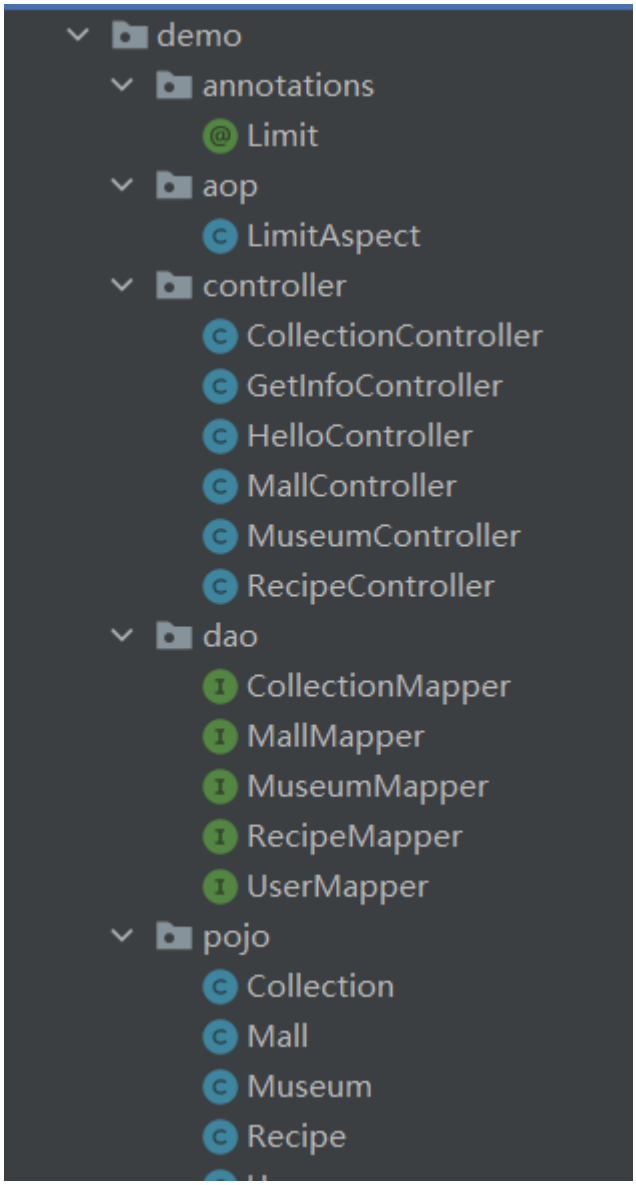
菜谱3.0版本（2022.6.17）

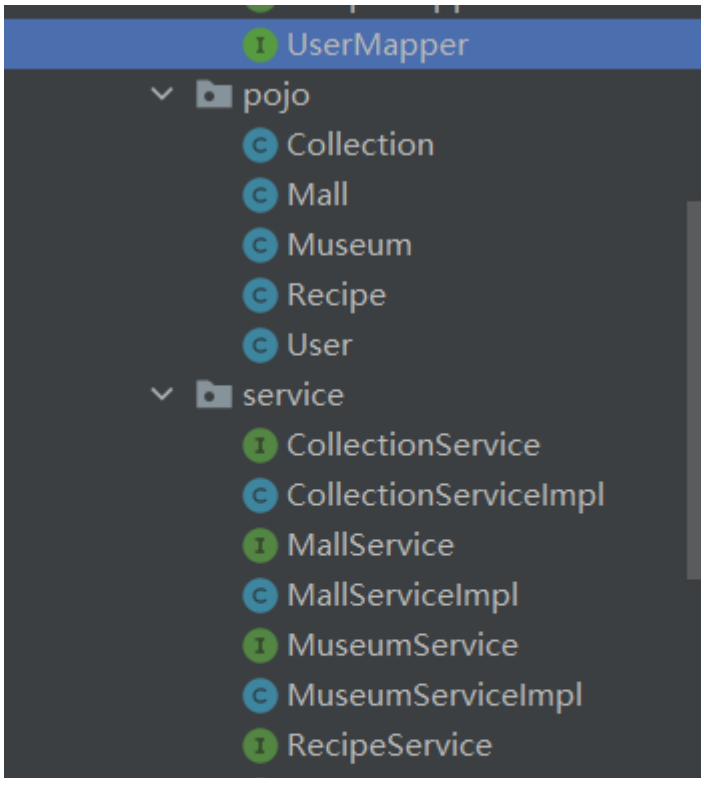
版本特点：

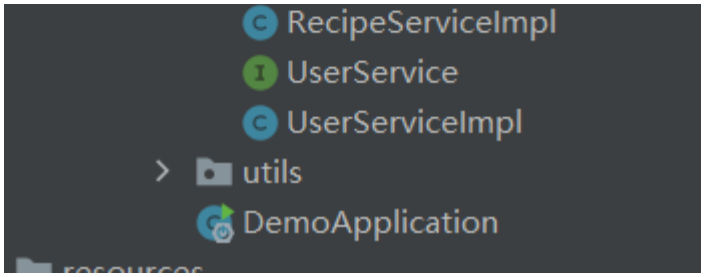
增加了日志功能
增加了限流功能

项目结构

```
  |  └com
|  |      └example
|  |          └demo
|  |              ├annotations
|  |              ├aop
|  |              ├controller
|  |              ├dao
|  |              ├pojo
|  |              ├service
|  |              └utils
```


demo
annotations
Limit
aop
LimitAspect
controller
CollectionController
GetInfoController
HelloController
MallController
MuseumController
RecipeController
dao
CollectionMapper
MallMapper
MuseumMapper
RecipeMapper
UserMapper
pojo
Collection
Mall
Museum
Recipe


UserMapper
pojo
Collection
Mall
Museum
Recipe
User
service
CollectionService
CollectionServiceImpl
MallService
MallServiceImpl
MuseumService
MuseumServiceImpl
RecipeService



代码

annotations层
Limit.java

```
package com.example.demo.annotations;

import java.lang.annotation.*;
import java.util.concurrent.TimeUnit;

// 学习网站：https://blog.csdn.net/qq_34217386/article/details/122100904

/**
 * @author czh
 */@Retention(RetentionPolicy.RUNTIME)
@Target({ElementType.METHOD})
@Documented
public @interface Limit {

    // 资源key
    String key() default "";

    // 最多访问次数
    double permitsPerSecond();

    // 时间
    long timeout();

    // 时间类型
    TimeUnit timeunit() default TimeUnit.MILLISECONDS;

    // 提示信息
    String msg() default "请求过于频繁,请稍后再试";

}
```

