载

<build>  
 <resources>  
 <!-- 编译之后包含xml -->  
 <resource>  
 <directory>src/main/java</directory>  
 <includes>  
 <include>\*\*/\*.xml</include>  
 </includes>  
 <filtering>true</filtering>  
 </resource>  
 <!-- 编译之后包含xml和properties -->  
 <resource>  
 <directory>src/main/resources</directory>  
 <includes>  
 <include>\*\*/\*</include>  
 </includes>  
 <filtering>true</filtering>  
 </resource>  
 </resources>  
</build>

连接池的配置，为了成功连接数据库的依赖

<!-- jdbc驱动包 -->  
<dependency>  
 <groupId>mysql</groupId>  
 <artifactId>mysql-connector-java</artifactId>  
 <version>8.0.11</version>  
</dependency>  
<!-- 添加连接池Druid支持 -->  
<dependency>  
 <groupId>com.alibaba</groupId>  
 <artifactId>druid</artifactId>  
 <version>1.0.16</version>  
</dependency>

添加Mybatis依赖

<!-- 添加mybatis支持 -->  
<dependency>  
 <groupId>org.mybatis</groupId>  
 <artifactId>mybatis</artifactId>  
 <version>3.4.0</version>  
</dependency>  
<dependency>  
 <groupId>org.mybatis</groupId>  
 <artifactId>mybatis-spring</artifactId>  
 <version>1.3.0</version>  
</dependency>

添加junit依赖，可以用于管理测试用例、定义测试代码、定义测试环境、检测测试结果

<!-- 添加junit支持 -->  
<dependency>  
 <groupId>junit</groupId>  
 <artifactId>junit</artifactId>  
 <version>4.12</version>  
 <scope>test</scope>  
</dependency>  
<dependency>  
 <groupId>commons-fileupload</groupId>  
 <artifactId>commons-fileupload</artifactId>  
 <version>1.3.3</version>  
</dependency>  
<dependency>  
 <groupId>commons-io</groupId>  
 <artifactId>commons-io</artifactId>  
 <version>2.4</version>  
</dependency>

添加jackson依赖，jackson可以轻松的将Java对象转换成json对象和xml文档，同样也可以将json、xml转换成Java对象

<!-- jackson -->  
<dependency>  
 <groupId>com.fasterxml.jackson.core</groupId>  
 <artifactId>jackson-databind</artifactId>  
 <version>2.9.0</version>  
</dependency>  
<dependency>  
 <groupId>com.fasterxml.jackson.core</groupId>  
 <artifactId>jackson-core</artifactId>  
 <version>2.9.0</version>  
</dependency>  
<dependency>  
 <groupId>com.fasterxml.jackson.core</groupId>  
 <artifactId>jackson-annotations</artifactId>  
 <version>2.9.0</version>  
</dependency>

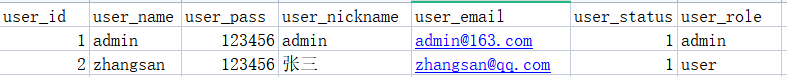
添加servlet，servlet的作用是处理请求，服务器会把接收到的请求交给servlet来处理，在servlet中通常需要：接收请求数据；处理请求；完成响应。

<!-- 添加sevlet支持 -->  
<dependency>  
 <groupId>javax.servlet</groupId>  
 <artifactId>javax.servlet-api</artifactId>  
 <version>3.1.0</version>  
 <scope>provided</scope>  
</dependency>

<!--spring test支持-->  
<dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-test</artifactId>  
 <version>${spring.version}</version>  
</dependency>  
<!--spring mvc支持-->  
<dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-webmvc</artifactId>  
 <version>${spring.version}</version>  
</dependency>

**2.5.2建立数据库表**

User表格：

****

其中

1) user\_id作为主键

2）user\_name作为用户名

3）user\_pass为用户密码

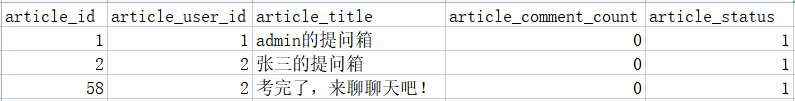
4) user\_nickname为用户昵称

5）user\_email为用户邮箱

6）user\_status为用户状态，表示是否被禁言的状态

7）user\_role为用户权限，有管理员和普通用户两种权限

Article表格：



其中

1）article\_id为提问箱id

2) article\_user\_id为提问箱所有者的id

3) article\_title为提问箱的标题

4）article\_comment\_count为提问箱的评论数量

5）article\_status表示提问箱是否可见

Comment表格：

****

其中

1）comment\_id为评论id

2）comment\_pid为当前评论上一个评论（回复上一个评论）的id

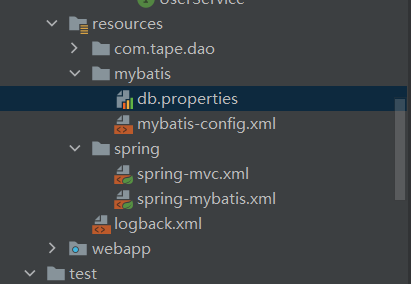
3）comment\_article为评论所在的文章id

4）comment\_content为评论的内容

5）comment\_role为评论的权限，表示是否可见

6）comment\_user\_id为评论作者的用户id

**2.5.3 Spring整合Mybtis和Spring MVC**



（1）db.properies文件配置数据库

#MySQL  
mysql.driver=com.mysql.cj.jdbc.Driver  
mysql.url=jdbc:mysql://127.0.0.1:3306/tape?\  
 useUnicode=true&characterEncoding=utf8&\  
 serverTimezone=UTC&useSSL=false  
mysql.username=root  
mysql.password=130218

（2）mybatis-config负责全局配置，包括JDBC连接数据库等配置

<?xml version="1.0" encoding="UTF-8" ?>  
<!DOCTYPE configuration  
 PUBLIC "-//mybatis.org//DTD Config 3.0//EN"  
 "http://mybatis.org/dtd/mybatis-3-config.dtd">  
<configuration>  
 <!-- 全局配置 -->  
 <settings>  
 <!--允许 JDBC 支持自动生成主键-->  
 <setting name="useGeneratedKeys" value="false"/>  
 <!--是否开启自动驼峰命名规则（camel case）映射，即从经典数据库列名 A\_COLUMN 到经典  
 Java 属性名 aColumn 的类似映射。 -->  
 <setting name="mapUnderscoreToCamelCase" value="true"/>  
  
 </settings>  
  
 <plugins>  
 <plugin interceptor="com.github.pagehelper.PageHelper">  
 <!--<property name="dialect" value="mysql"/>-->  
 <property name="offsetAsPageNum" value="false"/>  
 <property name="rowBoundsWithCount" value="false"/>  
 <property name="pageSizeZero" value="true"/>  
 <property name="reasonable" value="true"/>  
 <property name="supportMethodsArguments" value="false"/>  
 <property name="returnPageInfo" value="none"/>  
 </plugin>  
 </plugins>  
  
</configuration>

1. spring-mybatis，Spring整合Mybatis的配置文件，包括数据库的连接、映射文件的扫描和mybatis的事务扫描等的配置

数据库的连接配置：

<!--加载db.properties-->  
<context:property-placeholder location="classpath:mybatis/db.properties"/>  
  
<!--配置druid连接池-->  
<bean id="dataSource" class="com.alibaba.druid.pool.DruidDataSource" init-method="init" destroy-method="close">  
 <!-- 基本属性 url、user、password -->  
 <property name="driverClassName" value="${mysql.driver}" />  
 <property name="url" value="${mysql.url}" />  
 <property name="username" value="${mysql.username}" />  
 <property name="password" value="${mysql.password}" />  
  
 <!-- 配置初始化大小、最小、最大 -->  
 <property name="initialSize" value="1" />  
 <property name="minIdle" value="1" />  
 <property name="maxActive" value="20" />  
  
 <!-- 配置获取连接等待超时的时间 -->  
 <property name="maxWait" value="60000" />  
  
 <!-- 配置间隔多久才进行一次检测，检测需要关闭的空闲连接，单位是毫秒 -->  
 <property name="timeBetweenEvictionRunsMillis" value="60000" />  
  
 <!-- 配置一个连接在池中最小生存的时间，单位是毫秒 -->  
 <property name="minEvictableIdleTimeMillis" value="300000" />  
  
 <property name="validationQuery" value="SELECT 'x'" />  
 <property name="testWhileIdle" value="true" />  
 <property name="testOnBorrow" value="false" />  
 <property name="testOnReturn" value="false" />  
  
 <!-- 打开PSCache，并且指定每个连接上PSCache的大小 -->  
 <property name="poolPreparedStatements" value="true" />  
 <property name="maxPoolPreparedStatementPerConnectionSize" value="20" />  
  
 <!-- 配置监控统计拦截的filters，去掉后监控界面sql无法统计 -->  
 <property name="filters" value="stat" />  
</bean>

mybatis的核心配置：

<bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">  
 <!--数据库连接池-->  
 <property name="dataSource" ref="dataSource"/>  
 <!--加载mybatis全局配置文件-->  
 <property name="configLocation" value="classpath:mybatis/mybatis-config.xml"/>  
 <!--mapper.xml所在位置-->  
 <property name="mapperLocations" value="classpath\*:com/tape/dao/\*.xml" />  
 <!--指定需要使用别名的PO类所在的包-->  
 <property name="typeAliasesPackage" value="com.tape.entity" />  
</bean>

配置映射文件，注入Bean：

<!--mapper扫描器-->  
<bean class="org.mybatis.spring.mapper.MapperScannerConfigurer">  
 <!--如果需要扫描多个包，中间使用半角逗号隔开-->  
 <property name="basePackage" value="com.tape.dao"></property>  
 <property name="sqlSessionFactoryBeanName" value="sqlSessionFactory"/>  
</bean>

