2022-01-19---96期《改进探花交友》技术亮点

Jaden代码进度

- 前端app接口：day05-圈子互动-发布评论---done

  Todo：day05-16小视频：查询评论列表；
- 后端管理系统：分页查询app用户列表(tanhua库-tb_user、tb_user_info表)---done

  Todo：day09-15小视频：查看用户详情。

1. maven管理项目

- 管理jar包；
- maven模块，管理彼此之间的相互依赖关系。

  具体参看《Jaden探花交友架构图》

2. Yapi、SwaggerUI---接口文档

根据代码自动生成接口文档。【前后端分离开发必需】

3. lombok---自动生成Getter/Setter方法、构造器、日志

domain类上添加注解即可自动生成Getter/Setter方法、构造器、日志等。

常用注解：@Data、@NoArgsConstructor、@AllArgsConstructor、@Slf4j。

```
<dependency>
   <groupId>org.projectlombok</groupId>
   <artifactId>lombok</artifactId>
   <version>1.18.8</version>
</dependency>
```

4. Mybatis-Plus---基础增删改查、分页查询、统一设置创建和更新时间。

对应的maven坐标

```
<dependency>
    <groupId>com.baomidou</groupId>
    <artifactId>mybatis-plus</artifactId>
    <version>3.4.1</version>
</dependency>
```

- 基础增删改查dao层代码已自动生成。只要继承BaseMapper即可。

```
import com.baomidou.mybatisplus.core.mapper.BaseMapper;
import com.tanhua.model.domain.User;

public interface UserMapper extends BaseMapper<User> {
}
```

- 支持分页查询

  1>在容器中注入一个MybatisPlusInterceptor即可使用。

```
@Configuration
public class EnableMybatisPlusPageQuery {

    @Bean
    public MybatisPlusInterceptor mybatisPlusInterceptor() {
        MybatisPlusInterceptor interceptor = new MybatisPlusInterceptor();
        interceptor.addInnerInterceptor(new
PaginationInnerInterceptor(DbType.MYSQL));
        return interceptor;
    }
}
```

  2>代码使用

```
import com.baomidou.mybatisplus.extension.plugins.pagination.Page;

    @Override
    public IPage<UserInfo> findBlackListPageByUserId(Long userId, int page,
int pageSize) {
        Page pages = new Page(page, pageSize);  //分页查询前---创建一个Page对
象，设置分页参数
        return userInfoMapper.findBlackListPageByUserId(pages, userId);
    }
```

```
package com.tanhua.dubbo.mappers;

import com.baomidou.mybatisplus.core.mapper.BaseMapper;
import com.baomidou.mybatisplus.core.metadata.IPage;
import com.baomidou.mybatisplus.extension.plugins.pagination.Page;
import com.tanhua.model.domain.UserInfo;
import org.apache.ibatis.annotations.Param;
import org.apache.ibatis.annotations.Select;

public interface UserInfoMapper extends BaseMapper<UserInfo> {

    @Select("select * from tb_user_info where id in (\n" +
```

```
            "  SELECT black_user_id FROM tb_black_list where user_id=#
{userId}\n" +
            ")")
    IPage<UserInfo> findBlackListPageByUserId(@Param("pages") Page pages,
@Param("userId") Long userId);
}
```

```
        IPage<UserInfo> iPage =
blackListApi.findBlackListPageByUserId(userId, page, pageSize);
        //3、对象转化，将查询的Ipage对象的内容封装到PageResult中
        PageResult pr = new PageResult(page, pageSize,
iPage.getTotal(),iPage.getRecords());
        //4、返回
        return pr;
```

- 自动填充---记录的创建时间createdTime、更新时间updateTime

  1>实现mybatisplus的MetaObjectHandler接口

```
package com.tanhua.dubbo.handler;

import com.baomidou.mybatisplus.core.handlers.MetaObjectHandler;
import org.apache.ibatis.reflection.MetaObject;
import org.springframework.stereotype.Component;

import java.util.Date;

@Component
public class MyMetaObjectHandler implements MetaObjectHandler {

    @Override
    public void insertFill(MetaObject metaObject) {
        Object created = getFieldValByName("created", metaObject);
        if (null == created) {
            //字段为空，可以进行填充
            setFieldValByName("created", new Date(), metaObject);
        }

        Object updated = getFieldValByName("updated", metaObject);
        if (null == updated) {
            //字段为空，可以进行填充
            setFieldValByName("updated", new Date(), metaObject);
        }
    }

    @Override
    public void updateFill(MetaObject metaObject) {
```

```
            //更新数据时，直接更新字段
            setFieldValByName("updated", new Date(), metaObject);
        }
    }
```