aop层
LimitAspect.java

```
package com.example.demo.aop;

import com.alibaba.fastjson.JSONObject;
import com.example.demo.annotations.Limit;

import com.example.demo.utils.IPUtils;
import com.example.demo.utils.StatusCode;
import com.fasterxml.jackson.core.JsonProcessingException;
import com.fasterxml.jackson.databind.ObjectMapper;
import com.google.common.collect.Maps;
import com.google.common.util.concurrent.RateLimiter;
import lombok.extern.slf4j.Slf4j;
import org.aspectj.lang.ProceedingJoinPoint;
import org.aspectj.lang.annotation.Around;
import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.reflect.MethodSignature;
import org.springframework.stereotype.Component;
import org.springframework.web.context.request.RequestContextHolder;
import org.springframework.web.context.request.ServletRequestAttributes;
import org.springframework.web.util.WebUtils;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;
import java.io.PrintWriter;
import java.lang.reflect.Method;
import java.util.Map;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

@Slf4j
@Aspect
@Component

public class LimitAspect {
    private final Map<String, RateLimiter> limitMap =
Maps.newConcurrentMap();
    private Logger logger = LoggerFactory.getLogger(this.getClass());
    @Around("@annotation(com.example.demo.annotations.Limit)")
    public Object around(ProceedingJoinPoint pjp) throws Throwable {
        MethodSignature signature = (MethodSignature)pjp.getSignature();
        Method method = signature.getMethod();
        //拿limit的注解
```

```
        Limit limit = method.getAnnotation(Limit.class);
        if (limit != null) {
            //key作用：不同的接口，不同的流量控制
            String key=limit.key();
            RateLimiter rateLimiter;
            //验证缓存是否有命中key
            if (!limitMap.containsKey(key)) {
                // 创建令牌桶
                rateLimiter = RateLimiter.create(limit.permitsPerSecond());
                limitMap.put(key, rateLimiter);
                log.info("新建了令牌桶={}，容量=
{}",key,limit.permitsPerSecond());
            }
            rateLimiter = limitMap.get(key);
            // 拿令牌
            boolean acquire = rateLimiter.tryAcquire(limit.timeout(),
limit.timeunit());
            // 拿不到命令，直接返回异常提示
            if (!acquire) {
                log.debug("令牌桶={}，获取令牌失败",key);
                //throw new Exception(limit.msg());
                responseFail(limit.msg());
            }
        }
        return pjp.proceed();
    }

    /**
     * 直接向前端抛出异常
     * @param msg 提示信息
     */
    private void responseFail(String msg) throws IOException {
        HttpServletResponse response=((ServletRequestAttributes)
RequestContextHolder.getRequestAttributes()).getResponse();
        Map<String, Object> map = StatusCode.error(5001,msg);
        ServletRequestAttributes attr = (ServletRequestAttributes)
RequestContextHolder.getRequestAttributes();
        HttpServletRequest request = attr.getRequest();
        logger.error("IP为"+IPUtils.getIpAddress(request)+": 访问频率过高");

        response.setCharacterEncoding("UTF-8");
        response.setContentType("application/json; charset=utf-8");
        PrintWriter out = response.getWriter();
        JSONObject json = new JSONObject(map);
        out.append(json.toString());
        out.close();
```

```
    }
}
```

controller层
CollectionController.java

```
package com.example.demo.controller;

import com.example.demo.annotations.Limit;
import com.example.demo.dao.CollectionMapper;
import com.example.demo.pojo.Collection;
import com.example.demo.service.CollectionService;
import com.example.demo.utils.JwtUtils;
import com.example.demo.utils.StatusCode;
import io.jsonwebtoken.Claims;
import io.swagger.annotations.ApiOperation;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;


import javax.validation.Valid;
import java.util.List;
import java.util.Map;

@RestController          //注解可以使结果以Json字符串的形式返回给客户端
@RequestMapping(value = "/api/collection")          //使链接还有一个 /api/public
class CollectionController {
    @Autowired
    CollectionService collectionService;

    @Limit(key = "collection_type", permitsPerSecond = 1, timeout = 500, msg
= "请求过于频繁,请稍后再试！")
    @PostMapping("/type")
    @ApiOperation(value = "返回收藏页面")
    public Map<String, Object> listTypeCollection(@RequestParam("type")
String type, @RequestParam("token") String token) { //获取前端传过来的code
        //获取请求时的token
        System.out.println(type);
        JwtUtils jwt = JwtUtils.getInstance();
        Claims claims = jwt.check(token);
        if (claims != null) {
            String openid = (String) claims.get("openid");
            try {
                List<Collection> tmp = null;
                if( "mall".equals(type) ){
                    tmp = collectionService.listCollectionMall(openid);
```

```
                    System.out.println(type);
                }else if( "recipe".equals(type) ){
                    tmp = collectionService.listCollectionRecipe(openid);
                }
                Map<String, Object> data = StatusCode.success(tmp);
                return data;
            } catch (Exception e) {
                e.printStackTrace();
                return StatusCode.error(3001, "服务器内部错误: " +
e.toString());
            }
        }else{
            //非法token
            return StatusCode.error(2001, "用户未登录");
        }
    }

    @Limit(key = "collection_insert", permitsPerSecond = 1, timeout = 500,
msg = "请求过于频繁,请稍后再试! ")
    @PostMapping("/insert")
    @ApiOperation(value = "添加收藏信息")
    public Map<String, Object> insertCollection(@Valid Collection
collection,@RequestParam("token") String token) {

        //获取请求时的token
        JwtUtils jwt = JwtUtils.getInstance();
        Claims claims = jwt.check(token);
        if (claims != null) {
            try {
                String openid = (String) claims.get("openid");
                collection.setOpenid(openid);
                if( collectionService.insertCollection(collection) == true )
{
                    return StatusCode.success("插入成功");
                }else{
                    return StatusCode.success("插入失败，内容已存在");
                }

            } catch (Exception e) {
                e.printStackTrace();
                Map<String, Object> data = StatusCode.error(3001, "服务器内部错
误: " + e.toString());
                return data;
            }
        }else{
            //非法token
            return StatusCode.error(2001, "用户未登录");
        }
```

```
    }

    @Limit(key = "collection_delete", permitsPerSecond = 1, timeout = 500,
msg = "请求过于频繁,请稍后再试！")
    @PostMapping("/delete")
    @ApiOperation(value = "删除收藏信息")
    public Map<String, Object> deleteCollection(@RequestParam("id") String
id,@RequestParam("type") String type,@RequestParam("token") String token) {

        //获取请求时的token
        JwtUtils jwt = JwtUtils.getInstance();
        Claims claims = jwt.check(token);
        if (claims != null) {
            try {
                String openid = (String) claims.get("openid");
                if( collectionService.deleteCollection(id,type,openid) ==
true){
                    return StatusCode.success("删除成功");
                }else{
                    return StatusCode.success("删除失败，内容不存在");
                }

            } catch (Exception e) {
                e.printStackTrace();
                Map<String, Object> data = StatusCode.error(3001, "服务器内部错
误：" + e.toString());
                return data;
            }
        }else{
            //非法token
            return StatusCode.error(2001, "用户未登录");
        }
    }

    @Limit(key = "collection_exist", permitsPerSecond = 1, timeout = 500, msg
= "请求过于频繁,请稍后再试！")
    @PostMapping("/exist")
    @ApiOperation(value = "是否存在收藏信息")
    public Map<String, Object> existCollection(@RequestParam("id") String
id,@RequestParam("type") String type,@RequestParam("token") String token) {

        //获取请求时的token
        JwtUtils jwt = JwtUtils.getInstance();
        Claims claims = jwt.check(token);
        if (claims != null) {
            try {
                String openid = (String) claims.get("openid");
                if( collectionService.Collection(id,type,openid) == true){
```

```
                    return StatusCode.success("该收藏存在");
                }else{
                    return StatusCode.success("该收藏不存在");
                }

            } catch (Exception e) {
                e.printStackTrace();
                Map<String, Object> data = StatusCode.error(3001, "服务器内部错
误: " + e.toString());
                return data;
            }
        }else{
            //非法token
            return StatusCode.error(2001, "用户未登录");
        }
    }

}
```