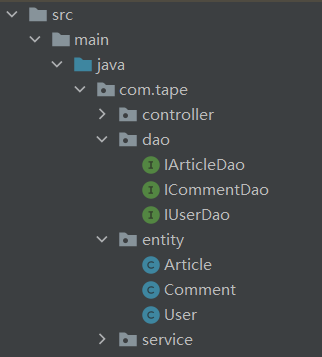
配置事务处理器：

<!-- 对mybatis操作数据事务控制，spring使用jdbc的事务控制类 -->  
<bean id="transactionManager" class="org.springframework.jdbc.datasource.DataSourceTransactionManager">  
 <!-- 数据源dataSource在spring-mybatis.xml中配置了 -->  
 <property name="dataSource" ref="dataSource"/>  
</bean>

1. spring-mvc，spring整合spring MVC的配置文件

<context:component-scan base-package="com.tape"/>  
  
<!-- 解决映射器和适配器的配置注解配置 -->  
<mvc:annotation-driven></mvc:annotation-driven>  
<bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">  
 <property name="prefix" value="/WEB-INF/pages/"></property>  
 <property name="suffix" value=".jsp"></property>  
</bean>

**2.5.4 数据持久层**



DAO层主要是做数据持久层的工作，负责与数据库进行联络的一些任务都封装在此，DAO层的设计首先是设计DAO的接口，然后在Spring的配置文件中定义此接口的实现类，然后就可在模块中调用此接口来进行数据业务的处理，而不用关心此接口的具体实现类是哪个类，显得结构非常清晰，DAO层的数据源配置，以及有关数据库连接的参数都在Spring的配置文件中进行配置。

下面以User类中部分代码为例介绍数据持久层的创建过程和调用逻辑，详细的接口介绍在开发文档中将会介绍。

（1）entity创建实体类，包括文章类，评论类和用户类，其中声明它们的属性，如用户的User类如下,其中Serializable是提供序列化的接口，其作用是将各种对象的状态保存下来，并且可以在需要时再将对象恢复：

public class User implements Serializable {  
  
 private Integer userId;  
 private String userName;  
 private String userPass;  
 private String userNickname;  
 private String userEmail;  
 private Integer userStatus;  
 */\*\*  
 \* 用户角色：admin/user  
 \*/* private String userRole;  
 */\*\*  
 \* 用户角色：admin/user  
 \*/* public Integer getUserId() {return userId;}  
  
 public void setUserId(Integer userId) { this.userId = userId; }  
  
 public String getUserName() {return userName; }  
  
 public void setUserName(String userName) { this.userName = userName; }  
  
 public String getUserPass() {return userPass;}  
  
 public void setUserPass(String userPass) {this.userPass = userPass;}  
  
 public String getUserNickname() { return userNickname;}  
  
 public void setUserNickname(String userNickname)

{this.userNickname = userNickname;}  
  
 public String getUserEmail() {return userEmail; }  
  
 public void setUserEmail(String userEmail) { this.userEmail = userEmail;}  
  
 public Integer getUserStatus() { return userStatus;}  
  
 public void setUserStatus(Integer userStatus) {this.userStatus = userStatus; }  
  
 public String getUserRole() {return userRole;}  
  
 public void setUserRole(String userRole) { this.userRole = userRole;}  
  
 @Override  
 public String toString() {  
 return "User{" +  
 "userId=" + userId +  
 ", userName='" + userName + '\'' +  
 ", userPass='" + userPass + '\'' +  
 ", userNickname='" + userNickname + '\'' +  
 ", userEmail='" + userEmail + '\'' +  
 ", userStatus=" + userStatus +  
 ", userRole='" + userRole + '\'' +  
 '}';  
 }  
}

其他类的声明同样

（2）在dao中创建数据持久层的函数声明接口，以User为例，添加Mapper注解，取其中的两个函数作为演示例子，声明insert（）、ListUser（)和getUserByName()的函数接口，其中@Mapper注解是mybatis的注解，是用来说明这个是一个Mapper：

package com.tape.dao;  
  
import com.tape.entity.User;  
import org.apache.ibatis.annotations.Mapper;  
  
import java.util.List;  
  
*/\*\*  
 \* 用户的持久层接口  
 \*/*@Mapper  
public interface IUserDao {

*/\*\*  
 \* 添加  
 \** ***@param*** *user 用户  
 \** ***@return*** *影响行数  
 \*/* int insert(User user);

*/\*\*  
 \* 获得用户列表  
 \*  
 \** ***@return*** *用户列表  
 \*/* List<User> listUser() ;  
  
 */\*\*  
 \* 根据用户名查用户  
 \*  
 \** ***@param*** *name 用户名  
 \** ***@return*** *用户  
 \*/* User getUserByName(String name) ;

}

1. 再将函数声明与对数据库操作的映射文件联系在一起，从而达到项目可以利用接口访问数据库的功能，映射文件.xml如下，其中都是sql语句，对数据库的表格进行CRUD的操作：

*<mapper namespace="com.tape.dao.IUserDao">  
 <resultMap id="BaseResultMap" type="com.tape.entity.User">  
 <id column="user\_id" property="userId" jdbcType="INTEGER"/>  
 <result column="user\_name" property="userName" jdbcType="VARCHAR"/>  
 <result column="user\_pass" property="userPass" jdbcType="VARCHAR"/>  
 <result column="user\_nickname" property="userNickname" jdbcType="VARCHAR"/>  
 <result column="user\_email" property="userEmail" jdbcType="VARCHAR"/>  
 <result column="user\_status" property="userStatus" jdbcType="INTEGER"/>  
 <result column="user\_role" property="userRole" jdbcType="VARCHAR"/>  
 </resultMap>  
  
 <sql id="tb">user</sql>  
  
 <sql id="Base\_Column\_List">  
 user\_id, user\_name, user\_pass, user\_nickname, user\_email, user\_url, user\_avatar,  
 user\_last\_login\_ip, user\_register\_time, user\_last\_login\_time, user\_status, user\_role  
 </sql>*

*<!--添加用户-->*

*<insert id="insert" parameterType="com.liuyanzhao.ssm.blog.entity.User" useGeneratedKeys="true" keyProperty="userId">*

*insert into*

*<include refid="tb"/>*

*(user\_id, user\_name, user\_pass,*

*user\_nickname, user\_email, user\_url,*

*user\_avatar, user\_last\_login\_ip, user\_register\_time,*

*user\_last\_login\_time, user\_status, user\_role)*

*values (#{userId,jdbcType=INTEGER}, #{userName,jdbcType=VARCHAR}, #{userPass,jdbcType=VARCHAR},*

*#{userNickname,jdbcType=VARCHAR}, #{userEmail,jdbcType=VARCHAR}, #{userUrl,jdbcType=VARCHAR},*

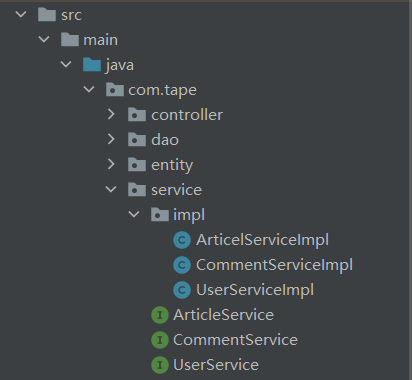
*#{userAvatar,jdbcType=VARCHAR}, #{userLastLoginIp,jdbcType=VARCHAR}, #{userRegisterTime,jdbcType=TIMESTAMP},*

*#{userLastLoginTime,jdbcType=TIMESTAMP}, #{userStatus,jdbcType=INTEGER}, #{userRole,jdbcType=VARCHAR})*

*</insert>  
  
 <!--用户列表查询-->  
 <select id="listUser" parameterType="com.tape.entity.User"  
 resultType="com.tape.entity.User">  
 SELECT  
 <include refid="Base\_Column\_List"/>  
 FROM  
 <include refid="tb"/>  
 ORDER BY `user\_status` ASC  
 </select>  
  
 <!--根据用户名查用户-->  
 <!--通过这个查出来的用户再getuserPass和userId，因为查article要用userId不能用username直接查-->  
 <select id="getUserByName" parameterType="String" resultType="com.tape.entity.User">  
 SELECT  
 <include refid="Base\_Column\_List"/>  
 FROM  
 <include refid="tb"/>  
 <where>  
 user\_name=#{value}  
 </where>  
 limit 1  
 </select>  
  
</mapper>*

<resultMap>实现了实体类和数据库中的名称转换，而后的数据库操作都与之前搭建好的Dao层的声明相对应，再结合之前spring对mybatis的配置和约束，自此实现了数据持久层的配置，之后程序在业务逻辑层只要调用Dao层的接口即可使用这些函数声明。

**2.5.5 业务逻辑层**



Service层主要负责业务模块的逻辑应用设计。同样是首先设计接口，再设计其实现的类，接着再Spring的配置文件中配置其实现的关联。这样我们就可以在应用中调用Service接口来进行业务处理。Service层的业务实现，具体要调用到已定义的DAO层的接口，封装Service层的业务逻辑有利于通用的业务逻辑的独立性和重复利用性，程序显得非常简洁。

下面同样以UserService中部分代码介绍业务逻辑层的创建过程和调用逻辑，详细的接口介绍在开发文档中将会介绍。

1. UserService接口声明函数，以UserService为例，添加Service注解，声明listUser()和getUserById（）函数接口，其中@Service("userService")注解表示给当前类命名一个别名，方便注入到其他需要用到的类中其,@Override 注解是用来指定方法重写的，只能修饰方法并且只能用于方法重写，不能修饰其它的元素，它可以强制一个子类必须重写父类方法或者实现接口的方法：

package com.tape.service.impl;  
import com.tape.dao.IUserDao;  
import com.tape.entity.User;  
import com.tape.service.UserService;  
  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Service;  
  
import java.util.List;  
  
