JancoBlog 个人博客 V2.0

```
import com.baomidou.mybatisplus.annotation.TableField;
import com.baomidou.mybatisplus.annotation.TableId;
import com.baomidou.mybatisplus.annotation.TableName;
import com.baomidou.mybatisplus.extension.activerecord.Model;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.EqualsAndHashCode;
import lombok.NoArgsConstructor;
import lombok.experimental.Accessors;
import java.io.IOException;
import java.io.UnsupportedEncodingException;
import java.time.LocalDateTime;
import java.io.Serializable;
import java.util.Date;
import java.util.List;
import java.util.Objects;
import java.util.UUID;
@Data
@EqualsAndHashCode(callSuper = true)
@AllArgsConstructor
@NoArgsConstructor
@Accessors(chain = true)
@TableName("tbl_article")
public class Article extends Model<Article> {
    private static final long serialVersionUID = 1L;
    @TableId(value = "article_id")
    private String articleId;
    private String articleTitle;
    private Integer articleAuthor;
    private Integer articleType;
    private String articleSummary;
    private String articleHtml;
    private String articleMd;
    private Integer articleIsComment;
    private Integer articleRank;
    private Date articlePostTime;
    private Date articleEditTime;
    private Integer articleViewCount;
    private Integer articleCommentCount;
    private Integer articleLikeCount;
    @TableField(exist = false)
    private String userName;
    @TableField(exist = false)
    private String typeName;
}
package com.jancoyan.jancoblog.pojo;
        import com.baomidou.mybatisplus.annotation.TableId;
        import com.baomidou.mybatisplus.annotation.TableName;
        import com.baomidou.mybatisplus.extension.activerecord.Model;
        import lombok.Data;
        import lombok.EqualsAndHashCode;
        import java.util.Date;
@EqualsAndHashCode(callSuper = true)
@TableName("tbl_article_image")
@Data
public class ArticleImage extends Model<ArticleImage> {
    private static final long serialVersionUID = 1L;
    @TableId(value = "filename")
    private String filename;
    private String articleId;
    private Date insertDate;
}
package com.jancoyan.jancoblog.pojo;
        import com.baomidou.mybatisplus.annotation.TableField;
        import com.baomidou.mybatisplus.annotation.TableName;
        import com.baomidou.mybatisplus.annotation.IdType;
        import com.baomidou.mybatisplus.extension.activerecord.Model;
        import com.baomidou.mybatisplus.annotation.TableId;
        import lombok.AllArgsConstructor;
        import lombok.Data;
        import lombok.EqualsAndHashCode;
        import lombok.NoArgsConstructor;
        import lombok.experimental.Accessors;
        import java.time.LocalDateTime;
        import java.io.Serializable;
        import java.util.Date;
@Data
@EqualsAndHashCode(callSuper = true)
@AllArgsConstructor
@NoArgsConstructor
@Accessors(chain = true)
@TableName("tbl_comment")
public class Comment extends Model<Comment> {
    private static final long serialVersionUID = 1L;
    @TableId(value = "comment_id", type = IdType.AUTO)
    private Integer commentId;
    private Integer preCommentId;
    private String commentArticleId;
    private Integer commentAuthorId;
    private String commentAuthorName;
    private String commentAuthorEmail;
    private String commentContent;
    private Date commentDate;
    private String commentAuthorIp;
    private Integer commentLikeCount;
    @TableField(exist = false)
    private String articleTitle;
}
package com.jancoyan.jancoblog.pojo;
        import com.baomidou.mybatisplus.annotation.TableField;
        import com.baomidou.mybatisplus.annotation.TableName;
        import com.baomidou.mybatisplus.annotation.IdType;
        import com.baomidou.mybatisplus.extension.activerecord.Model;
        import com.baomidou.mybatisplus.annotation.TableId;
        import lombok.AllArgsConstructor;
        import lombok.Data;
        import lombok.EqualsAndHashCode;
        import lombok.NoArgsConstructor;
        import lombok.experimental.Accessors;
        import java.time.LocalDateTime;
        import java.io.Serializable;
        import java.util.Date;
@Data
@EqualsAndHashCode(callSuper = true)
@AllArgsConstructor
@NoArgsConstructor
@Accessors(chain = true)
@TableName("tbl_like_record")
public class LikeRecord extends Model<LikeRecord> {
    private static final long serialVersionUID = 1L;
    @TableId(value = "like_id", type = IdType.AUTO)
    private Integer likeId;
    private String articleId;
    private Integer authorId;
    private Date likeDate;
    @TableField(exist = false)
    private String userName;
    @TableField(exist = false)
    private String articleTitle;
    @TableField(exist = false)
    private String articleAuthor;
}
package com.jancoyan.jancoblog.pojo;
        import com.baomidou.mybatisplus.extension.activerecord.Model;
        import lombok.AllArgsConstructor;
        import lombok.Data;
        import lombok.EqualsAndHashCode;
        import lombok.NoArgsConstructor;
        import lombok.experimental.Accessors;
        import java.util.Date;
@Data
@AllArgsConstructor
@NoArgsConstructor
@EqualsAndHashCode(callSuper = true)
@Accessors(chain = true)
public class PageArticle extends Model<PageArticle>{
    private String articleId;
    private Integer articleAuthor;
    private String articleTitle;
    private Date articlePostTime;
```

```
}
package com.jancoyan.jancoblog.pojo;
        import com.baomidou.mybatisplus.extension.activerecord.Model;
        import lombok.AllArgsConstructor;
        import lombok.Data;
        import lombok.EqualsAndHashCode;
        import lombok.NoArgsConstructor;
        import lombok.experimental.Accessors;
        import java.util.Date;
@EqualsAndHashCode(callSuper = true)
@Data
@AllArgsConstructor
@NoArgsConstructor
@Accessors(chain = true)
public class PageComment extends Model<LikeRecord> {
    private String commentAuthorName;
    private String articleId;
    private Integer articleAuthor;
    private String articleTitle;
    private Date commentDate;
}
package com.jancoyan.jancoblog.pojo;
        import com.baomidou.mybatisplus.annotation.TableField;
        import com.baomidou.mybatisplus.annotation.TableName;
        import com.baomidou.mybatisplus.annotation.IdType;
        import com.baomidou.mybatisplus.extension.activerecord.Model;
        import com.baomidou.mybatisplus.annotation.TableId;
        import lombok.AllArgsConstructor;
        import lombok.Data;
        import lombok.EqualsAndHashCode;
        import lombok.NoArgsConstructor;
        import lombok.experimental.Accessors;
        import java.util.Date;
@Data
@EqualsAndHashCode(callSuper = true)
@AllArgsConstructor
@NoArgsConstructor
@Accessors(chain = true)
@TableName("tbl_user")
public class User extends Model<User> {
    @TableId(value = "user_id", type = IdType.AUTO)
    private Integer userId;
    private String userName;
    private String userSignature;
    private String userPassword;
    private Integer userRole;
    private Date userCreateDate;
    private Date userLastLoginDate;
    private String userIp;
    @TableField(exist = false)
    private String roleName;
}
package com.jancoyan.jancoblog.pojo;
        import com.baomidou.mybatisplus.annotation.IdType;
        import com.baomidou.mybatisplus.annotation.TableId;
        import com.baomidou.mybatisplus.annotation.TableName;
        import com.baomidou.mybatisplus.extension.activerecord.Model;
        import lombok.AllArgsConstructor;
        import lombok.Data;
        import lombok.EqualsAndHashCode;
        import lombok.NoArgsConstructor;
        import lombok.experimental.Accessors;
        import java.util.Date;
@Data
@EqualsAndHashCode(callSuper = true)
@AllArgsConstructor
@NoArgsConstructor
@Accessors(chain = true)
@TableName("tbl_user_info")
public class UserInfo extends Model<UserInfo> {
    private static final long serialVersionUID = 1L;
    @TableId(value = "user_id", type = IdType.AUTO)
    private long userId;
    private String userEmail;
    private Integer userSex;
    private String userRegion;
    private Date userBirthdate;
    private String userTelephone;
    private String userRealName;
    private String userSchool;
    private String userMajor;
    private Date userEnterSchoolDate;
    private String userAcademicDegree;
    private String userCompany;
    private String userPosition;
    private String userField;
}
package com.jancoyan.jancoblog.pojo;
        import com.baomidou.mybatisplus.annotation.TableName;
        import com.baomidou.mybatisplus.annotation.IdType;
        import com.baomidou.mybatisplus.extension.activerecord.Model;
        import com.baomidou.mybatisplus.annotation.TableId;
        import lombok.AllArgsConstructor;
        import lombok.Data;
        import lombok.EqualsAndHashCode;
        import lombok.NoArgsConstructor;
        import lombok.experimental.Accessors;
        import java.time.LocalDateTime;
        import java.io.Serializable;
        import java.util.Date;
```

```
@EqualsAndHashCode(callSuper = true)
@Data
@AllArgsConstructor
@NoArgsConstructor
@Accessors(chain = true)
@TableName("tbl_user_login")
public class UserLogin extends Model<UserLogin> {
    private static final long serialVersionUID = 1L;
    @TableId(value = "id", type = IdType.AUTO)
    private Integer id;
    private Integer loginUser;
    private Date loginDate;
    private String loginIp;
    private String loginAddress;
    private String userAgent;
    private String browserName;
    private String browserVersion;
    private String osName;
    private String osVersion;
}
package com.jancoyan.jancoblog.controller;
        import com.baomidou.mybatisplus.core.metadata.IPage;
        import com.jancoyan.jancoblog.pojo.*;
        import com.jancoyan.jancoblog.service.ArticleService;
        import com.jancoyan.jancoblog.service.CommentService;
        import com.jancoyan.jancoblog.service.LikeRecordService;
        import com.jancoyan.jancoblog.utils.ArticleUtils;
        import com.jancoyan.jancoblog.utils.Msg;
        import com.jancoyan.jancoblog.utils.RedisUtil;
        import com.jancoyan.jancoblog.utils.TimeUtils;
        import org.springframework.beans.factory.annotation.Autowired;
        import org.springframework.web.bind.annotation.RequestMapping;
        import org.springframework.web.bind.annotation.RequestMethod;
        import org.springframework.web.bind.annotation.RequestParam;
        import org.springframework.web.bind.annotation.RestController;
        import org.springframework.web.multipart.MultipartFile;
        import javax.servlet.http.HttpServletRequest;
        import java.io.File;
        import java.io.IOException;
        import java.io.UnsupportedEncodingException;
        import java.util.Date;
        import java.util.List;
        import java.util.Objects;
        import java.util.UUID;
@RestController
@RequestMapping("/article")
public class ArticleController {
    @Autowired
    ArticleService service;
    @Autowired
```

```
CommentService   commentService;
@Autowired
LikeRecordService  likeRecordService;
@Autowired
RedisUtil  redisUtil;
@RequestMapping(value  =  "/all")
public  Msg  listArticleIndex(
        @RequestParam(value  =  "pn")String  pn,
        @RequestParam(value  =  "limit",  defaultValue  =  "10")String  limit,
        @RequestParam(value  =  "condition",  defaultValue  =  "")String  condition
){
    IPage<Article>  iPage  =  service.listArticleIndex(Integer.parseInt(pn),
            Integer.parseInt(limit),
            condition);
    return  Msg.success().add("pageInfo",  iPage);
}
@RequestMapping(value  =  "/manage")
public  Msg  listArticleManageAll(
        @RequestParam(value  =  "pn")String  pn,
        @RequestParam(value  =  "limit",  defaultValue  =  "10")String  limit,
        @RequestParam(value  =  "condition",  defaultValue  =  "")String  condition,
        HttpServletRequest  request
){
    String  token  =  request.getHeader("token");
    if(null  ==  token){
        return  Msg.expire();
    }
    IPage<Article>  iPage  =  service.listArticleManage(null,
            Integer.parseInt(pn),
            Integer.parseInt(limit),
            condition);
    return  Msg.success().add("pageInfo",  iPage);
}
@RequestMapping(value  =  "/user",  method  =  RequestMethod.GET)
public  Msg  listArticleManageUser(
        @RequestParam(value  =  "pn")String  pn,
        @RequestParam(value  =  "limit",  defaultValue  =  "10")String  limit,
        @RequestParam(value  =  "condition",  defaultValue  =  "")String  condition,
        HttpServletRequest  request){
    String  token  =  request.getHeader("token");
    if(null  ==  token){
        return  Msg.expire();
    }
    User  user  =  (User)  redisUtil.get(token);
    if(null  ==  user){
        return  Msg.fail();
    }
    IPage<Article>  iPage  =  service.listArticleManage(
            user.getUserId(),
            Integer.parseInt(pn),
                Integer.parseInt(limit),
                condition);
        return Msg.success().add("pageInfo", iPage);
    }
    @RequestMapping(value = "/deleted/all", method = RequestMethod.GET)
    public Msg listDeletedAll(
            @RequestParam(value = "pn")String pn,
            @RequestParam(value = "limit", defaultValue = "10")String limit,
            @RequestParam(value = "condition", defaultValue = "")String condition,
            HttpServletRequest request
    ){
        String token = request.getHeader("token");
        if(null == token){
            return Msg.expire();
        }
        IPage<Article> iPage = service.listDeleted(
                null,
                Integer.parseInt(pn),
                Integer.parseInt(limit),
                condition);
        return Msg.success().add("pageInfo", iPage);
    }
    @RequestMapping(value = "/deleted/user", method = RequestMethod.GET)
    public Msg listDeletedUser(
            @RequestParam(value = "pn")String pn,
            @RequestParam(value = "limit", defaultValue = "10")String limit,
            @RequestParam(value = "condition", defaultValue = "")String condition,
            HttpServletRequest request
    ){
        String token = request.getHeader("token");
        if(null == token){
            return Msg.fail();
        }
        User user = (User) redisUtil.get(token);
        if(null == user){
            return Msg.expire();
        }
        IPage<Article> iPage = service.listDeleted(
                user.getUserId(),