GetInfoController.java

```
package com.example.demo.controller;

//获取信息

import com.alibaba.fastjson.JSONObject;

import com.example.demo.dao.UserMapper;
import com.example.demo.pojo.User;
import com.example.demo.service.UserService;
import com.example.demo.utils.JwtUtils;
import com.example.demo.utils.RequestUtils;

import com.example.demo.utils.StatusCode;
import io.swagger.annotations.ApiOperation;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

import java.util.HashMap;
import java.util.Map;
import java.util.UUID;

@RestController          //注解可以使结果以Json字符串的形式返回给客户端
@RequestMapping(value = "/api/getinfo")          //使链接还有一个 /api/public
class GetInfoController {

```

```
    /**
     *  小程序的测试号
     *  appid
     *  secret     * */    public static String appid = "wx023e3a4297441859";
    public static String secret = "e7056611f6888dfd25745ff9b9dae882" ;

    @Autowired
    UserService userService;

    @GetMapping("/cs")
    public Map<String, Object> cs(){
        Map<String, Object> map = new HashMap<>();
        map.put("msg","helloworld");
        return map;
    }

    /**
     * 获取电话号码
     * 请求：POST
     * 链接：地址/api/getinfo/getphone
     * 参数：code
     *  Content-Type: application/json;     * 返回
     json {
     "msg":"ok"     }     * */    @PostMapping("getphone")
    @ApiOperation(value="获取用户手机号码")
    public Map<String, Object> getPhone(@RequestParam("code") String code){
//获取前端传过来的code

        try{
            System.out.println(code);

            //获取获取小程序全局唯一后台接口调用凭据（access_token）
            String getTokenUrl = "https://api.weixin.qq.com/cgi-bin/token?
grant_type=client_credential&appid="+appid+"&secret="+secret;

            //调用get请求去访问微信小程序自带的链接，将返回结果存储到jsonStringa中
            String jsonStringa =  RequestUtils.doGet(getTokenUrl);

            //String转JSON
            JSONObject jsonObject = JSONObject.parseObject(jsonStringa);

            //获取JSON数据中的access_token
            String access_token = jsonObject.getString("access_token");
            //提交参数
            String getPhoneUrl =
"https://api.weixin.qq.com/wxa/business/getuserphonenumber?
access_token="+access_token;
```

```
            Map<String,Object> map =  new HashMap<String,Object>();

            //将code放到map中
            map.put("code",code);
            //Map格式转化成JSON格式
            JSONObject json = new JSONObject(map);

            //向微信小程序接口提交Post请求得到结果
            String jsonStringb = RequestUtils.doPostForm(getPhoneUrl,json);

            //String转JSON
            JSONObject jsonObject2 = JSONObject.parseObject(jsonStringb);
            HashMap hashMap =
JSONObject.parseObject(jsonObject2.toJSONString(), HashMap.class);
            //请求成功
            if( 0 == (int)hashMap.get("errcode") ){
                Map<String, String> data = new HashMap<>();
                JSONObject tmp2 = (JSONObject)hashMap.get("phone_info");
                data.put("phone",(String)tmp2.get("phoneNumber"));
                //将结果存储下来
                return StatusCode.success(data);
            }else{
                return StatusCode.error((int)hashMap.get("errcode"),"获取失
败");
            }
        }catch (Exception e){
            System.out.println(e);
            return StatusCode.error(3001,"服务器内部错误: "+e.toString());
        }

    }

    //https://blog.csdn.net/qq_41432730/article/details/123617323
    //    @PostMapping(value = "login")
    @ApiOperation(value="登录")
    public @ResponseBody Map<String,Object> login(@RequestParam("code")
String code/*,@RequestParam("avatarUrl") String
avatarUrl,@RequestParam("avatarUrl") String name*/) {
        Map<String, String> data = new HashMap<String, String>();
        try {
            //Get请求（登录凭证校验）
            String getAuthUrl =
"https://api.weixin.qq.com/sns/jscode2session?appid=" + appid + "&secret=" +
secret + "&js_code=" + code + "&grant_type=authorization_code";
            //进行get请求
            String jsonString = RequestUtils.doGet(getAuthUrl);
            //String转JSON，再json转为map
            JSONObject jsonObject = JSONObject.parseObject(jsonString);
```

```
            HashMap hashMap =
JSONObject.parseObject(jsonObject.toJSONString(), HashMap.class);

            //注意这里要加上 hashMap.get("errcode") == null            if (
hashMap.get("errcode") == null || 0 == (int) hashMap.get("errcode")) {
//请求成功
                //得到openid和session_key去生成3rd_session
                //这个生成3rd_session的方式自己决定即可，比如使用SHA或Base64算法都可
以。例如：将session_key或openid+session_key作为SHA或Base64算法的输入，输出结果做为
3rd_session来使用，同时要将openid，session_key，3rd_session三者关联存储到数据库中，
方便下次拿3rd_session获取session_key或openid做其他处理。
                String openid = (String) hashMap.get("openid");
                String session_key = (String) hashMap.get("session_key");

                //判断是否注册过
                int tmp = userService.isUser(openid);
                if ( tmp == 0 ) {
                    //没有注册过
                    User user = new User(openid);
                    //插入数据库
                    userService.insertUser(user);
                }
                //生成token
                JwtUtils jwt = JwtUtils.getInstance();
                String token = jwt
                        .setClaim("openid",openid)
                        .generateToken();

                Map<String, String> tmp3 = new HashMap<>();
                tmp3.put("token",token);
                //将结果存储下来
                return StatusCode.success(tmp3);

            } else {
                return StatusCode.error((int) hashMap.get("errcode"),"获取失
败");
            }
        }
        catch(Exception e){
            e.printStackTrace();
            return StatusCode.error(3001,"服务器内部错误: "+e.toString());
        }

    }
}
```

HelloController.java

```
package com.example.demo.controller;

import com.example.demo.annotations.Limit;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;

import java.util.HashMap;
import java.util.Map;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

@RestController          //注解可以使结果以Json字符串的形式返回给客户端
public class HelloController {
    @GetMapping("/hello")
    public String hello() {
        return "hello SpringBoot";
    }
    private Logger logger = LoggerFactory.getLogger(this.getClass());
    @Limit(key = "cs", permitsPerSecond = 1, timeout = 500, msg = "请求过于频
繁,请稍后再试! ")
    @GetMapping("/cs")
    public Map<String, Object> cs() {
        Map<String, Object> map = new HashMap<>();
        map.put("msg", "helloworld");
        logger.error("cs");
        logger.info("cs");
        logger.debug("cs");

        return map;
    }

}
```

