

**河南工业职业技术学院  
 Henan Polytechnic Institute**

**毕业设计**

题 目： 基于SSM个人博客系统

院 系： 电子信息工程系

专 业： A软件技术

班 级： A软件技术2003

学 号： 0634200322

学 生 姓 名： 史毅博

指 导 教 师： 马世欢 职 称 副教授

2022 年 10 月 23日

河南工业职业技术学院教务处制

# 前言

基于SSM框架的个人博客系统搭建，选用这套框架的优势不言而喻，作为当今市面最主流的web应用开发框架，其轻量级，可拔插，扩展性强的优势可以极大地提高项目地开发效率，降低开发和后期维护地成本和时间，因此，这里我选用SSM框架作为博客搭建的首选。

虽然选用的设计主题是个人博客系统搭建，但在实际程序设计中，我又根据不同用户合理增加了社区功能，将不同用户博客网页展示在同一页面。功能主要包含：前端登录系统，前端主页面，前端广场页面，前端博客上传系统，即后台数据管理系统，主要根据用户不同需求进行增删改查等操作。

* 其中前端登录系统使用Vue封装组合组件，实现注册和登录之间的切换
* 个人主页与社区广场通过thymeleaf解析器对用户博客进行展示，配合Vue进行增删改查操作
* 博客上传使用iframe向后台发送请求

在系统设计部分对码流监测系统的总体架构进行了设计，对系统数据库的表和字段进行了详细设计，并列举出相关字段含义。包括处理流程和相关类的设计等。章节的最后介绍了各个子模块之间的通信协议的设计，整个协议是基于XML实现的。在系统的实现章节中，本文给出了部分核心模块的算法设计和代码实现，并以文字的方式阐述了相关调用过程。

设计理由：对于个人而言博客的作用很就是记录生活，记录自己都日常。发发家人的照片，随手记录下每个不同人生阶段的感悟。同时也是分享给更多有共同兴趣爱好的小伙伴一起参与自己做的事。对于技术能力来言博客的目的是在互联网上提供内容，回答潜在客户的问题并帮助他们了解您的产品或服务，从而达到引流的效果。所以，在本次设计中我希望通过构建一个博客网站，来记录自己的技能增长提高过程。在后期过程中随着学习能力深入和技能不断提高，我会不断完善该项目。

目录

[前言 1](#_Toc117553195)

[一. 技术分析及简介 2](#_Toc117553196)

[1. 实现项目技术需求 2](#_Toc117553197)

[2. 项目所用到的技术 3](#_Toc117553198)

[二. 数据库设计和需求分析 3](#_Toc117553199)

[1. 需求分析 4](#_Toc117553200)

[2. 数据库设计 5](#_Toc117553201)

[三. 项目环境搭建 7](#_Toc117553202)

[1. 配置jar包 7](#_Toc117553203)

[1.1 项目所需jar包 7](#_Toc117553204)

[1.2 pom.xml完整代码 8](#_Toc117553205)

[2. 项目核心配置文件 11](#_Toc117553206)

[2.1 web.xml配置 11](#_Toc117553207)

[2.2 配置springmvc.xml 13](#_Toc117553208)

[2.3 配置Spring.xml文件 15](#_Toc117553209)

[2.4 配置mybatis-config.xml文件 18](#_Toc117553210)

[四. 项目前端代码实现 18](#_Toc117553211)

[1. 用户登录界面实现 18](#_Toc117553212)

[2. 个人主页页面实现 22](#_Toc117553213)

[3. 广场页面搭建 32](#_Toc117553214)

[五. 项目后端代码实现 34](#_Toc117553215)

[1. 后端项目分层结构 35](#_Toc117553216)

[2. 用户登录界面后端实现 37](#_Toc117553217)

[3. 个人主页页面后端实现 40](#_Toc117553218)

[4. 广场页面后端实现 44](#_Toc117553219)

[六. 程序测试与评价 44](#_Toc117553220)

[1. 软件测试步骤 45](#_Toc117553221)

[2. 测试用例 46](#_Toc117553222)

[3. 项目存在不足和改进方式 46](#_Toc117553223)

[七. 结语 47](#_Toc117553224)

# 一. 技术分析及简介

## 1. 实现项目技术需求

1. 通过mybatis实现用户添加和登录操作
2. 使用SpringMVC拦截用户请求，根据不同请求来操作数据库
3. 通过Spring配置文件，配置数据库连接池(Druid连接池)和事务管理
4. 配置SpringMVC配置文件，使用thymeleaf视图解析器
5. 根据不同的需求对数据库进行增删改查操作
6. 通过RESTful风格设计项目

## 2. 项目所用到的技术

| 名称 | 简介 |
| --- | --- |
| Spring | 通过控制反转(IOC)，实现对象的可插拨。Spring是SSM的**核心** |
| SpringMVC | Spring的一个子项目。通过MVC软件架构的思想，将软件按照模型、视图、控制器来划分。更轻松做到分层开发 |
| MyBatis | 支持定制化SQL、存储过程以及高级映射，避免传统JDBC开发过程中性能和结果集不对应的复杂问题 |
| thymeleaf | 视图解析器，比JSP解析器功能更丰富。灵活性更高。 |
| Vue | 通过组件化模式提高代码复用率、且让代码更好维护 |
| Ajax | 通过使用Jquery封装的AJax，实现异步操作。 |
| element-ui | 使用组件库美化网页 |
| Bootstrap | 美化网页组件元素，提高开发效率和专业度。 |

# 

# 二. 数据库设计和需求分析

需求分析是设计数据库的起点，需求分析结果是否准确反映用户的实际要求将直接直接影响到后面各阶段的设计，并影响到设计结果是否合理和实用。通过前期的需求分析，来设计对应的数据库表和各个字段。易于后期维护。一个不良的数据库设计，必然会造成很多问题，轻则增减字段，重则系统无法运行。所以，在写代码之前，花费时间进行需求分析和数据库设计十分有必要。

而数据库是整个软件应用的根基，是软件设计的起点，它起着决定性的质变作用，因此我们必须对数据库设计高度重视起来，培养设计良好数据库的习惯，是一个优秀的软件设计师所必须具备的基本素质条件！所以我们花费大量时间设计数据库并不为过。设计数据库时应该遵循以下原则：

* 数据库设计最起码要占用整个项目开发的40%以上的时间如果需求不明确，就要分析不确定的因素，设计表时就要事先预留出可变通的字段，正所谓“有备无患”。
* 数据库设计不仅仅停留于页面demo的表面。页面内容所需要的字段，在数据库设计中只是一部分，还有系统运转、模块交互、中转数据、表之间的联系等等所需要的字段，因此数据库设计绝对不是简单的基本数据存储，还有逻辑数据存储。
* 每个字段的设计都是有他必要的意义的，你在设计每一个字段的同时，就应该已经想清楚程序中如何去运用这些字段，多张表的联系在程序中是如何体现的。换句话说，你完成数据库设计后，程序中所有的实现思路和实现方式在你的脑海中就已经考虑过了。如果达不到这种程度，那当进入编码阶段后，才发现要运用的技术或实现的方式数据库无法支持，这时再改动数据库就会很麻烦，会造成一系列不可预测的问题。
* 数据库设计时就要考虑到效率和优化问题，添加必要的（冗余）字段。一些冗余字段是为了便于日后维护、分析、拓展而添加的
* 设计合理的表关联。若多张表之间的关系复杂，建议采用第三张映射表来关联维护两张表之间的关系，以降低表之间的直接耦合度。若多张表涉及到大数据量的问题，表结构尽量简单，关联也要尽可能避免。

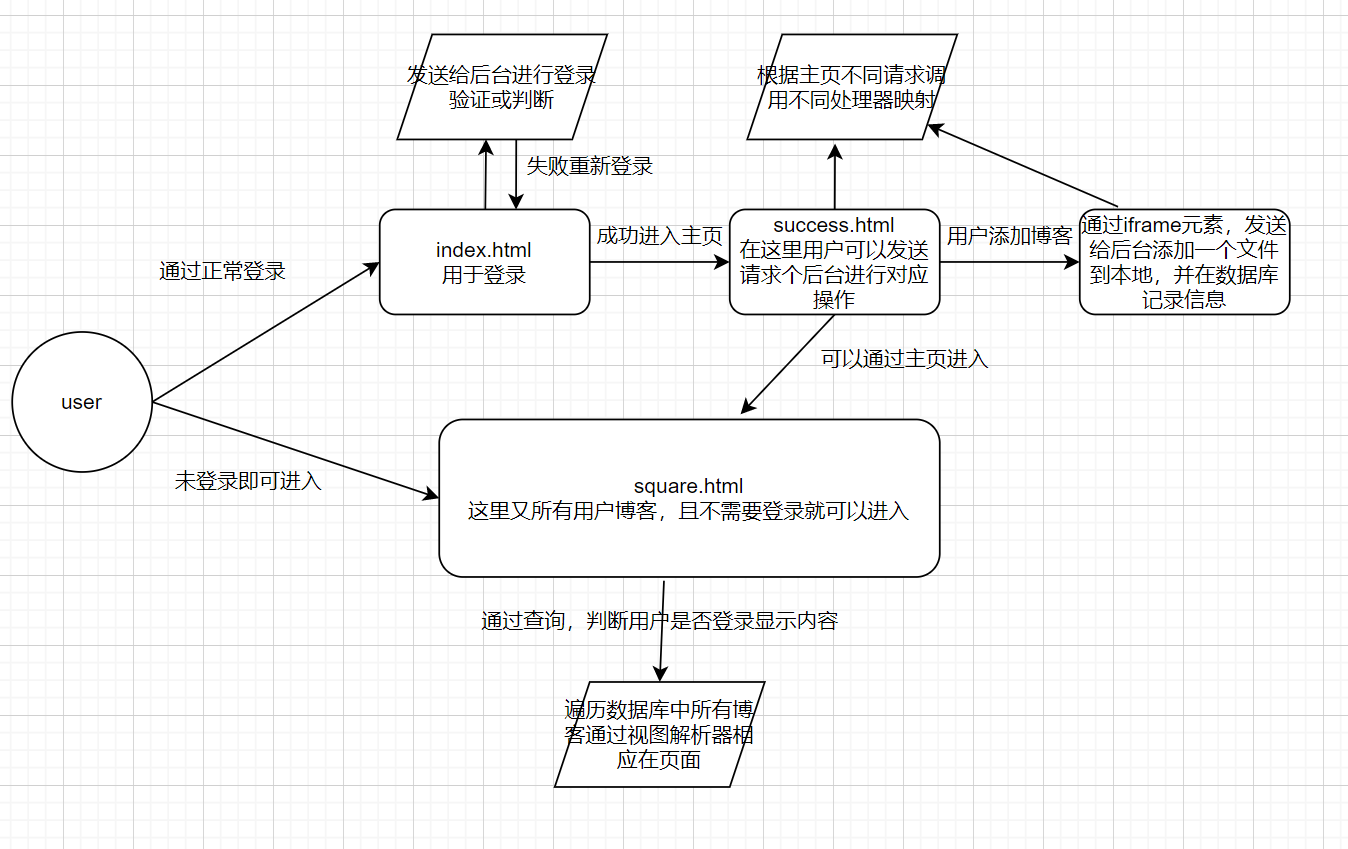
终上所述，我们可见数据库设计在整个软件开发的起到的举足轻重的作用，尤其是我说的设计原则的第一点，数据库与需求是相辅相成的，我经常把软件开发比作汽车制造。汽车制造会经过图纸设计，模型制作，样车制造，小批量试生产，最后是批量生产等步骤。整个过程环环相扣，后一过程是建立在前一过程正确的前提基础之上的。如果在图纸设计阶段发现了一个纰漏，我们可以重新进行图纸设计，如果到了样车制造阶段发现这个错误，那么我们就要把从图纸设计到样车制造的阶段重来，越到后面发现设计上的问题，所付出的代价越大，修改的难度也越大。

## 1. 需求分析

* 新用户可以进行注册，如果已有账号，可以直接登录。
* 登录和注册完成后会直接进入个人主页页面，访问登录页面或者网站首页，会直接跳转到个人主页。在主页中用户可以查看博客还可以在广场中查看他人博客。
* 用户需要根据是否登录来划分。如果只是浏览一些博客，不进行上传等一些进一步的操作，则可以不用登录浏览他人博客。
  + 如果不登陆，则可以访问广场(square)，浏览他人博客。
  + 如果登录，可以进入个人主页，查看之前上传的网页。也可以在主页点击上传按钮，实现博客文件的增删改查。
* 为了方便登录，使用会话存储技术，将会话保存在Cookie中，Cookie的有效期为7天。期间用户再次登录会重新计算时间。即用户超过7天未登录就会过期。
* 通过RESTful风格在个人主页用户可以对博客进行修改和删除。进行修改或删除后会刷新当前页面重新显示信息。
* 当登录用户进入广场，导航栏会显示和非登录用户不同界面
* 在使用手机或屏幕宽带小于1080时，导航到会启用另一套布局，且博客上传时未上传照片的，则左侧不显示照片