2>实体类-配合设置

```
package com.tanhua.model.domain;

import com.baomidou.mybatisplus.annotation.FieldFill;
import com.baomidou.mybatisplus.annotation.TableField;
import lombok.Data;

import java.io.Serializable;
import java.util.Date;

@Data
public class BasePojo implements Serializable {
    @TableField(fill = FieldFill.INSERT)
    private Date created;
    @TableField(fill = FieldFill.INSERT_UPDATE)
    private Date updated;
}
```

5. Hutool工具包---生成随机验证码、md5加密等

- 从实体list中，获取**指定属性**构成的list 或 map等。

```
List<Movement> items = (List<Movement>) pageResult.getItems();
List<Long> friendUserIds = CollUtil.getFieldValues(items, "userId",
Long.class);
```

- 生成随机验证码图片，且获取图片中的验证码。

```
//获取验证码图片
    @GetMapping("/verification")
    public void verification(String uuid, HttpServletResponse response) throws
IOException {
        //1、生成验证码对象
        LineCaptcha captcha = CaptchaUtil.createLineCaptcha(299, 97);
        //2、验证码存入Redis
        String code = captcha.getCode();
        redisTemplate.opsForValue().set(Constants.CAP_CODE+uuid,code);
        //3、输出验证码图片
        captcha.write(response.getOutputStream());
    }
```

- md5加密字符串，得到对应密文

```
//将password-明文字符串，进行md5加密
String password = SecureUtil.md5(password);
```

6. JWT格式的token令牌

格式的登录令牌：三部分

```
package com.tanhua.commons.utils;

import io.jsonwebtoken.Claims;
import io.jsonwebtoken.Jwts;
import io.jsonwebtoken.SignatureAlgorithm;
import org.apache.commons.lang3.StringUtils;

import java.util.Date;
import java.util.Map;

public class JwtUtils {

    // TOKEN的有效期1小时（S）
    private static final int TOKEN_TIME_OUT = 3600 * 24;

    // 加密KEY
    private static final String TOKEN_SECRET = "itcast";


    // 生成Token
    public static String getToken(Map params){
        long currentTime = System.currentTimeMillis();
        return Jwts.builder()
                .signWith(SignatureAlgorithm.HS512, TOKEN_SECRET) //加密方式
                .setExpiration(new Date(currentTime + TOKEN_TIME_OUT * 1000))
//过期时间戳
                .addClaims(params)
                .compact();
    }

    /**
     * 获取Token中的claims信息
     */
    public static Claims getClaims(String token) {
        return Jwts.parser()
                .setSigningKey(TOKEN_SECRET)
                .parseClaimsJws(token).getBody();
    }
```

```
    /**
     * 是否有效 true-有效，false-失效
     */
    public static boolean verifyToken(String token) {
        if(StringUtils.isEmpty(token)) {
            return false;
        }
        try {
            Claims claims = Jwts.parser()
                    .setSigningKey("itcast")
                    .parseClaimsJws(token)
                    .getBody();
        }catch (Exception e) {
            return false;
        }

        return true;
    }
}
```

7. Spring---拦截器解析当前登录用户信息、@ControllerAdvice统一异常处理、@Async

- 自定义Spring拦截器，从jwt格式的token中 获取当前登录用户id，保存到ThreadLocal中

  1> 定义ThreadLocal工具类

```
package com.tanhua.server.interceptor;

import com.tanhua.model.domain.User;

/**
 * 工具类:向ThreadLocal存储数据
 */
public class UserHolder {
    private static ThreadLocal<User> tl = new ThreadLocal<>();

    //将用户对象存入ThreadLocal
    public static void set(User user) {
        tl.set(user);
    }

    //从当前线程，获取用户对象
    public static User get() {
        return tl.get();
    }

    //从当前线程，获取用户对象的id
```

```
    public static Long getUserId() {
        return tl.get().getId();
    }

    //从当前线程，获取用户对象的手机号码
    public static String getMobile() {
        return tl.get().getMobile();
    }

    //清空
    public static void remove() {
        tl.remove();
    }
}
```