MallController.java

```
package com.example.demo.controller;

import com.example.demo.annotations.Limit;
import com.example.demo.pojo.Mall;
import com.example.demo.service.MallService;
import com.example.demo.utils.FileUploadUtils;
import com.example.demo.utils.StatusCode;
import io.swagger.annotations.ApiOperation;
```

```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;
import org.springframework.web.multipart.MultipartFile;

import javax.servlet.http.HttpServletRequest;
import javax.validation.Valid;
import java.util.List;
import java.util.Map;

@RestController          //注解可以使结果以Json字符串的形式返回给客户端
@RequestMapping(value = "/api/mall")          //使链接还有一个 /api/public class
MallController {
    @Autowired
    MallService mallService;

    @PostMapping("/")
    @ApiOperation(value = "获取全部商品的关键信息")
    public Map<String, Object> listMall(@RequestParam("page") int page) { //
获取前端传过来的code
        try {
            List<Mall> tmp = mallService.listMall(page, 8);
            Map<String, Object> data = StatusCode.success(tmp);
            return data;
        } catch (Exception e) {
            e.printStackTrace();
            return StatusCode.error(3001, "服务器内部错误: " + e.toString());
        }
    }

    @PostMapping("/type")
    @Limit(key = "mall_type", permitsPerSecond = 1, timeout = 500, msg = "请求
过于频繁,请稍后再试! ")
    @ApiOperation(value = "获取指定类型商品的关键信息")
    public Map<String, Object> listTypeMall(@RequestParam("page") int
page,@RequestParam("type") String type) { //获取前端传过来的code
        try {
            List<Mall> tmp = mallService.listTypeMall(page, 8,type);
            Map<String, Object> data = StatusCode.success(tmp);
            return data;
        } catch (Exception e) {
            e.printStackTrace();
            return StatusCode.error(3001, "服务器内部错误: " + e.toString());
        }
    }

    @PostMapping("/{id}")
    @Limit(key = "mall_id", permitsPerSecond = 1, timeout = 500, msg = "请求过
于频繁,请稍后再试! ")
```

```
    @ApiOperation(value = "获取具体商品信息")
    public Map<String, Object> getMall(@PathVariable(name = "id") String id)
{
        //limit 为8
        try {
            Mall tmp = mallService.getMall(id);
            Map<String, Object> data = StatusCode.success(tmp);
            return data;
        } catch (Exception e) {
            e.printStackTrace();
            return StatusCode.error(3001, "服务器内部错误: " + e.toString());
        }
    }

    @PostMapping("/insert")
    @Limit(key = "mall_insert", permitsPerSecond = 1, timeout = 500, msg =
"请求过于频繁,请稍后再试! ")
    @ApiOperation(value = "添加具体菜谱信息")
    public Map<String, Object> insertMall(@RequestParam MultipartFile file,
@Valid Mall mall, HttpServletRequest request) {
        try {
            //保存文件
            String filename = FileUploadUtils.SaveServer(file, request);
            //用UUID来生成唯一的id
            //String id = UUID.randomUUID().toString();             //把实体类
中的picture设置成文件路径
            mall.setGoodsPicture(filename);
            //Mall.setId(id);

            mallService.insertMall(mall);
            Map<String, Object> data = StatusCode.success("插入成功");
            return data;
        } catch (Exception e) {
            e.printStackTrace();
            Map<String, Object> data = StatusCode.error(3001, "服务器内部错
误: " + e.toString());
            return data;
        }
    }


    @PostMapping("/search")
    @Limit(key = "mall_search", permitsPerSecond = 1, timeout = 500, msg =
"请求过于频繁,请稍后再试! ")
    @ApiOperation(value = "搜索商品信息")
    public Map<String, Object> search(@RequestParam("find") String
find,@RequestParam("page") int page) {
        //limit 为8
```

```
        try {
            List<Mall> tmp = mallService.searchMall(find,page,8);
            Map<String, Object> data = StatusCode.success(tmp);
            return data;
        } catch (Exception e) {
            e.printStackTrace();
            Map<String, Object> data = StatusCode.error(3001, "服务器内部错
误: " + e.toString());
            return data;
        }
    }


}
```

MuseumController.java

```
package com.example.demo.controller;

import com.example.demo.annotations.Limit;
import com.example.demo.pojo.Museum;
import com.example.demo.service.MuseumService;
import com.example.demo.utils.FileUploadUtils;
import com.example.demo.utils.StatusCode;
import io.swagger.annotations.ApiOperation;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;
import org.springframework.web.multipart.MultipartFile;

import javax.servlet.http.HttpServletRequest;
import javax.validation.Valid;
import java.util.List;
import java.util.Map;

@RestController          //注解可以使结果以Json字符串的形式返回给客户端
@RequestMapping(value = "/api/museums")          //使链接还有一个 /api/public
class MuseumController {
    @Autowired
    MuseumService museumService;

    @PostMapping("/")
    @ApiOperation(value = "获取全部博物馆的关键信息")
    public Map<String, Object> listMuseum(@RequestParam("page") int page) {
//获取前端传过来的code
        try {
            List<Museum> tmp = museumService.listMuseum(page, 8);
```

```
            Map<String, Object> data = StatusCode.success(tmp);
            return data;
        } catch (Exception e) {
            e.printStackTrace();
            Map<String, Object> data = StatusCode.error(3001, "服务器内部错
误: " + e.toString());
            return data;
        }
    }

    @PostMapping("/type")
    @Limit(key = "museum_type", permitsPerSecond = 1, timeout = 500, msg =
"请求过于频繁,请稍后再试! ")
    @ApiOperation(value = "获取指定类型博物馆的关键信息")
    public Map<String, Object> listTypeMall(@RequestParam("page") int
page,@RequestParam("type") String type) { //获取前端传过来的code
        try {
            System.out.println(type);
            List<Museum> tmp = museumService.listTypeMuseum(type, page,8);
            Map<String, Object> data = StatusCode.success(tmp);
            return data;
        } catch (Exception e) {
            e.printStackTrace();
            Map<String, Object> data = StatusCode.error(3001, "服务器内部错
误: " + e.toString());
            return data;
        }
    }

    @PostMapping("/{id}")
    @Limit(key = "museum_id", permitsPerSecond = 1, timeout = 500, msg = "请求
过于频繁,请稍后再试! ")
    @ApiOperation(value = "获取具体博物馆信息")
    public Map<String, Object> getMuseum(@PathVariable(name = "id") String
id) {
        //limit 为8
        try {
            Museum tmp = museumService.getMuseum(id);
            Map<String, Object> data = StatusCode.success(tmp);
            return data;
        } catch (Exception e) {
            e.printStackTrace();
            Map<String, Object> data = StatusCode.error(3001, "服务器内部错
误: " + e.toString());
            return data;
        }
    }
```

```
    @PostMapping("/insert")
    @Limit(key = "museum_insert", permitsPerSecond = 1, timeout = 500, msg =
"请求过于频繁,请稍后再试！")
    @ApiOperation(value = "添加具体博物馆信息")
    public Map<String, Object> insertMuseum(@RequestParam MultipartFile file,
@Valid Museum museum, HttpServletRequest request) {
        try {
            //保存文件
            String filename = FileUploadUtils.SaveServer(file, request);
            //用UUID来生成唯一的id
            //String id = UUID.randomUUID().toString();            //把实体类
中的picture设置成文件路径
            museum.setPicture(filename);
            //Museum.setId(id);

            museumService.insertMuseum(museum);
            Map<String, Object> data = StatusCode.success("插入成功");
            return data;
        } catch (Exception e) {
            e.printStackTrace();
            Map<String, Object> data = StatusCode.error(3001, "服务器内部错
误：" + e.toString());
            return data;
        }
    }

}
```