                Integer.parseInt(pn),
                Integer.parseInt(limit),
                condition);
        return Msg.success().add("pageInfo", iPage);
    }
    @RequestMapping(value = "/deleted/delete", method = RequestMethod.POST)
    public Msg deleteArticleDeleted(
            String ids,
            HttpServletRequest request
    ){
        String token = request.getHeader("token");
        if(null == token){
            return Msg.fail();
        }
        boolean suc = service.deleteCompletely(ids);
        return Msg.success().add("suc", suc);
    }
    @RequestMapping(value = "/deleted/recover", method = RequestMethod.POST)
    public Msg recoverArticle(
            String ids,
            HttpServletRequest request
    ){
        String token = request.getHeader("token");
        if(null == token){
            return Msg.fail();
        }
        boolean suc = service.batchRecoverDeletedArticle(ids);
        commentService.recoverCommentByArticle(ids);
        return Msg.success().add("suc", suc);
    }
    @RequestMapping(value = "/delete", method = RequestMethod.POST)
    public Msg deleteArticle(
            String ids,
            HttpServletRequest request
    ){
        String token = request.getHeader("token");
        if(null == token){
            return Msg.fail();
        }
        boolean suc = service.batchDeleteArticle(ids);
        commentService.deleteCommentByArticle(ids);
        return Msg.success().add("suc", suc);
    }
    @RequestMapping(value = "/edit", method = RequestMethod.GET)
    public Msg getArticleEdit(
            @RequestParam(value = "id") String id,
            HttpServletRequest request){
        String token = request.getHeader("token");
        if(null == token){
            return Msg.fail();
        }
        Article article = service.getArticleEdit(id);
        return Msg.success().add("article", article);
    }
    @RequestMapping(value = "/single", method = RequestMethod.GET)
    public Msg getArticleSingle(
            @RequestParam(value = "id") String articleId
    ){
        Article article = service.getArticleSingle(articleId);
        return Msg.success().add("article", article);
    }
```

```
@RequestMapping(value = "/single/deleted", method = RequestMethod.GET)
public Msg getArticleSingleDeleted(
        @RequestParam(value = "id") String articleId
){
    Article article = service.getArticleSingleDeleted(articleId);
    return Msg.success().add("article", article);
}
@RequestMapping(value = "/recent", method = RequestMethod.GET)
public Msg listArticleUserRecently(
        @RequestParam(value = "id") String id,
        @RequestParam(value = "pn", defaultValue = "1")Integer pn,
        @RequestParam(value = "limit" ,defaultValue = "10")Integer limit
){
    IPage<PageArticle> iPage = service.listArticleUserRecently(id, pn, limit);
    return Msg.success().add("pageInfo", iPage);
}
@RequestMapping(value = "/like", method = RequestMethod.POST)
public Msg addLikeCount(
        @RequestParam(value = "id")String id,
        HttpServletRequest request
){
    String token = request.getHeader("token");
    if(null == token){
        return Msg.loginNeeded();
    }
    User user = (User) redisUtil.get(token);
    if(null == user){
        return Msg.expire();
    }
    service.addLikeCount(id);
    likeRecordService.insertRecord(user.getUserId(), id);
    return Msg.success();
}
@RequestMapping(value = "/dislike", method = RequestMethod.POST)
public Msg subLikeCount(
        @RequestParam(value = "id")String id,
        HttpServletRequest request
){
    String token = request.getHeader("token");
    if(null == token){
        return Msg.loginNeeded();
    }
    User user = (User) redisUtil.get(token);
    service.subLikeCount(id);
    likeRecordService.deleteRecord(user.getUserId(), id);
    return Msg.success();
}
@RequestMapping(value = "/view", method = RequestMethod.GET)
public Msg addViewCount(@RequestParam(value = "id")String id){
    service.addViewCount(id);
```

```
        return Msg.success();
    }
    @RequestMapping(value = "/toggle/comment", method = RequestMethod.POST)
    public Msg updateIsComment(
            @RequestParam(value = "id")String id,
            HttpServletRequest request
    ){
        String token = request.getHeader("token");
        if(null == token){
            return Msg.fail();
        }
        boolean suc = service.updateIsComment(id);
        if(suc) {
            return Msg.success();
        } else {
            return Msg.fail();
        }
    }
    @RequestMapping(value = "/toggle/top", method = RequestMethod.POST)
    public Msg updateArticleTop(
            @RequestParam(value = "id")String id,
            HttpServletRequest request
    ){
        String token = request.getHeader("token");
        if(null == token){
            return Msg.fail();
        }
        boolean suc = service.updateIsTop(id);
        if(suc) {
            return Msg.success();
        } else {
            return Msg.fail();
        }
    }
    @RequestMapping(value = "/picture", method = RequestMethod.POST)
    public Msg uploadPicture(
            @RequestParam(value = "file") MultipartFile file,
            HttpServletRequest request
    ) throws IOException {
        if (file == null) {
            return Msg.fail().add("msg", "请选择要上传的图片");
        }
        if (file.getSize() > 1024 * 1024 * 10) {
            return Msg.fail().add("msg", "文件大小不能大于 10M");
        }
        String suffix = Objects.requireNonNull(file.getOriginalFilename()).substring(file.getOriginalFile
name().lastIndexOf(".") + 1);
        if (!"jpg,jpeg,gif,png".toUpperCase().contains(suffix.toUpperCase())) {
            return Msg.fail().add("msg", "请选择 jpg,jpeg,gif,png 格式的图片");
        }
        String nowMonth = TimeUtils.getCurrentTimeString().substring(0, 7);
        String savePath =
                new File(".").getCanonicalPath() + "\\target\\classes\\static\\p\\" +
                        nowMonth + "\\";
        File savePathFile = new File(savePath);
        if (!savePathFile.exists()) {
            savePathFile.mkdir();
        }
        String filename = UUID.randomUUID().toString().replaceAll("-","") + "." + suffix;
        try {
            file.transferTo(new File(savePath + filename));
        } catch (Exception e) {
            e.printStackTrace();
            return Msg.fail().add("msg", "保存文件异常");
        }
        String url = "http:
        return Msg.success().add("url", url);
    }
    @RequestMapping(value = "/post", method = RequestMethod.POST)
    public Msg insertArticle(
            @RequestParam(value = "title") String title,
            @RequestParam(value = "type") String type,
            @RequestParam(value = "summary") String summary,
            @RequestParam(value = "comment") String comment,
            @RequestParam(value = "md") String md,
            @RequestParam(value = "html") String html,
            HttpServletRequest request
    ) throws UnsupportedEncodingException {
        request.setCharacterEncoding("utf-8");
        String token = request.getHeader("token");
        User user;
        if(null == token){
            return Msg.fail();
        }else{
            user = (User) redisUtil.get(token);
            if(null == user) {
                return Msg.expire();
            }
        }
        Article article = new Article();
        article.setArticleTitle(title)
                .setArticleAuthor(user.getUserId())
                .setArticleType(Integer.parseInt(type))
                .setArticleHtml(ArticleUtils.simplifyImages(html))
                .setArticleMd(ArticleUtils.replaceSingleSlash(md))
                .setArticleIsComment("true".equals(comment) ? 1 : 0)
                .setArticleRank(0);
        long now = System.currentTimeMillis();
        article.setArticleId(ArticleUtils.getArticleId(user.getUserId(), now))
                .setArticlePostTime(new Date(now))
                .setArticleEditTime(new Date(now));
        if(!"".equals(summary)){
            article.setArticleSummary(summary);
        }else{
            article.setArticleSummary(ArticleUtils.getArticleDefaultSummary(html));
        }
        List<String> images = ArticleUtils.getPicturesInArticle(html);
        ArticleImage articleImage = new ArticleImage();
        for (String image : images) {
            articleImage.setInsertDate(new Date(now));
            articleImage.setArticleId(article.getArticleId());
            articleImage.setFilename(image);
            articleImage.insert();
        }
        boolean suc = article.insert();
        return Msg.success().add("suc", suc).add("id", article.getArticleId());
    }
    @RequestMapping(value = "/update", method = RequestMethod.POST)
    public Msg updateArticle(
            @RequestParam(value = "id") String id,
            @RequestParam(value = "title") String title,
            @RequestParam(value = "type") String type,
            @RequestParam(value = "summary") String summary,
            @RequestParam(value = "comment") String comment,
            @RequestParam(value = "md") String md,
            @RequestParam(value = "html") String html,
            HttpServletRequest request
    ) throws UnsupportedEncodingException {
        request.setCharacterEncoding("utf-8");
        String token = request.getHeader("token");
        User user;
        if(null == token){
            return Msg.fail();
        }else{
            user = (User) redisUtil.get(token);
            if(null == user) {
                return Msg.expire();
            }
        }
        Article article = new Article();
        article.setArticleId(id)
                .setArticleTitle(title)
                .setArticleAuthor(user.getUserId())
                .setArticleType(Integer.parseInt(type))
                .setArticleHtml(ArticleUtils.simplifyImages(html))
                .setArticleMd(ArticleUtils.replaceSingleSlash(md))
                .setArticleIsComment("true".equals(comment) ? 1 : 0)
                .setArticleRank(0);
        long now = System.currentTimeMillis();
        article.setArticleEditTime(new Date(now));
        if(!"".equals(summary)){
            article.setArticleSummary(summary);
        }else{
            article.setArticleSummary(ArticleUtils.getArticleDefaultSummary(html));
        }
        List<String> images = ArticleUtils.getPicturesInArticle(html);
        ArticleImage articleImage = new ArticleImage();
        for (String image : images) {
            articleImage.setInsertDate(new Date(now));
            articleImage.setArticleId(article.getArticleId());
            articleImage.setFilename(image);
            articleImage.insert();
        }
        boolean suc = article.updateById();
        return Msg.success().add("suc", suc).add("id", article.getArticleId());
    }
}
package com.jancoyan.jancoblog.controller;
        import org.springframework.web.bind.annotation.RequestMapping;
        import org.springframework.stereotype.Controller;
@Controller
@RequestMapping("/article_image")
public class ArticleImageController {
}
package com.jancoyan.jancoblog.controller;
        import com.baomidou.mybatisplus.core.metadata.IPage;
        import com.jancoyan.jancoblog.pojo.Article;
        import com.jancoyan.jancoblog.pojo.Comment;
        import com.jancoyan.jancoblog.pojo.PageComment;
        import com.jancoyan.jancoblog.pojo.User;
        import com.jancoyan.jancoblog.service.CommentService;
        import com.jancoyan.jancoblog.utils.Msg;
        import com.jancoyan.jancoblog.utils.RedisUtil;
        import org.springframework.beans.factory.annotation.Autowired;
        import org.springframework.web.bind.annotation.RequestMapping;
        import org.springframework.web.bind.annotation.RequestMethod;
        import org.springframework.web.bind.annotation.RequestParam;
        import org.springframework.web.bind.annotation.RestController;
        import javax.servlet.http.HttpServletRequest;
        import java.util.Date;
@RestController
@RequestMapping("/comment")
public class CommentController {
    @Autowired
    CommentService service;
    @Autowired
    RedisUtil redisUtil;
    @RequestMapping(value = "/all")
    public Msg listAll(
            @RequestParam(value = "pn")String pn,
            @RequestParam(value = "limit", defaultValue = "10")String limit,
            @RequestParam(value = "condition", defaultValue = "")String condition
    ){
        IPage<Comment> iPage = service.listAll(null, Integer.parseInt(pn),
                Integer.parseInt(limit),
                condition);
        return Msg.success().add("pageInfo", iPage);
    }
    @RequestMapping(value = "/receive")
    public Msg listCommentByUserReceive(
            @RequestParam(value = "pn")String pn,
            @RequestParam(value = "limit", defaultValue = "10")String limit,
            @RequestParam(value = "condition", defaultValue = "")String condition,
            HttpServletRequest request
    ){
        String token = request.getHeader("token");
        if(null == token){
            return Msg.expire();
        }
        User user = (User) redisUtil.get(token);
        IPage<Comment> iPage = service.listAll(
                String.valueOf(user.getUserId()),
                Integer.parseInt(pn),
                Integer.parseInt(limit),
                condition);
        return Msg.success().add("pageInfo", iPage);
    }
    @RequestMapping(value = "/posted")
    public Msg listCommentByUserPosted(
            @RequestParam(value = "pn")String pn,
            @RequestParam(value = "limit", defaultValue = "10")String limit,
            @RequestParam(value = "condition", defaultValue = "")String condition,
            HttpServletRequest request
    ){
        String token = request.getHeader("token");
        if(null == token){
            return Msg.expire();
        }
        User user = (User) redisUtil.get(token);
        if(null == user){
            return Msg.fail().add("msg", "获取失败");
        }
        IPage<Comment> iPage = service.listCommentByUserPosted(
                String.valueOf(user.getUserId()),
                Integer.parseInt(pn),
                Integer.parseInt(limit),
                condition);
        return Msg.success().add("pageInfo", iPage);
    }
    @RequestMapping(value = "/delete", method = RequestMethod.POST)
```

```
public Msg deleteComment(
        @RequestParam(value = "ids") String ids
){
    boolean suc = service.batchDeleteComment(ids);