@Service("userService")  
public class UserServiceImpl implements UserService {  
  
 @Autowired  
 private IUserDao userDao;  
  
 @Override  
 public List<User> listUser() {  
 List<User> userList = userDao.listUser();  
 return userList;  
 }  
  
 @Override  
 public User getUserByName(String username){  
 User user = userDao.getUserByName(username);  
 return user;  
 }  
}

（2）ServiceImpl实现Service中的接口，向顶层调用数据持久层的函数接口轻松地实现对数据库的操作，@Autowired是一个spring容器配置的一个注解，它可以自动装配已经装载进bean中的类，因此生成对象时不需要多次new对象，只需要使用此注解即可从装载好的工厂中生成一个需要的对象：

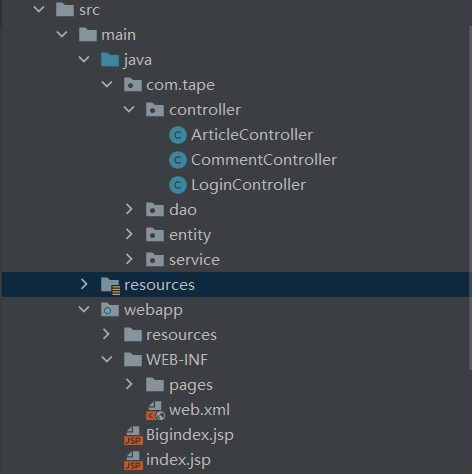
package com.tape.service.impl;  
import com.tape.dao.IUserDao;  
import com.tape.entity.User;  
import com.tape.service.UserService;  
  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Service;  
import java.util.List;  
  
@Service("userService")  
public class UserServiceImpl implements UserService {  
  
 @Autowired  
 private IUserDao userDao;  
  
 @Override  
 public List<User> listUser() {  
 List<User> userList = userDao.listUser();  
 return userList;  
 }  
  
 @Override  
 public User getUserByName(String username){  
 User user = userDao.getUserByName(username);  
 return user;  
 }  
}

加上spring对Service的配置，自此业务逻辑层也已经完成，

**2.5.6控制逻辑层**

Controller层即SpringMVC中的核心控制器部分，负责具体的前端url请求后的处理，在此层里面要调用Serice层的接口来控制业务流程，控制的配置也同样是在Spring的配置文件里面进行，针对具体的业务流程，会有不同的控制器，我们具体的设计过程中可以将流程进行抽象归纳，设计出可以重复利用的子单元流程模块，这样不仅使程序结构变得清晰，也大大减少了代码量。

Controller模块下的三个控制器如下图所示，包含三个核心控制器：ArticleController（负责处理每个提问箱相关的请求）、CommentController（负责处理每个留言相关的请求）、LoginController（负责处理登录请求）。



下面以ArticleController中部分代码介绍控制逻辑层的创建过程和调用逻辑，以及前端页面的转发逻辑，详细的函数介绍在开发文档中将会介绍。

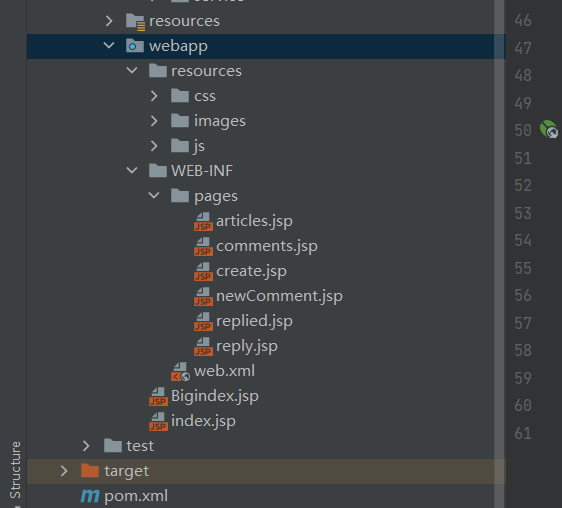
ArticleController中调用Service层中的函数接口，从而可以对底层的数据进行处理，同时可以将数据与前端进行交互，将数据传到对应的页面，具体的方法如下，其中使用@Controller 注解，在对应的方法上，视图解析器可以解析return 的jsp、html页面，并且跳转到相应页面，使用 @RequestMapping 来映射请求，也就是通过它来指定控制器可以处理哪些URL请求，相当于Servlet中在web.xml中配置的映射作用一致：

package com.tape.controller;  
  
import com.tape.entity.Article;  
import com.tape.service.ArticleService;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Controller;  
  
import org.springframework.web.bind.annotation.RequestMapping;  
import org.springframework.web.servlet.ModelAndView;  
  
import java.util.List;  
  
@Controller("ArticleController")  
@RequestMapping("article")  
public class ArticleController {  
 @Autowired  
 public ArticleService articleService;  
  
 @RequestMapping("")  
 public ModelAndView showAll(int userId)  
 {  
 System.*out*.println("articlecontroller执行了");  
 List<Article> articles = articleService.getArticleByUserIdService(userId);  
 ModelAndView mv = new ModelAndView();  
 mv.addObject("articles",articles);  
 mv.addObject("userId",userId);  
 mv.addObject("who","我的");  
 mv.setViewName("articles");  
 return mv;  
 }  
  
 @RequestMapping("create")  
 public ModelAndView create(int userId)  
 {  
 ModelAndView mv = new ModelAndView();  
 mv.addObject("userId",userId);  
 mv.setViewName("create");  
 return mv;  
 }  
 @RequestMapping("commit")  
 public String createCommit(Article article)  
 {  
 //articleService.insert(article);  
 System.*out*.println("创建成功"+article.getArticleUserId());  
 articleService.insertService(article);  
 return "redirect:/article?userId="+article.getArticleUserId();  
 }  
  
 @RequestMapping("all")  
 public ModelAndView showOthers(int userId)  
 {  
 ModelAndView mv =new ModelAndView();  
 List<Article> articles = articleService.listAllNotWithContent();  
 mv.addObject("articles",articles);  
 mv.addObject("userId",userId);  
 mv.addObject("who","全部的");  
 mv.setViewName("articles");  
 return mv;  
 }  
}

1. @RequestMapping的使用。该注解可以用在类上，也可以用在方法上。特别需要注意的是跳转路径的url是使用相对路径还是绝对路径，@RequestMapping注解会将所有路径解析成相对路径，而在转发或重定向时以及在View中（jsp文件中）会根据是否以/开头解析成相对路径或绝对路径。
2. 返回值的多种方式。上图中articleController中的函数用到了两种不同的返回值类型，String和ModelAndView。String类型的返回值会被自动解析成请求路径，ModelAndView返回值则更符合MVC架构，可以将模型与视图绑定，一同发给视图层。
3. 转发与重定位。转发是指同一个请求在服务器端将其发送给另一个handler处理，在客户端并没有产生新的请求，属于完全的服务端处理；重定位是指服务器接受到请求后向客户端发出一个响应，使客户端重新发出一个指定的请求到服务器，这个过程需要客户端和服务器共同参与，在客户端的眼中属于重新发送了一个请求。返回值为String类型的函数实现转发与重定位需要在url前加上“forward：”或“redirect：”，返回值为ModelAndView类型的函数实现转发可直接设置视图名，实现重定位也需要在视图名前加“redirect：”。

**2.5.7视图层**

View层即MVC架构中的v，也即视图层，是Web服务器处理完客户端请求后返回给客户端的信息。在SpringMVC中，View层的内容由jsp文件组成。jsp文件格式是一种在html页面内部嵌入java语句的格式，SpringMVC就是基于servlet与jsp开发的工具，能够使我们方便的在jsp中使用Controller中的数据，并进行页面间数据的交互。如下图所示，webapp下面的文件结构为resources（保存为静态资源，需要设置为资源文件夹）、WEB-INF（保存tomcat加载时需要读取的配置信息web.xml以及被安全访问的pages文件夹），根目录下保存有index页面作为tomcat默认加载的初始页面。



1. web.xml配置文件为tomcat启动时读取的配置文件，在该文件中需要配置SpringMVC的前端控制器、Spring配置文件的位置、监听器、过滤器等web应用的内容，具体如下所示：

servlet标签下配置SpringMVC的前端控制器：

<servlet>  
 <servlet-name>SpringMVCDispatcherServlet</servlet-name>  
<servlet-class>  
 org.springframework.web.servlet.DispatcherServlet  
</servlet-class>  
<!-- 配置初始化参数，用于读取 SpringMVC 的配置文件 -->  
<init-param>  
 <param-name>contextConfigLocation</param-name>  
 <param-value>classpath:spring/spring-mvc.xml</param-value>  
</init-param>  
<!-- 配置 servlet 的对象的创建时间点：应用加载时创建。  
取值只能是非 0 正整数，表示启动顺序 -->  
<load-on-startup>1</load-on-startup>  
</servlet>

配置SpringMVC前端控制器的响应请求匹配模式，“/”表示全部匹配：

<servlet-mapping>  
<servlet-name>SpringMVCDispatcherServlet</servlet-name>  
<url-pattern>/</url-pattern>  
</servlet-mapping>