RecipeController.java

```
package com.example.demo.controller;

import com.example.demo.annotations.Limit;
import com.example.demo.pojo.Recipe;
import com.example.demo.service.RecipeService;
import com.example.demo.utils.FileUploadUtils;
import com.example.demo.utils.StatusCode;
import io.swagger.annotations.ApiOperation;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;
import org.springframework.web.multipart.MultipartFile;

import javax.servlet.http.HttpServletRequest;
import javax.validation.Valid;
import java.util.List;
import java.util.Map;
```

```java
@RestController          //注解可以使结果以Json字符串的形式返回给客户端
@RequestMapping(value = "/api/recipes")          //使链接还有一个 /api/public
class RecipeController {
    @Autowired
    RecipeService recipeService;

    @PostMapping("/")
    @ApiOperation(value = "获取全部菜谱的关键信息")
    public Map<String, Object> listRecipe(@RequestParam("page") int page) {
//获取前端传过来的code
        try {
            List<Recipe> tmp = recipeService.listRecipe(page, 8);
            Map<String, Object> data = StatusCode.success(tmp);
            return data;
        } catch (Exception e) {
            e.printStackTrace();
            Map<String, Object> data = StatusCode.error(3001, "服务器内部错
误: " + e.toString());
            return data;
        }
    }

    @PostMapping("/type")
    @Limit(key = "recipe_type", permitsPerSecond = 1, timeout = 500, msg =
"请求过于频繁,请稍后再试! ")
    @ApiOperation(value = "获取指定类型菜谱的关键信息")
    public Map<String, Object> listTypeMall(@RequestParam("page") int
page,@RequestParam("type") String type) { //获取前端传过来的code
        try {
            //System.out.println(type);
            List<Recipe> tmp = recipeService.listTypeRecipe(page, 8,type);
            Map<String, Object> data = StatusCode.success(tmp);
            return data;
        } catch (Exception e) {
            e.printStackTrace();
            Map<String, Object> data = StatusCode.error(3001, "服务器内部错
误: " + e.toString());
            return data;
        }
    }

    @PostMapping("/{id}")
    @Limit(key = "recipe_id", permitsPerSecond = 1, timeout = 500, msg = "请求
过于频繁,请稍后再试! ")
    @ApiOperation(value = "获取具体菜谱信息")
    public Map<String, Object> getRecipe(@PathVariable(name = "id") String
id) {
```

```
        //limit 为8
        try {
            Recipe tmp = recipeService.getRecipe(id);
            Map<String, Object> data = StatusCode.success(tmp);
            return data;
        } catch (Exception e) {
            e.printStackTrace();
            Map<String, Object> data = StatusCode.error(3001, "服务器内部错
误: " + e.toString());
            return data;
        }
    }

    @PostMapping("/insert")
    @Limit(key = "recipe_insert", permitsPerSecond = 1, timeout = 500, msg =
"请求过于频繁,请稍后再试! ")
    @ApiOperation(value = "添加具体菜谱信息")
    public Map<String, Object> insertRecipe(@RequestParam MultipartFile file,
@Valid Recipe recipe, HttpServletRequest request) {
        try {
            //保存文件
            String filename = FileUploadUtils.SaveServer(file, request);
            //用UUID来生成唯一的id
            //String id = UUID.randomUUID().toString();             //把实体类
中的picture设置成文件路径
            recipe.setPicture(filename);
            //recipe.setId(id);

            recipeService.insertRecipe(recipe);
            Map<String, Object> data = StatusCode.success("插入成功");
            return data;
        } catch (Exception e) {
            e.printStackTrace();
            Map<String, Object> data = StatusCode.error(3001, "服务器内部错
误: " + e.toString());
            return data;
        }
    }


    @PostMapping("/search")
    @Limit(key = "recipe_search", permitsPerSecond = 1, timeout = 500, msg =
"请求过于频繁,请稍后再试! ")
    @ApiOperation(value = "搜索菜谱信息")
    public Map<String, Object> search(@RequestParam("find") String
find,@RequestParam("page") int page) {
        //limit 为8
        try {
```

```
            List<Recipe> tmp = recipeService.searchRecipe(find,page,8);
            Map<String, Object> data = StatusCode.success(tmp);
            return data;
        } catch (Exception e) {
            e.printStackTrace();
            Map<String, Object> data = StatusCode.error(3001, "服务器内部错
误: " + e.toString());
            return data;
        }
    }


}
```

dao层
CollectionMapper.java

```
package com.example.demo.dao;

import com.example.demo.pojo.Collection;
import com.example.demo.pojo.Mall;
import org.apache.ibatis.annotations.Delete;
import org.apache.ibatis.annotations.Insert;
import org.apache.ibatis.annotations.Mapper;
import org.apache.ibatis.annotations.Select;

import java.util.List;

//create table collection(id int not null AUTO_INCREMENT,itemid varchar(50)
not null,openid varchar(50) not null,type varchar(50),primary key(id));
@Mapper
public interface CollectionMapper {
    /**
     * 筛选商店收藏信息
     */
    @Select("select
m.goodsName,m.goodsDescribe,m.goodsId,m.goodsPicture,m.goodsPrice from mall m
inner join collection c on m.goodsId = c.itemid  where c.type='mall' and
c.openid=#{openid} ;")
    List<Collection> listCollectionMall(String openid);



    /**
     * 筛选菜谱收藏信息
     */
```

```
    @Select("select f.picture,f.dishes,f.id,f.describes from food f inner
join collection c on f.id = c.itemid  where c.type='recipe' and c.openid=#
{openid} ;")
    List<Collection> listCollectionRecipe(String openid);

    /**
     * 判断是否已存在
     */
    @Select("select count(*) from  collection where type=#{type} and openid=#
{openid} and itemid=#{itemid} ;")
    int Collection(String type,String openid,String itemid);

    /**
     * 添加新的收藏信息
     */
    @Insert("insert into collection(itemid,openid,type) values(#{itemid},#
{openid},#{itemtype})")
    int insertCollection(Collection collection);

    /**
     * 删除收藏信息
     */
    @Delete("delete from collection where itemid=#{id} and type=#{type} and
openid=#{openid}")
    void deleteCollection(String id,String type,String openid);

}
```