    return Msg.success().add("suc", suc ? "success" : "fail");
}
@RequestMapping(value = "/article")
public Msg listCommentByArticle(
        @RequestParam(value = "id") String id,
        @RequestParam(value = "pn")Integer pn,
        @RequestParam(value = "limit", defaultValue = "7") Integer limit
){
    IPage<Comment> iPage = service.listCommentByArticle(id, pn, limit);
    return Msg.success().add("pageInfo", iPage);
}
@RequestMapping(value = "/post", method = RequestMethod.POST)
public Msg insertComment(
        @RequestParam(value = "articleId")String articleId,
        @RequestParam(value = "name", defaultValue = "") String userName,
        @RequestParam(value = "email", defaultValue = "")String email,
        @RequestParam(value = "content")String content,
        HttpServletRequest request
){
    Comment comment = new Comment();
    String token = request.getHeader("token");
    User user = (User) redisUtil.get(token);
    if(null == user && null != token) {
        return Msg.expire();
    }
    if(null == token || "".equals(token)){
        comment.setCommentAuthorEmail(email);
        comment.setCommentAuthorName(userName);
    } else {
        comment.setCommentAuthorName(user.getUserName());
        comment.setCommentAuthorId(user.getUserId());
    }
    comment.setCommentArticleId(articleId);
    comment.setCommentDate(new Date());
    comment.setCommentContent(content);
    comment.setCommentAuthorIp(request.getScheme());
    Article article = new Article();
    article.setArticleId(articleId);
    article = article.selectById();
    article.setArticleCommentCount(article.getArticleCommentCount() + 1);
    article.updateById();
    comment.insert();
    return Msg.success();
}
@RequestMapping(value = "/like", method = RequestMethod.POST)
public Msg likeComment(
                @RequestParam(value = "id") Integer id
        ){
            service.likeComment(id);
            return Msg.success();
        }
        @RequestMapping(value = "/recent", method = RequestMethod.GET)
        public Msg listCommentByUserRecentlyReceive(
                @RequestParam(value = "id") String authorId,
                @RequestParam(value = "pn", defaultValue = "1") String pn,
                @RequestParam(value = "limit", defaultValue = "10") String limit
        ){
            IPage<PageComment> iPage  = service.listCommentByUserRecently(authorId);
            return Msg.success().add("pageInfo", iPage);
        }
}
package com.jancoyan.jancoblog.controller;
            import com.baomidou.mybatisplus.core.metadata.IPage;
            import com.jancoyan.jancoblog.pojo.Comment;
            import com.jancoyan.jancoblog.pojo.DeletedComment;
            import com.jancoyan.jancoblog.service.DeletedCommentService;
            import com.jancoyan.jancoblog.utils.Msg;
            import org.springframework.beans.factory.annotation.Autowired;
            import org.springframework.web.bind.annotation.RequestMapping;
            import org.springframework.stereotype.Controller;
            import org.springframework.web.bind.annotation.RequestMethod;
            import org.springframework.web.bind.annotation.RequestParam;
            import org.springframework.web.bind.annotation.RestController;
@RestController
@RequestMapping("/deleted/comment")
public class DeletedCommentController {
        @Autowired
        DeletedCommentService service;
        @RequestMapping(value = "/article", method = RequestMethod.GET)
        public Msg getCommentByArticle(
                @RequestParam(value = "id") String id,
                @RequestParam(value = "pn", defaultValue = "1")Integer pn,
                @RequestParam(value = "limit", defaultValue = "7") Integer limit
        ){
            IPage<DeletedComment> iPage = service.getCommentByArticle(id, pn, limit);
            return Msg.success().add("pageInfo", iPage);
        }
}
package com.jancoyan.jancoblog.controller;
            import com.baomidou.mybatisplus.core.conditions.query.QueryWrapper;
            import com.baomidou.mybatisplus.core.metadata.IPage;
            import com.baomidou.mybatisplus.extension.plugins.pagination.Page;
            import com.jancoyan.jancoblog.pojo.User;
            import com.jancoyan.jancoblog.pojo.UserLogin;
            import com.jancoyan.jancoblog.service.UserLoginService;
            import com.jancoyan.jancoblog.utils.Msg;
            import com.jancoyan.jancoblog.utils.RedisUtil;
            import org.springframework.beans.factory.annotation.Autowired;
            import org.springframework.web.bind.annotation.RequestMapping;
            import org.springframework.stereotype.Controller;
            import org.springframework.web.bind.annotation.RequestMethod;
            import org.springframework.web.bind.annotation.RequestParam;
            import org.springframework.web.bind.annotation.RestController;
            import javax.servlet.http.HttpServletRequest;
            import java.util.List;
@RestController
@RequestMapping("/login")
public class UserLoginController {
        @Autowired
        UserLoginService service;
        @Autowired
        RedisUtil redisUtil;
        @RequestMapping(value = "/all", method = RequestMethod.GET)
        public Msg getAll(
                @RequestParam(value = "id", defaultValue = "") String userId,
                HttpServletRequest request
        ){
            String token = request.getHeader("token");
            if(null == token){
                return Msg.fail().add("msg", "用户验证失败");
            }
            User user = (User) redisUtil.get(token);
            if(null == user){
                return Msg.expire();
            }
            UserLogin userLogin = new UserLogin();
            QueryWrapper<UserLogin> wrapper = new QueryWrapper<>();
            wrapper.eq("login_user", user.getUserId());
            wrapper.orderByDesc("login_date");
            IPage<UserLogin> iPage = new Page<>(1, 10);
            iPage =  userLogin.selectPage(iPage, wrapper);
            return Msg.success().add("pageInfo", iPage);
        }
}
package com.jancoyan.jancoblog.utils;
            import com.baomidou.mybatisplus.core.conditions.query.QueryWrapper;
            import com.jancoyan.jancoblog.pojo.Article;
            import java.util.ArrayList;
            import java.util.List;
            import java.util.regex.Matcher;
            import java.util.regex.Pattern;
public class ArticleUtils {
        public static QueryWrapper<Article> generateManageArticleWrapperByCondition(QueryWrapper<A
rticle> wrapper,
                                                                                    String c
ondition){
            String[] split = condition.split("--");
            for (String item : split) {
                String[] split2 = item.split("=");
                if(split2.length < 2){
                    continue;
                }
                if("article_author_name".equals(split2[0])){
                    wrapper.like("user_name", split2[1]);
                }else if("article_title".equals(split2[0])){
                    wrapper.like("article_title", split2[1]);
                }else if("type".equals(split2[0])){
                    wrapper.eq("article_type", split2[1]);
                }else if("start".equals(split2[0])){
                    wrapper.gt("article_post_time", split2[1]);
                }else if("end".equals(split2[0])){
                    wrapper.lt("article_post_time", split2[1]);
                }else if("rank_view".equals(split2[0])){
                    if ("1".equals(split2[1])) {
                        wrapper.orderByAsc("article_view_count");
                    } else {
                        wrapper.orderByDesc("article_view_count");
                    }
                }else if("rank_like".equals(split2[0])){
                    if ("1".equals(split2[1])) {
                        wrapper.orderByAsc("article_like_count");
                    } else {
                        wrapper.orderByDesc("article_like_count");
                    }
                }else if("rank_comment".equals(split2[0])){
                    if ("1".equals(split2[1])) {
                        wrapper.orderByAsc("article_comment_count");
                    } else {
                        wrapper.orderByDesc("article_comment_count");
                    }
                }
            }
            return wrapper;
        }
        public static String nextLineToText(String str){
            return str.replaceAll("\n", "\\\\n");
        }
        public static String replaceSingleSlash(String str){
            return str.replaceAll("'", "\\\\'");
        }
        public static String getArticleDefaultSummary(String innerHTML){
            String rst = innerHTML.replaceAll("<.*?>", "");
            rst = rst.replaceAll("\\s", " ");
            if(rst.length() >= 100){
                rst = rst.substring(0, 100) + "......";
            }
            return rst;
        }
        public static String getArticleId(Integer userId, long now){
            String str = String.valueOf(userId);
            str += String.valueOf(now);
            return str;
        }
        public static String simplifyImages(String html){
            return html.replaceAll("<img src", "<img style=\"max-width:70%\" src");
        }
        public static List<String> getPicturesInArticle(String html){
            String regex = "src=\".*?\"";
            Pattern p = Pattern.compile(regex);
            Matcher matcher = p.matcher(html);
            List<String> fileNames = new ArrayList<>(16);
            while (matcher.find()){
                String fileName = matcher.group();
                fileName = fileName.substring(fileName.lastIndexOf('/') + 1, fileName.length() - 1);
                fileNames.add(fileName);
                System.out.println(fileName);
            }
            return fileNames;
        }
    }
    package com.jancoyan.jancoblog.utils;
            import java.io.File;
            import java.io.IOException;
            import java.text.SimpleDateFormat;
            import java.util.Date;
    public class FileUtils {
        public static void deleteImageIfExists(Date date, String imageName)  {
            SimpleDateFormat simpleDateFormat = new SimpleDateFormat("yyyy-MM");
            String format = simpleDateFormat.format(date);
            String path = null;
            try {
                path = new File(".").getCanonicalPath() + "\\target\\classes\\static\\p\\" +
                        format + "\\" + imageName;
                File file = new File(path);
                if(file.exists()){
                    file.delete();
                }
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
    package com.jancoyan.jancoblog.utils;
            import com.auth0.jwt.JWT;
            import com.auth0.jwt.JWTVerifier;
            import com.auth0.jwt.algorithms.Algorithm;
            import java.util.Calendar;
            import java.util.HashMap;
            import java.util.Map;
    public class JsonWebTokenUtils {
        public static final int calendarField = Calendar.DATE;
        public static final int calendarInterval = 1;
        public static String createToken(Long user_id){
            Date iatDate = new Date();
            Calendar nowTime = Calendar.getInstance();
            nowTime.add(calendarField, calendarInterval);
            Date expiresDate = nowTime.getTime();
            Map<String, Object> map = new HashMap<>();
            map.put("alg", "HS256");
            map.put("typ", "JWT");
            String token = JWT.create().withHeader(map)
                    .withClaim("iss", "Service")
                    .withClaim("aud", "APP")
                    .withClaim("user_id", null == user_id ? null : user_id.toString())
                    .withIssuedAt(iatDate)
                    .withExpiresAt(expiresDate)
                    .sign(Algorithm.HMAC256(SECRET));
            return token;
        }
        public static Map<String, Claim> verifyToken(String token) {
            DecodedJWT jwt = null;
            try {
                JWTVerifier verifier = JWT.require(Algorithm.HMAC256(SECRET)).build();
                jwt = verifier.verify(token);
            } catch (Exception e) {
                e.printStackTrace();
                System.out.println("Token 非法");
            }
            return jwt.getClaims();
        }
        public static Long getAppUID(String token) {
            Map<String, Claim> claims = verifyToken(token);
            Claim user_id_claim = claims.get("user_id");
            if (null == user_id_claim || "".equals(user_id_claim.asString())) {
                System.out.println("token 验证非法");
            }
            return Long.valueOf(user_id_claim.asString());
        }
    }
    package com.jancoyan.jancoblog.utils;
            import java.security.MessageDigest;
            import java.security.NoSuchAlgorithmException;
    public class MD5Util {
        public static String getMD5(String password) {
            try {
                MessageDigest digest = MessageDigest.getInstance("md5");
                byte[] result = digest.digest(password.getBytes());
                StringBuffer buffer = new StringBuffer();
                for (byte b : result) {
                    int number = b & 0xff;
                    String str = Integer.toHexString(number);
                    if (str.length() == 1) {
                        buffer.append("0");
                    }
                    buffer.append(str);
                }
                return buffer.toString();
            } catch (NoSuchAlgorithmException e) {
                e.printStackTrace();
                return "";
            }
        }
    }
    package com.jancoyan.jancoblog.utils;
            import lombok.Data;
            import java.util.HashMap;
            import java.util.Map;
    @Data
    public class Msg {
        private int code;
        private String msg;
        private Map<String, Object> extend = new HashMap<String, Object>();
        public static Msg expire(){
            Msg result = new Msg();
            result.setCode(601);
            result.setMsg("用户信息已过期");
            return result;
        }
        public static Msg illegalToken(){
            Msg result = new Msg();
            result.setCode(602);
            result.setMsg("登录信息无效");
            return result;
        }
        public static Msg success(){