2>自定义Spring拦截器，实现HandlerInterceptor接口

```
package com.tanhua.server.interceptor;

import com.tanhua.commons.utils.JwtUtils;
import com.tanhua.model.domain.User;
import io.jsonwebtoken.Claims;
import org.springframework.lang.Nullable;
import org.springframework.web.servlet.HandlerInterceptor;
import org.springframework.web.servlet.ModelAndView;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

/**
 * 自定义拦截器:拦截器主要拦截请求controller请求
 */
public class TokenInterceptor implements HandlerInterceptor {
    //前置拦截
    public boolean preHandle(HttpServletRequest request,
HttpServletResponse response, Object handler) throws Exception {
        //1、获取请求头
        String token = request.getHeader("Authorization");
        //2、判断token是否有效  如果token失效，返回状态码401，拦截
        boolean verifyToken = JwtUtils.verifyToken(token);
        if(!verifyToken) {
            response.setStatus(401);
            return false;
        }
        //3、如果token正常可用，放行

        //解析token,获取id和手机号,构造User对象,存入ThreadLocal
        Claims claims = JwtUtils.getClaims(token);
```

```
        Integer id = (Integer) claims.get("id");
        String mobile = (String) claims.get("phone");

        User user = new User();
        user.setMobile(mobile);
        user.setId(Long.valueOf(id));
        UserHolder.set(user);
        return true;
    }

    //响应处理
    public void postHandle(HttpServletRequest request,
                           HttpServletResponse response,
                           Object handler,
                           @Nullable ModelAndView modelAndView) throws
Exception {
    }

    //最终增强
    public void afterCompletion(HttpServletRequest request,
HttpServletResponse response, Object handler, @Nullable Exception ex)
throws Exception {
        UserHolder.remove();
    }

}
```

3> 配置自定义Spring拦截器的，拦截规则

```
package com.tanhua.server.interceptor;

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 WebConfig implements WebMvcConfigurer {
    /**
     * 将自定义的拦截器加入到springmvc的拦截器链中
     * @param registry
     */
    @Override
    public void addInterceptors(InterceptorRegistry registry) {
        registry.addInterceptor(new TokenInterceptor())
                .addPathPatterns("/**")
                .excludePathPatterns(new String[]
{"/user/login","/user/loginVerification"});
```

```
    }
}
```

- @ControllerAdvise + @ExceptionHandler 统一异常处理

```
package com.tanhua.server.exception;

import com.tanhua.model.vo.ErrorResult;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.ControllerAdvice;

/**
 * 全局异常处理
 *   1、通过注解，声明异常处理类
 *   2、编写方法，在方法内部处理异常，构造响应数据
 *   3、方法上编写注解，指定此方法可以处理的异常类型
 */
@ControllerAdvice
public class ExceptionHandler {

    //业务自定义异常处理

 @org.springframework.web.bind.annotation.ExceptionHandler(BusinessExceptio
n.class)
    public ResponseEntity handlerException(BusinessException
businessException){
        businessException.printStackTrace();
        ErrorResult errorResult = businessException.getErrorResult();
        return
ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).body(errorResult);
    }

    //其他异常

 @org.springframework.web.bind.annotation.ExceptionHandler(Exception.class)
    public ResponseEntity handlerException1(Exception e) {
        e.printStackTrace();
        return
ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).body(ErrorResult.er
ror());
    }
}
```

- Spring的@Async注解的作用---发布动态，根据friends表写入好友时间线表movement_timeline中。

  【项目中多线程的使用，真正起作用吗？】

8. Redis

在项目中的使用---5种数据类型

- string数据类型
  - 登录时的短信验证码，保存到redis中，同时设置5分钟过期；eg: 18737104575---123456

    ```
    String code = "123456";
    //3.将验证码存入redis
    redisTemplate.opsForValue().set(phone,code, Duration.ofMinutes(5));
    ```