MallMapper.java

```
package com.example.demo.dao;


import com.example.demo.pojo.Mall;
import org.apache.ibatis.annotations.Insert;
import org.apache.ibatis.annotations.Mapper;
import org.apache.ibatis.annotations.Select;

import java.util.List;

@Mapper
public interface MallMapper {
    /**
     * 页面只返回一张图片，名字，id
     */    @Select("select
goodsName,goodsDescribe,goodsId,goodsPicture,goodsPrice from mall limit #
```

```
{first},#{second};")
    List<Mall> listMall(int first, int second);

    /**
     * 页面返回部分信息
     */
    @Select("select goodsName,goodsDescribe,goodsId,goodsPicture,goodsPrice
from mall where goodsType=#{type} limit #{first},#{second};")
    List<Mall> listTypeMall(String type, int first, int second);

    /**
     * 页面返回具体信息
     */
    @Select("select * from mall where goodsId=#{id};")
    Mall getMall(String id);


    /**
     * 页面返回查找的信息（显示于页面上）
     */
    @Select("select goodsName,goodsDescribe,goodsId,goodsPicture,goodsPrice
from mall where goodsName like CONCAT('%',#{find},'%') limit #{first},#
{second};")
    List<Mall> searchMall(String find,int first,int second);

    /**
     * 页面返回查找的信息（显示于页面上）
     */
    @Select("select goodsName,goodsDescribe,goodsId,goodsPicture,goodsPrice
from mall where goodsName like CONCAT('%',#{find},'%');")
    List<Mall> searchSepicMall(String find);


    /**
     * 添加新的商城信息
     */
    @Insert("insert into
mall(goodsName,goodsPlace,goodsDescribe,goodsType,goodsDetails,goodsPicture,g
oodsPrice) values(#{goodsName},#{goodsPlace},#{goodsDescribe},#{goodsType},#
{goodsDetails},#{goodsPicture},#{goodsPrice})")
    int insertMall(Mall mall);


}
```

MuseumMapper.java

```java
package com.example.demo.dao;

import com.example.demo.pojo.Museum;
import org.apache.ibatis.annotations.Mapper;
import org.apache.ibatis.annotations.Insert;
import org.apache.ibatis.annotations.Mapper;
import org.apache.ibatis.annotations.Select;

import java.util.List;

@Mapper
public interface MuseumMapper {
    /**
     * 页面只返回一张图片，名字，id
     */    @Select("select id,picture,title from museum limit #{first},#{second};")
    List<Museum> listMuseum(int first, int second);

    /**
     * 页面返回部分信息
     */
    @Select("select id,picture,title from museum where type=#{type} limit #{first},#{second};")
    List<Museum> listTypeMuseum(String type, int first, int second);


    /**
     * 页面返回具体信息
     */
    @Select("select * from museum where id=#{id};")
    Museum getMuseum(String id);

    /**
     * 插入页面
     * */
    @Insert("insert into museum(id,title,body,picture) values(#{id},#{title},#{body},#{picture})")
    int insertMuseum(Museum museum);

}
```

RecipeMapper.java

```java
package com.example.demo.dao;
```

```java
import com.example.demo.pojo.Recipe;
import org.apache.ibatis.annotations.Insert;
import org.apache.ibatis.annotations.Mapper;
import org.apache.ibatis.annotations.Select;

import java.util.List;

@Mapper
public interface RecipeMapper {


    /**
     * 页面只返回一张图片，名字，id
     */    @Select("select picture,dishes,id,describes from food limit #
{first},#{second};")
    List<Recipe> listRecipe(int first, int second);

    /**
     * 页面返回部分信息
     */
    @Select("select picture,dishes,id,describes from food where type=#{type}
limit #{first},#{second};")
    List<Recipe> listTypeRecipe(String type, int first, int second);

    /**
     * 页面返回具体信息
     */
    @Select("select * from food where id=#{id};")
    Recipe getRecipe(String id);

    /**
     * 页面返回查找的信息（显示于页面上）
     */
    @Select("select picture,dishes,id,describes from food where dishes like
CONCAT('%',#{find},'%') limit #{first},#{second};")
    List<Recipe> searchRecipe(String find,int first,int second);

    /**
     * 添加新的菜谱信息
     */
    @Insert("insert into
food(dishes,regional,culture,efficacy,materials,practice,type,id,picture,desc
ribes) values(#{dishes},#{regional},#{culture},#{efficacy},#{materials},#
{practice},#{type},#{id},#{picture},#{describes})")
    int insertRecipe(Recipe recipe);
```

```
}
```

UserMapper.java

```
package com.example.demo.dao;

import com.example.demo.pojo.User;
import org.apache.ibatis.annotations.Insert;
import org.apache.ibatis.annotations.Mapper;
import org.apache.ibatis.annotations.Select;

@Mapper
public interface UserMapper {
    /**
     * 判断用户是否存在
     */
    @Select("SELECT count(*) FROM users WHERE openid=#{openid}")
    int isUser(String openid);

    /**
     * 添加新的用户信息
     */
    @Insert("insert into users(openid) values(#{openid})")
    int insertUser(User user);
    //@Insert("insert into users(avatarUrl,name,openid) values(#{dishes},#
{avatarUrl},#{name},#{openid})")
    //int insertUser(User user);
}
```

pojo层
Collection.java

```
package com.example.demo.pojo;

import com.fasterxml.jackson.annotation.JsonInclude;
import lombok.Data;

import javax.validation.constraints.NotBlank;
import java.io.Serializable;

/**
 * 收藏页面
 * itemid 物品id
```

```
 * openid 用户id
 * type 类型（菜谱还是商店）
*/
//create table collection(id int not null AUTO_INCREMENT,itemid varchar(50)
not null,openid varchar(50) not null,type varchar(50),primary key(id));
@Data       //使用这个注解可以省去代码中大量的get()、 set()、 toString()等方法;
@JsonInclude(JsonInclude.Include.NON_NULL)   // 忽略返回参时值为null的字段
public class Collection implements Serializable {
    String itemid;
    String openid;
    String itemtype;
    //菜谱
    private String picture;
    private String dishes;
    private String type;
    private String id;

    //商品
    private String goodsPicture;
    private String goodsName;
    private String goodsPlace;
    private String goodsId;
    private String goodsType;
    private String goodsPrice;

}
```

Mall.java

```
package com.example.demo.pojo;

import com.fasterxml.jackson.annotation.JsonInclude;
import lombok.Data;

import java.io.Serializable;

/**
 * 商城实体类
 * 0.图片集
 * 1.商品名字
 * 2.商品产地
 * 3.商品描述
 * 4.商品详情
 * 5.id唯一标识符
 * @author czh
 *///create table mall(goodsId int not null AUTO_INCREMENT,goodsName
```

```
varchar(50) not null,
//goodsPlace varchar(80),goodsDescribe longtext,goodsType varchar(50) not
null,goodsPicture longtext,goodsDetails longtext,primary key (goodsId));

@Data       //使用这个注解可以省去代码中大量的get()、 set()、 toString()等方法;
@JsonInclude(JsonInclude.Include.NON_NULL)   // 忽略返回参时值为null的字段
public class Mall implements Serializable {
    private String goodsPicture;
    private String goodsName;
    private String goodsPlace;
    private String goodsDescribe;
    private String goodsId;
    private String goodsType;
    private String goodsDetails;
    private String goodsPrice;
}
```