            Msg result = new Msg();
            result.setCode(100);
            result.setMsg("成功");
            return result;
        }
        public static Msg loginNeeded(){
            Msg result = new Msg();
            result.setCode(603);
            result.setMsg("用户未登录");
            return result;
        }
        public static Msg fail(){
            Msg result = new Msg();
            result.setCode(200);
            result.setMsg("失败！");
            return result;
        }
        public Msg add(String key,Object value){
            this.getExtend().put(key, value);
            return this;
        }
    }
    package com.jancoyan.jancoblog.utils;
            import java.text.ParseException;
            import java.text.SimpleDateFormat;
            import java.util.Date;
    public class TimeUtils {
        public static String getCurrentTimeString(){
            SimpleDateFormat df = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
            return df.format(new Date());
        }
        public static String castDateTypeToDateString(Date date){
            SimpleDateFormat simpleDateFormat = new SimpleDateFormat("yyyy-MM-dd");
            return simpleDateFormat.format(date);
        }
        public static Date castDateStringToDateTypeYMD(String dateStr){
            Date date = null;
            SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd");
            try {
                date = sdf.parse(dateStr);
            } catch (ParseException e) {
                e.printStackTrace();
            }
            return date;
        }
    }
    package com.jancoyan.jancoblog.config;
            import org.springframework.web.servlet.HandlerInterceptor;
            import org.springframework.web.servlet.ModelAndView;
            import javax.servlet.http.HttpServletRequest;
            import javax.servlet.http.HttpServletResponse;
    public class AllowOriginIntercepter implements HandlerInterceptor {
        @Override
        public boolean preHandle(HttpServletRequest request, HttpServletResponse response, Object handl
    er) {
            response.setHeader("Access-Control-Allow-Origin", request.getHeader("Origin"));
            response.setHeader("Access-Control-Allow-Credentials", "true");
            response.setHeader("Access-Control-Allow-Methods","GET, POST, OPTIONS, PUT");
            response.setHeader("Access-Control-Allow-Headers","Authorization, Content-Type, token");
            return true;
        }
        @Override
        public void postHandle(HttpServletRequest request, HttpServletResponse response, Object handler,
                                       ModelAndView modelAndView) throws Exception {
        }
        @Override
        public void afterCompletion(HttpServletRequest request, HttpServletResponse response, Object ha
    ndler, Exception ex)
                throws Exception {
        }
    }
    package com.jancoyan.jancoblog.config;
            import org.springframework.context.annotation.Configuration;
            import org.springframework.web.servlet.config.annotation.InterceptorRegistry;
            import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;
    @Configuration
    public class MvcConfig implements WebMvcConfigurer {
        @Override
        public void addInterceptors(InterceptorRegistry registry) {
            registry.addInterceptor(new AllowOriginIntercepter());
        }
    }
    package com.jancoyan.jancoblog.config;
            import com.baomidou.mybatisplus.annotation.DbType;
            import com.baomidou.mybatisplus.extension.plugins.MybatisPlusInterceptor;
            import com.baomidou.mybatisplus.extension.plugins.inner.PaginationInnerInterceptor;
            import org.mybatis.spring.annotation.MapperScan;
            import org.springframework.context.annotation.Bean;
            import org.springframework.context.annotation.Configuration;
            import org.springframework.transaction.annotation.EnableTransactionManagement;
    @EnableTransactionManagement
    @Configuration
    @MapperScan("com.jancoyan.jancoblog.mapper")
    public class MybatisPlusConfig {
        @Bean
        public MybatisPlusInterceptor mybatisPlusInterceptor() {
            MybatisPlusInterceptor interceptor = new MybatisPlusInterceptor();
            interceptor.addInnerInterceptor(new PaginationInnerInterceptor(DbType.MYSQL));
            return interceptor;
        }
    }
    package com.jancoyan.jancoblog.config;
            import com.fasterxml.jackson.annotation.JsonAutoDetect;
            import com.fasterxml.jackson.annotation.PropertyAccessor;
            import com.fasterxml.jackson.databind.ObjectMapper;
            import org.springframework.context.annotation.Bean;
            import org.springframework.context.annotation.Configuration;
            import org.springframework.data.redis.connection.RedisConnectionFactory;
            import org.springframework.data.redis.core.RedisTemplate;
            import org.springframework.data.redis.serializer.Jackson2JsonRedisSerializer;
            import org.springframework.data.redis.serializer.StringRedisSerializer;
```

```
@Configuration
public class RedisConfig {
    @Bean
    @SuppressWarnings("all")
    public RedisTemplate<String, Object> redisTemplate(
            RedisConnectionFactory factory) {
        RedisTemplate<String, Object> template =
                new RedisTemplate<String, Object>();
        template.setConnectionFactory(factory);
        Jackson2JsonRedisSerializer jackson2JsonRedisSerializer =
                new Jackson2JsonRedisSerializer(Object.class);
        ObjectMapper om = new ObjectMapper();
        om.setVisibility(PropertyAccessor.ALL, JsonAutoDetect.Visibility.ANY);
        om.enableDefaultTyping(ObjectMapper.DefaultTyping.NON_FINAL);
        jackson2JsonRedisSerializer.setObjectMapper(om);
        StringRedisSerializer stringRedisSerializer =
                new StringRedisSerializer();
        template.setKeySerializer(stringRedisSerializer);
        template.setHashKeySerializer(stringRedisSerializer);
        template.setValueSerializer(jackson2JsonRedisSerializer);
        template.setHashValueSerializer(jackson2JsonRedisSerializer);
        template.afterPropertiesSet();
        return template;
    }
}
package com.jancoyan.jancoblog.service;
        import com.baomidou.mybatisplus.core.metadata.IPage;
        import com.jancoyan.jancoblog.pojo.Article;
        import com.baomidou.mybatisplus.extension.service.IService;
        import com.jancoyan.jancoblog.pojo.PageArticle;
public interface ArticleService extends IService<Article> {
    IPage<Article> listArticleIndex(Integer pn, Integer limit, String condition);
    IPage<Article> listArticleManage(Integer userId, Integer pn, Integer limit, String condition);
    Article getArticleSingle(String articleId);
    Article getArticleSingleDeleted(String articleId);
    IPage<Article> listDeleted(Integer userId, Integer pn, Integer limit, String condition);
    boolean deleteCompletely(String ids);
    boolean batchRecoverDeletedArticle(String ids);
    boolean batchDeleteArticle(String ids);
    IPage<PageArticle> listArticleUserRecently(String id, Integer pn, Integer limit);
    void addLikeCount(String id);
    void subLikeCount(String id);
    void addViewCount(String id);
    boolean updateIsComment(String id);
    boolean updateIsTop(String id);
    Article getArticleEdit(String id);
}
package com.jancoyan.jancoblog.service;
        import com.baomidou.mybatisplus.core.metadata.IPage;
        import com.jancoyan.jancoblog.pojo.Comment;
        import com.baomidou.mybatisplus.extension.service.IService;
        import com.jancoyan.jancoblog.pojo.PageComment;
public interface CommentService extends IService<Comment> {
    IPage<Comment> listAll(String userId, Integer pn, Integer limit, String condition);
    IPage<Comment> listCommentByArticle(String id, Integer pn, Integer limit);
    IPage<Comment> listCommentByUserPosted(String id, Integer pn, Integer limit,
                                        String condition);
    IPage<PageComment> listCommentByUserRecently(String authorId);
    void deleteCommentByArticle(String articleId);
    void recoverCommentByArticle(String ids);
    boolean batchDeleteComment(String ids);
    void likeComment(Integer id);
}
package com.jancoyan.jancoblog.service;
        import com.baomidou.mybatisplus.core.metadata.IPage;
        import com.jancoyan.jancoblog.pojo.LikeRecord;
        import com.baomidou.mybatisplus.extension.service.IService;
public interface LikeRecordService extends IService<LikeRecord> {
    IPage<LikeRecord> getUserReceive(String userId, Integer pn, Integer limit);
    void insertRecord(Integer userId, String articleId);
    void deleteRecord(Integer userId, String articleId);
}
package com.jancoyan.jancoblog.service;
        import com.baomidou.mybatisplus.core.metadata.IPage;
        import com.jancoyan.jancoblog.pojo.User;
        import com.baomidou.mybatisplus.extension.service.IService;
        import com.jancoyan.jancoblog.pojo.UserInfo;
        import com.jancoyan.jancoblog.pojo.VUserTotalData;
        import java.util.List;
public interface UserService extends IService<User> {
    IPage<User> getAll(Integer pn, Integer limit, String condition);
    VUserTotalData getUserTotalData(String userId);
    UserInfo getUserInfo(Integer userId);
    User login(String username, String password);
}
package com.jancoyan.jancoblog.service.impl;
        import com.baomidou.mybatisplus.core.conditions.query.QueryWrapper;
        import com.baomidou.mybatisplus.core.metadata.IPage;
        import com.baomidou.mybatisplus.extension.plugins.pagination.Page;
        import com.jancoyan.jancoblog.pojo.Article;
        import com.jancoyan.jancoblog.mapper.ArticleMapper;
        import com.jancoyan.jancoblog.pojo.ArticleImage;
        import com.jancoyan.jancoblog.pojo.DeletedComment;
        import com.jancoyan.jancoblog.pojo.PageArticle;
        import com.jancoyan.jancoblog.service.ArticleService;
        import com.baomidou.mybatisplus.extension.service.impl.ServiceImpl;
        import com.jancoyan.jancoblog.utils.ArticleUtils;
        import com.jancoyan.jancoblog.utils.FileUtils;
        import org.springframework.stereotype.Service;
        import java.util.List;
```

```
@Service
public class ArticleServiceImpl extends ServiceImpl<ArticleMapper, Article> implements ArticleServic
e {
    @Override
    public IPage<Article> listArticleIndex(Integer pn, Integer limit, String condition) {
        IPage<Article> iPage = new Page<>(pn, limit);
        QueryWrapper<Article> wrapper = new QueryWrapper<>();
        wrapper.orderByDesc("article_rank");
        wrapper = ArticleUtils.generateManageArticleWrapperByCondition(wrapper, condition);
        wrapper.orderByDesc("article_post_time");
        return baseMapper.getIndexList(iPage, wrapper);
    }
    @Override
    public IPage<Article> listArticleManage(Integer userId,
                                            Integer pn,
                                            Integer limit,
                                            String condition) {
        IPage<Article> iPage = new Page<>(pn, limit);
        QueryWrapper<Article> wrapper = new QueryWrapper<>();
        if(null != userId) {
            wrapper.eq("user_id", userId);
        }
        wrapper = ArticleUtils.generateManageArticleWrapperByCondition(wrapper, condition);
        wrapper.orderByDesc("article_post_time");
        return baseMapper.getManageList(iPage, wrapper);
    }
    @Override
    public Article getArticleSingle(String articleId) {
        return baseMapper.getSingleArticle(articleId);
    }
    @Override
    public Article getArticleSingleDeleted(String articleId) {
        return baseMapper.getSingleArticleDeleted(articleId);
    }
    @Override
    public IPage<Article> listDeleted(Integer userId,
                                      Integer pn,
                                      Integer limit,
                                      String condition) {
        IPage<Article> iPage = new Page<>(pn, limit);
        QueryWrapper<Article> wrapper = new QueryWrapper<>();
        if(null != userId) {
            wrapper.eq("user_id", userId);
        }
        wrapper = ArticleUtils.generateManageArticleWrapperByCondition(wrapper, condition);
        wrapper.orderByDesc("article_post_time");
        return baseMapper.getDeletedList(iPage, wrapper);
    }
    @Override
    public boolean deleteCompletely(String ids) {
        QueryWrapper<ArticleImage> wrapper = new QueryWrapper<>();
        ArticleImage articleImage = new ArticleImage();
        QueryWrapper<DeletedComment> deletedCommentQueryWrapper = new QueryWrapper<>();
        DeletedComment deletedComment = new DeletedComment();
        if(!ids.contains("&")){
            wrapper.eq("article_id", ids);
            deletedCommentQueryWrapper.eq("comment_article_id", ids);
            List<ArticleImage> articleImages = articleImage.selectList(wrapper);
            for (ArticleImage image : articleImages) {
                FileUtils.deleteImageIfExists(image.getInsertDate(), image.getFilename());
            }
            articleImage.delete(wrapper);
            deletedComment.delete(deletedCommentQueryWrapper);
            baseMapper.deleteCompletely(ids);
        } else {
            String[] id = ids.split("&");
            for (String item : id) {
                wrapper.eq("article_id", item);
                deletedCommentQueryWrapper.eq("comment_article_id", item);
                List<ArticleImage> articleImages = articleImage.selectList(wrapper);
                for (ArticleImage image : articleImages) {
                    FileUtils.deleteImageIfExists(image.getInsertDate(), image.getFilename());
                }
                articleImage.delete(wrapper);
                deletedComment.delete(deletedCommentQueryWrapper);
                baseMapper.deleteCompletely(item);
            }
        }
        return true;
    }
    @Override
    public boolean batchRecoverDeletedArticle(String ids) {
        if(!ids.contains("&")){
            baseMapper.batchRecover(ids);
        } else {
            String[] id = ids.split("&");
            for (String item : id) {
                baseMapper.batchRecover(item);
            }
        }
        return true;
    }
    @Override
    public boolean batchDeleteArticle(String ids) {
        Article article = new Article();