配置SpringMVC的监听器，在tomcat启动时读取spring配置文件，初始化bean容器：

<!-- 配置 spring 提供的监听器，用于启动服务时加载容器 。  
该间监听器只能加载 WEB-INF 目录中名称为 applicationContext.xml 的配置文件 -->  
 <listener>  
 <listener-class>  
 org.springframework.web.context.ContextLoaderListener  
 </listener-class>  
 </listener>  
 <!-- 手动指定 spring 配置文件位置 -->  
 <context-param>  
 <param-name>contextConfigLocation</param-name>  
 <param-value>classpath:spring/spring-mybatis.xml</param-value>  
 </context-param>

配置springMVC的字符过滤器，用于处理请求参数的乱码问题，同时设置其过滤所有请求。

<filter>  
<filter-name>CharacterEncodingFilter</filter-name>  
<filter-class>  
 org.springframework.web.filter.CharacterEncodingFilter  
</filter-class><!-- 设置过滤器中的属性值 -->  
 <init-param>  
 <param-name>encoding</param-name>  
 <param-value>UTF-8</param-value>  
 </init-param>  
 <!-- 启动过滤器 -->  
 <init-param>  
 <param-name>forceEncoding</param-name>  
 <param-value>true</param-value>  
 </init-param>  
</filter>  
<!-- 过滤所有请求 -->  
<filter-mapping>  
 <filter-name>CharacterEncodingFilter</filter-name>  
 <url-pattern>/\*</url-pattern>  
</filter-mapping>

设置静态资源使用默认过滤器，即不被字符过滤器过滤。

<servlet-mapping>  
 <servlet-name>default</servlet-name>  
 <url-pattern>\*.css</url-pattern>  
</servlet-mapping>  
<servlet-mapping>  
 <servlet-name>default</servlet-name>  
 <url-pattern>\*.js</url-pattern>  
</servlet-mapping>  
<servlet-mapping>  
 <servlet-name>default</servlet-name>  
 <url-pattern>\*.ico</url-pattern>  
</servlet-mapping>  
<servlet-mapping>  
 <servlet-name>default</servlet-name>  
 <url-pattern>\*.jpg</url-pattern>  
</servlet-mapping>  
<servlet-mapping>  
 <servlet-name>default</servlet-name>  
 <url-pattern>\*.png</url-pattern>  
</servlet-mapping>  
<servlet-mapping>  
 <servlet-name>default</servlet-name>  
 <url-pattern>\*.gif</url-pattern>  
</servlet-mapping>

1. 以index.jsp为例，说明视图层jsp页面的结构：

<%@ **page** contentType="text/html; charset=UTF-8" pageEncoding="UTF-8" %>  
<!doctype html>  
<html>  
<head>  
 <meta charset="utf-8">  
 <title>Whisper-Login</title>  
 <link type="text/css" href="resources/css/style.css" rel="stylesheet" />  
</head>

上图为index.xml的头部，注明该文件是html格式，并引入webapp根目录下resources文件夹内的css资源。需要注意的是在文档开头需要用jsp格式注明编码方式为UTF-8和html格式，否则仅仅靠html中表明文档使用utf-8还是会造成页面乱码。

<body>  
  
<script src="resources/js/anime.min.js"></script>  
  
<div class="page">  
 <div class="container">  
 <div class="left">  
 <div class="login">登录</div>  
 <div class="eula">欢迎使用！</div>  
 </div>  
 <div class="right">  
 <svg viewBox="0 0 320 300">  
 <defs>  
 <linearGradient  
 inkscape:collect="always"  
 id="linearGradient"  
 x1="13"  
 y1="193.49992"  
 x2="307"  
 y2="193.49992"  
 gradientUnits="userSpaceOnUse">  
 <stop  
 style="stop-color:#ff00ff;"  
 offset="0"  
 id="stop876" />  
 <stop  
 style="stop-color:#ff0000;"  
 offset="1"  
 id="stop878" />  
 </linearGradient>  
 </defs>  
 <path d="m 40,120.00016 239.99984,-3.2e-4 c 0,0 24.99263,0.79932 25.00016,35.00016 0.008,34.20084 -25.00016,35 -25.00016,35 h -239.99984 c 0,-0.0205 -25,4.01348 -25,38.5 0,34.48652 25,38.5 25,38.5 h 215 c 0,0 20,-0.99604 20,-25 0,-24.00396 -20,-25 -20,-25 h -190 c 0,0 -20,1.71033 -20,25 0,24.00396 20,25 20,25 h 168.57143" />  
 </svg>

上图为index.jsp下面的部分内容，可以看到就是html内容，其中引入了js资源，并设置了一些布局信息，与本课程无关，在此不再赘述。

<div class="form" ><form method="post" action="login">  
 <input type="hidden" value="1" name="userId"/>  
 <label for="userName">账号</label>  
 <input type="text" id="userName" name="userName">  
 <label for="userPass">密码</label>  
 <input type="password" id="userPass" name="userPass">  
 <input type="submit" id="submit" value="提交"></form>  
</div>

上图为index.jsp表单提交。使用<form>标签制作一个表单，action属性为提交后的请求路径。在这里使用的是SpringMVC的实体类绑定方法，只要input标签的name属性与实体类属性一致，即可自动绑定成功。

1. 以articles.jsp为例，说明View层如何通过Controller获得model的属性：

<**c:forEach** items="${articles}" var="article" varStatus="st">  
 <div class="c-sites\_\_item">  
 <div class="c-sites\_\_itemWrapper">  
 <div class="c-sites\_\_itemInner">  
 <div class="c-sites\_\_body"><h2 class="c-sites\_\_title">  
 <a href="/comment?userId=${userId}&articleId=${article.articleId}">  
 ${article.articleTitle}  
 </a>  
 </div>  
 </div>  
 </div>  
 </div></**c:forEach**>

如上图所示，在articles.jsp页面中，可以使用${}的方式访问在Controller中返回的model内的属性，因为model的属性是以键值对的方式，所以在这里访问时只需在${}中引用model的key（例如${userId},userId为model中的一个key），即可得到保存进该model中的value。如下图：

mv.addObject("userId",userId);

（4）配置spring-mvc.xml

在spring-mvc.xml中，首先要设置前端控制器要扫描的核心控制器（即Controller）的位置，在spring-mvc.xml中可以使用如下配置：

<context:component-scan base-package="com.tape"/>

之后需要配置映射器和适配器，在springmvc中可以使用简单的语句实现：

<!-- 一个配置节解决映射器和适配器的配置注解配置 -->  
<mvc:annotation-driven></mvc:annotation-driven>

配置视图解析器，将视图放在安全路径WEB-INF下：

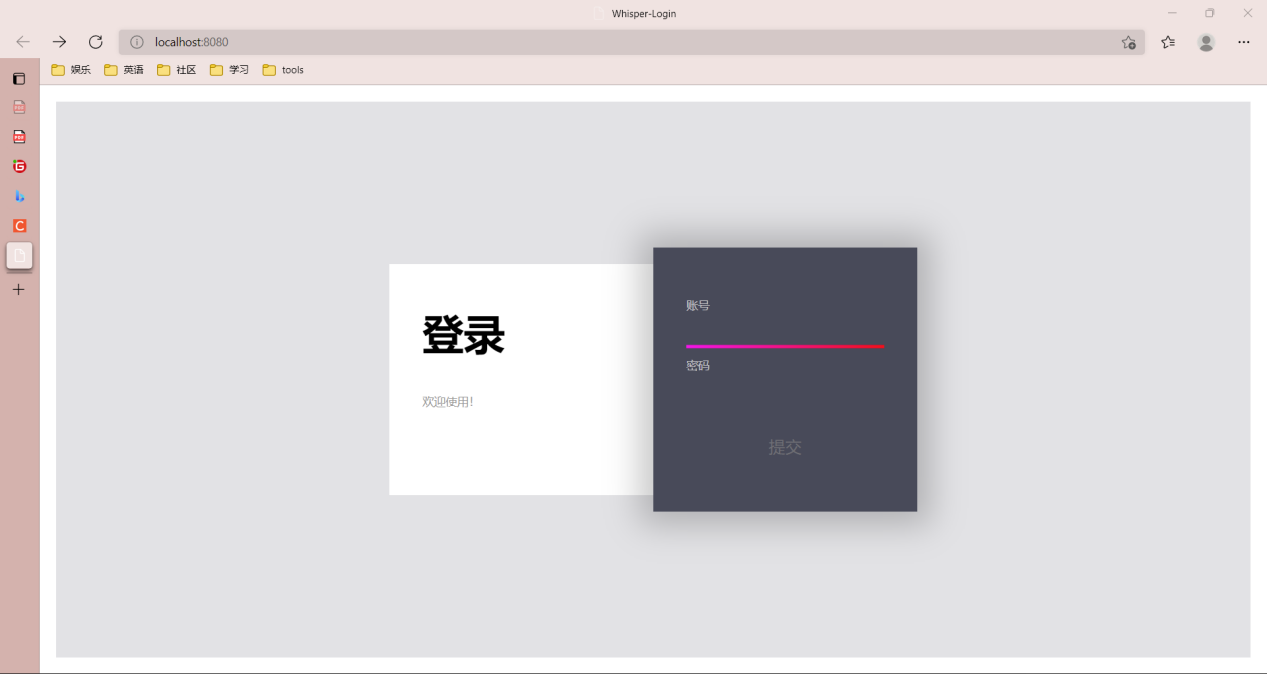
<bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">  
 <property name="prefix" value="/WEB-INF/pages/"></property>  
 <property name="suffix" value=".jsp"></property>  
</bean>