Museum.java

```
package com.example.demo.pojo;

import com.fasterxml.jackson.annotation.JsonInclude;
import lombok.Data;

import java.io.Serializable;

/**
 * 图片
 * 标题
 * id
 * 正文
 * 类型
 * */

//create table museum(picture longtext,body longtext,title varchar(50),id int
not null AUTO_INCREMENT,primary key (id));

@Data       //使用这个注解可以省去代码中大量的get()、 set()、 toString()等方法;
@JsonInclude(JsonInclude.Include.NON_NULL)   // 忽略返回参时值为null的字段
public class Museum implements Serializable {
    private String picture;
    private String title;
    private String id;
    private String body;
    private String type;
}
```

Recipe.java

```
package com.example.demo.pojo;

import com.fasterxml.jackson.annotation.JsonInclude;
import lombok.Data;

import javax.validation.constraints.NotBlank;
import java.io.Serializable;
import java.util.List;
import java.util.Map;


//create table food()

//create table food(id int not null AUTO_INCREMENT,dishes varchar(50) not
null,regional varchar(50),culture longtext,efficacy longtext,
// efficacy longtext,materials longtext,practice longtext,type varchar(30)
not null,picture longtext,pirmary key(id));

/**
 * 菜谱实体类
 * 0.图片集
 * 1.菜名
 * 2.所属地域及地域特色
 * 3.菜品文化
 * 4.菜品功效
 * 5.菜品原材料（链接到商城）
 * 6.做法分享
 * 7.类型
 * 8.id唯一
 * 9.菜品描述
 */
@Data       //使用这个注解可以省去代码中大量的get()、 set()、 toString()等方法;
@JsonInclude(JsonInclude.Include.NON_NULL)    // 忽略返回参时值为null的字段
public class Recipe implements Serializable {
    // 如果是字符串类型的数据,使用 @NotBlank 比 @NoNull 更好,因为 @NotBlank 不仅会
校验 null 值,它还会校验空字符串
    private String picture;
    @NotBlank
    private String dishes;
    private String regional;
    private String culture;
    private String efficacy;
    private String materials;
```

```
    private String practice;
    private String practicePicture;
    @NotBlank
    private String type;
    private String id;
    private String describes;
    private List<Mall> mall;
}
```

User.java

```
package com.example.demo.pojo;

import lombok.Data;

import java.io.Serializable;

//create table users(avatarUrl longtext not null,name varchar(50),openid
varchar(70),primary key (openid));

/**
 * 用户实体类
 * avatarUrl 头像链接
 * name 昵称
 * @author czh
 */@Data       //使用这个注解可以省去代码中大量的get()、 set()、 toString()等方法;
public class User implements Serializable {
    //private String avatarUrl;
    //private String name;    private String openid;

    /*public User(String _avatarUrl, String _name, String _openid) {
        avatarUrl = _avatarUrl;        name = _name;        openid = _openid;
}*/
    public User(String _openid) {
        openid = _openid;
    }
}
```

service层
CollectionService.java

```
package com.example.demo.service;

import com.example.demo.pojo.Collection;
```

```
import com.example.demo.pojo.Mall;
import com.example.demo.pojo.Recipe;
import java.util.List;

public interface CollectionService {
    /**
     * 功能:显示商店部分信息
     * @return 返回每张页面的信息
     */
    List<Collection> listCollectionMall(String openid);

    /**
     * 功能:显示菜谱部分信息
     * @return 返回每张页面的信息
     */
    List<Collection> listCollectionRecipe(String openid);

    /**
     * 功能:判断当前是否收藏
     * @return 返回每张页面的信息
     */
    boolean Collection(String id,String type,String openid);

    /**
     * 添加新的收藏信息
     */
    boolean insertCollection(Collection collection);

    /**
     * 删除收藏信息
     */
    boolean deleteCollection(String id,String type,String openid);


}
```

CollectionServiceImpl.java

```
package com.example.demo.service;

import com.example.demo.dao.CollectionMapper;
import com.example.demo.pojo.Collection;
import com.example.demo.pojo.Mall;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
```

```java
import java.util.List;
import java.util.concurrent.TimeUnit;

@Service
public class CollectionServiceImpl implements CollectionService {
    @Autowired
    CollectionMapper collectionMapper;


    /**
     * 功能:显示部分信息
     * @return 返回每张页面的信息
     */
    @Override
    public List<Collection> listCollectionMall(String openid) {
        List<Collection> listMall =
collectionMapper.listCollectionMall(openid);
        //只返回第一张图片
        for( Collection t : listMall ){
            String picture = t.getGoodsPicture();
            int f = picture.indexOf("|");
            if( f != -1 ) {
                t.setGoodsPicture(picture.substring(0, f));
            }
        }
        return listMall;
    }

    /**
     * 功能:显示部分信息
     * @return 返回每张页面的信息
     */
    @Override
    public List<Collection> listCollectionRecipe(String openid) {
        List<Collection> listRecipe =
collectionMapper.listCollectionRecipe(openid);
        //只返回第一张图片
        for( Collection t : listRecipe ){
            String picture = t.getPicture();
            int f = picture.indexOf("|");
            if( f != -1 ) {
                t.setPicture(picture.substring(0, f));
            }
        }
        return listRecipe;
    }

    /**
```

```
     * 添加新的收藏信息
     */
    @Override
    public boolean insertCollection(Collection collection) {
        if(
collectionMapper.Collection(collection.getItemtype(),collection.getOpenid(),c
ollection.getItemid()) <= 0 ) {
            collectionMapper.insertCollection(collection);
            return true;        }else{
            return false;
        }
    }

    /**
     * 删除收藏信息
     */
    @Override
    public boolean deleteCollection(String id,String type,String openid) {
        if( collectionMapper.Collection(type,openid,id) <= 0 ){
            //不存在
            return false;
        }else{
            collectionMapper.deleteCollection(id,type,openid);
            return true;        }
    }

    @Override
    public boolean Collection(String id,String type,String openid){
        if( collectionMapper.Collection(type,openid,id) <= 0 ){
            return false;
        }else{
            return true;
        }
    }
}
```

MallService.java

```
package com.example.demo.service;

import com.example.demo.pojo.Mall;

import java.util.List;

public interface MallService {
    /**
```

```
     * 功能:显示部分信息
     *
     * @param page 页数
     * @return 返回每张页面的信息
     */
    List<Mall> listMall(int page, int limit);

    /**
     * 功能:根据类型显示部分信息
     *
     * @param page 页数
     * @param type 类型
     * @return 返回每张页面的信息
     */
    List<Mall> listTypeMall(int page, int limit, String type);

    /**
     * 显示具体信息
     *
     * @param id 标识符
     * @return 返回页面的具体信息
     */
    Mall getMall(String id);

    /**
     * 插入具体菜谱
     *
     * @param find 查找内容
     * @return 返回插入成功的数组
     */
    List<Mall> searchMall(String find,int page, int limit);

    /**
     * 插入具体菜谱
     *
     * @param recipe 实体类
     * @return 返回插入成功的列
     */
    int insertMall(Mall recipe);


}
```