整个需求样板如下：

[需求样板：](https://gitee.com/Acido/images/raw/master/image/202210231827140.pn)



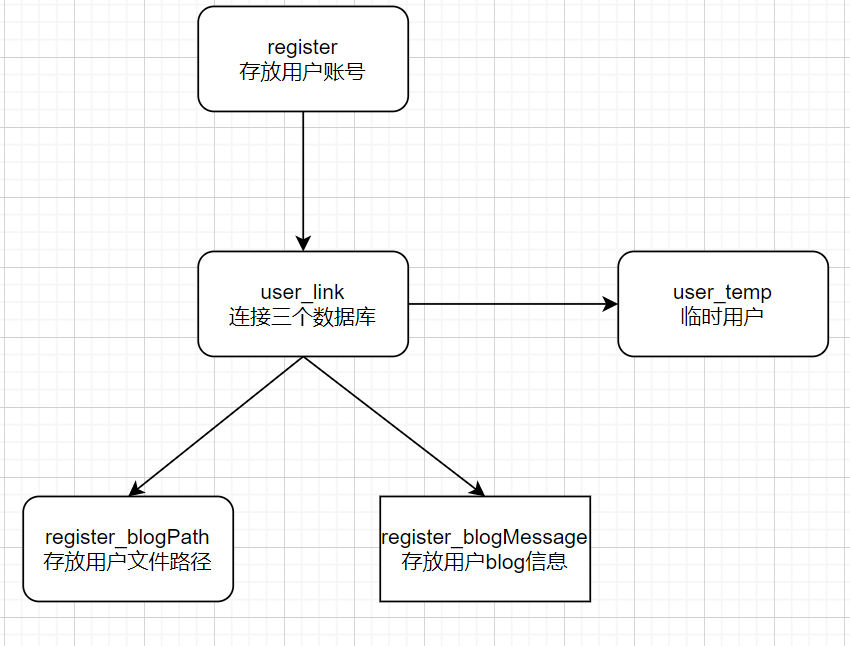
blog需求分析图

## 2. 数据库设计

* 根据创建用户表register。其中字段有：id、username、password、email、home\_pageId。
* 其中主键为id，home\_pageId为外链，主键自增，且各个字段都不能为null。
* CREATE TABLE register(  
   register\_id INT NOT NULL AUTO\_INCREMENT PRIMARY KEY,  
   register\_username VARCHAR(10) NOT NULL,  
   register\_password VARCHAR(20) NOT NULL,  
   email VARCHAR(20) NOT NULL,  
   home\_pageId INT NOT NULL  
  )
* register\_blogPath 用来存放用户文件路径。字段有：blog\_id、blogPath\_id、blogPath\_path
* 主键为blog\_id，自增，各个字段都不为null
* CREATE TABLE register\_blogPath(  
   blog\_id INT NOT NULL AUTO\_INCREMENT PRIMARY KEY,  
   blogPath\_id INT NOT NULL,  
   blogPath\_path VARCHAR(50) NOT NULL  
  )
* register\_blogMessage 用来存放用户blog信息。字段有：blog\_ids、blog\_title、blog\_message、blog\_data
* 主键为blog\_ids，blog\_data为TIMESTAMP类型，用于记录用户修改该行的时间，其他字段都不为null
* CREATE TABLE register\_blogMessage(  
   blog\_ids INT NOT NULL AUTO\_INCREMENT PRIMARY KEY,  
   blog\_title VARCHAR(50) NOT NULL,  
   blog\_message VARCHAR(100) NOT NULL,  
   blog\_data TIMESTAMP NOT NULL DEFAULT CURRENT\_TIMESTAMP ON UPDATE CURRENT\_TIMESTAMP  
  )

数据库ERD图如下，其中包含未来可能需要设计数据库表.

[数据库ERD图：](https://gitee.com/Acido/images/raw/master/image/202210231841450.png)



blog数据库设计

# 三. 项目环境搭建

开发编辑器：IDEA

Tomcat版本：9.0.58

Mysql版本：8.0

Maven版本：1.7

## 1. 配置jar包

添加jar包方式很多种，这里我们用Maven仓，通过配置pom.xml配置文件方式添加jar包，Maven在联网状态下会自动下载所需要的jar包。首先在本地仓库中找，找不到就在网上进行下载。这样省去网上找jar包时间，节省来发流程。并且Maven核心优势是会自动将被依赖的jar包导进来。这样省去寻找依赖包繁琐流程。同时也可以借助Maven可以将一个项目拆分成多个工程。

### 1.1 项目所需jar包

导入Spring核心jar包

| 包名 | 简介 |
| --- | --- |
| spring-context、spring-beans | Spring核心模块 |

配置SpringMVC相关jar包

| 包名 | 简介 |
| --- | --- |
| spring-web、spring-webmvc | SpringMVC的核心运行环境 |
| spring-jdbc | 虽然我们用了MyBatis，但事务管理需要用到 |
| spring-aspects | 管理切面 |
| javax.servlet-api | SpringMVC的前端控制器DispatcherServlet父类 |

junit测试模块

| 包名 | 简介 |
| --- | --- |
| spring-test | 整合Spring和测试模块 |
| junit | 测试模块所需包 |

MyBatis核心依赖

| 包名 | 简介 |
| --- | --- |
| mybatis-spring | Spring整合MyBatis，工厂Bean可以直接在配置文件中实现 |
| mybatis | MyBatis核心依赖 |
| mysql-connector-java | MySql驱动 |

数据库连接池

| 包名 | 简介 |
| --- | --- |
| druid | 数据写入量大，数据压缩效果好 |

日志配置和视图解析器配置

| 包名 | 简介 |
| --- | --- |
| logback-classic | log4j依赖文件 |
| thymeleaf-spring5 | Spring和Thymeleaf整合包 |

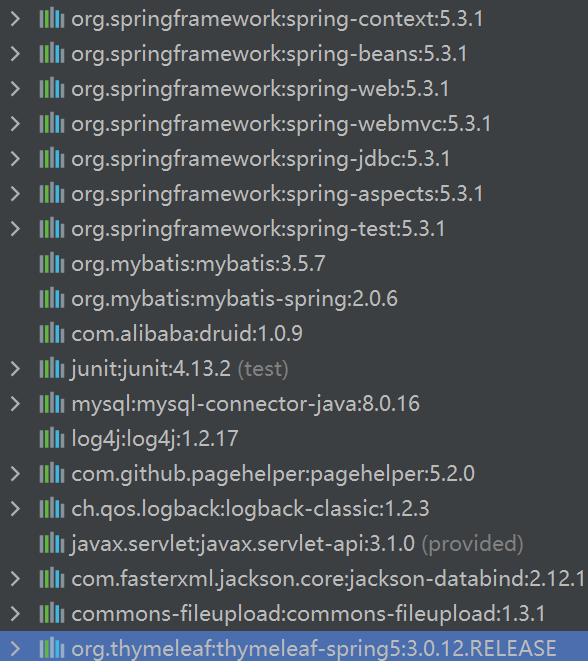
### 1.2 pom.xml完整代码

<dependencies>  
 <dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-context</artifactId>  
 <version>${spring.version}</version>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-beans</artifactId>  
 <version>${spring.version}</version>  
 </dependency>  
 <!--springmvc-->  
 <dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-web</artifactId>  
 <version>${spring.version}</version>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-webmvc</artifactId>  
 <version>${spring.version}</version>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-jdbc</artifactId>  
 <version>${spring.version}</version>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-aspects</artifactId>  
 <version>${spring.version}</version>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-test</artifactId>  
 <version>${spring.version}</version>  
 </dependency>  
 <!-- Mybatis核心 -->  
 <dependency>  
 <groupId>org.mybatis</groupId>  
 <artifactId>mybatis</artifactId>  
 <version>3.5.7</version>  
 </dependency>  
 <!--mybatis和spring的整合包-->  
 <dependency>  
 <groupId>org.mybatis</groupId>  
 <artifactId>mybatis-spring</artifactId>  
 <version>2.0.6</version>  
 </dependency>  
 <!-- 连接池 -->  
 <dependency>  
 <groupId>com.alibaba</groupId>  
 <artifactId>druid</artifactId>  
 <version>1.0.9</version>  
 </dependency>  
 <!-- junit测试 -->  
 <dependency>  
 <groupId>junit</groupId>  
 <artifactId>junit</artifactId>  
 <version>4.13.2</version>  
 <scope>test</scope>  
 </dependency>  
 <!-- MySQL驱动 -->  
 <dependency>  
 <groupId>mysql</groupId>  
 <artifactId>mysql-connector-java</artifactId>  
 <version>8.0.16</version>  
 </dependency>  
 <!-- log4j日志 -->  
 <dependency>  
 <groupId>log4j</groupId>  
 <artifactId>log4j</artifactId>  
 <version>1.2.17</version>  
 </dependency>  
 <!-- https://mvnrepository.com/artifact/com.github.pagehelper/pagehelper -->  
 <dependency>  
 <groupId>com.github.pagehelper</groupId>  
 <artifactId>pagehelper</artifactId>  
 <version>5.2.0</version>  
 </dependency>  
 <!-- 日志 -->  
 <dependency>  
 <groupId>ch.qos.logback</groupId>  
 <artifactId>logback-classic</artifactId>  
 <version>1.2.3</version>  
 </dependency>  
 <!-- ServletAPI -->  
 <dependency>  
 <groupId>javax.servlet</groupId>  
 <artifactId>javax.servlet-api</artifactId>  
 <version>3.1.0</version>  
 <scope>provided</scope>  
 </dependency>  
 <!-- 处理JSON数据-->  
 <dependency>  
 <groupId>com.fasterxml.jackson.core</groupId>  
 <artifactId>jackson-databind</artifactId>  
 <version>2.12.1</version>  
 </dependency>  
 <!-- 文件上传依赖-->  
 <dependency>  
 <groupId>commons-fileupload</groupId>  
 <artifactId>commons-fileupload</artifactId>  
 <version>1.3.1</version>  
 </dependency>  
 <!-- Spring5和Thymeleaf整合包 -->  
 <dependency>  
 <groupId>org.thymeleaf</groupId>  
 <artifactId>thymeleaf-spring5</artifactId>  
 <version>3.0.12.RELEASE</version>  
 </dependency>  
</dependencies>

这里我们可以通过**配置自定义属性管理版本**

<properties>  
 <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  
 <maven.compiler.source>1.7</maven.compiler.source>  
 <maven.compiler.target>1.7</maven.compiler.target>  
 <spring.version>5.3.1</spring.version>  
</properties>

[项目完整jar包：](https://gitee.com/Acido/images/raw/master/image/202210231605688.png)



项目完整jar包

## 2. 项目核心配置文件

### 2.1 web.xml配置

web.xml是JavaWeb中极其关键的文件，用来**初始化配置信息**。web项目启动的时候,首先对web.xml文件进行一个加载，只有此文件没有异常的时候,我们加载的web项目才算是真正的跑起来，而web.xml文件的内容加载也是由一定的顺序的。顺序如下：