自此页面前端通过Controller与底层逻辑建立了连接，页面上可以显示我们对数据库处理后得到的文件，前端也可以轻松对数据库进行操作。

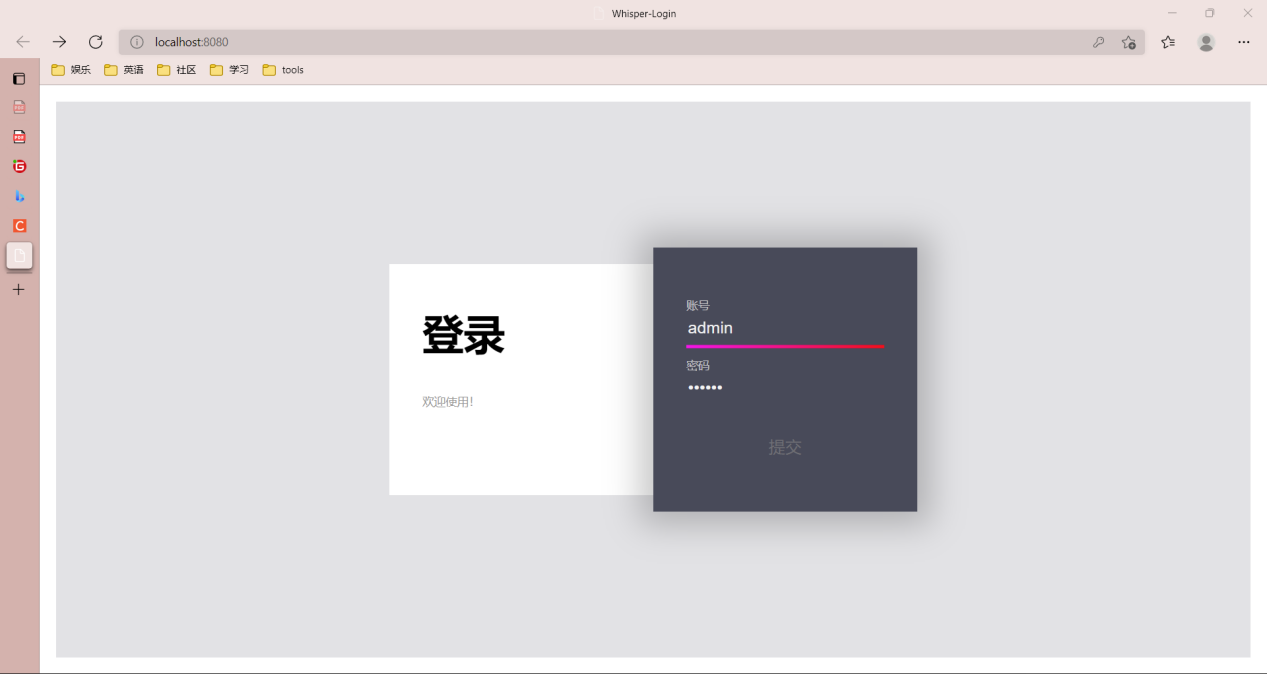
1. **实验结论**

**3.1实验结果**

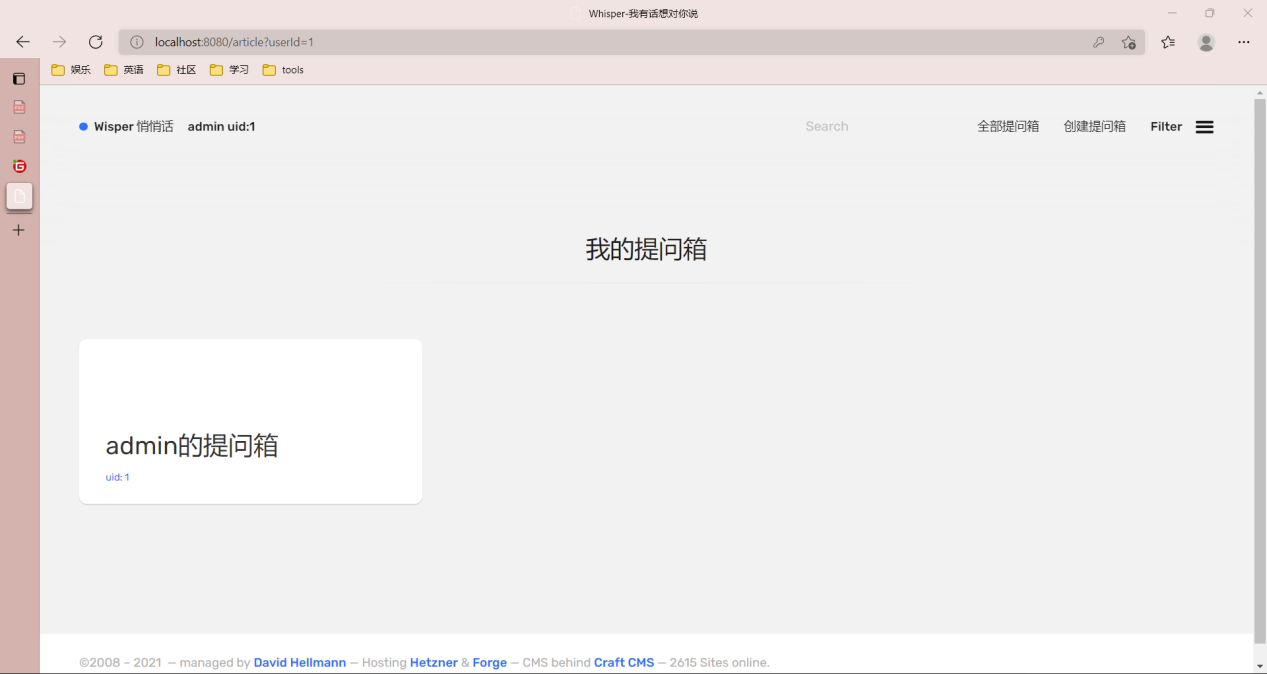
**3.1.1用户登录**

****

登录界面



用户admin登录

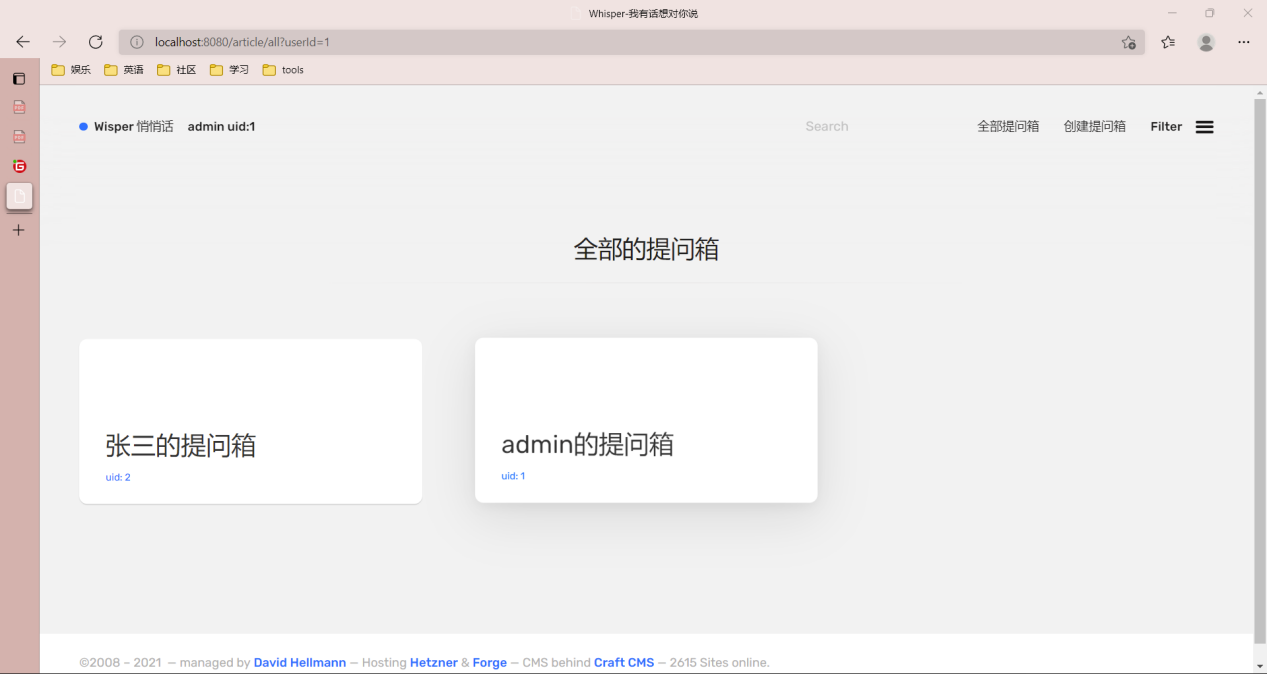


用户“admin”登录成功

进入程序界面，主界面是用户自己的提问箱

右上角显示当前用户的名字和专属uid，这也是我们项目的主题即“匿名”