        if(!ids.contains("&")){
            article.setArticleId(ids);
            article.deleteById();
        } else {
            String[] id = ids.split("&");
            for (String item : id) {
                article.setArticleId(item);
                article.deleteById();
            }
        }
        return true;
    }
    @Override
    public IPage<PageArticle> listArticleUserRecently(String id, Integer pn, Integer limit) {
        IPage<PageArticle> iPage = new Page<>(pn, limit);
        QueryWrapper<PageArticle> wrapper = new QueryWrapper<>();
        wrapper.eq("article_author", id);
        wrapper.orderByDesc("article_post_time");
        return baseMapper.getArticleByUserRecently(iPage, wrapper);
    }
    @Override
    public void addLikeCount(String id) {
        Article article = new Article();
        article.setArticleId(id);
        article = article.selectById();
        article.setArticleLikeCount(article.getArticleLikeCount() + 1);
        article.updateById();
    }
    @Override
    public void subLikeCount(String id) {
        Article article = new Article();
        article.setArticleId(id);
        article = article.selectById();
        article.setArticleLikeCount(article.getArticleLikeCount() - 1);
        article.updateById();
    }
    @Override
    public void addViewCount(String id) {
        Article article = new Article();
        article.setArticleId(id);
        article = article.selectById();
        article.setArticleViewCount(article.getArticleViewCount() + 1);
        article.updateById();
    }
    @Override
    public boolean updateIsComment(String id) {
        Article article = new Article();
        article.setArticleId(id);
        article = article.selectById();
        article.setArticleIsComment(1 - article.getArticleIsComment());
        return article.updateById();
    }
    @Override
    public boolean updateIsTop(String id) {
        Article article = new Article();
        article.setArticleId(id);
        article  =  article.selectById();
        article.setArticleRank(1  -  article.getArticleRank());
        return  article.updateById();
    }
    @Override
    public  Article  getArticleEdit(String  id)  {
        return  baseMapper.getArticleEdit(id);
    }
}
<template>
  <div  class="cropper-content">
    <div  class="cropper-box">
      <div  class="cropper">
        <vue-cropper
          ref="cropper"
          :img="option.img"
          :outputSize="option.outputSize"
          :outputType="option.outputType"
          :info="option.info"
          :canScale="option.canScale"
          :autoCrop="option.autoCrop"
          :autoCropWidth="option.autoCropWidth"
          :autoCropHeight="option.autoCropHeight"
          :fixed="option.fixed"
          :fixedNumber="option.fixedNumber"
          :full="option.full"
          :fixedBox="option.fixedBox"
          :canMove="option.canMove"
          :canMoveBox="option.canMoveBox"
          :original="option.original"
          :centerBox="option.centerBox"
          :height="option.height"
          :infoTrue="option.infoTrue"
          :maxImgSize="option.maxImgSize"
          :enlarge="option.enlarge"
          :mode="option.mode"
          @realTime="realTime"
          @imgLoad="imgLoad"
        >
        </vue-cropper>
      </div>
      <!--底部操作工具按钮-->
      <div  class="footer-btn">
        <div  class="scope-btn">
          <label
            class="btn"
            for="uploads"
          >选择图片</label>
          <input
        type="file"
        id="uploads"
        style="position:absolute; clip:rect(0 0 0 0);"
        accept="image/png, image/jpeg, image/gif, image/jpg"
        @change="selectImg($event)"
      >
      <el-button
        size="mini"
        type="danger"
        plain
        icon="el-icon-zoom-in"
        @click="changeScale(1)"
      >放大</el-button>
      <el-button
        size="mini"
        type="danger"
        plain
        icon="el-icon-zoom-out"
        @click="changeScale(-1)"
      >缩小</el-button>
      <el-button
        size="mini"
        type="danger"
        plain
        @click="rotateLeft"
      >↺ 左旋转</el-button>
      <el-button
        size="mini"
        type="danger"
        plain
        @click="rotateRight"
      >↻ 右旋转</el-button>
    </div>
    <div class="upload-btn">
      <el-button
        size="mini"
        type="success"
        @click="uploadImg('blob')"
      >上传 <i class="el-icon-upload"></i></el-button>
    </div>
  </div>
</div>
<!--预览效果图-->
<div class="show-preview">
  <div
    :style="previews.div"
    class="preview"
  >
    <img
      :src="previews.url"
```

```
          :style="previews.img"
        >
      </div>
    </div>
  </div>
</template>
<script>
import { VueCropper } from 'vue-cropper'
import { getToken } from '@/utils/auth'
export default {
  name: 'CropperImage',
  components: {
    VueCropper,
  },
  props: ['Name'],
  data() {
    return {
      name: this.Name,
      previews: {},
      option: {
        img: '',
        outputSize: 1,
        outputType: 'png',
        info: true,
        canScale: true,
        autoCrop: true,
        autoCropWidth: 256,
        autoCropHeight: 256,
        fixed: true,
        fixedNumber: [1, 1],
        full: false,
        fixedBox: true,
        canMove: false,
        canMoveBox: true,
        original: false,
        centerBox: false,
        height: true,
        infoTrue: false,
        maxImgSize: 2000,
        enlarge: 1,
        mode: '256px 256px',
      },
    }
  },
  methods: {
    imgLoad(msg) {},
    changeScale(num) {
      num = num || 1
      this.$refs.cropper.changeScale(num)
    },
    rotateLeft() {
      this.$refs.cropper.rotateLeft()
    },
    rotateRight() {
      this.$refs.cropper.rotateRight()
    },
    realTime(data) {
      this.previews = data
    },
    selectImg(e) {
      let file = e.target.files[0]
      if (!/\.(jpg|jpeg|png|JPG|PNG)$/.test(e.target.value)) {
        this.$message({
          message: '图片类型要求：jpeg、jpg、png',
          type: 'error',
        })
        return false
      }
      let reader = new FileReader()
      reader.onload = (e) => {
        let data
        if (typeof e.target.result === 'object') {
          data = window.URL.createObjectURL(new Blob([e.target.result]))
        } else {
          data = e.target.result
        }
        this.option.img = data
      }
      reader.readAsDataURL(file)
    },
    uploadImg(type) {
      let _this = this
      if (type === 'blob') {
        this.$refs.cropper.getCropBlob(async (data) => {
          let formData = new FormData()
          formData.append('file', data, 'file.png')
          const ajax = new XMLHttpRequest()
          ajax.open('POST', 'http:
          ajax.setRequestHeader('token', getToken())
          ajax.send(formData)
          ajax.onreadystatechange = function () {
            if(ajax.readyState === 4){
              _this.$emit('uploadImgSuccess', ajax.responseText)
            }
          }
        })
      }
    },
  },
}
```

```
</script>
<style scoped lang="scss">
.cropper-content {
  display: flex;
  display: -webkit-flex;
  justify-content: flex-end;
  .cropper-box {
    flex: 1;
    width: 100%;
    .cropper {
      width: auto;
      height: 300px;
    }
  }
  .show-preview {
    flex: 1;
    -webkit-flex: 1;
    display: flex;
    display: -webkit-flex;
    justify-content: center;
    .preview {
      overflow: hidden;
      border: 1px solid #67c23a;
      background: #cccccc;
    }
  }
}
.footer-btn {
  margin-top: 30px;
  display: flex;
  display: -webkit-flex;
  justify-content: flex-end;
  .scope-btn {
    display: flex;
    display: -webkit-flex;
    justify-content: space-between;
    padding-right: 10px;
  }
  .upload-btn {
    flex: 1;
    -webkit-flex: 1;
    display: flex;
    display: -webkit-flex;
    justify-content: center;
  }
  .btn {
    outline: none;
    display: inline-block;
    line-height: 1;
    white-space: nowrap;
```

```
        cursor: pointer;
        -webkit-appearance: none;
        text-align: center;
        -webkit-box-sizing: border-box;
        box-sizing: border-box;
        outline: 0;
        -webkit-transition: 0.1s;
        transition: 0.1s;
        font-weight: 500;
        padding: 8px 15px;
        font-size: 12px;
        border-radius: 3px;
        color: #fff;
        background-color: #409eff;
        border-color: #409eff;
        margin-right: 10px;
    }
}
</style>
package com.jancoyan.jancoblog.service.impl;
        import com.baomidou.mybatisplus.core.conditions.query.QueryWrapper;
        import com.baomidou.mybatisplus.core.metadata.IPage;
        import com.baomidou.mybatisplus.extension.plugins.pagination.Page;
        import com.jancoyan.jancoblog.pojo.Article;
        import com.jancoyan.jancoblog.pojo.Comment;
        import com.jancoyan.jancoblog.mapper.CommentMapper;
        import com.jancoyan.jancoblog.pojo.DeletedComment;
        import com.jancoyan.jancoblog.pojo.PageComment;
        import com.jancoyan.jancoblog.service.CommentService;
        import com.baomidou.mybatisplus.extension.service.impl.ServiceImpl;
        import io.lettuce.core.dynamic.annotation.CommandNaming;
        import org.springframework.stereotype.Service;
        import java.util.List;
@Service
public class CommentServiceImpl extends ServiceImpl<CommentMapper, Comment> implements Com
mentService {
    @Override
    public IPage<Comment> listAll(String userId, Integer pn, Integer limit, String condition) {
        IPage<Comment> iPage = new Page<>(pn, limit);
        QueryWrapper<Comment> wrapper = new QueryWrapper<>();
        if(userId != null) wrapper.eq("user_id", userId);
        String[] split = condition.split("--");
        for (String item : split) {
            String[] split2 = item.split("=");
            if(split2.length < 2){
                continue;
            }
            if("comment_author_name".equals(split2[0])){
                wrapper.like("comment_author_name", split2[1]);
            }else if("article_title".equals(split2[0])){
                wrapper.like("article_title", split2[1]);
            }else if("start".equals(split2[0])){
                wrapper.gt("comment_date", split2[1]);
            }else if("end".equals(split2[0])){
                wrapper.lt("comment_date", split2[1]);
            }else if("rank_like".equals(split2[0])){
                if ("1".equals(split2[1])) {
                    wrapper.orderByAsc("comment_like_count");
                } else {
                    wrapper.orderByDesc("comment_like_count");
                }
            }
        }
        return baseMapper.getAll(iPage, wrapper);
    }
    @Override
    public IPage<Comment> listCommentByArticle(String id, Integer pn, Integer limit) {
        QueryWrapper<Comment> wrapper = new QueryWrapper<>();
        IPage<Comment> page = new Page<>(pn, limit);
        wrapper.eq("comment_article_id", id);
        wrapper.orderByDesc("comment_date");
        return baseMapper.getCommentByArticle(page, wrapper);
    }
    @Override
    public IPage<Comment> listCommentByUserPosted(String id, Integer pn, Integer limit, String co
ndition) {
        IPage<Comment> iPage = new Page<>(pn, limit);
        QueryWrapper<Comment> wrapper = new QueryWrapper<>();
        if(id != null) {
            wrapper.eq("comment_author_id", id);
        }
        String[] split = condition.split("--");
        for (String item : split) {
            String[] split2 = item.split("=");
            if(split2.length < 2){
                continue;
            }
            if("comment_author_name".equals(split2[0])){
                wrapper.like("user_id", split2[1]);
            }else if("article_title".equals(split2[0])){
                wrapper.like("article_title", split2[1]);
            }else if("start".equals(split2[0])){
                wrapper.gt("comment_date", split2[1]);
            }else if("end".equals(split2[0])){
                wrapper.lt("comment_date", split2[1]);
            }else if("rank_like".equals(split2[0])){
                if ("1".equals(split2[1])) {
                    wrapper.orderByAsc("comment_like_count");
                } else {
                    wrapper.orderByDesc("comment_like_count");
                }
            }
        }
        return baseMapper.getCommentByUserPosted(iPage, wrapper);
    }
    @Override
    public IPage<PageComment> listCommentByUserRecently(String authorId) {
        IPage<PageComment> iPage = new Page<>(1, 10);
        QueryWrapper<PageComment> wrapper = new QueryWrapper<>();
        wrapper.eq("article_author", authorId);
        wrapper.orderByDesc("comment_date");
        return baseMapper.getCommentByUserRecently(iPage, wrapper);
    }
    @Override
    public void deleteCommentByArticle(String ids) {
        QueryWrapper<Comment> wrapper = new QueryWrapper<>();
        Comment comment = new Comment();
        Article article = new Article();
        if(!ids.contains("&")){
            wrapper.eq("comment_article_id", ids);
            List<Comment> list = comment.selectList(wrapper);
            if(!list.isEmpty()){
                for (Comment item : list){
                    baseMapper.deleteComment(item.getCommentId());
                }
            }
        } else {
            String[] id = ids.split("&");
            for (String articleId : id) {
                wrapper.eq("comment_article_id", articleId);
                List<Comment> list = comment.selectList(wrapper);
                if(!list.isEmpty()){
                    for (Comment item : list){
                        baseMapper.deleteComment(item.getCommentId());
                    }
                }
            }
        }
    }
    @Override
    public void recoverCommentByArticle(String ids) {
        QueryWrapper<DeletedComment> wrapper = new QueryWrapper<>();
        DeletedComment comment = new DeletedComment();
        if(!ids.contains("&")){
            wrapper.eq("comment_article_id", ids);
            List<DeletedComment> list = comment.selectList(wrapper);
            if(!list.isEmpty()){
                for (DeletedComment item : list){
                    baseMapper.recoverComment(item.getCommentId());
                }
                }
            } else {
                String[] id = ids.split("&");
                for (String article : id) {
                    wrapper.eq("comment_article_id", article);
                    List<DeletedComment> list = comment.selectList(wrapper);