ServletContextcontext-paramlistenerfilterservletspring

* 配置Spring的编码过滤器
* 这里一定要先配置Spring的编码过滤器，因为在设置编码的之前，获取过其他请求参数，那么这时得到的参数就会出现乱码问题。所以该配置项一定要在所以配置之前配置。
* <filter>  
   <filter-name>CharacterEncodingFilter</filter-name>  
   <filter-class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>  
   <init-param>  
   <param-name>encoding</param-name>  
   <param-value>UTF-8</param-value>  
   </init-param>  
   <init-param>  
   <param-name>forceEncoding</param-name>  
   <param-value>true</param-value>  
   </init-param>  
  </filter>  
  <filter-mapping>  
   <filter-name>CharacterEncodingFilter</filter-name>  
   <url-pattern>/\*</url-pattern>  
  </filter-mapping>
* 上面代码中5~6行设置自定义编码来处理请求编码，8~10配置设置后，既能处理响应编码，又能处理请求编码。13~16，配置编码处理范围为整个web项目。
* 配置处理请求方式PUT和DELETE的过滤器
* 该配置项是将网页请求方式扩展了PUT和DELETE请求。应用与RESTful风格(表现层资源状态转移)，使代码前后端分离。易于后期维护。此时对于服务器资源分类分别对应四种基本操作：GET 用来获取资源，POST 用来新建资源，PUT 用来更新资源，DELETE 用来删除资源。配置如下：
* <filter>  
   <filter-name>HiddenHttpMethodFilter</filter-name>  
   <filter-class>org.springframework.web.filter.HiddenHttpMethodFilter</filter-class>  
  </filter>  
  <filter-mapping>  
   <filter-name>HiddenHttpMethodFilter</filter-name>  
   <url-pattern>/\*</url-pattern>  
  </filter-mapping>
* REST 风格提倡 URL 地址使用统一的风格设计，从前到后各个单词使用斜杠分开，不使用问号键值对方式携带请求参数，而是将要发送给服务器的数据作为 URL 地址的一部分，以保证整体风格的一致性。
* 配置SpringMVC的前端控制器DispatcherServlet
* DispatcherServlet是Spring MVC最核心的类，是前端控制器（Front Controller）设计模式的实现，正是这个核心组件接收所有传输到Web应用的HTTP请求。其作用：
  + 把一个HTTP request交给它真正的处理方法
  + 解析HTTP request的header和body中的数据，并把它们转换为DTO(数据传输对象)
  + Model-View-Controller三方的交互
  + 再把业务逻辑返回的DTO转换成HTTP response
  + 渲染具体的视图等；
* <servlet>  
   <servlet-name>DispatcherServlet</servlet-name>  
   <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>  
   <!-- 设置SpringMVC的配置文件的位置和名称 -->  
   <init-param>  
   <param-name>contextConfigLocation</param-name>  
   <param-value>classpath:springmvc.xml</param-value>  
   </init-param>  
   <load-on-startup>1</load-on-startup>  
  </servlet>  
  <servlet-mapping>  
   <servlet-name>DispatcherServlet</servlet-name>  
   <url-pattern>/</url-pattern>  
  </servlet-mapping>
* 上面第5~8行设置SpringMVC的配置文件的位置和名称。第9行代码将DispatcherServlet的初始化时间提前到服务器启动时。后面的<servlet-mapping>同样将处理域作用到整个web项目。
* 配置Spring的监听器
* 其作用是可以在服务器启动时加载Spring配置文件
* <listener>  
   <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>  
  </listener>
* 设置Spring的配置文件的位置和名称
* <context-param>  
   <param-name>contextConfigLocation</param-name>  
   <param-value>classpath:spring.xml</param-value>  
  </context-param>

到此web.xml文件的配置就完成了。

### 2.2 配置springmvc.xml

该文件主要是配置处理器映射信息和视图解析器。

处理器映射器有两种配置方式，一种是基于 XML 的资源配置，也就是常说的非注解方式。另外一种方式是基于 Annotation 注解方式的配置。

SpringMVC的一个重要特性：将控制器中处理请求的逻辑 和 视图中渲染实现 解耦。 控制器方法和视图实现 只会在模型内容上保持一致，这是两者最大的关联。每种视图解析器对应不同的web解析技术，这里我们用thymeleaf解析器。

* 扫描控制层组件
* 用来扫描控制层的映射处理器，将所有的映射处理加载到SpringMVC中。
* <context:component-scan base-package="com.blog.cn"></context:component-scan>
* com.blog.cn是本次项目的主目录包
* 配置视图解析器
* 用于解析页面视图，将结果以视图的形式呈现在页面。
* <bean id="viewResolver"  
   class="org.thymeleaf.spring5.view.ThymeleafViewResolver">  
   <property name="order" value="1"/>  
   <property name="characterEncoding" value="UTF-8"/>  
   <property name="templateEngine">  
   <bean class="org.thymeleaf.spring5.SpringTemplateEngine">  
   <property name="templateResolver">  
   <bean class="org.thymeleaf.spring5.templateresolver.SpringResourceTemplateResolver">  
   <!-- 视图前缀 -->  
   <property name="prefix" value="/WEB-INF/view/"/>  
   <!-- 视图后缀 -->  
   <property name="suffix" value=".html"/>  
   <property name="templateMode" value="HTML5"/>  
   <property name="characterEncoding" value="UTF-8"/>  
   </bean>  
   </property>  
   </bean>  
   </property>  
  </bean>
* 以上核心代码为8~15行，其分别使用prefix和suffix代表视图前缀和视图后缀。前缀值为静态网页存放目录。
* 其他配置
* 由于SpringMVC的可拔插性，我们可以根据自己的需求进行配置。
* <!-- 配置默认的servlet处理静态资源 -->  
  <mvc:default-servlet-handler/>  
    
  <!-- 配置访问首页的视图控制 -->  
  <mvc:view-controller path="/" view-name="index"></mvc:view-controller>  
    
  <!-- 开启MVC的注解驱动 -->  
  <mvc:annotation-driven/>
* 上面我们配置处理静态资源配置项，配置访问首页的视图控制，首页视图会默认为/请求。之后开启MVC的注解驱动，使注解和配置项可以同时使用，否则SpringMVC就只能处理首页请求，其他在control层中的请求映射将全部失效。
* 配置文件上传
* 由于用户需要上传自己的博客文件，所以我们还需要配置文件上传解析器。
* <bean id="multipartResolver"  
   class="org.springframework.web.multipart.commons.CommonsMultipartResolver">  
  </bean>
* 必须通过文件解析器的解析才能将文件转换为MultipartFile对象

### 2.3 配置Spring.xml文件

* 配置组件扫描
* 有时候通过配置文件的方式创建Bean对象过于繁琐，我们还可以通过注解方式简化配置，针对Bean管理中创建对象提供以下注解：

| * 注解 | * 说明 |
| --- | --- |
| * @Component | * 推荐使用在类上用于实例化Bean |
| * @Controller | * 推荐使用在web层类上用于实例化bean |
| * @Service | * 推荐使用在service用于实例化bean |
| * @Repository | * 推荐使用在dao层类上用于实例化bean |

* 注意：这四个注解的功能是一样的都可以用来创建bean实例，用哪个都可以，只是为了区分，我们使用相对应的。
* <!--扫描组件-->  
  <context:component-scan base-package="com.blog.cn">  
   <context:exclude-filter type="annotation" expression="org.springframework.stereotype.Controller"/>  
  </context:component-scan>
* 同样扫面范围为整个com.blog.cn包下，里面的<context:exclude-filter>为不扫描控制层组件，因为控制层已经在springmvc.xml中进行扫描了。
* 配置数据库连接池
* 为了简化数据库连接配置，和后期可维护性，我们通常将数据库的连接属性写在一个jdbc.properties文件中：
* jdbc.driver=com.mysql.cj.jdbc.Driver  
  jdbc.url=jdbc:mysql://localhost:3306/blogs?serverTimezone=UTC&useSSL=true&useUnicode=true&characterEncoding=utf8&zeroDateTimeBehavior=round  
  jdbc.username=root  
  jdbc.password=123456
* 这里要注意数据库版本在8.0以上要设置字符集和时区。否则会导致数据库异常。
* 数据库连接池这里选用Druid连接池，因为其强大的监控特性，通过Druid提供的监控功能，可以清楚知道连接池和SQL的工作情况。监控SQL的执行时间、ResultSet持有时间、返回行数、更新行数、错误次数、错误堆栈信息。监控连接池的物理连接创建和销毁次数、逻辑连接的申请和关闭次数、非空等待次数、PSCache命中率等。其次，方便扩展。Druid提供了Filter-Chain模式的扩展API，可以自己编写Filter拦截JDBC中的任何方法，可以在上面做任何事情，比如说性能监控、SQL审计、用户名密码加密、日志等等。配置Druid连接池如下：
* <bean id="dataSource" class="com.alibaba.druid.pool.DruidDataSource">  
   <property name="driverClassName" value="${jdbc.driver}"></property>  
   <property name="url" value="${jdbc.url}"></property>  
   <property name="username" value="${jdbc.username}"></property>  
   <property name="password" value="${jdbc.password}"></property>  
  </bean>
* 配置用于创建SqlSessionFactory的工厂bean
* SqlSessionFactory是Mybatis的关键对象，它是个单个数据库映射关系经过编译后的内存镜像。SqlSessionFactory对象的实例可以通过SqlSessionFactoryBuilder对象类获得，而SqlSessionFactoryBuilder则可以从XML配置文件或者一个预先定制的Configuration的实例构建出SqlSessionFactory的实例。每一个Mybaits的应用给程序都以一个SqlSessionFactory对象的实例为核心，同时SqlSessionFactory也是线程安全的，SqlSessionFactory一旦被创建，应该在应用执行期间都存在，在应用运行期间不要重复创建多次，建议使用单例模式。SqlSessionFactory是创建SqlSession的工厂。我们通过配置文件配置可以直接在spring的IOC中获取SqlSessionFactory。这样我们就不需要用传统方式获取，我们可以直接在IOC容器中自动装配一个SqlSessionFactory对象。
* <bean class="org.mybatis.spring.SqlSessionFactoryBean">  
   <!-- 设置MyBatis配置文件的路径（可以不设置） -->  
   <property name="configLocation" value="classpath:mybatis-config.xml"></property>  
   <!-- 设置数据源 -->  
   <property name="dataSource" ref="dataSource"></property>  
   <!-- 设置类型别名所对应的包 -->  
   <property name="typeAliasesPackage" value="com.blog.cn"></property>  
   <!--设置映射文件的路径 ,若映射文件所在路径和mapper接口所在路径一致，则不需要设置-->  
   <!--<property name="mapperLocations" value="classpath:mapper/\*.xml"></property>-->  
  </bean>
* 实际上我们简化了mybatis-config.xml配置，将核心配置文件中配置转移到了这里，用spring整合。其中configLocation设置MyBatis配置文件的路径（可以不设置)。dataSource引用之前Druid连接池数据源。typeAliasesPackage设置类型别名所对应的包
* 设置映射文件的路径
* 如果若映射文件所在路径和mapper接口所在路径不一致，则需要设置
* <property name="mapperLocations" value="classpath:mapper/\*.xml"></property>
* 配置mapper接口的扫描配置
* 之前配置用于创建SqlSessionFactory的工厂bean，获取对象方式仍然很麻烦，所以这里我们有一种更加简便方式。我们可以轻松替代sqlsessionfactory对象的获取操作，即自动获取IOC工厂对象。我们直接自动装配即可。但必须要精确到mapper包下。即可以将指定包下所有的mapper接口创建动态代理并将这些动态代理作为IOC容器的bean管理。
* <bean class="org.mybatis.spring.mapper.MapperScannerConfigurer">  
   <property name="basePackage" value="com.blog.cn.primer.mapper,com.blog.cn.register.mapper"></property>  
  </bean>
* Control中可直接设置自动注入，生活获取工厂对象等一系列操作
* @Controller  
  public class RegisterControl {  
   @Autowired  
   private RegisterService registerService;  
   @Autowired  
   private BlogPathService blogPathService;  
   @Autowired  
   private BlogMessageService blogMessageService;  
  }

### 2.4 配置mybatis-config.xml文件

* 配置全局配置
* 由于数据库字段会和实体类字段不一致，此时我们需要在mybatis中用到resultMap，为了简化代码，我们可以设置驼峰映射，将web应用字段中的下划线转换为驼峰命名。
* <settings>  
   <!--将下划线映射为驼峰-->  
   <setting name="mapUnderscoreToCamelCase" value="true"/>  
   <!--开启延迟加载-->  
   <setting name="lazyLoadingEnabled" value="true"/>  
   <!--按需加载-->  
   <setting name="aggressiveLazyLoading" value="false"/>  
  </settings>