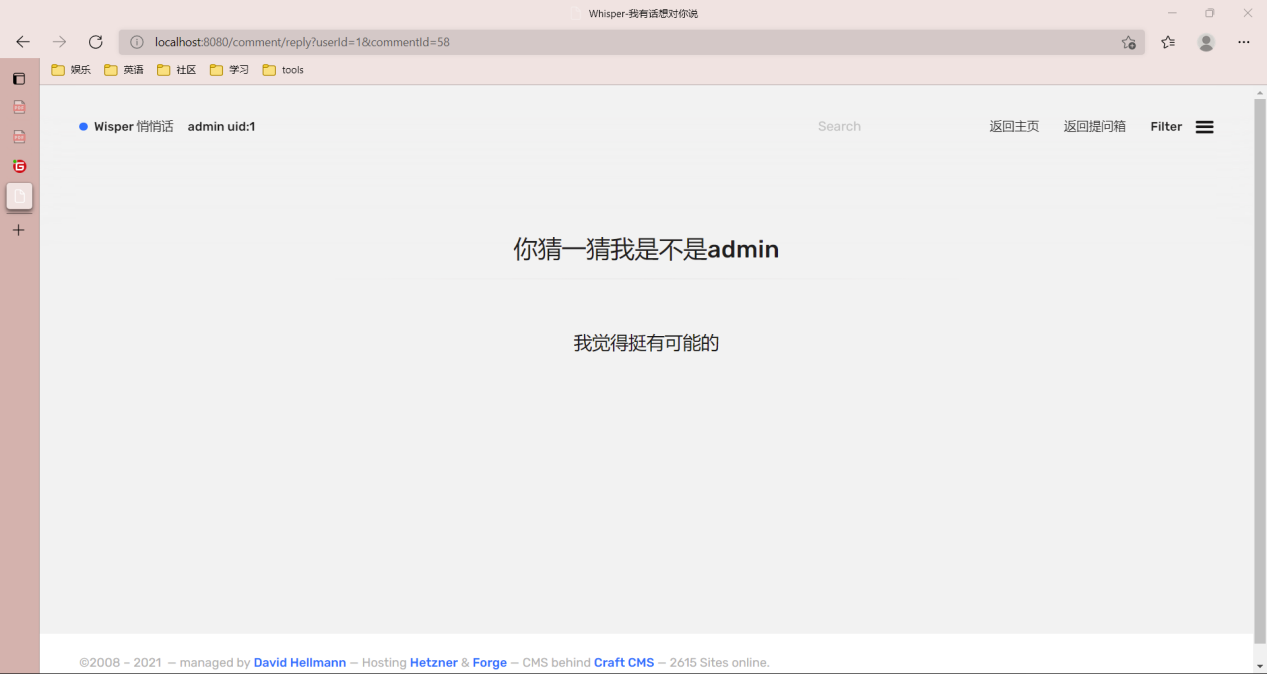
**3.1.2用户查看提问箱**

****

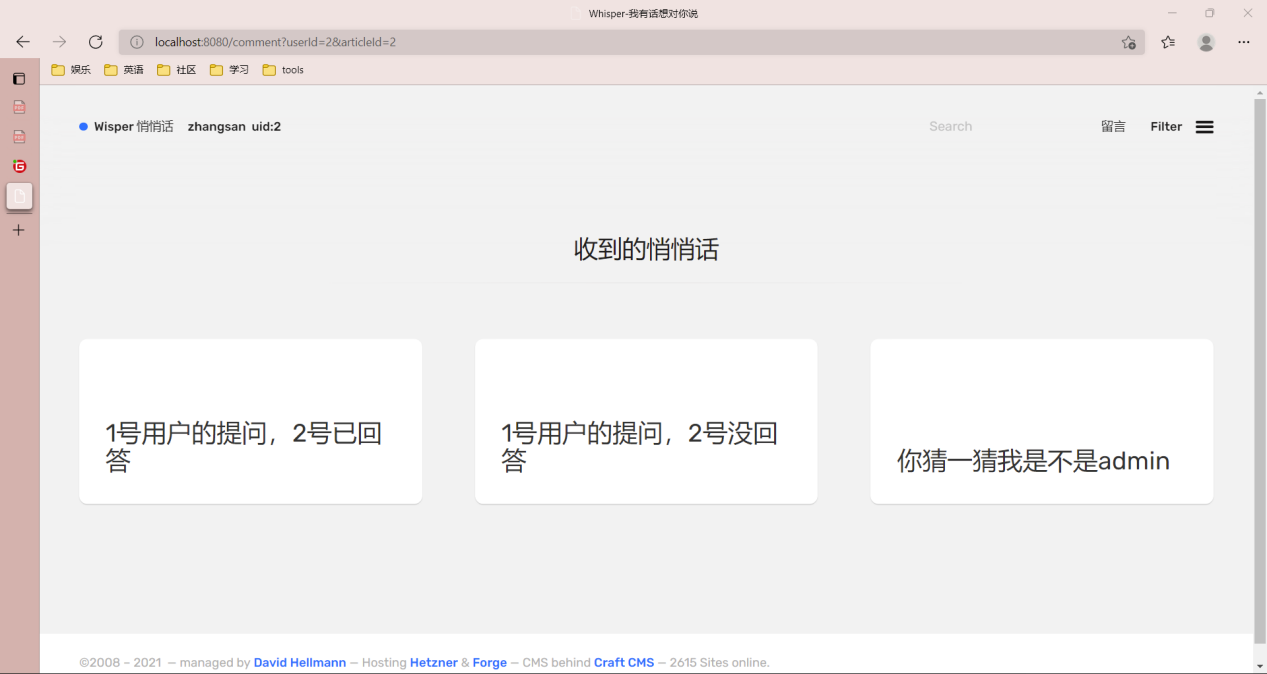
用户“admin”查看全部提问箱，提问箱下方显示发布此提问箱的用户uid

点进“张三的提问箱”，里面是zhangsan已经回答的问题

点进去可以查看详情



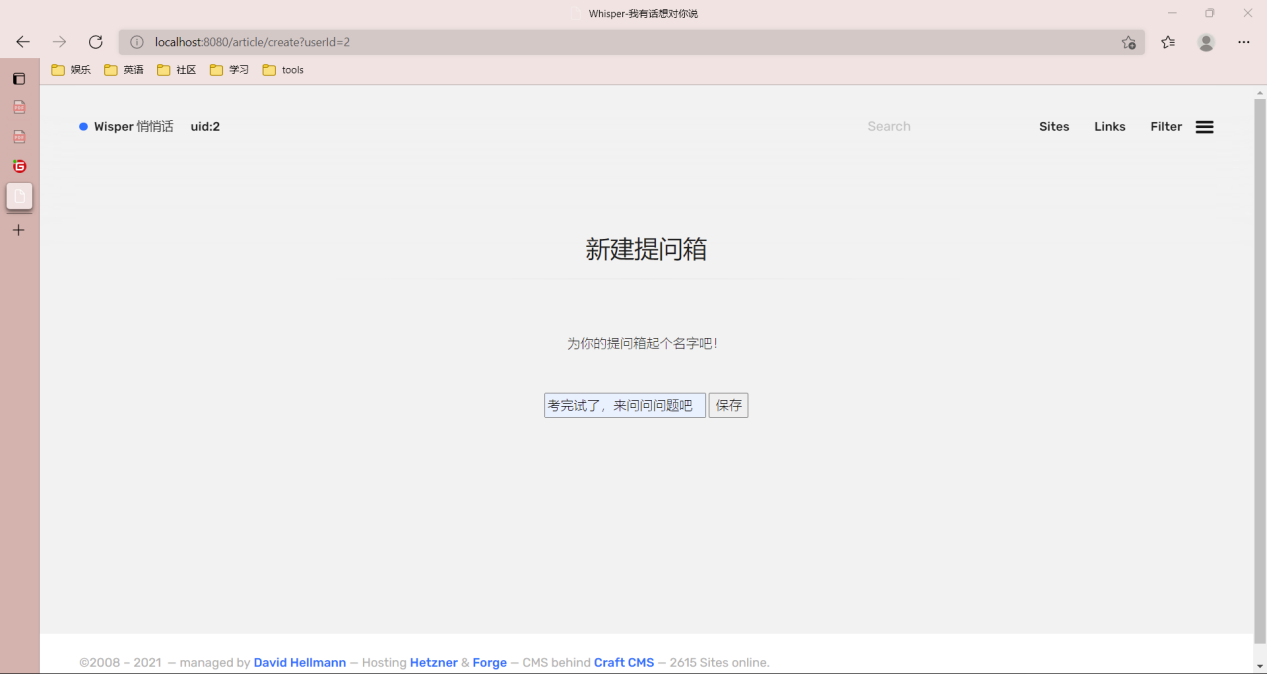
点开其中一条问题显示问题和回答

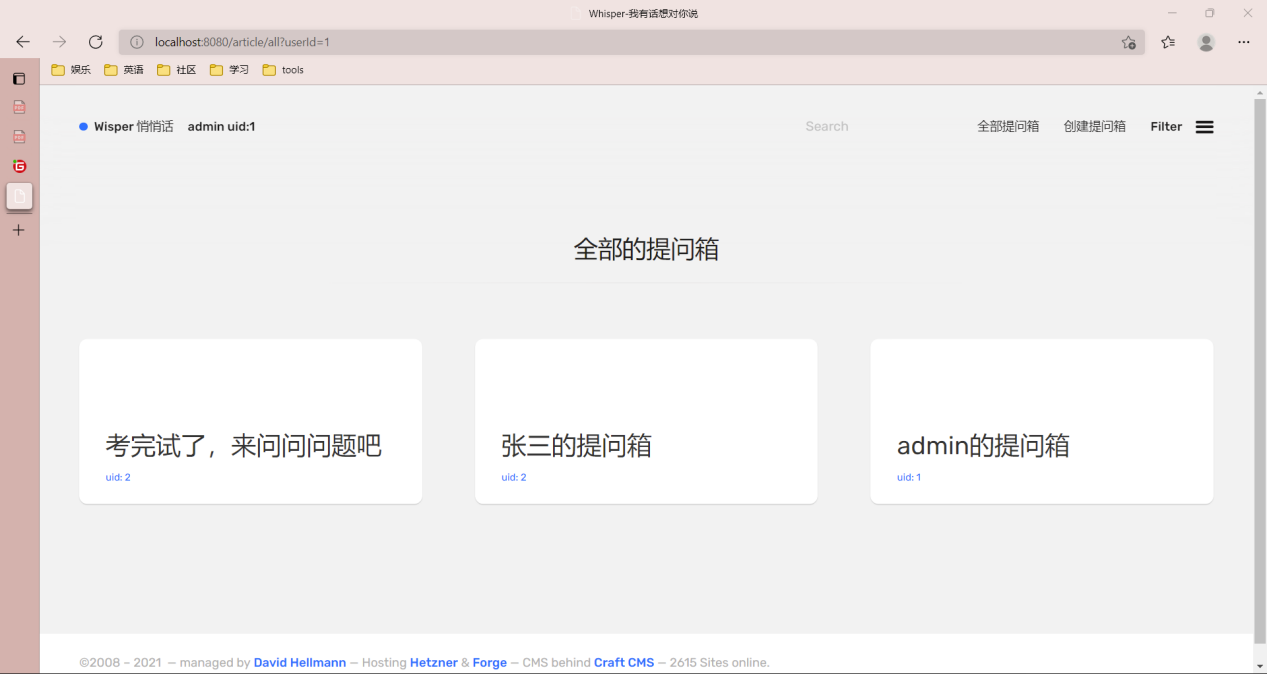


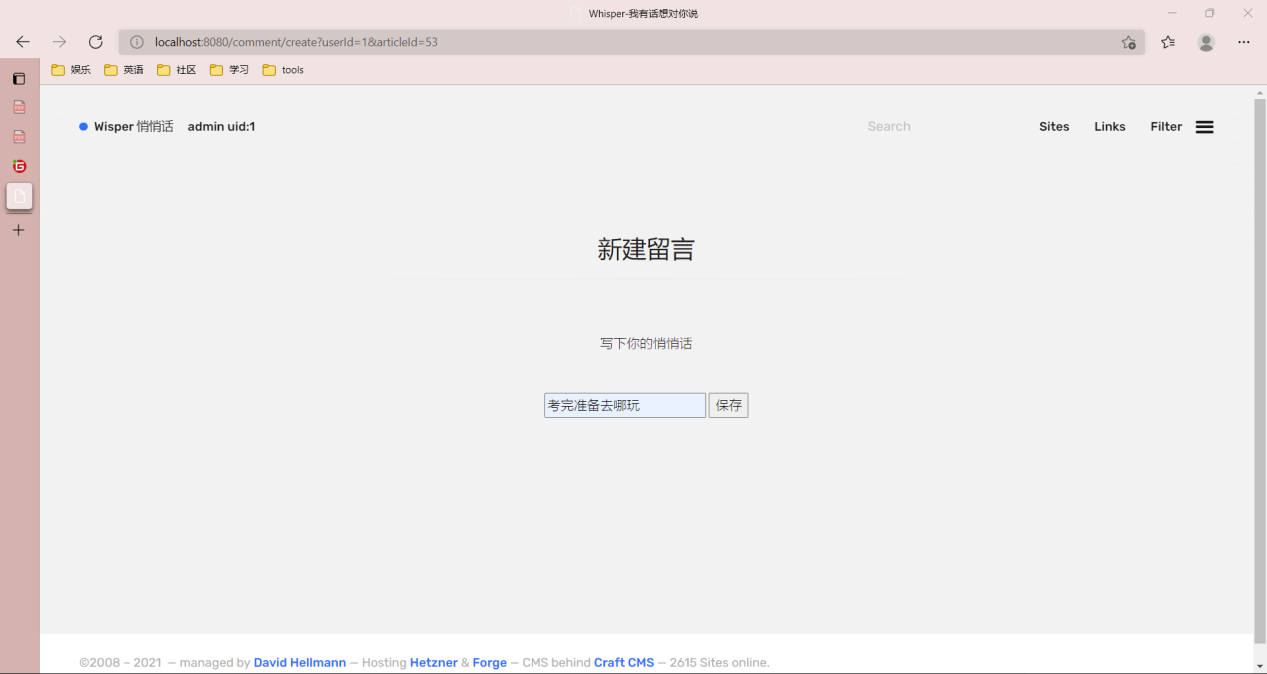
用户“zhangsan”的提问箱内容

此时还没回答第二条问题，所以用户“admin”看不到此条问题

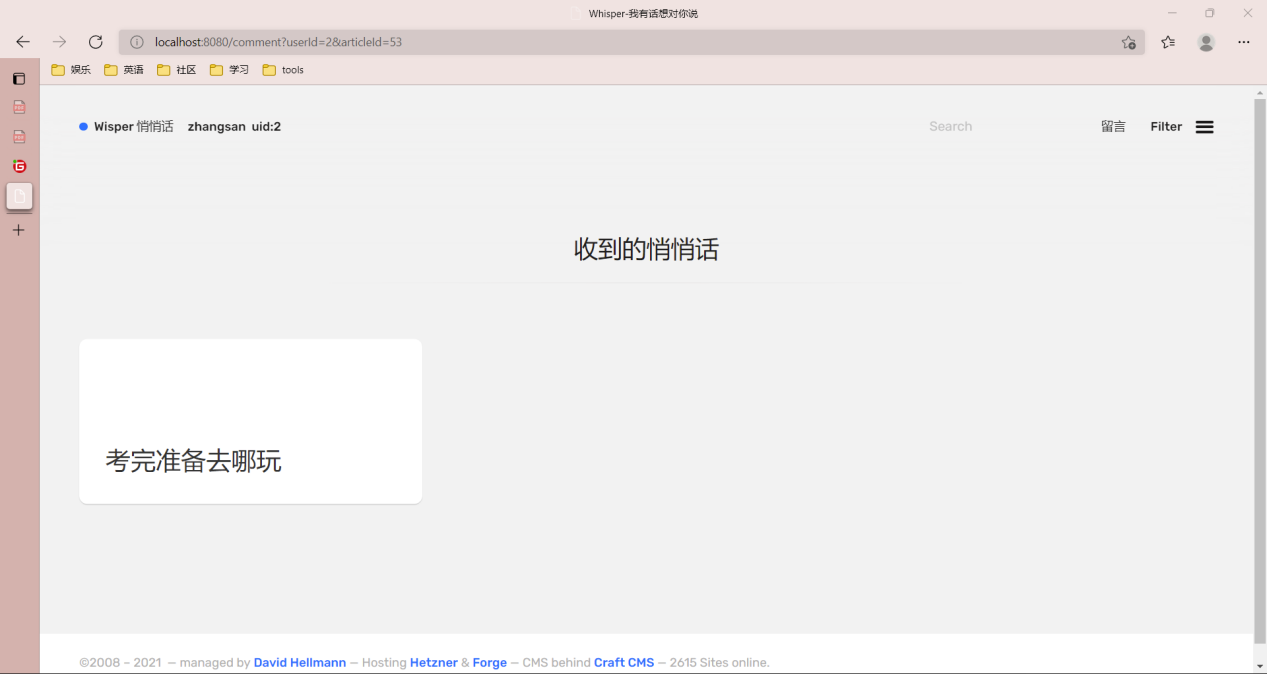
**3.1.3匿名提问流程**

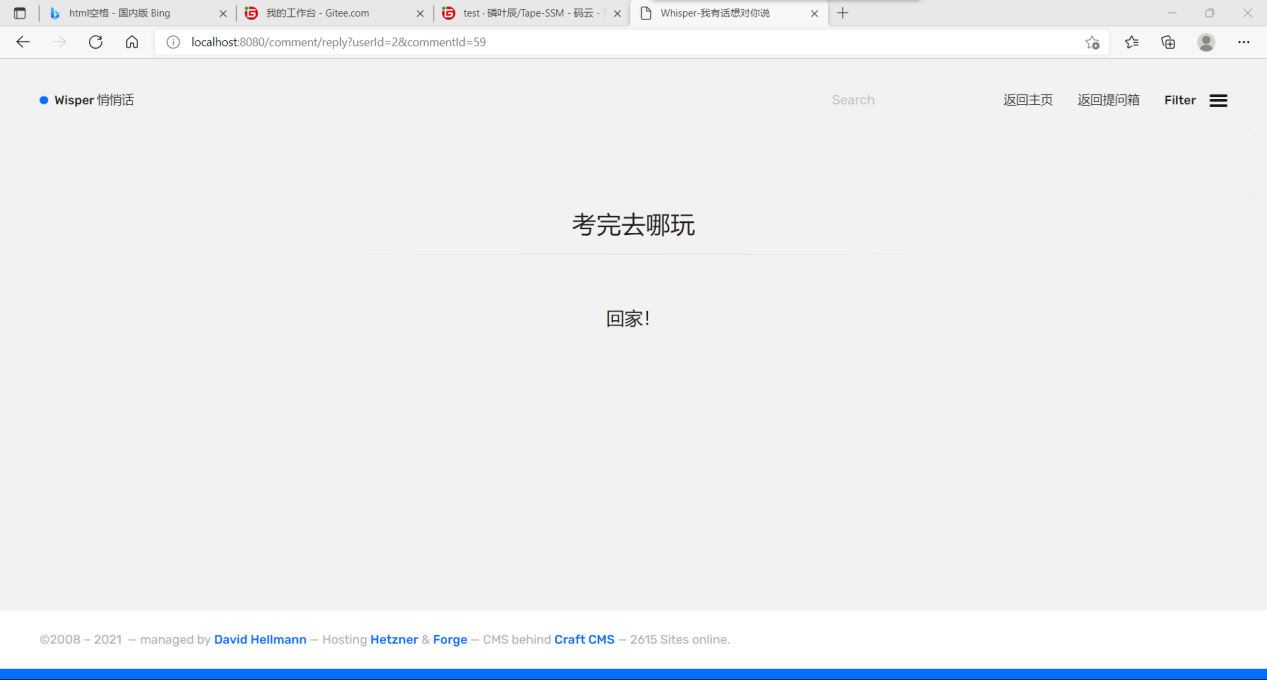


用户“zhangsan”创建提问箱



用户“admin”提问



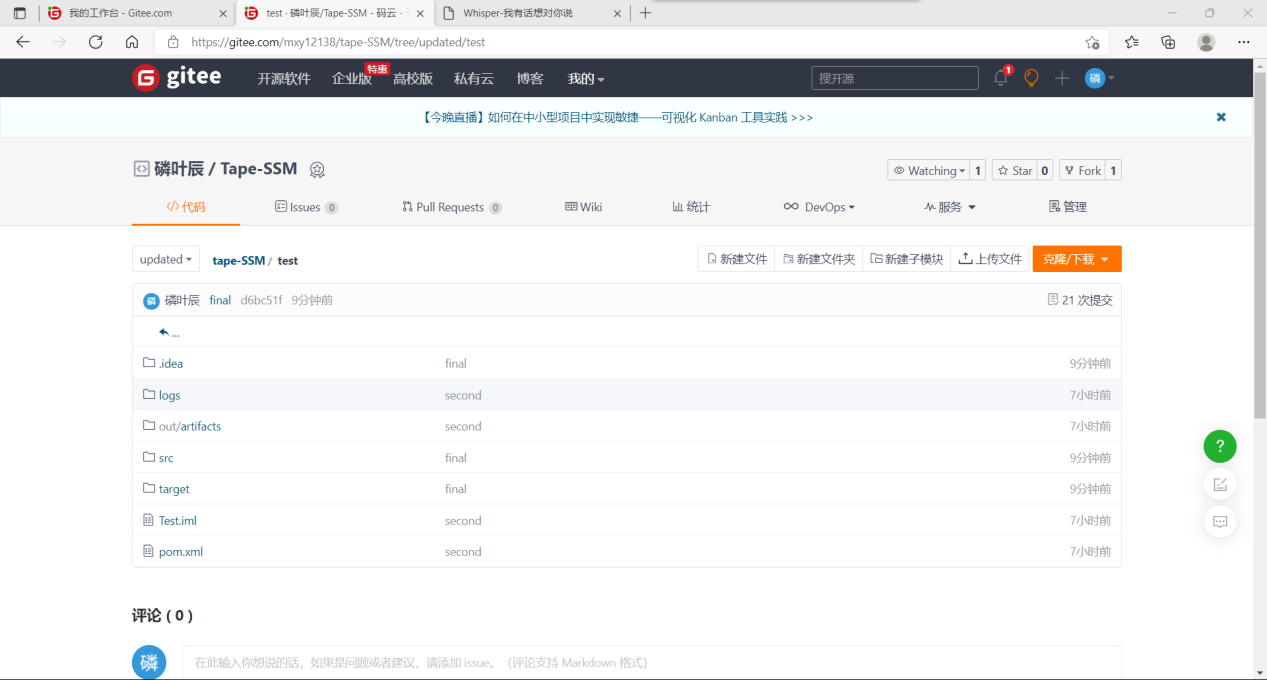


用户“zhangsan”收到此条问题，进行回答

**3.2实验总结**

本实验基于SSM的框架，相较直接使用SpringBoot而言更为繁琐，但基于SSM框架的开发使整个流程更加具有层次感，也让我们分工完成任务时更加方便，项目分为底层和前端交互开发这两部分，底层将实体类、数据库和映射文件进行设计并通过测试将完好的接口提供给控制层，控制层使用接口与前端进行交互，使后端的接口的功能得以展示。

同时我们小组的项目从一开始进行独立开发，从新建项目到最后的项目成品出炉都是我们一步一步开发的过程，独立开发是我们小组的思维活跃一直保持在一个较高的水平，小组的前后端经常进行交流讨论，使我们独立开发的项目更有自己的特色，但独立开发也有其困难所在，中间碰到了不少的问题和坑，单是配置文件出现问题就会耽误很长时间，但当整个框架搭建起来后，只需要在其上添砖加瓦就好了，这也是使用框架的好处。



同时在项目的合并更新过程中，我们使用了Git进行项目的整理，也为日后我们团队开发项目打下了良好的基础。

**3.3项目特点**

完全基于SSM框架进行开发,从配置Maven文件到Spring整合Spring MVC和MyBatis再到项目中接口函数的相互调用，都基于配置文件的约束；