  - 推荐动态：推荐系统为用户推荐合适的动态pid

    eg：MOVEMENTS_RECOMMEND_106---2562,3639,2063,3448
- map数据类型
  - 圈子互动 之 本人已喜欢/已点赞，前端对应动态喜欢/点赞图标高亮显示

    eg：MOVEMENTS_INTERACT_60dc0ee36f1f5f170ddb0813---[MOVEMENT_LOVE_106---1]
  - 记录用户-查看访客列表的时间

    eg：VISITOR_USER---[106---1612148600271、1---1612148600300、111---1612148600331、2---1612148600279]
- set数据类型
  - 探花左右滑-喜欢/不喜欢

    eg：

    USER_LIKE_KEY_106---1,2,3,4

    USER_NOT_LIKE_KEY---45,67,78

    使用set数据类型，保存喜欢/不喜欢的人好处：

    1>redis的set数据类型保证了元素值唯一；

    2>针对用户“共同喜欢的人”需求时，可使用redis支持的集合交集、并集、差集。

9. mongoDB

- mongoDB分页查询：skip、limit的使用

```
// 1.创建查询对象,设置分页查询
// 跳过多少个数据应该根据 当前页数-1 * 每页需要展示的数据个数
Query query = Query.query(Criteria.where("toUserId").is(userId))
        .with(Sort.by(Sort.Order.desc("score")))
        .skip((page - 1) * pageSize)
        .limit(pageSize);

// 2.调用mongoTemplate进行查询,获取list集合
List<RecommendUser> recommendUsers = mongoTemplate.find(query,
RecommendUser.class);
```

- mongDB字段自增，使用sequence单独一个集合表来实现

```
package com.tanhua.model.mongo;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
import org.bson.types.ObjectId;
import org.springframework.data.mongodb.core.mapping.Document;

/**
 * 确保mongo中的字段自增的类
 */
@Document(collection = "sequence")
@Data
@AllArgsConstructor
@NoArgsConstructor
public class Sequence {

    private ObjectId id;

    private long seqId; //自增序列

    private String collName;  //集合名称
}
```

```
package com.tanhua.dubbo.utils;


import com.tanhua.model.mongo.Sequence;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.mongodb.core.FindAndModifyOptions;
import org.springframework.data.mongodb.core.MongoTemplate;
import org.springframework.data.mongodb.core.query.Criteria;
import org.springframework.data.mongodb.core.query.Query;
import org.springframework.data.mongodb.core.query.Update;
import org.springframework.stereotype.Component;

/**
 * 这个工具类用来实现mongo中的字段的自动增长
 */
@Component
public class IdWorker {

    @Autowired
    private MongoTemplate mongoTemplate;

    public Long getNextId(String collName) {
```

```
        Query query = new Query(Criteria.where("collName").is(collName));

        Update update = new Update();
        update.inc("seqId", 1);  //inc: increment 自动增长

        //FindAndModifyOptions---保证并发情况的线程安全
        FindAndModifyOptions options = new FindAndModifyOptions();
        options.upsert(true);//表示更新数据，如果此数据不存在，表示插入一条新数据
        options.returnNew(true);//将更新后的数据返回
        //返回更新后的最新数据
        Sequence sequence = mongoTemplate.findAndModify(query, update,
options, Sequence.class);
        return sequence.getSeqId();
    }
}
```

- mongDB多集合操作，事务保证---???【movement、movement_timeline】

10. 项目文件存储方案

- 阿里云OSS---付费
- FastDFS---ing
- 分布式存储系统Minio

11. 探花后端请求---Nginx+Gateway

[后端浏览器访问端口-8088]

[Nginx---nginx.conf]

```
server {
        listen 8088;

        location / {
                root /Users/jaden/Jaden_Data/1_Work/code/tanhua/html;
                index index.html;
        }

        location  /management {
                #转发后台地址(网关地址)
                        proxy_pass http://127.0.0.1:8888/admin;

        }
}
```

[Gateway---/app或/admin]

```
    routes:
      # 探花移动端
      - id: tanhua-app-server
        uri: lb://tanhua-app-server
        predicates:
          - Path=/app/**
        filters:
          - StripPrefix= 1  # 表示在将请求发送到下游之前从请求中剥离的路径个数
      # 探花后台系统
      - id: tanhua-admin
        uri: lb://tanhua-admin
        predicates:
          - Path=/admin/**
        filters:
          - StripPrefix= 1
```