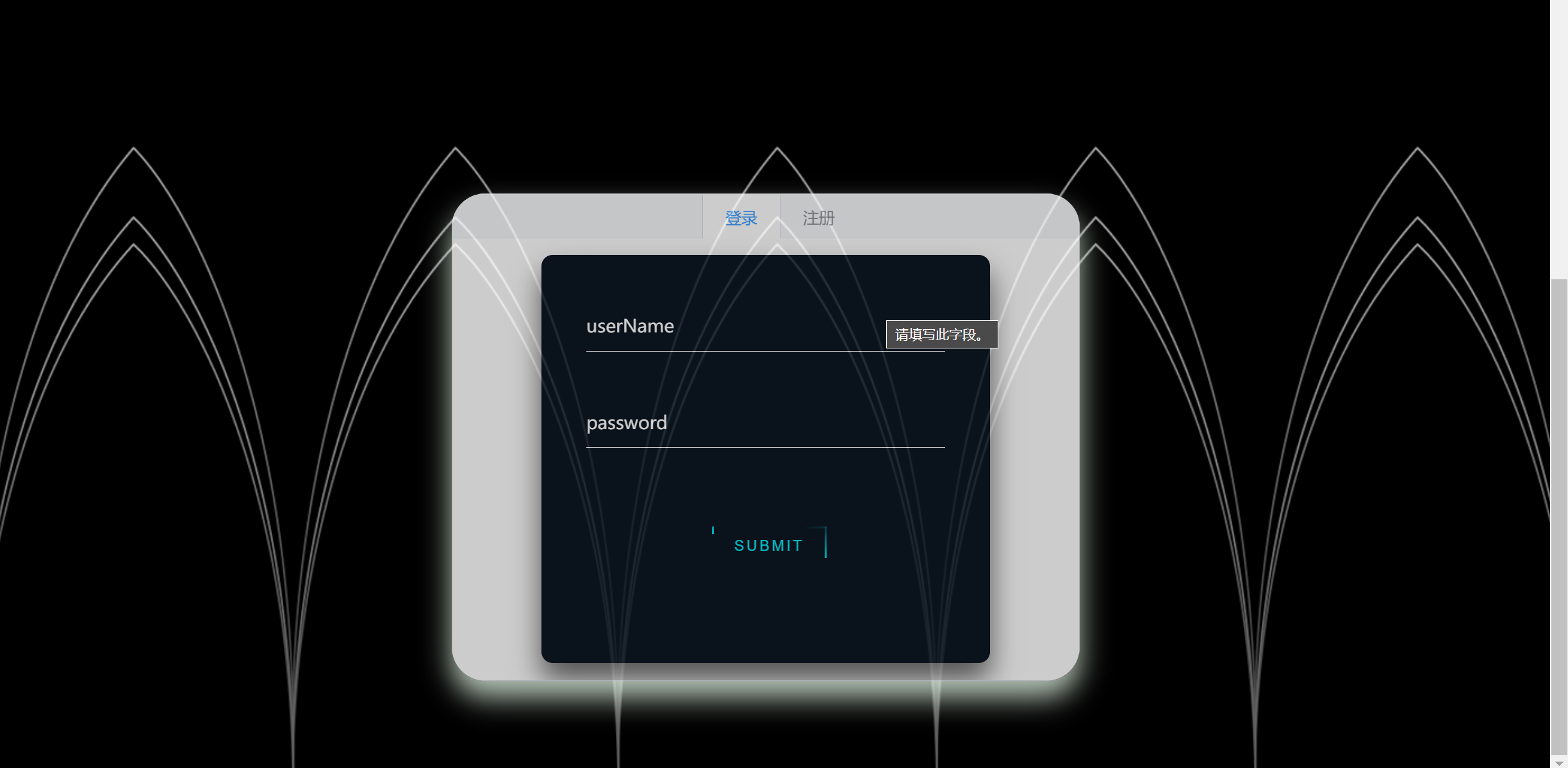
至此我们已将整个项目环境和配置文件搭建完毕。

# 四. 项目前端代码实现

## 1. 用户登录界面实现

通过之前springmvc.xml文件的配置，我们主页为index.html。其路径位于/WEB-INF/view目录下。这里也是所有静态网页存放的地方。使用element-ui组件通过Vue组合模式实现网页复用。

主页主要分为两部分：

* 页头部分主要放置网页标题和一些动态特效。以及登录按钮、广场按钮、注册按钮。连接跳转通过thymeleaf视图解析器语法引用。
* <div id="app" style="height: 100%">   
   <!-- <a th:href="@{/}">-->   
   <!-- <img src="../../static/images/home.svg" alt="">-->   
   <!-- </a>-->   
   <el-container>   
   <el-main>   
   <div id="titles">Blogs</div>   
   <div id="intrduce">一个精简的博客网站</div>   
   <el-row style="line-height: 10px;">   
   <el-button type="success" round>   
   <el-link href="#app2" class="link-r">登录</el-link>   
     
   </el-button>   
   <el-button type="info" round>   
   <el-link type="warning" th:href="@{/square/}">广场</el-link>   
   </el-button>   
   <el-button type="warning" round>   
   <el-link href="#app2">注册</el-link>   
   </el-button>   
   </el-row>   
   <span id="sbTitle"></span>   
   </el-main>   
   </el-container>   
  </div>
* 主页展示：
* [主页展示：](https://gitee.com/Acido/images/raw/master/image/202210231853901.png)
* 
* bliog\_主页展示
* 第二部分为登录和注册部分，点击登录按钮，或者注册按钮会下滑至登录或者注册页面。这里使用Bootstrap组件美化表单。当用户登录或者注册时，网页通过thymeleaf语法解析，会匹配Control层下的处理器映射将请求和处理请求的控制器方法关联起来,建立映射关系，然后根据Service层逻辑，返回数据响应给浏览器，或者跳转到其他页面。
* <div id="app2">  
   <el-tabs type="border-card" class="register-top">  
   <el-tab-pane label="登录" class="register">  
   <div class="loginBox">  
   <form th:action="@{/register}" method="post">  
   <div class="item">  
   <input type="text" id="registerUsername" name="registerUsername" required>  
   <label for="registerUsername" >userName</label>  
   </div>  
   <div class="item">  
   <input type="password" id="registerPassword" name="registerPassword" required>  
   <label for="registerPassword">password</label>  
   </div>  
   <button class="btn">submit  
   <span></span>  
   <span></span>  
   <span></span>  
   <span></span>  
   </button>  
   </form>  
   </div>  
   </el-tab-pane>  
    
   <el-tab-pane label="注册" class="register">  
   <div class="loginBox" style="height: 500px;">  
   <form th:action="@{/login}" method="post">  
   <div class="item">  
   <input type="text" name="registerUsername" required>  
   <label for="registerUsername">userName</label>  
   </div>  
   <div class="item">  
   <input type="password" name="registerPassword" required>  
   <label for="registerPassword">password</label>  
   </div>  
   <div class="item">  
   <input type="text" name="registerPassword2" required>  
   <label >password2</label>  
   </div>  
   <div class="item">  
   <input type="text" name="email" required>  
   <label >email</label>  
   </div>  
   <button class="btn">submit  
   <span></span>  
   <span></span>  
   <span></span>  
   <span></span>  
   </button>  
   </form>  
   </div>  
   </el-tab-pane>  
   </el-tabs>  
  </div>
* 登录展示
* [登录展示：](https://gitee.com/Acido/images/raw/master/image/202210231907213.png)
* 
* blog\_登录展示
* 注册展示
* [注册展示：](https://gitee.com/Acido/images/raw/master/image/202210231908425.png)
* 
* Blog\_注册展示
* 通过Vue组合组件，使用JS动画库美化页面
* new Vue().$mount('#app')  
  new Vue().$mount('#app2')  
  var typed = new Typed("#sbTitle", {  
   strings: ['stay hungry stay foolish ❤❤ ❤❤', 'good good study,day day up！!'],  
   startDelay: 0,  
   typeSpeed: 100,  
   backSpeed: 100,  
   loop: true,  
   showCursor: true,  
   shuffle: false  
  });

至此主页主要结构以实现。

## 2. 个人主页页面实现

头部导航栏采用两种配置方式，应对不同屏幕尺寸，主页展示用户名字和头像。下拉即为博客展示页，这里采用thymeleaf语法后台获取用户博客信息后，通过th:each遍历展示博客。此网页可分为四分部：顶部导航栏，主页，博客展示页，页脚。

* 第一部分：导航栏这里用户可以根据需求退出登录，或者进入广场。也可以点击博客上传按钮上传博客。上传博客时候，通过layui的iframe美化组件显示上传分页。这里用户需要上传博客文件、博客标题及博客内容简介。在分页中点击上传后，通过表单提交到后台，后台根据文件是否重复来进行保存操作。结果会返回一个JSON字符出。iframe分页会根据返回数据通过Ajax异步方式，关闭分页并刷新父页面。
* <div class="arlo\_tm\_topbar">  
   <div class="container">  
   <div class="topbar\_inner">  
   <div class="logo">  
   <a th:href="@{/}">  
   <img class="light"src="../../static/images/logo/classic/light.png" alt="" />  
   </a>  
   <a href="#"><img class="dark" src="../../static/images/logo/classic/dark.png" alt="" /></a>  
   </div>  
   <div class="menu">  
   <ul class="anchor\_nav">  
   <li class="current"><a href="#home">主页</a></li>  
   <li><a href="#about">博客</a></li>  
   <li><a th:href="@{/square/}">广场</a></li>  
   <li><a href="javascript:;" id="blogUplod">博客上传</a></li>  
   <li><a th:href="@{/blog/logout}">退出登录</a></li>  
   <li><a href="#contact"></a></li>  
   </ul>  
   </div>  
   </div>  
   </div>  
  </div>
* 可以看到在列表中，根据不同需求点击部分按钮即可。下面是博客上传按钮触发的iframe分页的js代码。当我们点击上传之后服务器会向前台发送OK(1)代表成功，或发生ERROR(2)代表错误。
* var apps = document.querySelectorAll('.app')  
  $("#blogUplod").click(function() {  
   layer.open({  
   title: '上传',  
   type: 2,  
   area: ['700px', '530px'],  
   fix: false, //不固定  
   maxmin: true,  
   content: 'http://localhost:8080/blog/blogUplod',  
   success: function(layero, index) {},  
   end: function() {  
   var handle\_status = $("#handle\_status").val();  
   if (handle\_status == '1') {  
   layer.msg('添加成功！', {  
   icon: 1,  
   time: 2000 //2秒关闭（如果不配置，默认是3秒）  
   }, function() {  
   history.go(0);  
   });  
   } else if (handle\_status == '2') {  
   layer.msg('添加失败！', {  
   icon: 2,  
   time: 2000 //2秒关闭（如果不配置，默认是3秒）  
   }, function() {  
   history.go(0);  
   });  
   }  
   }  
   });  
  })
* 由于iframe子页需要提供访问页面，我们还需要创建upload.html。iframe会通过http://localhost:8080/blog/blogUplod请求，将upload.html页面展现于iframe分页中。我们上传博客后在发送请求，返回JSON。
* <div class="content" style="margin-top: 20%;">  
   <form th:action="@{'/blog/uploadBlog/'+${session.loginUser.registerUsername}}" method="post"  
   enctype="multipart/form-data" style="text-align: center;margin:auto;" onsubmit="greeting()">  
   <input type="text" name="blogTitle" required placeholder="博客标题"  
   style="width: 250px;height: 50px;" /><br />  
   <textarea cols="35" name="blogText" rows="5" required placeholder="博客介绍100字以内"></textarea><br />  
   <input type="submit" value="上传" style="margin-left: 150px;" id="closes" />  
   <input type="file" id="upload" name="upload" required style="margin: 0 auto;" />  
    