完全独立开发，从项目需求的提出、数据库的建立到数据持久层、业务逻辑的编写、再到控制逻辑层和前端页面代码的编写和对应转发都是小组成员自主独立开发，虽不如网上展示的项目一样有着精美的前端界面，也不像它们一样有着复杂多样的功能，但基本完成了我们的项目开始的需求，完成了我们一开始的简洁美观、分享生活的创作理念，而这个项目也是我们辛苦独立创作的成果，我们从中也收获到了很多开发的经验和创作思想；

项目最终基本完成了最初设想的需求，实现了匿名提问，当发布人对问题进行回答后进行公开的功能，同时基本的功能也都实现了，底层的逻辑清晰易用，为以后的维护开发提供了很大的开发空间。

**3.4需要完善的方面**

首先因为项目的开发时间较短，我们没有完成很多计划内的拓展功能，比如追问、禁言等，但基本的数据库实现都已经完成，所以日后有机会进行项目的更新开发应当是很简单的；

前端并未做到像大型项目那样高端，我们小组的初衷是竭尽全力希望我们的项目能够做到简洁美观，但是因为在此之前我们并未尝试过对前端的开发，导致对前端资源手忙脚乱，但我们最后以退为进，选择用简单的HTML进行简单的前端设计，虽没有那么高端，但整体仍保持简洁欸，日后我们学习了前端相关知识后可以对项目前端进行美化更新；

开发文档和PPT没有很多时间进行整理，因为独立开发时间较紧，任务较重，小组的重心都放在了项目的完成上，但是实验文档中对整体的项目有一个比较好的概括整理，日后进行开发或者项目回顾时都能很好的理解此时我们开发项目的思路与心境；

在一开始我们有计划将此项目部署到服务器上，让其成为一个真正的可用的网站，但最后因为开发时间受限，此想法停留在了计划内，但我们有思考使用免费的Heroku服务器进行网站部署，此想法也加入到了我们的日后开发的计划表中。

1. **实验心得**

|  |  |
| --- | --- |
| 小组成员 | 心得体会 |
| 马薪宇 | 这次的JavaEE项目是很完整的一次独立开发项目的经历，从一开始的项目选题到数据库的设计到整个底层的开发和与前端的交互转发，每一步都充满着我们小组成员的辛苦和头发，而且我们小组内的成员都没有很深入的学习过Java，所以一开始也只是在跟着视频和老师的课堂内容进行框架知识的学习，在开始项目之后遇到了很多始料未及的坑。  我在这次项目开发中主要负责底层逻辑的开发，包括数据持久层、映射文件和业务逻辑层的开发，我觉得其中的难点就是一开始底层的配置即Spring整合MyBatis的部分，但每次解决一个头晕的的bug让人绝处逢生的感觉也很难得，经过这次的项目开发，我对SSM整个的框架流程也有了更清晰的认识，对框架的使用也更加得心应手，也决心在以后的时间里多多锻炼对这些好用框架的运用，这次项目中我们也用到了Git进行项目的合并和更新，对Git这一工具的使用更加熟练了。  在此我也要感谢我的队友不辞辛劳，对底层的需求与衔接十分清晰，让我处理起来得心应手，同时也感谢老师这学期的细心教导。 |