                    if(!list.isEmpty()){
                        for (DeletedComment item : list){
                            baseMapper.recoverComment(item.getCommentId());
                        }
                    }
                }
            }
        }
        @Override
        public boolean batchDeleteComment(String ids) {
            Comment comment = new Comment();
            boolean suc = false;
            if(!ids.contains("&")){
                comment.setCommentId(Integer.parseInt(ids));
                suc = comment.deleteById();
            } else {
                String[] id = ids.split("&");
                for (String item : id) {
                    comment.setCommentId(Integer.parseInt(item));
                    suc = comment.deleteById();
                }
            }
            return suc;
        }
        @Override
        public void likeComment(Integer id) {
            Comment comment = new Comment();
            comment.setCommentId(id);
            comment = comment.selectById();
            comment.setCommentLikeCount(comment.getCommentLikeCount() + 1);
            comment.updateById();
        }
}
package com.jancoyan.jancoblog.service.impl;
            import com.baomidou.mybatisplus.core.conditions.query.QueryWrapper;
            import com.baomidou.mybatisplus.core.metadata.IPage;
            import com.baomidou.mybatisplus.extension.plugins.pagination.Page;
            import com.jancoyan.jancoblog.pojo.Comment;
            import com.jancoyan.jancoblog.pojo.DeletedComment;
            import com.jancoyan.jancoblog.mapper.DeletedCommentMapper;
            import com.jancoyan.jancoblog.service.DeletedCommentService;
            import com.baomidou.mybatisplus.extension.service.impl.ServiceImpl;
            import org.springframework.stereotype.Service;
@Service
```

```
public class DeletedCommentServiceImpl extends ServiceImpl<DeletedCommentMapper, DeletedComm
ent> implements DeletedCommentService {
    @Override
    public IPage<DeletedComment> getCommentByArticle(String id, Integer pn, Integer limit) {
        QueryWrapper<DeletedComment> wrapper = new QueryWrapper<>();
        IPage<DeletedComment> page = new Page<>(pn, limit);
        wrapper.eq("comment_article_id", id);
        wrapper.orderByDesc("comment_date");
        return baseMapper.getCommentByArticle(page, wrapper);
    }
}
package com.jancoyan.jancoblog.service.impl;
        import com.baomidou.mybatisplus.core.conditions.query.QueryWrapper;
        import com.baomidou.mybatisplus.core.metadata.IPage;
        import com.baomidou.mybatisplus.extension.plugins.pagination.Page;
        import com.jancoyan.jancoblog.pojo.LikeRecord;
        import com.jancoyan.jancoblog.mapper.LikeRecordMapper;
        import com.jancoyan.jancoblog.service.LikeRecordService;
        import com.baomidou.mybatisplus.extension.service.impl.ServiceImpl;
        import org.springframework.stereotype.Service;
        import java.util.Date;
@Service
public class LikeRecordServiceImpl extends ServiceImpl<LikeRecordMapper, LikeRecord> implements
 LikeRecordService {
    @Override
    public IPage<LikeRecord> getUserReceive(String userId, Integer pn, Integer limit) {
        IPage<LikeRecord> iPage = new Page<>(pn, limit);
        QueryWrapper<LikeRecord> wrapper = new QueryWrapper<>();
        wrapper.eq("article_author", userId);
        wrapper.orderByDesc("like_date");
        return baseMapper.getUserReceive(iPage, wrapper, userId);
    }
    @Override
    public void insertRecord(Integer userId, String articleId) {
        LikeRecord record = new LikeRecord();
        record.setAuthorId(userId);
        record.setLikeDate(new Date());
        record.setArticleId(articleId);
        record.insert();
    }
    @Override
    public void deleteRecord(Integer userId, String articleId) {
        LikeRecord record = new LikeRecord();
        QueryWrapper<LikeRecord> wrapper = new QueryWrapper<>();
        wrapper.eq("article_id", articleId);
        wrapper.eq("author_id", userId);
        record.delete(wrapper);
    }
}
package com.jancoyan.jancoblog.service.impl;
```

```
        import com.baomidou.mybatisplus.core.conditions.query.QueryWrapper;
@Service
public class UserServiceImpl extends ServiceImpl<UserMapper, User> implements UserService {
    @Override
    public IPage<User> getAll(Integer pn, Integer limit, String condition) {
        IPage<User> iPage = new Page<>(pn, limit);
        QueryWrapper<User> wrapper = new QueryWrapper<>();
        String[] split = condition.split("--");
        for (String item : split) {
            String[] split2 = item.split("=");
            if(split2.length < 2){
                continue;
            }
            if("user_name".equals(split2[0])){
                wrapper.like("user_name", split2[1]);
            }else if("start".equals(split2[0])){
                wrapper.gt("user_create_date", split2[1]);
            }else if("end".equals(split2[0])){
                wrapper.lt("user_create_date", split2[1]);
            }
        }
        return baseMapper.getAll(iPage, wrapper);
    }
    @Override
    public VUserTotalData getUserTotalData(String userId) {
        return baseMapper.getUserTotalData(userId);
    }
    @Override
    public UserInfo getUserInfo(Integer userId) {
        return baseMapper.getUserInfo(userId);
    }
    @Override
    public User login(String username, String password) {
        password = MD5Util.getMD5(password);
        return baseMapper.login(username, password);
    }
}
package com.jancoyan.jancoblog.mapper;
public interface ArticleMapper extends BaseMapper<Article> {
    IPage<Article> getIndexList(IPage<Article> iPage, Wrapper ew);
    IPage<Article> getManageList(IPage<Article> iPage, Wrapper ew);
    Article getSingleArticle(String id);
    IPage<Article> getDeletedList(IPage<Article> iPage, Wrapper ew);
    void deleteCompletely(String articleId);
    void batchRecover(String articleId);
    IPage<PageArticle> getArticleByUserRecently(IPage<PageArticle> iPage, Wrapper ew);
    Article getSingleArticleDeleted(String id);
    Article getArticleEdit(String id);
}
package com.jancoyan.jancoblog.mapper;
```

```
        import com.baomidou.mybatisplus.core.conditions.Wrapper;
        import com.baomidou.mybatisplus.core.conditions.query.QueryWrapper;
        import com.baomidou.mybatisplus.core.metadata.IPage;
        import com.jancoyan.jancoblog.pojo.Comment;
        import com.baomidou.mybatisplus.core.mapper.BaseMapper;
        import com.jancoyan.jancoblog.pojo.PageComment;
public interface CommentMapper extends BaseMapper<Comment> {
    IPage<Comment> getAll(IPage<Comment> iPage, Wrapper ew);
    IPage<Comment> getCommentByArticle(IPage<Comment> page, QueryWrapper<Comment> ew);
    IPage<Comment> getCommentByUserPosted(IPage<Comment> iPage, Wrapper ew);
    IPage<PageComment> getCommentByUserRecently(IPage<PageComment> iPage, Wrapper ew);
    void deleteComment(Integer commentId);
    void recoverComment(Integer commentId);
}
package com.jancoyan.jancoblog.mapper;
        import com.baomidou.mybatisplus.core.conditions.Wrapper;
        import com.baomidou.mybatisplus.core.metadata.IPage;
        import com.jancoyan.jancoblog.pojo.User;
        import com.baomidou.mybatisplus.core.mapper.BaseMapper;
        import com.jancoyan.jancoblog.pojo.UserInfo;
        import com.jancoyan.jancoblog.pojo.VUserTotalData;
        import java.util.List;
public interface UserMapper extends BaseMapper<User> {
    IPage<User> getAll(IPage<User> iPage, Wrapper ew);
    VUserTotalData getUserTotalData(String userId);
    UserInfo getUserInfo(Integer userId);
    User login(String username, String password);
}
package com.jancoyan.jancoblog.utils;
        import javax.imageio.ImageIO;
        import javax.servlet.http.HttpServletRequest;
        import javax.servlet.http.HttpServletResponse;
        import javax.servlet.http.HttpSession;
        import java.awt.*;
        import java.awt.image.BufferedImage;
        import java.util.Random;
public class VerifyCodeUtil {
    public static final String RANDOMCODEKEY= "RANDOMVALIDATECODEKEY";
    private String randString = "0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ";
    private int width = 160;
    private int height = 40;
    private int lineSize = 40;
    private int stringNum = 4;
    private Random random = new Random();
    private Font getFont() {
        return new Font("Fixedsys", Font.CENTER_BASELINE, 28);
    }
    private Color getRandColor(int fc, int bc) {
        if (fc > 255)
            fc = 255;
        if (bc > 255)
            bc = 255;
        int r = fc + random.nextInt(bc - fc - 16);
        int g = fc + random.nextInt(bc - fc - 14);
        int b = fc + random.nextInt(bc - fc - 18);
        return new Color(r, g, b);
    }
    public String getRandcode(HttpServletRequest request, HttpServletResponse response) {
        HttpSession session = request.getSession();
        BufferedImage image = new BufferedImage(width, height, BufferedImage.TYPE_INT_BGR);
        Graphics g = image.getGraphics();
        g.fillRect(0, 0, width, height);
        g.setFont(new Font("Times New Roman", Font.ROMAN_BASELINE, 18));
        g.setColor(getRandColor(110, 133));
        for (int i = 0; i <= lineSize; i++) {
            drowLine(g);
        }
        String randomString = "";
        for (int i = 1; i <= stringNum; i++) {
            randomString = drowString(g, randomString, i);
        }
        session.removeAttribute(RANDOMCODEKEY);
        session.setAttribute(RANDOMCODEKEY, randomString);
        g.dispose();
        try {
            ImageIO.write(image, "JPEG", response.getOutputStream());
        } catch (Exception e) {
            e.printStackTrace();
        }
        return randomString;
    }
    private String drowString(Graphics g, String randomString, int i) {
        g.setFont(getFont());
        g.setColor(new Color(random.nextInt(101), random.nextInt(111), random
                .nextInt(121)));
        String rand = String.valueOf(getRandomString(random.nextInt(randString
                .length())));
        randomString += rand;
        g.translate(random.nextInt(3), random.nextInt(3));
        g.drawString(rand, 30 * i, 21);
        return randomString;
    }
    private void drowLine(Graphics g) {
        int x = random.nextInt(width);
        int y = random.nextInt(height);
        int xl = random.nextInt(13);
        int yl = random.nextInt(15);
        g.drawLine(x, y, x + xl, y + yl);
    }
    public String getRandomString(int num) {
        return  String.valueOf(randString.charAt(num));
    }
}
package  com.jancoyan.jancoblog.utils;
        import  eu.bitwalker.useragentutils.Browser;
        import  eu.bitwalker.useragentutils.OperatingSystem;
        import  eu.bitwalker.useragentutils.UserAgent;
        import  org.apache.commons.lang3.StringUtils;
        import  org.slf4j.Logger;
        import  org.slf4j.LoggerFactory;
        import  javax.servlet.http.HttpServletRequest;
public  class  UserAgentUtils  {
    private  static  Logger  logger  =  LoggerFactory.getLogger(UserAgentUtils.class);
    public  static  String  getUserAgent(HttpServletRequest  request)  {
        String  userAgent=request.getHeader("User-Agent");
        return  userAgent;
    }
    public  static  String  getOsVersion(HttpServletRequest  request)  {
        String  userAgent  =  getUserAgent(request);
        return  getOsVersion(userAgent);
    }
    public  static  String  getOsVersion(String  userAgent)  {
        String  osVersion  =  "";
        if(StringUtils.isBlank(userAgent))
            return  osVersion;
        String[]  strArr  =  userAgent.substring(userAgent.indexOf("(")+1,
                userAgent.indexOf(")")).split(";");
        if(null  ==  strArr  ||  strArr.length  ==  0)
            return  osVersion;
        osVersion  =  strArr[1];
        logger.info("osVersion  is:{}",  osVersion);
        return  osVersion;
    }
    private  static  OperatingSystem  getOperatingSystem(String  userAgent)  {
        UserAgent  agent  =  UserAgent.parseUserAgentString(userAgent);
        OperatingSystem  operatingSystem  =  agent.getOperatingSystem();
        return  operatingSystem;
    }
    public  static  String  getOs(HttpServletRequest  request)  {
        String  userAgent  =  getUserAgent(request);
        return  getOs(userAgent);
    }
    public  static  String  getOs(String  userAgent)  {
        OperatingSystem  operatingSystem  =   getOperatingSystem(userAgent);
        String  os  =  operatingSystem.getGroup().getName();
        logger.info("os  is:{}",  os);
        return  os;
    }
    public  static  String  getDevicetype(HttpServletRequest  request)  {
        String  userAgent  =  getUserAgent(request);
        return getDevicetype(userAgent);
    }
    public static String getDevicetype(String userAgent) {
        OperatingSystem operatingSystem =  getOperatingSystem(userAgent);
        String deviceType = operatingSystem.getDeviceType().toString();
        logger.info("deviceType is:{}", deviceType);
        return deviceType;
    }
    public static String getOsName(HttpServletRequest request) {
        String userAgent = getUserAgent(request);
        return getOsName(userAgent);
    }
    public static String getOsName(String userAgent) {
        OperatingSystem operatingSystem =  getOperatingSystem(userAgent);
        String osName = operatingSystem.getName();
        logger.info("osName is:{}", osName);
        return osName;
    }
    public static String getDeviceManufacturer(HttpServletRequest request) {
        String userAgent = getUserAgent(request);
        return getDeviceManufacturer(userAgent);
    }
    public static String getDeviceManufacturer(String userAgent) {
        OperatingSystem operatingSystem =  getOperatingSystem(userAgent);
        String deviceManufacturer = operatingSystem.getManufacturer().toString();
        logger.info("deviceManufacturer is:{}", deviceManufacturer);
        return deviceManufacturer;
    }
    public static Browser getBrowser(String agent) {
        UserAgent userAgent = UserAgent.parseUserAgentString(agent);
        Browser browser = userAgent.getBrowser();
        return browser;
    }
    public static String getBorderName(HttpServletRequest request) {
        String userAgent = getUserAgent(request);
        return getBorderName(userAgent);
    }