   </form>  
   </form>  
  </div>
* 其中enctype="multipart/form-data"代表表单为上传文件。
* 第二部分：主页展示，这里通过session获取用户名，和数据库中保存用户的头像。使用init.js美化页面，使页面有打字机效果。同时通过animate.min.css给用户头像添加动态特效。
* <div class="arlo\_tm\_hero" id="home">  
   <div class="background">  
   <div class="image" data-img-url="../../static/images/slider/1.jpg"></div>  
   <div class="overlay"></div>  
   </div>  
   <div class="content">  
   <div class="image\_wrap">  
   <div class="main" id="imgs" data-img-url="@{https://images.unsplash.com/photo-1652758978641-9fad66f4fa01?crop=entropy&cs=tinysrgb&fm=jpg&ixlib=rb-1.2.1&q=80&raw\_url=true&ixid=MnwxMjA3fDB8MHxwaG90by1wYWdlfHx8fGVufDB8fHx8&auto=format&fit=crop&w=735}">  
   </div>  
   </div>  
   <div class="name\_holder">  
   <h3>你好！<span th:text="${session.loginUser.registerUsername}"></span></h3>  
   </div>  
   <div class="text\_typing">  
   <p><span class="arlo\_tm\_animation\_text\_word"></span>|</p>  
   </div>  
   </div>  
   <div class="arlo\_tm\_arrow\_wrap bounce anchor">   
   <a href="#about">  
   <i class="xcon-angle-double-down"></i>  
   </a>   
   </div>  
  </div>
* js部分
* //头像动画  
  $('#imgs').mousemove(function(){  
   $(this).addClass('animate\_\_hinge');  
  });  
  $('#imgs').mouseover(function(){  
   $(this).removeClass('animate\_\_hinge')  
  })
* 第三部分：为页面主要部分，这里展示用户博客。然后用element-ui组件库给每篇博客添加删除修改按钮。当用户点击任意按钮时，通过Vue绑定事件，弹出对话框。当用户选择确定时，会触发对应异步请求。最终在请求完成后，会刷新主页面。
* <div class="right">  
   <div class="about\_title app">  
   <h2 th:text="${blogPathMessage.value.getBlogTitle}"></h2>  
   <h3>I'm   
   <span th:text="${session.loginUser.registerUsername}"></span> and   
   <span class="arlo\_tm\_animation\_text\_word"></span>  
   <!-- <a href="#" style="float: right;">点击删除</a> -->  
   <el-dropdown @command="handleCommand">  
   <span class="el-dropdown-link">  
   编辑<i class="el-icon-arrow-down el-icon--right"></i>  
   </span>  
   <el-dropdown-menu slot="dropdown">  
   <el-dropdown-item>  
   <el-button type="text" @click="openDelete">  
   <a th:href="@{'/blog/delete/'+${blogPathMessage.key.getBlogId}+'/'+${blogPathMessage.value.getBlogId}}">删除</a>  
   </el-button>  
   </el-dropdown-item>  
   <el-dropdown-item>  
   <el-button type="text" @click="openAlter">修改</el-button>  
   </el-dropdown-item>  
   </el-dropdown-menu>  
   </el-dropdown>  
   </h3>  
   </div>  
   <div class="text">  
   <p th:text="${blogPathMessage.value.getBlogData}">  
   <p th:text="${blogPathMessage.value.getBlogMessage}"></p>  
   </div>  
   <div class="about\_short\_contact\_wrap">  
    
   </div>  
   <div class="arlo\_tm\_button">   
   <a th:href="@{'/blog/'+${blogPathMessage.key.getBlogPathPath}}"style="background-color: #67c23a !important;">  
   <span>点击查看笔记->></span>  
   </a>   
   </div>  
  </div>
* 以上为主要代码。当我们通过点击Vue绑定事件@click="openDelete"或者@click="openAlter"时候，会触发对应的Vue方法。且当用户使用手机或屏幕较小的设备观看时，通过JavaScript获取body页面大小，如果小于1080，则会不显示无图片博客左侧页面框。优化观看体验。
* var Main = {  
   methods: {  
   openDelete() {  
   // this.$message('click on item ' + command);  
   this.$confirm('此操作将永久删除该文件, 是否继续?', '提示', {  
   confirmButtonText: '确定',  
   cancelButtonText: '取消',  
   type: 'warning'  
   }).then(() => {  
   this.$message({  
   type: 'success',  
   message: '删除成功!',  
    
   })  
   this.$router.replace("@{'/blog/delete/'+${blogPathMessage.key.getBlogId}+'/'+${blogPathMessage.value.getBlogId}}").then(function o2() {  
   $.axios("@{'/blog/delete/'+${blogPathMessage.key.getBlogId}+'/'+${blogPathMessage.value.getBlogId}}"  
   )  
   })  
   }).catch(() => {  
   this.$message({  
   type: 'info',  
   message: '已取消删除'  
   });  
   });  
   },  
   openAlter() {  
   this.$confirm('您将要修改次博客, 是否继续?', '提示', {  
   confirmButtonText: '确定',  
   cancelButtonText: '取消',  
   type: 'warning',  
   }).then(() => {  
   this.$message({  
   type: 'success',  
   message: 'localhost:8080 404!!'  
   });  
   }).catch(() => {  
   this.$message({  
   type: 'info',  
   message: '已取消修改'  
   });  
   });  
   }  
   }  
  }  
  window.onload=function(){  
   var width=document.body.scrollWidth  
   var left=document.querySelectorAll('.left')  
   if(width<1080){  
   for(var i=0;i<left.length;i++){  
   left[i].setAttribute('style','display:none')  
   }  
    
   }  
  }
* 第四部分为页脚部分，这里的实现十分简单，主要是放一些备案信息和连接。
* <div class="arlo\_tm\_section">   
   <div class="arlo\_tm\_copyright">   
   <div class="container">   
   <div class="inner">   
   <div class="copy wow fadeInLeft" data-wow-duration="1.5s">   
   <p>&copy;Copyright 2019-2021. All rights are By <a th:href="@{/}">Acid.Stephen.Boer</a>.   
   </p>   
   </div>   
   <div class="social wow fadeInLeft" data-wow-duration="1.5s" data-wow-delay=".2s">   
   <ul>   
   <li><a href="#"><img class="svg" src="../../static/images/svg/social/facebook.svg"   
   alt="" /></a>   
   </li>   
   <li><a href="#"><img class="svg" src="../../static/images/svg/social/twitter.svg"   
   alt="" /></a></li>   
   <li><a href="#"><img class="svg" src="../../static/images/svg/social/behance.svg"   
   alt="" /></a></li>   
   <li><a href="#"><img class="svg" src="../../static/images/svg/social/dribbble.svg"   
   alt="" /></a>   
   </li>   
   <li><a href="#"><img class="svg" src="../../static/images/svg/social/tik-tok.svg"   
   alt="" /></a></li>   
   </ul>   
   </div>   
   </div>   
   </div>   
   </div>   
  </div>

主页展示：

当用户登录时：

[用户登录：](https://gitee.com/Acido/images/raw/master/image/202210231953275.png)



主页用户登录

博客展示页:

[博客展示页：](https://gitee.com/Acido/images/raw/master/image/202210231954391.png)



博客展示页

博客上传页：

[博客上传页：](https://gitee.com/Acido/images/raw/master/image/202210231957664.png)



博客上传页

博客上传成功：

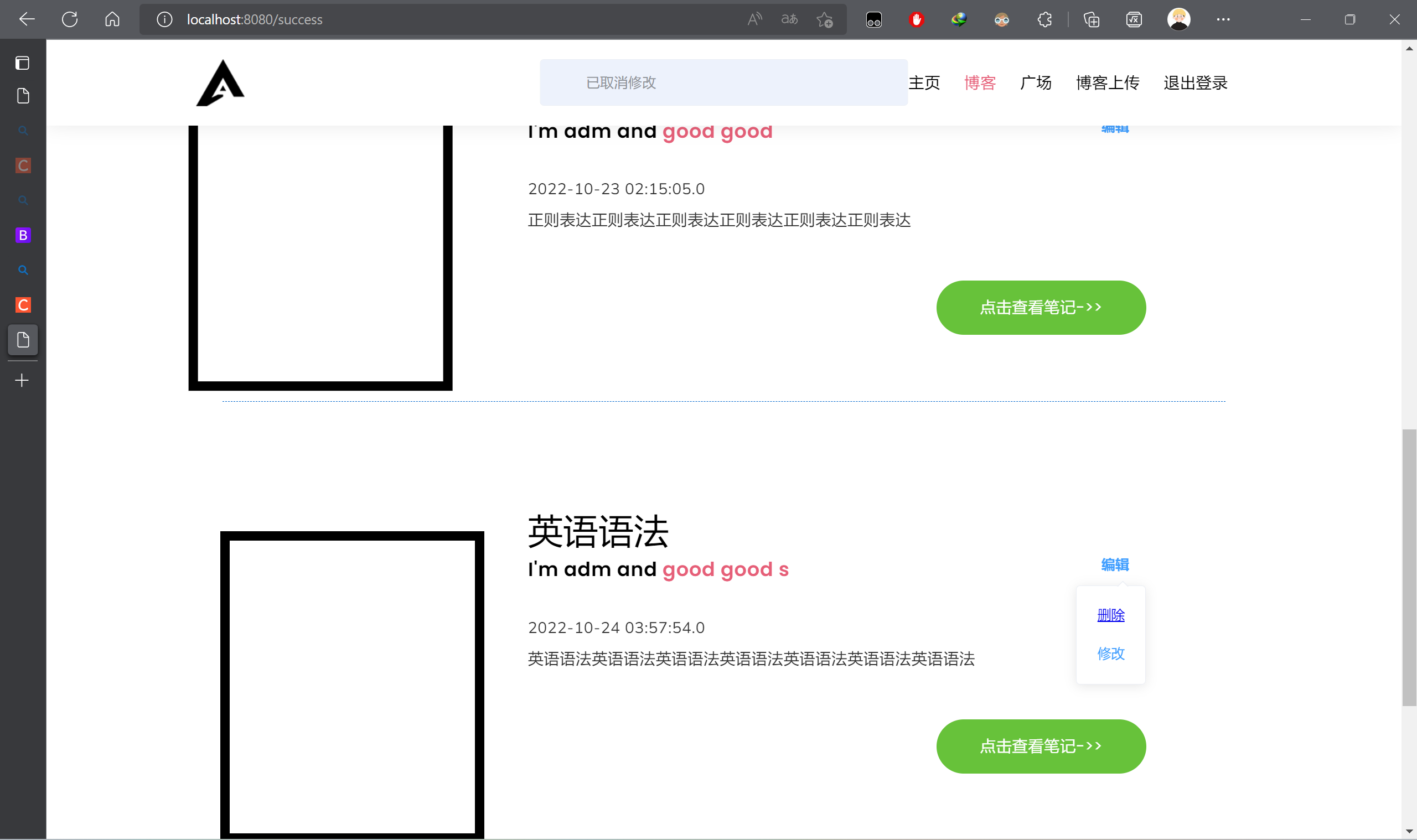
[博客上传成功：](https://gitee.com/Acido/images/raw/master/image/202210231958550.png)



博客上传成功

修改或删除：

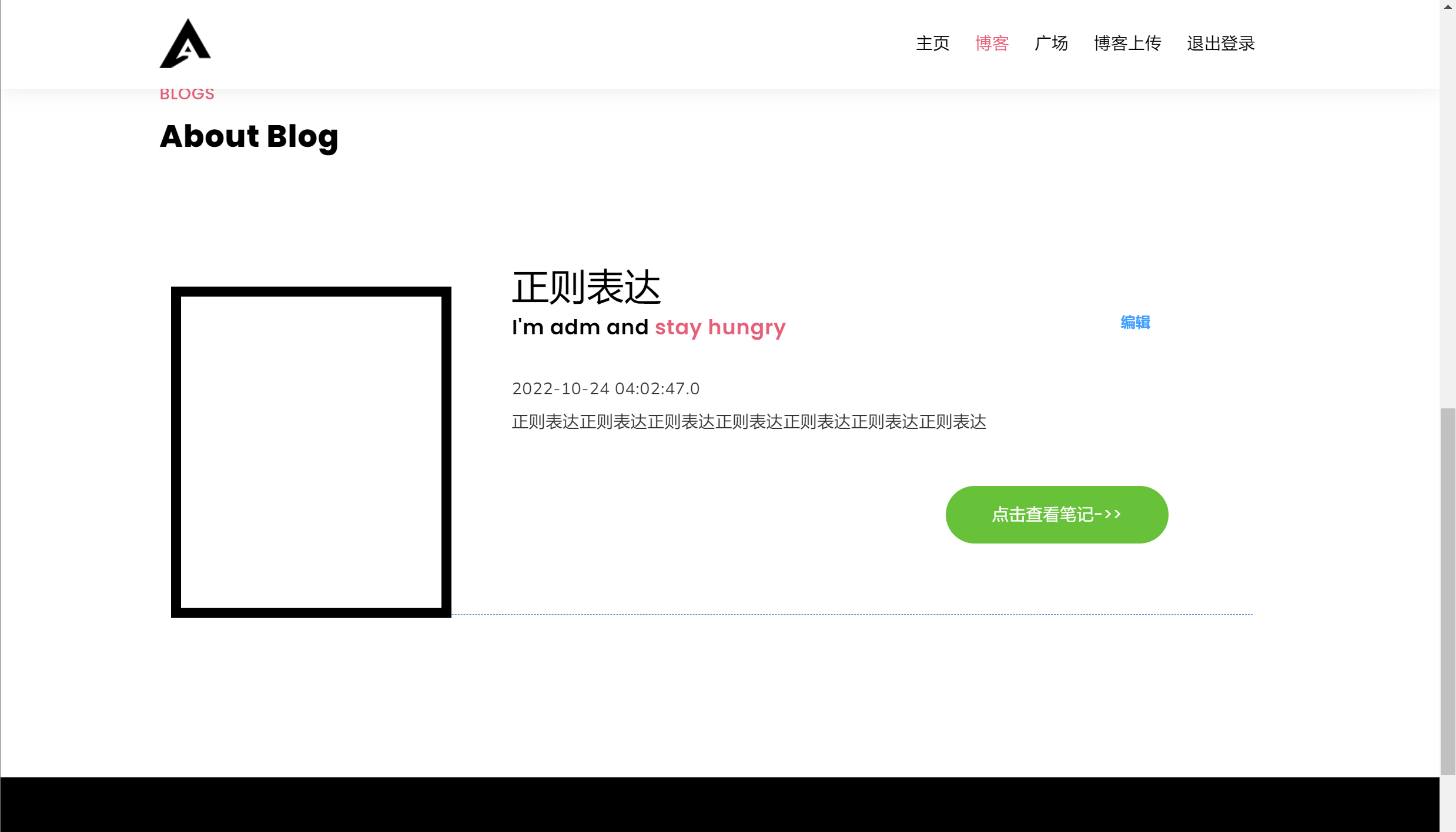
[修改或删除：](https://gitee.com/Acido/images/raw/master/image/202210232000317.png)



