附　录

附录 1 用户登录和注册功能实现

```
/**
* 用户注册
*
* @param user 用户信息
* @return {@link Result<>}
*/
@ApiOperation(value = "用户注册")
@PostMapping("/register")
public Result<?> register(@Valid @RequestBody UserVO user) {
userAuthService.register(user);
return Result.ok();
}
@Transactional(rollbackFor = Exception.class)
    @Override
    public void register(UserVO user) {
        // 校验账号是否合法
        if (checkUser(user)) {
            throw new BizException("邮箱已被注册！");
        }
        // 新增用户信息
        UserInfo userInfo = UserInfo.builder()
                .email(user.getUsername())
                .nickname(CommonConst.DEFAULT_NICKNAME +
IdWorker.getId())
                .avatar(blogInfoService.getWebsiteConfig().getUserAvatar())
```

```
        .build();

userInfoDao.insert(userInfo);

// 绑定用户角色

UserRole userRole = UserRole.builder()

        .userId(userInfo.getId())

        .roleId(RoleEnum.USER.getRoleId())

        .build();

userRoleDao.insert(userRole);

// 新增用户账号

UserAuth userAuth = UserAuth.builder()

        .userInfoId(userInfo.getId())

        .username(user.getUsername())

        .password(BCrypt.hashpw(user.getPassword(), BCrypt.gensalt()))

        .loginType(LoginTypeEnum.EMAIL.getType())

        .build();

userAuthDao.insert(userAuth);

}
```

附录 2 验证码功能实现

```
@Override

    public void sendCode(String username) {

        // 校验账号是否合法

        if (!checkEmail(username)) {

            throw new BizException("请输入正确邮箱");

        }

        // 生成六位随机验证码发送

        String code = getRandomCode();

        // 发送验证码

        EmailDTO emailDTO = EmailDTO.builder()

                .email(username)

                .subject("验证码")

                .content("您的验证码为 " + code + " 有效期 15 分钟，请不要
告诉他人哦！")

                .build();

        rabbitTemplate.convertAndSend(EMAIL_EXCHANGE, "*", new
Message(JSON.toJSONBytes(emailDTO), new MessageProperties()));

        // 将验证码存入 redis，设置过期时间为 15 分钟

        redisService.set(USER_CODE_KEY + username, code,
CODE_EXPIRE_TIME);

    }
```

附录 3 前端文章添加功能实现

```
import * as imageConversion from "image-conversion";

export default {

  created() {

    const path = this.$route.path;

    const arr = path.split("/");

    const articleId = arr[2];

    if (articleId) {

      this.axios.get("/api/admin/articles/" + articleId).then(({ data }) => {

        this.article = data.data;

      });

    } else {

      const article = sessionStorage.getItem("article");

      if (article) {

        this.article = JSON.parse(article);

      }

    }

  },

    openModel() {

      if (this.article.articleTitle.trim() == "") {

        this.$message.error("文章标题不能为空");

        return false;

      }

      if (this.article.articleContent.trim() == "") {

        this.$message.error("文章内容不能为空");

        return false;
```

附录 4 后端文章添加功能实现

```
<select id="listArticlesByCondition" resultMap="articlePreviewResultMap">
    SELECT
    a.id,
    article_cover,
    article_title,
    a.create_time,
    a.category_id,
    category_name,
    t.id AS tag_id,
    t.tag_name
    FROM
    (
    SELECT
    id,
    article_cover,
    article_title,
    article_content,
    create_time,
    category_id
    FROM
    tb_article
    <where>
        <if test="condition.categoryId != null">
            category_id = #{condition.categoryId}
        </if>
        <if test="condition.tagId != null">
```

```
id IN (

SELECT

article_id

FROM

tb_article_tag

WHERE

tag_id = #{condition.tagId})

</if>

</where>

AND is_delete = 0

AND status = 1

ORDER BY id DESC

LIMIT #{current},#{size}

) a

JOIN tb_category c ON a.category_id = c.id

JOIN tb_article_tag atg ON a.id = atg.article_id

JOIN tb_tag t ON t.id = atg.tag_id

</select>
```

附录 5 文章搜索功能实现

```
@Override
    PublicArticlePreviewListDTO listArticlesByCondition(ConditionVO condition)
{
        // 查询文章
        List<ArticlePreviewDTO> articlePreviewDTOList =
articleDao.listArticlesByCondition(PageUtils.getLimitCurrent(), PageUtils.getSize(),
condition);
        // 搜索条件对应名(标签或分类名)
        String name;
        if (Objects.nonNull(condition.getCategoryId())) {
            name = categoryDao.selectOne(new
LambdaQueryWrapper<Category>()
                            .select(Category::getCategoryName)
                            .eq(Category::getId, condition.getCategoryId()))
                    .getCategoryName();
        } else {
            name = tagService.getOne(new LambdaQueryWrapper<Tag>()
                            .select(Tag::getTagName)
                            .eq(Tag::getId, condition.getTagId()))
                    .getTagName();
        }
        return ArticlePreviewListDTO.builder()
                .articlePreviewDTOList(articlePreviewDTOList)
                .name(name)
                .build();
    }
```