    public static String getBorderName(String userAgent) {
        Browser browser =  getBrowser(userAgent);
        String borderName = browser.getName();
        logger.info("borderName is:{}", borderName);
        return borderName;
    }
    public static String getBorderType(HttpServletRequest request) {
        String userAgent = getUserAgent(request);
        return getBorderType(userAgent);
    }
    public static String getBorderType(String userAgent) {
        Browser browser =  getBrowser(userAgent);
        String borderType = browser.getBrowserType().getName();
        logger.info("borderType is:{}", borderType);
        return borderType;
    }
    public static String getBorderGroup(HttpServletRequest request) {
        String userAgent = getUserAgent(request);
        return getBorderGroup(userAgent);
    }
    public static String getBorderGroup(String userAgent) {
        Browser browser =  getBrowser(userAgent);
        String browerGroup = browser.getGroup().getName();
        logger.info("browerGroup is:{}", browerGroup);
        return browerGroup;
    }
    public static String getBrowserManufacturer(HttpServletRequest request) {
        String userAgent = getUserAgent(request);
        return getBrowserManufacturer(userAgent);
    }
    public static String getBrowserManufacturer(String userAgent) {
        Browser browser =  getBrowser(userAgent);
        String browserManufacturer = browser.getManufacturer().getName();
        logger.info("browserManufacturer is:{}", browserManufacturer);
        return browserManufacturer;
    }
    public static String getBorderRenderingEngine(HttpServletRequest request) {
        String userAgent = getUserAgent(request);
        return getBorderRenderingEngine(userAgent);
    }
    public static String getBorderRenderingEngine(String userAgent) {
        Browser browser =  getBrowser(userAgent);
        String renderingEngine = browser.getRenderingEngine().name();
        logger.info("renderingEngine is:{}", renderingEngine);
        return renderingEngine;
    }
    public static String getBrowserVersion(HttpServletRequest request) {
        String userAgent = getUserAgent(request);
        return getBrowserVersion(userAgent);
    }
    public static String getBrowserVersion(String userAgent) {
        Browser browser =  getBrowser(userAgent);
        String borderVersion = browser.getVersion( userAgent).toString();
        return borderVersion;
    }
}
package com.jancoyan.jancoblog.utils;
        import org.springframework.beans.factory.annotation.Autowired;
        import org.springframework.data.redis.core.RedisTemplate;
        import org.springframework.stereotype.Component;
        import org.springframework.util.CollectionUtils;
        import java.util.Collection;
        import java.util.List;
        import java.util.Map;
        import java.util.Set;
        import java.util.concurrent.TimeUnit;
@Component
public class RedisUtil {
    @Autowired
    private RedisTemplate<String, Object> redisTemplate;
    public boolean expire(String key, long time) {
        try {
            if (time > 0) {
                redisTemplate.expire(key, time, TimeUnit.SECONDS);
            }
            return true;
        } catch (Exception e) {
            e.printStackTrace();
            return false;
        }
    }
    public long getExpire(String key) {
        return redisTemplate.getExpire(key, TimeUnit.SECONDS);
    }
    public boolean hasKey(String key) {
        try {
            return redisTemplate.hasKey(key);
        } catch (Exception e) {
            e.printStackTrace();
            return false;
        }
    }
    @SuppressWarnings("unchecked")
    public void del(String... key) {
        if (key != null && key.length > 0) {
            if (key.length == 1) {
                redisTemplate.delete(key[0]);
            } else {
                redisTemplate.delete((Collection<String>) CollectionUtils.arrayToList(key));
            }
        }
    }
    public Object get(String key) {
        return key == null ? null : redisTemplate.opsForValue().get(key);
    }
    public boolean set(String key, Object value) {
        try {
            redisTemplate.opsForValue().set(key, value);
            return true;
        } catch (Exception e) {
            e.printStackTrace();
            return false;
        }
```

```
}
public boolean set(String key, Object value, long time) {
    try {
        if (time > 0) {
            redisTemplate.opsForValue().set(key, value, time,
                    TimeUnit.SECONDS);
        } else {
            set(key, value);
        }
        return true;
    } catch (Exception e) {
        e.printStackTrace();
        return false;
    }
}
public long incr(String key, long delta) {
    if (delta < 0) {
        throw new RuntimeException("递增因子必须大于 0");
    }
    return redisTemplate.opsForValue().increment(key, delta);
}
public long decr(String key, long delta) {
    if (delta < 0) {
        throw new RuntimeException("递减因子必须大于 0");
    }
    return redisTemplate.opsForValue().increment(key, -delta);
}
public Object hget(String key, String item) {
    return redisTemplate.opsForHash().get(key, item);
}
public Map<Object, Object> hmget(String key) {
    return redisTemplate.opsForHash().entries(key);
}
public boolean hmset(String key, Map<String, Object> map) {
    try {
        redisTemplate.opsForHash().putAll(key, map);
        return true;
    } catch (Exception e) {
        e.printStackTrace();
        return false;
    }
}
public boolean hmset(String key, Map<String, Object> map, long time) {
    try {
        redisTemplate.opsForHash().putAll(key, map);
        if (time > 0) {
            expire(key, time);
        }
        return true;
    } catch (Exception e) {
            e.printStackTrace();
            return false;
        }
    }
    public boolean hset(String key, String item, Object value) {
        try {
            redisTemplate.opsForHash().put(key, item, value);
            return true;
        } catch (Exception e) {
            e.printStackTrace();
            return false;
        }
    }
    public boolean hset(String key, String item, Object value, long time) {
        try {
            redisTemplate.opsForHash().put(key, item, value);
            if (time > 0) {
                expire(key, time);
            }
            return true;
        } catch (Exception e) {
            e.printStackTrace();
            return false;
        }
    }
    public void hdel(String key, Object... item) {
        redisTemplate.opsForHash().delete(key, item);
    }
    public boolean hHasKey(String key, String item) {
        return redisTemplate.opsForHash().hasKey(key, item);
    }
    public double hincr(String key, String item, double by) {
        return redisTemplate.opsForHash().increment(key, item, by);
    }
    public double hdecr(String key, String item, double by) {
        return redisTemplate.opsForHash().increment(key, item, -by);
    }
    public Set<Object> sGet(String key) {
        try {
            return redisTemplate.opsForSet().members(key);
        } catch (Exception e) {
            e.printStackTrace();
            return null;
        }
    }
    public boolean sHasKey(String key, Object value) {
        try {
            return redisTemplate.opsForSet().isMember(key, value);
        } catch (Exception e) {
            e.printStackTrace();
            return false;
        }
    }
    public long sSet(String key, Object... values) {
        try {
            return redisTemplate.opsForSet().add(key, values);
        } catch (Exception e) {
            e.printStackTrace();
            return 0;
        }
    }
    public long sSetAndTime(String key, long time, Object... values) {
        try {
            Long count = redisTemplate.opsForSet().add(key, values);
            if (time > 0)
                expire(key, time);
            return count;
        } catch (Exception e) {
            e.printStackTrace();
            return 0;
        }
    }
    public long sGetSetSize(String key) {
        try {
            return redisTemplate.opsForSet().size(key);
        } catch (Exception e) {
            e.printStackTrace();
            return 0;
        }
    }
    public long setRemove(String key, Object... values) {
        try {
            Long count = redisTemplate.opsForSet().remove(key, values);
            return count;
        } catch (Exception e) {
            e.printStackTrace();
            return 0;
        }
    }
    public List<Object> lGet(String key, long start, long end) {
        try {
            return redisTemplate.opsForList().range(key, start, end);
        } catch (Exception e) {
            e.printStackTrace();
            return null;
        }
    }
    public long lGetListSize(String key) {
        try {
            return redisTemplate.opsForList().size(key);
        } catch (Exception e) {
            e.printStackTrace();
            return 0;
        }
    }
    public Object lGetIndex(String key, long index) {
        try {
            return redisTemplate.opsForList().index(key, index);
        } catch (Exception e) {
            e.printStackTrace();
            return null;
        }
    }
    public boolean lSet(String key, Object value) {
        try {
            redisTemplate.opsForList().rightPush(key, value);
            return true;
        } catch (Exception e) {
            e.printStackTrace();
            return false;
        }
    }
    public boolean lSet(String key, Object value, long time) {
        try {
            redisTemplate.opsForList().rightPush(key, value);
            if (time > 0)
                expire(key, time);
            return true;
        } catch (Exception e) {
            e.printStackTrace();
            return false;
        }
    }
    public boolean lSet(String key, List<Object> value) {
        try {
            redisTemplate.opsForList().rightPushAll(key, value);
            return true;
        } catch (Exception e) {
            e.printStackTrace();
            return false;
        }
    }
    public boolean lSet(String key, List<Object> value, long time) {
        try {
            redisTemplate.opsForList().rightPushAll(key, value);
            if (time > 0)
                expire(key, time);
            return true;
        } catch (Exception e) {
            e.printStackTrace();
```

```
                return  false;
            }
        }
        public  boolean  lUpdateIndex(String  key,  long  index,  Object  value)  {
            try  {
                redisTemplate.opsForList().set(key,  index,  value);
                return  true;
            }  catch  (Exception  e)  {
                e.printStackTrace();
                return  false;
            }
        }
        public  long  lRemove(String  key,  long  count,  Object  value)  {
            try  {
                Long  remove  =  redisTemplate.opsForList().remove(key,  count,
                        value);
                return  remove;
            }  catch  (Exception  e)  {
                e.printStackTrace();
                return  0;
            }
        }
}
package  com.jancoyan.jancoblog.controller;
            import  com.baomidou.mybatisplus.core.conditions.query.QueryWrapper;
            import  com.baomidou.mybatisplus.core.metadata.IPage;
            import  com.jancoyan.jancoblog.pojo.User;
            import  com.jancoyan.jancoblog.pojo.UserInfo;
            import  com.jancoyan.jancoblog.pojo.UserLogin;
            import  com.jancoyan.jancoblog.pojo.VUserTotalData;
            import  com.jancoyan.jancoblog.service.UserService;
            import  com.jancoyan.jancoblog.utils.*;
            import  io.lettuce.core.output.ScanOutput;
            import  org.springframework.beans.factory.annotation.Autowired;
            import  org.springframework.web.bind.annotation.RequestMapping;
            import  org.springframework.web.bind.annotation.RequestMethod;
            import  org.springframework.web.bind.annotation.RequestParam;
            import  org.springframework.web.bind.annotation.RestController;
            import  org.springframework.web.multipart.MultipartFile;
            import  javax.imageio.ImageIO;
            import  javax.servlet.http.Cookie;
            import  javax.servlet.http.HttpServletRequest;
            import  javax.servlet.http.HttpServletResponse;
            import  javax.servlet.http.HttpSession;
            import  java.awt.*;
            import  java.awt.image.BufferedImage;
            import  java.io.File;
            import  java.io.IOException;
            import  java.io.OutputStream;
            import  java.util.Date;
            import java.util.Locale;
            import java.util.Objects;
            import java.util.UUID;
            @RestController
@RequestMapping("/user")
public class UserController {
        @Autowired
        UserService service;
        @Autowired
        RedisUtil redisUtil;
        @RequestMapping(value = "/login", method = RequestMethod.POST)
        public Msg login(
                @RequestParam(value = "username") String username,
                @RequestParam(value = "password") String password,
                HttpServletRequest request
        ){
            String token = null;
            User user = service.login(username, password);
            if(null != user){
                token = JsonWebTokenUtils.createToken(user.getUserId().longValue());
                redisUtil.set(token, user, 30 * 60);
                user.setUserLastLoginDate(new Date());
                user.updateById();
            } else {
                return Msg.fail().add("msg", "登录失败");
            }
            UserLogin log = new UserLogin();
            String userAgent = request.getHeader("User-Agent");
            log.setLoginUser(user.getUserId());
            log.setLoginDate(new Date());
            log.setLoginAddress(NetworkUtils.queryAddressByIp(request.getRemoteAddr()));
            log.setLoginIp(request.getRemoteAddr());
            log.setUserAgent(userAgent);
            log.setBrowserName(UserAgentUtils.getBorderName(userAgent));
            log.setBrowserVersion(UserAgentUtils.getBrowserVersion(userAgent));
            log.setOsName(UserAgentUtils.getOsName(userAgent));
            log.setOsVersion(UserAgentUtils.getOsVersion(userAgent));
            if(null != token){
                log.insert();
                return Msg.success().add("token", token);