修改或删除

成功删除英语语法博客：

[成功删除英语语法博客：](https://gitee.com/Acido/images/raw/master/image/202210232003148.png)



成功删除英语语法博客

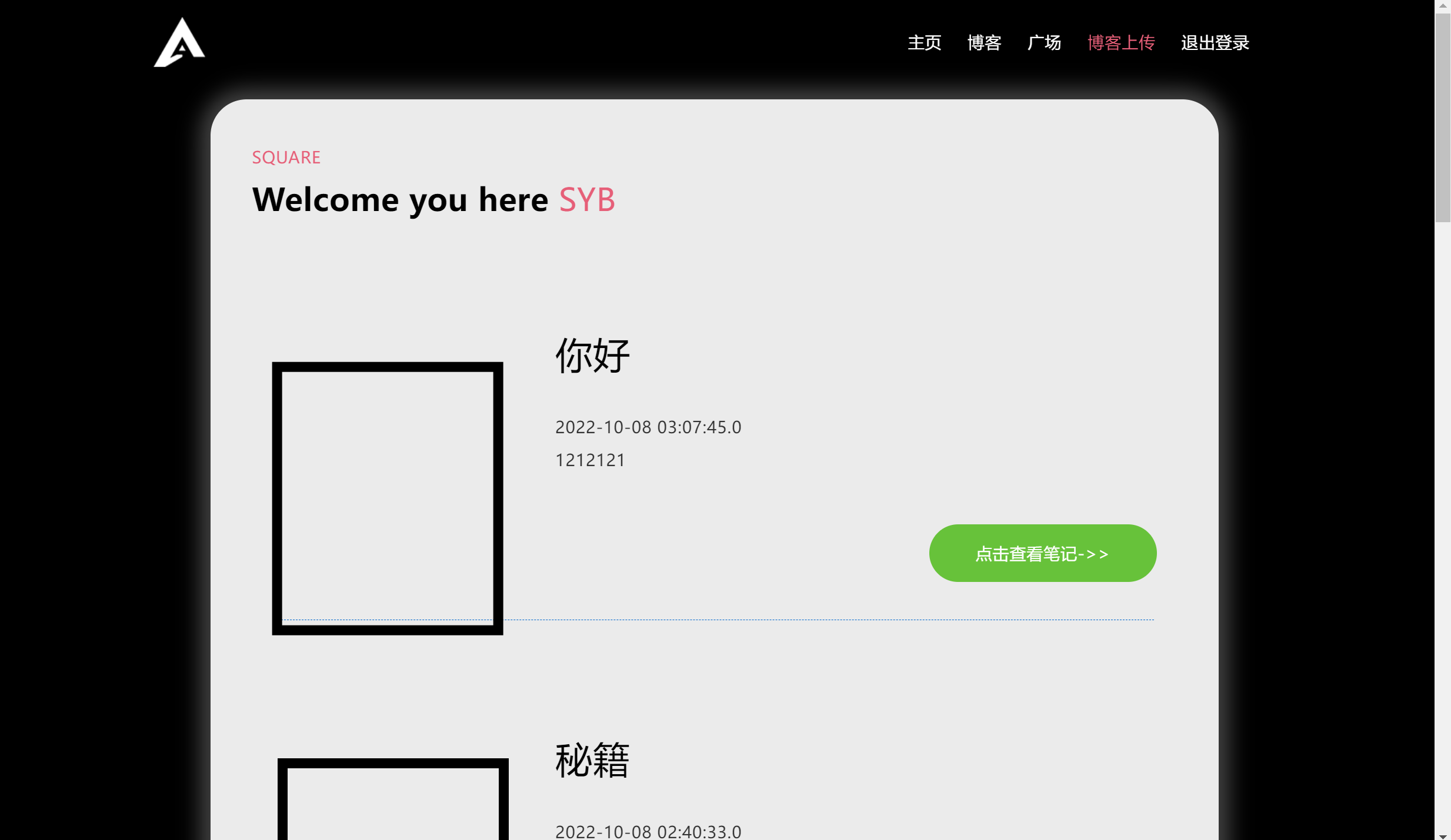
## 3. 广场页面搭建

广场页面实现也不复杂，对之前导航栏组件复用可以快速搭建。页面博客展示同样也是使用thymeleaf语法后台获取用户博客信息后，通过th:each遍历展示博客。通过判断用户是否登录来展示不同的导航栏。不同的是后台，后台需要对所有博客进行映射收集，传递给视图解析器进行解析。

<div th:each="blogPathMessages:${BlogPathMessages}">  
 <div class="about\_inner">  
 <div class="left">  
 <div class="about\_image\_wrap parallax" data-relative-input="true">  
 <div class="image layer" data-depth="0.1"> <img  
 src="../../static/images/ratios/550x650.jpg" alt="" />  
 <div class="inner" data-img-url="">  
 </div>  
 </div>  
 <div class="border layer" data-depth="0.2"> <img  
 src="../../static/images/ratios/550x650.jpg" alt="" />  
 <div class="inner"></div>  
 </div>  
 </div>  
 </div>  
 <div class="right">  
 <div class="about\_title app">  
 <h2 th:text="${blogPathMessages.value.getBlogTitle}"></h2>  
  
  
 </div>  
 <div class="text">  
 <p th:text="${blogPathMessages.value.getBlogData}">  
 <p th:text="${blogPathMessages.value.getBlogMessage}"></p>  
 </div>  
 <div class="about\_short\_contact\_wrap">  
  
 </div>  
 <div class="arlo\_tm\_button"> <a  
 th:href="@{'/blog/'+${blogPathMessages.key.getBlogPathPath}}"  
 style="background-color: #67c23a !important;"><span>点击查看笔记->></span></a> </div>  
 </div>  
 </div>  
 <hr style="height:1px;width: 95%; border:none;margin-top: 2%; border-top:1px dashed #0066CC;" />  
</div>

广场页面

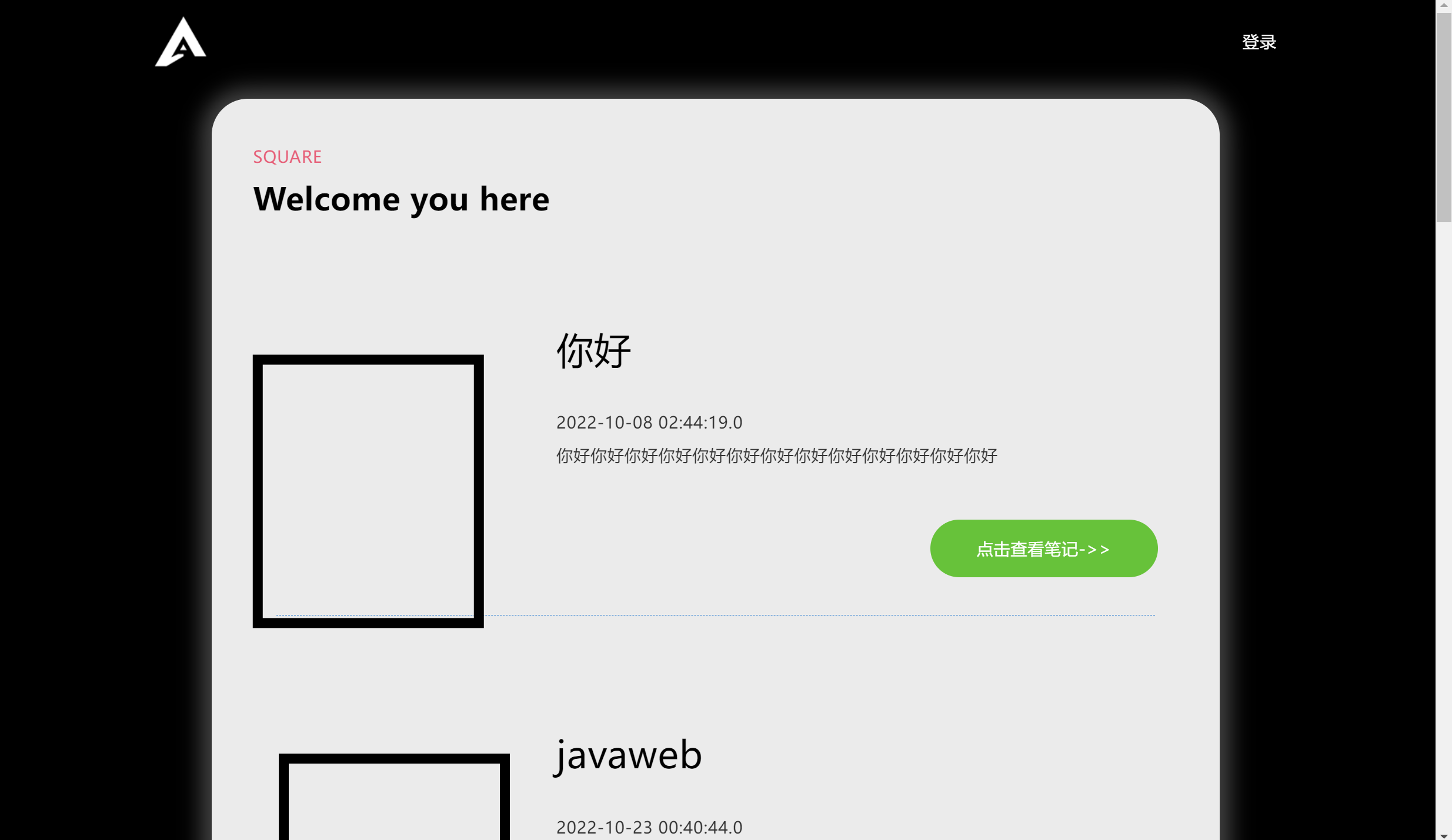
[广场页面：](https://gitee.com/Acido/images/raw/master/image/202210232146126.png)



广场页面

广场页面(未登陆)

[广场页面(未登陆)：](https://gitee.com/Acido/images/raw/master/image/202210232147878.png)



广场页面(未登录)

# 五. 项目后端代码实现

后端采用分层思想：

* pojo层主要放实体类
  + 类中定义了多个类属性，并与数据库表的字段保持一致，一张表对应一个model类。
  + 主要用于定义与数据库对象应的属性，提供get/set方法,tostring方法,有参无参构造函数。
* dao(mapper)层写接口，里面主要为操作数据库各种方法
  + 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。
* service层(业务层)实现接口层具体方法，所以service(业务层)调dao层。这两层统称为Model。
  + 首先设计接口，再设计其实现的类；
  + 接着在Spring的配置文件中配置其实现的关联，这样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。
* controller层，控制层用于控制请求，类似与servlet。该层调用service层。
  + 在此层里面要调用Service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，在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。

各层之间的联系：

Dao层、Service层都可以单独开发，互相的耦合度很低，完全可以独立进行，这样的一种模式在开发大项目的过程中尤其有优势；

Service逻辑层设计是建立在Dao层之上的，建立了Dao层后才可以建立Service层，而Service层又是在Controller层之下的，因而Service层应该既要调用Dao层的接口，又要提供接口给Controller层的类来进行调用，它刚好处于一个中间层的位置。每个模型都有一个Service接口，每个接口分别封装各自的业务处理方法；