| 张嘉伟 | 这门课程的大作业给我带来的挑战很大，第一是自己还有一门课程有大作业，所以时间并不充裕；第二是自己尽管负责前端和MVC层，但是前端基础很薄弱。在开始项目之前，自己首先通读了课上的的讲义和笔记，复习和熟悉Spring三件套的用法，并抽时间看了html和css的知识。但做项目时才发现各种错误迎面而来：配置SpringMVC的环境配置了一天。有的错误是因为把web静态资源放错了位置，有的错误是因为tomcat启动时没有把资源文件配全，甚至还有是tomcat版本的问题！自己在遇到这些问题时情绪会非常低落，多亏有朋友们的陪伴，让自己能耐心的调试、在网络上查找办法，最终解决了一个个问题。  当真正开始使用Spring写代码时，终于发现了Spring框架的厉害，自己在Spring使用上遇到的问题并不多，比较顺利地实现了一开始设计的功能。虽然时间压力很大，但是回顾这个过程，我感觉到自己的抗压能力和自主解决问题的能力都得到了锻炼。当然最重要的是，自己基本上掌握了企业级JavaWeb应用开发的本领，非常感谢老师一学期辛苦的付出。 |
| 李嘉琦 | 在团队其他两位朋友的指导下，我从对项目一无所知，无从下手，到开始着手自己的事情，最后我开始着手数据库的基本构想，一开始的主要问题时评论的评论该如何存储，前后讨论了几种方法，最后确定一个precomment作为评论的评论，再结合其所在文章的id和comment所在文章的id是否相，从而确定评论的下一个评论是否还属于当前文章的评论。  作为团队成员里的顺风车选手，没有有效地参与到团队的技术工作之中，参与到了一些团队的打杂事务之中，没有给团队其他成员承担到应有的责任，但团队其他两位朋友的乐观积极和无私的帮助也点亮了我。即使项目是从新建文件夹开始的，他们也没有说去放弃，只是一直地坚持，有好几次看到我的组员们在桌子上打瞌睡，真的是很感动，或许等很久以后我不记得参与过什么项目，但我会一直记得这种意志，并带着这种意志不断前行！  最后也非常感谢老师一个学期以来的的付出，真的学到了很多！ |

**5.教师评语评分**

评语：

评分：

评阅人： 年 月 日

（备注：对该实验报告给予优点和不足的评价，并给出百分制评分。）