            }
            return Msg.fail();
        }
        @RequestMapping(value = "/data/total", method = RequestMethod.GET)
        public Msg getUserTotalData(
                @RequestParam(value = "id", defaultValue = "-1")String userId,
                HttpServletRequest request
        ){
            if("-1".equals(userId)){
                String token = request.getHeader("token");
                if(null == token) {
                    return Msg.fail();
                }
                User user = (User)redisUtil.get(token);
                userId = String.valueOf(user.getUserId());
            }
            VUserTotalData data = service.getUserTotalData(userId);
            if (null != data){
                return Msg.success().add("data", data);
            } else {
                return Msg.fail().add("msg", "找不到用户");
            }
        }
        @RequestMapping(value = "/userinfo", method = RequestMethod.GET)
        public Msg getUserInfo(HttpServletRequest request){
            String token = request.getHeader("token");
            User user = (User)redisUtil.get(token);
            if(null != user){
                return Msg.success().add("user", user);
            }
            return Msg.fail().add("msg", "找不到用户");
        }
        @RequestMapping(value = "/userdetail", method = RequestMethod.GET)
        public Msg getUserDetailInfo(
                HttpServletRequest request
        ){
            String token = request.getHeader("token");
            User user = (User)redisUtil.get(token);
            if(null == user){
                return Msg.expire().add("msg", "用户已过期");
            }
            UserInfo info = service.getUserInfo(user.getUserId());
            if(null != info){
                return Msg.success().add("info", info);
            } else {
                return Msg.fail();
            }
        }
        @RequestMapping(value = "/authorinfo", method = RequestMethod.GET)
        public Msg getAuthorInfo(
                @RequestParam(value = "id")String userId
        ){
            VUserTotalData data = service.getUserTotalData(userId);
            if(null != data) {
                return Msg.success().add("data", data);
            } else {
                return Msg.fail().add("msg", "获取数据失败");
            }
        }
        @RequestMapping(value = "/register", method = RequestMethod.POST)
        public Msg register(
                @RequestParam(value = "username") String userName,
                @RequestParam(value = "password") String password,
                @RequestParam(value = "code") String code,
                HttpServletRequest request
        ){
            String verify = (String) request.getSession().getAttribute(VerifyCodeUtil.RANDOMCODEKE
    Y);
            if(!verify.toLowerCase(Locale.ROOT).equals(code.toLowerCase(Locale.ROOT))){
                return Msg.fail().add("msg", "验证码输入错误");
            }
            User user = new User();
            user.setUserName(userName)
                    .setUserLastLoginDate(null)
                    .setUserId(null)
                    .setUserSignature("Hello World")
                    .setUserPassword(MD5Util.getMD5(password))
                    .setUserRole(2)
                    .setUserCreateDate(new Date())
                    .setUserIp(request.getRemoteAddr());
            boolean insert = user.insert();
            if(!insert){
                return Msg.fail().add("msg", "用户注册失败");
            }
            return Msg.success().add("success", insert);
        }
        @RequestMapping(value = "/checkusername", method = RequestMethod.POST)
        public Msg checkUserNameUnique(
                @RequestParam(value = "username") String userName
        ){
            User user = new User();
            QueryWrapper<User> queryWrapper = new QueryWrapper<>();
            queryWrapper.eq("user_name", userName);
            int count = user.selectCount(queryWrapper);
            return Msg.success().add("unique", count == 0);
        }
        @RequestMapping(value = "/all", method = RequestMethod.GET)
        public Msg getAllUser(
                @RequestParam(value = "pn")String pn,
                @RequestParam(value = "limit", defaultValue = "10")String limit,
                @RequestParam(value = "condition", defaultValue = "")String condition
        ){
            IPage<User> iPage = service.getAll(Integer.parseInt(pn),
                    Integer.parseInt(limit),
                    condition);
            return Msg.success().add("pageInfo", iPage);
        }
        @RequestMapping(value = "/delete", method = RequestMethod.POST)
        public Msg batchDelte(
                @RequestParam(value = "ids") String ids
        ){
            User user = new User();
            boolean suc = false;
            if(!ids.contains("&")){
                user.setUserId(Integer.parseInt(ids));
                suc = user.deleteById();
            } else {
                String[] id = ids.split("&");
                for (String item : id) {
                    user.setUserId(Integer.parseInt(item));
                    suc = user.deleteById();
                }
            }
            return Msg.success().add("suc", suc ? "success" : "fail");
        }
        @RequestMapping(value = "/upload/avatar", method = RequestMethod.POST)
        public Msg uploadAvatar(
                @RequestParam(value = "file") MultipartFile file,
                HttpServletRequest request
        ){
            String token = request.getHeader("token");
            if (null == token){
                return Msg.fail();
            }
            User user = (User) redisUtil.get(token);
            if (null == user){
                return Msg.expire();
            }
            if (null == file) {
                return Msg.fail().add("msg", "请选择要上传的图片");
            }
            if (file.getSize() > 1024 * 1024 * 10) {
                return Msg.fail().add("msg", "文件大小不能大于 10M");
            }
            String suffix = Objects.requireNonNull(file.getOriginalFilename()).substring(file.getOriginalFile
    name().lastIndexOf(".") + 1);
            if (!"jpg,jpeg,gif,png".toUpperCase().contains(suffix.toUpperCase())) {
                return Msg.fail().add("msg", "请选择 jpg,jpeg,gif,png 格式的图片");
            }
            String savePath = null;
            try {
                savePath = new File(".").getCanonicalPath() + "\\target\\classes\\static\\avatar\\";
            } catch (IOException e) {
                e.printStackTrace();
            }
            File savePathFile = new File(savePath);
            if (!savePathFile.exists()) {
                savePathFile.mkdir();
            }
            String filename = user.getUserId() + "." + suffix;
            File oldFile = new File(savePath + filename);
            if(oldFile.exists()){
                oldFile.delete();
            }
            try {
                file.transferTo(new File(savePath + filename));
            } catch (Exception e) {
                e.printStackTrace();
                return Msg.fail().add("msg", "保存文件异常");
            }
            return Msg.success().add("suc", true);
        }
        @RequestMapping(value = "/changepwd", method = RequestMethod.POST)
        public Msg changePassword(
                @RequestParam(value = "id", defaultValue = "-1") String userId,
                @RequestParam(value = "old") String oldPassword,
                @RequestParam(value = "new") String newPassword,
                HttpServletRequest request
        ){
            boolean suc = false;
            if("-1".equals(userId)){
                String token = request.getHeader("token");
                if(null == token) {
                    return Msg.fail();
                }
                User user = (User) redisUtil.get(token);
                if(user.getUserPassword().equals(MD5Util.getMD5(oldPassword))){
                    user.setUserPassword(MD5Util.getMD5(newPassword));
                    suc = user.updateById();
                    return Msg.success().add("suc", suc);
                } else {
                    return Msg.success().add("suc", suc);
                }
            }
            User user = new User();
            user.setUserId(Integer.parseInt(userId));
            user.setUserPassword(MD5Util.getMD5(newPassword));
            suc = user.updateById();
            return Msg.success().add("suc", suc);
        }
        @RequestMapping(value = "/info/update", method = RequestMethod.POST)
        public Msg updateUserInfo(
                @RequestParam(value = "user")String userInfo,
                HttpServletRequest request
        ){
            String token = request.getHeader("token");
            if(null == token){
                return Msg.fail().add("msg", "用户未登录");
            }
            User user = new User();
```

```
user = (User) redisUtil.get(token);
if(null == user){
    return Msg.expire();
}
UserInfo info = new UserInfo();
userInfo = userInfo.substring(1, userInfo.length() - 1);
String[] kvs = userInfo.split(",");
for (String kv: kvs) {
    String[] nameAndValue = kv.split(":");
    String name = nameAndValue[0].substring(1, nameAndValue[0].length() - 1);
    String value = nameAndValue[1];
    if((!"null".equals(value)) && (!"".equals(value))){
        if("userId".equals(name)){
            info.setUserId(Integer.parseInt(value));
            continue;
        }
        if("userEmail".equals(name)){
            info.setUserEmail(value.substring(1, value.length() - 1));
            continue;
        }
        if("userSex".equals(name)){
            if("\"1\"".equals(value)){
                info.setUserSex(1);
            } else if("\"0\"".equals(value)){
                info.setUserSex(0);
            }
            continue;
        }
        if("userRegion".equals(name)){
            info.setUserRegion(value.substring(1, value.length() - 1));
            continue;
        }
        if("userBirthdate".equals(name)){
            info.setUserBirthdate(TimeUtils.castDateStringToDateTypeYMD(
                    value.substring(1, 11)));
            continue;
        }
        if("userTelephone".equals(name)){
            info.setUserTelephone(value.substring(1, value.length() - 1));
            continue;
        }
        if("userRealName".equals(name)){
            info.setUserRealName(value.substring(1, value.length() - 1));
            continue;
        }
        if("userSchool".equals(name)){
            info.setUserSchool(value.substring(1, value.length() - 1));
            continue;
        }
        if("userMajor".equals(name)){
```

```
                    info.setUserMajor(value.substring(1, value.length() - 1));
                    continue;
                }
                if("userEnterSchoolDate".equals(name)) {
                    info.setUserEnterSchoolDate(TimeUtils.castDateStringToDateTypeYMD(
                            value.substring(1, 11)));
                    continue;
                }
                if("userAcademicDegree".equals(name)){
                    info.setUserAcademicDegree(value.substring(1, value.length() - 1));
                    continue;
                }
                if("userCompany".equals(name)){
                    info.setUserCompany(value.substring(1, value.length() - 1));
                    continue;
                }
                if("userPosition".equals(name)){
                    info.setUserPosition(value.substring(1, value.length() - 1));
                    continue;
                }
                if("userField".equals(name)){
                    info.setUserField(value.substring(1, value.length() - 1));
                }
            }
        }
        boolean suc = info.updateById();
        return Msg.success().add("suc", suc);
    }
    @RequestMapping(value = "/info/user", method = RequestMethod.POST)
    public Msg updateUser(
            @RequestParam(value = "username") String username,
            @RequestParam(value = "signature") String signature,
            HttpServletRequest request
    ){
        String token = request.getHeader("token");
        if(null == token){
            return Msg.fail().add("msg", "用户未登录");
        }
        User user = new User();
        user = (User) redisUtil.get(token);
        if(null == user){
            return Msg.expire();
        }
        user.setUserSignature(signature);
        user.setUserName(username);
        boolean suc = user.updateById();
        if(suc){
            redisUtil.set(token, user, 20 * 60);
        }
        return Msg.success().add("suc", suc);
    }
    @RequestMapping(value = "/getverifycode", method = RequestMethod.GET)
    public void generateValidationCode(
            HttpServletRequest request,
            HttpServletResponse response,
            HttpSession session
    ){
        try {
            response.setContentType("image/jpeg");
            response.setHeader("Pragma", "No-cache");
            response.setHeader("Cache-Control", "no-cache");
            response.setDateHeader("Expire", 0);
            VerifyCodeUtil verifyCodeUtil = new VerifyCodeUtil();
            verifyCodeUtil.getRandcode(request, response);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    @RequestMapping(value = "/add", method = RequestMethod.POST)
    public Msg addUser(
            @RequestParam("username") String userName,
            @RequestParam("password") String password,
            HttpServletRequest request
    ){
        User user = new User();
        user.setUserName(userName)
                .setUserLastLoginDate(null)
                .setUserId(null)
                .setUserSignature("Hello World")
                .setUserPassword(MD5Util.getMD5(password))
                .setUserRole(2)
                .setUserCreateDate(new Date())
                .setUserIp(request.getRemoteAddr());
        boolean insert = user.insert();
        if(!insert){
            return Msg.fail().add("msg", "用户注册失败");
        }
        return Msg.success().add("success", insert);
    }
    @RequestMapping(value = "/logout", method = RequestMethod.POST)
    public Msg logout(HttpServletRequest request){
        String token = request.getHeader("token");
        if(null == token || "".equals(token)){
            return Msg.fail();
        }
        redisUtil.del(token);
        System.out.println("用户已退出");
        return Msg.success().add("suc", true);
    }
}
```