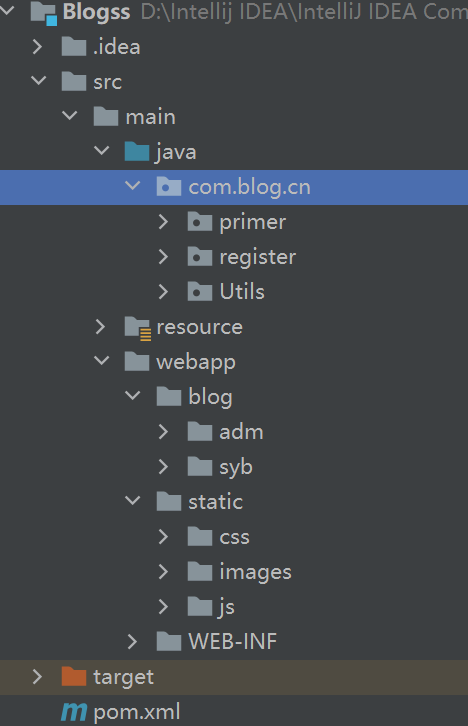
Controller层、View层因为耦合度比较高，因而要结合在一起开发，但是也可以看作一个整体独立于前两个层进行开发。这样，在层与层之前我们只需要知道接口的定义，调用接口即可完成所需要的逻辑单元应用，一切显得非常清晰简单。

## 1. 后端项目分层结构

我将整个项目分三个包。包中分为四层。这三个包分别是register(处理用户相关信息)、primer(处理博客相关内容)。Utils(用于存放工具类)。四层分别为pojo层、mapper层、service层、controller层。其中service层调用mapper层，mapper层是数据库接口。service层中的接口实现类调用mapper层，并对数据库中查询到的结果进行增强。最后Controller层处理映射请求，将用户请求合理调用service层中的方法，并对service层中返回结果做最后处理，通过视图传递给前端，或者返回JSON字符，展现在页面。

项目包结构如下：

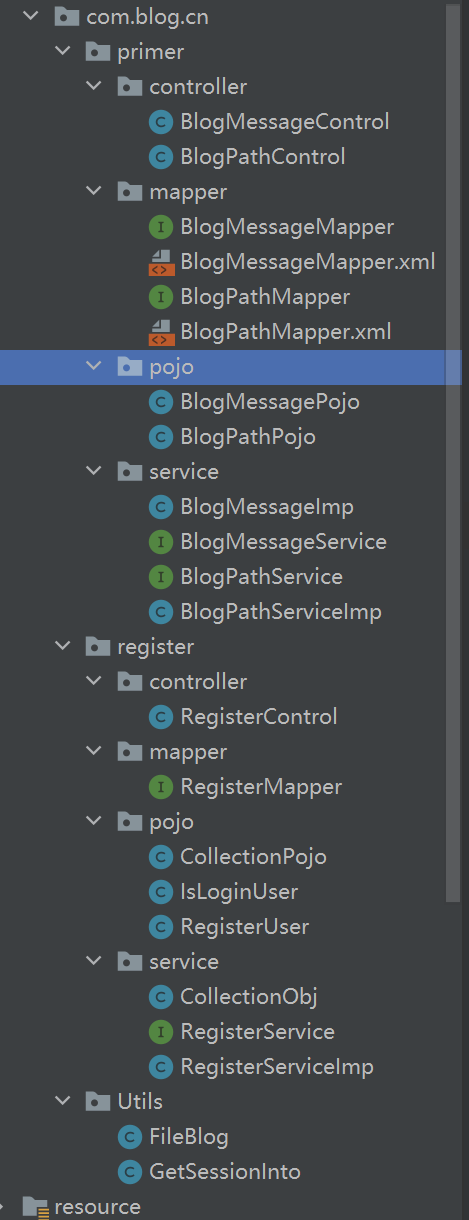
[项目包结构：](https://gitee.com/Acido/images/raw/master/image/202210242018684.png)



项目包结构

项目层结构如下：

[项目层结构：](https://gitee.com/Acido/images/raw/master/image/202210242019083.png)



项目层结构

## 2. 用户登录界面后端实现

* 先拦截index.html的请求，判断用户是否登录，如果用户已经登录，则直接跳转至success.html即个人主页。如果用户没有登录，则显示index.html页面，供用户登录，或者进入广场。判断方法是通过session查找里面是否有用户信息。为null则表示没有登录。
* @RequestMapping(value = "/")   
  public String isUser(HttpSession session){   
   if (session.getAttribute("loginUser")!=null){   
   return "redirect:/success";   
   }   
   return "index";   
  }
* 注意这里使用的登录成功是重定向视图。重定向和请求转发最大区别是请求转发会携带当前页面请求和参数在request域中，而重定向则不会共享数据。并且转发不会改变地址栏，但请求重定向会改变地址栏。
* 如果判断用户没有登录即session中存放用户字段为null，则显示index页面，此时用户可以选择注册或者登录。当用户登录时会用RESTful风格发送/register请求用于登录判断，通过service层调用mapper接口查询数据库中用户信息，获取数据库中用户账号和密码，判断是否与用户输入密码相同。错误则重新输入，成功则进入主页。然后通过工具类将用户账号信息，存放在session中，然后，通过cookie实现用户免登录。注意这里实际上应该至少通过MD5对密码进行加密放入会话和数据库但由于时间问题并未实行，希望以后可以完善安全方面不足吧
* @RequestMapping(value = "/register",method = RequestMethod.POST)   
  public String register(String registerUsername,String registerPassword,   
   HttpServletResponse response,HttpSession session){   
   RegisterUser registerUser=new RegisterUser(registerUsername,registerPassword,null);   
   List<RegisterUser> list=registerService.randomSelect(registerUser);   
   if (list!=null&&!list.isEmpty()){   
   for (RegisterUser r:list) {   
   if (r.getRegisterUsername().equals(registerUsername)&&r.getRegisterPassword().equals(registerPassword)){   
   IsLoginUser isLoginUser=new IsLoginUser(r.getRegisterUsername(),r.getRegisterPassword());   
   new GetSessionInto(response,null,session).getSessionInto("loginUser",isLoginUser);   
   return "redirect:/success";   
   }   
   }   
   }   
   return "redirect:/";   
  }
* 我们可以看到先获取到了用户输入账号和密码，然后用户表中查询数据库与之相匹配的信息，查到则，判断密码。下面是封装简单的工具类：
* package com.blog.cn.Utils;   
     
     
  import javax.servlet.http.Cookie;   
  import javax.servlet.http.HttpServletRequest;   
  import javax.servlet.http.HttpServletResponse;   
  import javax.servlet.http.HttpSession;   
     
  public class GetSessionInto {   
   private HttpServletResponse response;   
   private HttpServletRequest request;   
   private HttpSession session;   
     
   public GetSessionInto(HttpServletResponse response, HttpServletRequest request, HttpSession session) {   
   this.response = response;   
   this.request = request;   
   this.session = session;   
   }   
     
   public void getSessions(String sessionName,Object obj){   
   session.setAttribute(sessionName,obj);   
   }   
     
   public void getSessionInto(String sessionName,Object obj){   
   session.setAttribute(sessionName,obj);   
   session.setMaxInactiveInterval(7\*24\*60\*60);   
   Cookie cookie=new Cookie("JSESSIONID",session.getId());   
   cookie.setMaxAge(7\*24\*60\*60);   
   cookie.setPath("/");   
   response.addCookie(cookie);   
   }   
  }
* 如果用户选择注册账号，则会接收用户输入账号、密码和email，然后通过mapper接口将信息放入插入数据库。
* @RequestMapping(value = "/login",method = RequestMethod.POST)   
  public String liginUser(String registerUsername,String registerPassword,   
   String registerPassword2,String email,   
   HttpServletResponse response, HttpSession session){   
   if (registerPassword.equals(registerPassword2)){   
   RegisterUser registerUser=new RegisterUser(registerUsername,registerPassword,email);   
   new GetSessionInto(response,null,session).getSessionInto("loginUser",registerUser);   
   //添加到数据库   
   registerService.addUser(registerUser);   
   return "redirect:/success";   
   }   
   return "redirect:/";   
  }
* 这里用到了GetSessionInto类，这是一个工具类，主要是在用户注册完毕之后实现免登录用。

至此用户登录界面核心功能和代码已展示完毕。

## 3. 个人主页页面后端实现

* 博客展示是一个核心的处理器映射请求，在用户登录之后将用户的博客信息展示在页面。实现不复杂，主要是先通过session中存放的用户名，查询数据库获取用户ID，通过用户ID到register\_blogpath表中获取有关用户的所有博客页地址和博客的ID，再通过博客ID到register\_blogmessage表中根据博客ID获取博客标题和博客简介。将这两个表中所有字段通过pojo实体类映射获取结果集。返回结果是一个LIST集合。
* 下面是mapper接口数据库查询语言
* SELECT \* FROM   
  register JOIN register\_blogpath path   
  ON register.`register\_id`=path.`blogPath\_id`   
  WHERE register.`register\_id`=#{registerID}
* 下面是Control层业务代码
* @RequestMapping(value = "/success")   
  public String success(HttpSession session, Model model){   
   if (session.getAttribute("loginUser")!=null){   
   IsLoginUser reg= (IsLoginUser) session.getAttribute("loginUser");   
   List<RegisterUser> registerUsersName=registerService.randomSelect(reg);   
   if (registerUsersName!=null){   
   List<BlogPathPojo> blogPaths=blogPathService.selectById(registerUsersName.get(0).getRegisterId());   
   Integer arr[] = new Integer[blogPaths.size()],i=0;   
   for (BlogPathPojo bPojo:blogPaths) {   
   arr[i++]=bPojo.getBlogId();   
   }   
   if (arr.length!=0){   
   List<BlogMessagePojo> blogMessagePojos= blogMessageService.selectByIdEach(arr);   
   Map<BlogPathPojo,BlogMessagePojo> map= CollectionObj.BlogMap(blogPaths,blogMessagePojos);   
   if (map!=null){   
   model.addAttribute("BlogPathMessage",map);   
   }   
   }   
     
   }   
   return "/success";   
   }   
   return "redirect:/";   
  }
* 由于这一模块开发较早，所以没有使用连表查询，而是将用户博客ID查询出来后，放入一个数组，再将数组传递给mapper层，通过mybatis的<foreach>标签循环查询结果，封装在一个LIST集合中再通过map集合将两个LIST集合合并成一个map集合。使用model视图将封装的map信息通过request域传递给前端，再通过视图解析器循环展示给用户。
* 用户在主页可以选择退出登录。
* @RequestMapping(value = "/logout")   
  public String logout(HttpServletRequest request){   
   request.getSession().invalidate();   
   return "redirect:/";   
  }
* 实现十分简单，就是将session会话销毁，并删除保存用户会话的cookie。
* 博客上传功能实现也是一大核心模块。我们可以将用户通过文件表单，提交过来的二进制文件进行MD5加密，然后将请求中的博客标题，博客简介和加密的MD5值存储在数据库。这里的MD5加密仅仅为了防止用户发送重复博客文件。之后通过工具类FileBlog将文件保存在webapp/blog/用户名目录下。返回路径地址，同样存放在数据库中。这些存放在数据库中的信息就是register\_blogmessage表的核心字段。下面是代码实现：
* @RequestMapping(value = "/uploadBlog/{registerUsername}",method = RequestMethod.POST)   
  @ResponseBody   
  public String uploadBlog(@RequestParam("upload") MultipartFile blog,   
   @PathVariable("registerUsername")String registerUsername,   
   String blogTitle,String blogText,   
   HttpSession session) throws IOException{   
   String md5 = DigestUtils.md5DigestAsHex(blog.getInputStream());   
   List<RegisterUser> users=registerService.randomSelect(new RegisterUser(registerUsername,null,null));   
   BlogPathPojo blogPathPojoMd5=new BlogPathPojo(users.get(0).getRegisterId(),null,md5);   
   //判断上传文件md5值是否在数据库中存在   
   List<BlogPathPojo> md5s=blogPathService.randomBlogSelectStruct(blogPathPojoMd5);   
   if (md5s.isEmpty()){   
   String[] blogUp=new FileBlog(blog,registerUsername,session).renewFileName(null);   
   RegisterUser registerUser=new RegisterUser(registerUsername,null,null);   
   BlogMessagePojo blogMessagePojo=new BlogMessagePojo(blogTitle,blogText,null);   
   List<RegisterUser> registerId= registerService.randomSelect(registerUser);   
   if (!registerId.isEmpty()&&blogUp!=null){   
   for (RegisterUser itemId:registerId) {   
   if (itemId.getRegisterUsername().equals(registerUsername)){   
   BlogPathPojo blogPathPojo=new BlogPathPojo(itemId.getRegisterId(),blogUp[1],blogUp[0]);   
   BlogPathPojo pathPojo=new BlogPathPojo(blogPathPojo.getBlogPathId(),null,blogPathPojo.getBlogMd5());   
   //在service中判断path是否为null，只有满足用户ID和文件md5都不存在则添加   
   List<BlogPathPojo> path=blogPathService.randomBlogSelectStruct(pathPojo);   
   blogPathService.addBlog(blogPathPojo,blogMessagePojo,path);   
   return "OK";   
   }   
   }   
   }   
   }   
   return "error";   
  }
* 上面代码我们先获取用户提交过来的博客文件和信息，通过MD5对文件进行加密获取MD5值，再查询数据库中与用户对应博客信息表中md5值，如果返回结果为0则LIST集合为null表示在该用户的博客中没有上传过此类博客文件，接着封装博客信息，通过文件上传工具类将文件保存在上文目录下，并返回文件路径，存放在数据库中。通过@ResponseBody注解返回字符用于关闭前端iframe子页面。
* 下面是文件上传工具类：
* public String[] renewFileName(BlogPathPojo md5s)throws IOException {   
     