附录 6 评论功能实现

```java
Integer userId = BLOGGER_ID;

String id = Objects.nonNull(comment.getTopicId()) ?
comment.getTopicId().toString() : "";

if (Objects.nonNull(comment.getReplyUserId())) {

    userId = comment.getReplyUserId();

} else {

    switch
(Objects.requireNonNull(getCommentEnum(comment.getType()))) {

        case ARTICLE:

            userId =
articleDao.selectById(comment.getTopicId()).getUserId();

            break;

        case TALK:

            userId =
talkDao.selectById(comment.getTopicId()).getUserId();

            break;

        default:

            break;

    }

}

String email = userInfoDao.selectById(userId).getEmail();

if (StringUtils.isNotBlank(email)) {

    EmailDTO emailDTO = new EmailDTO();

    if (comment.getIsReview().equals(TRUE)) {

        // 评论提醒

        emailDTO.setEmail(email);
```

```
                    emailDTO.setSubject("评论提醒");

                    // 获取评论路径

                    String url = websiteUrl + getCommentPath(comment.getType())
+ id;

                    emailDTO.setContent("您收到了一条新的回复，请前往" + url
+ "\n 页面查看");

                } else {

                    // 管理员审核提醒

                    String adminEmail =
userInfoDao.selectById(BLOGGER_ID).getEmail();

                    emailDTO.setEmail(adminEmail);

                    emailDTO.setSubject("审核提醒");

                    emailDTO.setContent("您收到了一条新的回复，请前往后台管
理页面审核");

                }

                rabbitTemplate.convertAndSend(EMAIL_EXCHANGE, "*", new
Message(JSON.toJSONBytes(emailDTO), new MessageProperties()));

        }
```

附录 7 数据库连接

配置 mysql 数据库

spring:

　　datasource:

　　　　type: com.zaxxer.hikari.HikariDataSource

　　　　driver-class-name: com.mysql.cj.jdbc.Driver

　　　　url:

jdbc:mysql:/127.0.0.1:3306/blog?serverTimezone=Asia/Shanghai&allowMultiQueries=true

　　　　username: root

　　　　password: 123456789

　　　　hikari:

　　　　　　minimum-idle: 5

　　　　　　# 空闲连接存活最大时间，默认 600000（10 分钟）

　　　　　　idle-timeout: 180000

　　　　　　# 连接池最大连接数，默认是 10

　　　　　　maximum-pool-size: 10

　　　　　　# 此属性控制从池返回的连接的默认自动提交行为,默认值：true

　　　　　　auto-commit: true

　　　　　　# 连接池名称

　　　　　　pool-name: MyHikariCP

　　　　　　# 此属性控制池中连接的最长生命周期，值 0 表示无限生命周期，默认

1800000 即 30 分钟

　　　　　　max-lifetime: 1800000

　　　　　　# 数据库连接超时时间,默认 30 秒，即 30000

　　　　　　connection-timeout: 30000

　　　　　　connection-test-query: SELECT 1

附录 8 缓存技术配置和实现

\# redis 配置

redis:

    host: 127.0.0.1

    port: 6379

    password: 123456789

```
@Configuration
public class RedisConfig {

    @Bean
    public RedisTemplate<String, Object> redisTemplate(RedisConnectionFactory factory) {
        RedisTemplate<String, Object> redisTemplate = new RedisTemplate<>();
        redisTemplate.setConnectionFactory(factory);
        Jackson2JsonRedisSerializer<Object> jackson2JsonRedisSerializer = new Jackson2JsonRedisSerializer<>(Object.class);
        ObjectMapper mapper = new ObjectMapper();
        mapper.setVisibility(PropertyAccessor.ALL, JsonAutoDetect.Visibility.ANY);
        // mapper.enableDefaultTyping(ObjectMapper.DefaultTyping.NON_FINAL);
        mapper.activateDefaultTyping(LaissezFaireSubTypeValidator.instance,
                ObjectMapper.DefaultTyping.NON_FINAL,
                JsonTypeInfo.As.PROPERTY);
        jackson2JsonRedisSerializer.setObjectMapper(mapper);
        StringRedisSerializer stringRedisSerializer = new StringRedisSerializer();
        // key 采用 String 的序列化方式
```

```
        redisTemplate.setKeySerializer(stringRedisSerializer);

        // hash 的 key 也采用 String 的序列化方式

        redisTemplate.setHashKeySerializer(stringRedisSerializer);

        // value 序列化方式采用 jackson

        redisTemplate.setValueSerializer(jackson2JsonRedisSerializer);

        // hash 的 value 序列化方式采用 jackson

        redisTemplate.setHashValueSerializer(jackson2JsonRedisSerializer);

        redisTemplate.afterPropertiesSet();

        return redisTemplate;

    }

}
```