   String fileName = blog.getOriginalFilename();   
   String fileUserName=registerUsername;   
   String hzName = fileName.substring(fileName.lastIndexOf("."));   
     
   fileName = UUID.randomUUID().toString() + hzName;   
   ServletContext servletContext = session.getServletContext();   
   String blogPath = servletContext.getRealPath("blog/"+fileUserName);   
   File file = new File(blogPath);   
   if(!file.exists()){   
   file.mkdir();   
   }   
   String finalPath = blogPath + File.separator + fileName;   
   String md5=DigestUtils.md5DigestAsHex(blog.getInputStream());   
   if (md5s==null){   
   blog.transferTo(new File(finalPath));   
   String webBlogPath='/'+registerUsername+'/'+fileName;   
   String[] blogUp={md5,webBlogPath};   
   return blogUp;   
   }else {   
   return null;   
   }   
  }
* 博客删除功能是通过工具类获取文件路径根据路径拼接文件名，最后再通过文件类方法删除该博客即可
* 处理映射请求：
* @RequestMapping(value = "/delete/{blogPathId}/{blogMessageId}")   
  public String deleteBlog(@PathVariable("blogPathId")int blogPathId,   
   @PathVariable("blogMessageId") int blogMessageId,   
   HttpSession session){   
   if (session.getAttribute("loginUser")!=null){   
   IsLoginUser reg= (IsLoginUser) session.getAttribute("loginUser");   
   List<RegisterUser> registerUsersName=registerService.randomSelect(reg);   
   if (registerUsersName!=null){   
   List<BlogPathPojo> blogPathPojo= blogPathService.selectById(registerUsersName.get(0).getRegisterId());   
   for (BlogPathPojo pojoId:blogPathPojo) {   
   if (pojoId.getBlogId()==blogPathId){   
   List<BlogPathPojo> listBlog=blogPathService.selectByIdOne(blogPathId);   
   String fileName=listBlog.get(0).getBlogPathPath();   
   blogPathService.deleteBlog(blogPathId);   
   blogMessageService.deleteMessage(blogMessageId);   
   new FileBlog().deleteBlog(fileName,session);   
   return "redirect:/";   
   }   
   }   
   }   
     
   }   
   return "redirect:/error";   
  }
* 删除博客工具类
* public boolean deleteBlog(String fileName,HttpSession session){   
   ServletContext servletContext = session.getServletContext();   
   String blogPath = servletContext.getRealPath("blog");   
   String finalPath = blogPath + File.separator + fileName;   
   File file=new File(finalPath);   
   return file.delete();   
  }

至此博客个人主页后端实现核心代码展示完毕。

## 4. 广场页面后端实现

广场实现实际上就是将数据库中所有用户对应博客查询出来，通过map集合传递给前端，通过视图解析器遍历展示即可。代码实现如下：

//广场   
@RequestMapping("/")   
public String blogSuccess(Model model){   
   
 List<BlogPathPojo> pathAll =blogPathService.selectAll();   
 List<BlogMessagePojo> messageAll=blogMessageService.selectAllMessage();   
 Map<BlogPathPojo,BlogMessagePojo> map= CollectionObj.BlogMap(pathAll,messageAll);   
 model.addAttribute("BlogPathMessages",map);   
 System.out.println(map);   
 return "/square";   
}

# 六. 程序测试与评价

软件测试是检查实际结果与预期结果是否匹配并确保软件系统无缺陷的活动。软件测试还有助于识别产品与实际需求不符或是缺失项。测试活动既可以手动完成，也可以使用自动化工具完成。有些人更喜欢将软件测试称为白盒和黑盒测试。其主要目的是在给定的产品中尽可能多地发现错误(或bug)。演示一个给定的软件产品与它的需求规格匹配。使用最小的成本和努力来验证软件的质量。生成高质量的测试用例，执行有效的测试，并发布正确和有用的问题报告。测试很重要，因为软件错误可能很昂贵甚至很危险，一个bug越长越不被发现，它就可能带来更大的隐患。此外，如果测试团队错过了准确详尽地捕捉或识别风险和软件问题，会导致一些灾难发生，那么就需要付出昂贵的代价来检测bug，所以软件漏洞或是Bug可能会导致货币和人员的损失，历史上太多这样的例子。所以我们要再一次次工程中总结经验，通过不断测试，规避软件因BUG产生风险。

## 1. 软件测试步骤

* 单元测试
* 这是在开发人员级别使用的最基本的测试，测试人员专注于单元代码的单个部分，而它已经从任何外部交互或依赖于任何模块之前被隔离。这个测试要求开发人员检查他们编写的最小代码单元，并证明单元可以独立工作。
* 集成测试
* 在开发人员级别上，在单元测试之后，还应该仔细检查这些最小代码的组合(或集成)。集成测试提供了访问网络、数据库和文件系统的测试模块。
* 它们将指示数据库和网络在合并到整个系统时是否运行良好。最重要的是，在前一阶段测试的小代码单元之间的连接将在这个阶段被证明。
* 功能测试
* 毫无疑问，功能测试是更高级别的测试类型，应该在集成测试之后使用。功能测试检查输出与规范中定义的输入的准确性。对中间值不太重视，但对所创建的最终输出给予了更多的关注。
* 冒烟测试
* 冒烟测试的类比来自于电子产品，其中一个问题意味着电路板散发出烟雾。在功能测试完成之后，在新安装和更新的输入值之后，将在起始点执行一个简单的测试。
* 回归测试
* 当系统中出现复杂的bug时，通常会影响系统的核心区域，所以使用回归测试来重新测试系统的所有模块。
* UI测试
* 除了上面的核心测试类型， UI测试现在也是一个众所周知的，在软件工程行业非常流行。该图形用户界面测试确保了对所有用户友好的特定应用程序或产品。UI测试主要评估设计组件，如布局、颜色、字体、大小等等。另外， UI测试可以手动和自动执行。

测试策略的重要性，在于它是一个非常宏观的，对后期测试执行具有明确指导作用的方法论。就是说，在理解了用户需求的基础上，要能够预测到测试的重点、难点;测试执行的程度，需要借助什么工具，或者是否需要自研工具;测试对外部环境的依赖有哪些?如何解决这些依赖;测试中有哪些风险需要提前注意以及如何去应对这些风险;在测试完成后，如何对缺陷进行分析，如何对该产品做产品质量评估;在测试人员资源分配上如何更合理，测试进度如何控制等。

## 2. 测试用例

用户登录测试：

| 测试目的 | 测试用例 | 预测结果 | 实际结果 |
| --- | --- | --- | --- |
| 用户登录 | 用户名：1 密码：a | 登录失败 | 提示错误信息 |
|  | 用户名：adm 密码：123 | 登录成功 | 登陆成功，跳转至主页 |

个人主页页面测试：

| 测试目的 | 测试用例 | 预测结果 | 实际结果 |
| --- | --- | --- | --- |
| 判断用户登录/未登录情况下是否可以访问 | 通过连接直接访问(未登录) | 跳转至登录页面 | 跳转至登录页面 |
|  | 通过连接直接访问(已登录) | 成功进入页面 | 成功进入页面 |

个人主页上传博客测试：

| 测试目的 | 测试用例 | 预测结果 | 实际结果 |
| --- | --- | --- | --- |
| 判断文件是否成功上传 | 不输入博客标题和简介，上传不是html文件 | 子页面报错：ERROR | 报错重新上传 |
|  | 输入标题和简介，但上传不是html文件 | 子页面报错：ERROR | 报错重新上传，提示文件格式不正确 |
|  | 输入标题和简介，上传html文件 | 子页面显示OK并关闭 | 子页面显示OK并关闭 |

## 3. 项目存在不足和改进方式

系统存在不足主要为以下几个方面：

1. 后端代码过于冗长，由于平时练习项目较少，导致封装等思想体现较少。大部分代码以能够实现功能为主，所以代码不够美观。在一些可以重复套用方法上由于存在细微业务差距，而没有进行整体代码抽取封装，导致业务代码过于冗长，且功能越来越复杂，代码可读性越来越差，耦合度越来越高，且后期维护时间大大增加。这也是这次项目时间中暴露最大问题。再以后项目开发中会进行改进。
2. 分层不够清晰，由于web开发最关键的就是对各个层次结构分离实现，以提高代码内聚程度，降低代码耦合度。由于项目经验缺少，导致不能够灵活分析层次结构，而是套用课本分层结构，对业务，数据库接口，控制层等缺乏更深刻的体会，导致各种功能分布不一，造成"头重脚轻"的严重问题。
3. 缺乏数据库实践经验。由于课程上的安排，导致本人接触数据库较晚。虽然之前已经通过各种渠道掌握了数据库一些基本增删改查方法。但在实际的业务逻辑中，使用最底层方法写出的sql语句过于简单，不能实现复杂的数据结构查询，而是通过间接查询方式一层一层查询表数据。这样势必增加了因数据处理不当而导致整个系统崩溃的风险。如在上面上传博客后端实现一节中，本应该使用事务管理(ACID)方式实现上传，但由于经验缺乏导致并未用这种最优方案实现。之后会将这一模块重新设计完善。

解决方案

1. 针对后端代码过于冗长这一问题，我会多练习，平时多写一些小程序。不断培养自己封装思维。多积累经验。
2. 关于分层不够清晰这一点，我会多积累项目经验，做笔记进行问题总结，改正思维惯性。多分析优秀开源项目的源码，对其中的框架结构深入了解，一次提高自己能力和技能。
3. 数据库这一方面我会加大对数据库学习，通过不断练习，对本项目进行不断完善。并深入了解其运行机制和优化方案。减轻由不断后台不断访问数据库对服务器的压力，学习数据库框架，核心知识。通过对框架了解进一步结构清晰，通俗易懂实现mapper层数据库接口编写。优化pojo层由于字段较多，resultMap结果集映射复杂等情况下，数据库查询结构。改变后端各层之间调用部分代码冗长导致后期较难维护情况。

# 七. 结语

本次毕业设计介绍和展示到此已经结束。经过这次程序设计我十分清晰看到了自己不足和能提升的空间。这次项目设计从分析到完成只用了国庆假期一个星期时间，时间十分紧凑，所以前期在需求分析阶段和数据库设计阶段并没有充足时间进行思考，而是做了最基本的软件规划和流程导图，通过导图设计数据库中表和字段，这样做最大弊端就是后期增加需求时，要改变各层中代码，并改变数据库结构。这样做无疑增加后期维护成本。但也有好处，即通过网页流程导图可以快速实现网页核心功能，同时也能节省大部分代码调试时间。所以总的来说，这次毕业设计对我的提升很大。

对于这次设计增加我对这个专业和web开发热爱程度。通过一次次解决程序问题和实现需求过程，让我不仅收获了知识而且也锻炼了品质，通过这次认真而又细致的毕业设计，我对待事情的态度更加严谨更加有耐心，而且我更希望把所做的事情做好做完美，我想这将是一种很重要的财富。感谢本次设计，感谢遇见了麻烦和难题，毕业设计的结束也是另一种开始，相信本次毕业设计会令我走得更远也能取得更大的成就。也为我们在校理论知识的学习画上了句号。不由感慨颇多，宝贵的大学学习生活转瞬即逝，无论如何，我们的毕业设计已经结束了，我们要好好利用已经学到的知识并且要继续学习掌握新的知识，以便更新我们的知识系统。重要的是，我们要将所获取的各种知识运用于实践，并用实践来休整我们的不足和缺陷，使我们真正成为一名理论扎实而又具有很强的实践能力的工作者。