MallServiceImpl.java

```java
package com.example.demo.service;

import com.example.demo.dao.MallMapper;
import com.example.demo.pojo.Mall;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.redis.core.RedisTemplate;
import org.springframework.stereotype.Service;

import java.util.List;
import java.util.concurrent.TimeUnit;

@Service
public class MallServiceImpl implements MallService {
    @Autowired
    MallMapper mallMapper;
    @Autowired
    private RedisTemplate<Object,Object> redisTemplate;

    @Override
    public List<Mall> listMall(int page, int limit) {

        //为提升系统性能和用户体验
        //首先在Redis缓存中查询，如果有，直接使用；如果没有，去数据库查询并放入redis缓
存
        List<Mall> listMall = (List<Mall>)
redisTemplate.opsForValue().get("listMall_"+page);

        if (null == listMall) {
            //去数据库查询
            //{(page -1) * limit},#{limit};")
            int first = (page - 1) * limit;
            int second = limit;
            listMall = mallMapper.listMall(first, second);
            //只返回第一张图片
            for( Mall t : listMall ){
                String picture = t.getGoodsPicture();
                int f = picture.indexOf("|");
                if( f != -1 ) {
                    t.setGoodsPicture(picture.substring(0, f));
                }
            }
            //并放入redis缓存
            redisTemplate.opsForValue().set("listMall_"+page, listMall, 30,
TimeUnit.SECONDS);
        }
        return listMall;
    }
```

```
    @Override
    public List<Mall> listTypeMall(int page, int limit, String type) {
        //为提升系统性能和用户体验
        //首先在Redis缓存中查询，如果有，直接使用；如果没有，去数据库查询并放入redis缓
存
        List<Mall> listMall = (List<Mall>)
redisTemplate.opsForValue().get("listMall_"+page+"_"+type);

        if (null == listMall) {
            //去数据库查询
            //{(page -1) * limit},#{limit};")
            int first = (page - 1) * limit;
            int second = limit;
            listMall =  mallMapper.listTypeMall(type, first, second);
            //只返回第一张图片
            for( Mall t : listMall ){
                String picture = t.getGoodsPicture();
                int f = picture.indexOf("|");
                if( f != -1 ) {
                    t.setGoodsPicture(picture.substring(0, f));
                }
            }
            //并放入redis缓存
            redisTemplate.opsForValue().set("listMall_"+page+"_"+type,
listMall, 30, TimeUnit.SECONDS);
        }
        return listMall;
    }

    @Override
    public Mall getMall(String id) {

        //为提升系统性能和用户体验
        //首先在Redis缓存中查询，如果有，直接使用；如果没有，去数据库查询并放入redis缓
存
        Mall mall = (Mall) redisTemplate.opsForValue().get("Mall_"+id);
        if (null == mall) {
            //去数据库查询
            mall = mallMapper.getMall(id);
            //并放入redis缓存
            redisTemplate.opsForValue().set("Mall_"+id, mall, 30,
TimeUnit.SECONDS);
        }
        return mall;
    }

    @Override
```

```
    public int insertMall(Mall Mall) {
        return mallMapper.insertMall(Mall);
    }

    @Override
    public List<Mall> searchMall(String find,int page, int limit) {
        //为提升系统性能和用户体验
        //首先在Redis缓存中查询，如果有，直接使用；如果没有，去数据库查询并放入redis缓
存
        List<Mall>  searchMall = (List<Mall>)
redisTemplate.opsForValue().get("searchMall_"+find);

        if (null == searchMall) {
            //{(page -1) * limit},#{limit};")
            int first = (page - 1) * limit;
            int second = limit;
            //去数据库查询
            searchMall = mallMapper.searchMall(find,first,second);
            //只返回第一张图片
            for( Mall t : searchMall ){
                String picture = t.getGoodsPicture();
                int f = picture.indexOf("|");
                if( f != -1 ) {
                    t.setGoodsPicture(picture.substring(0, f));
                }
            }
            //并放入redis缓存
            redisTemplate.opsForValue().set("searchMall_"+find, searchMall,
30, TimeUnit.SECONDS);
        }
        return searchMall;
    }

}
```

MuseumService.java

```
package com.example.demo.service;

import com.example.demo.pojo.Museum;

import java.util.List;

public interface MuseumService {
    /**
     * 功能:显示部分信息
```

```
     *
     * @param page 页数
     * @return 返回每张页面的信息
     */
    List<Museum> listMuseum(int page, int limit);


    /**
     * 功能:根据类型显示部分信息
     * @param type 类型
     * @return 返回每张页面的信息
     */
    List<Museum> listTypeMuseum(String type, int page, int limit);


    /**
     * 显示具体信息
     * @param id 标识符
     * @return 返回页面的具体信息
     */
    Museum getMuseum(String id);


    /**
     * 插入具体博物馆
     *
     * @param museum 实体类
     * @return 返回插入成功的列
     */
    int insertMuseum(Museum museum);


}
```

MuseumServiceImpl.java

```
package com.example.demo.service;

import com.example.demo.dao.MuseumMapper;
import com.example.demo.pojo.Museum;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.redis.core.RedisTemplate;
import org.springframework.stereotype.Service;

import java.util.List;
import java.util.concurrent.TimeUnit;
```

```java
/**
 * @author czh
 */@Service
public class MuseumServiceImpl implements MuseumService {
    @Autowired
    MuseumMapper museumMapper;

    @Autowired
    private RedisTemplate<Object,Object> redisTemplate;

    @Override
    public List<Museum> listMuseum(int page, int limit) {
        //为提升系统性能和用户体验
        //首先在Redis缓存中查询，如果有，直接使用；如果没有，去数据库查询并放入redis缓
存
        List<Museum>  listmuseum = (List<Museum>)
redisTemplate.opsForValue().get("listMuseum_"+page);

        if (null == listmuseum) {
            //去数据库查询
            //{(page -1) * limit},#{limit};")
            int first = (page - 1) * limit;
            int second = limit;
            listmuseum = museumMapper.listMuseum(first, second);

            //只返回第一张图片
            for( Museum t :listmuseum ){
                String picture = t.getPicture();
                int f = picture.indexOf("|");
                if( f != -1 ) {
                    t.setPicture(picture.substring(0, f));
                }
            }
            //并放入redis缓存
            redisTemplate.opsForValue().set("listMuseum_"+page, listmuseum,
30, TimeUnit.SECONDS);
        }

        return listmuseum;
    }

    @Override
    public List<Museum> listTypeMuseum(String type,int page, int limit) {
        //为提升系统性能和用户体验
        //首先在Redis缓存中查询，如果有，直接使用；如果没有，去数据库查询并放入redis缓
存
        List<Museum> listmuseum = (List<Museum>)
```

```
redisTemplate.opsForValue().get("listMuseum_"+page+"_"+type);

        if (null == listmuseum) {
            //去数据库查询
            //{(page -1) * limit},#{limit};")
            int first = (page - 1) * limit;
            int second = limit;
            listmuseum =  museumMapper.listTypeMuseum(type, first, second);
            //只返回第一张图片
            for( Museum t : listmuseum ){
                String picture = t.getPicture();
                int f = picture.indexOf("|");
                if( f != -1 ) {
                    t.setPicture(picture.substring(0, f));
                }
            }
            //并放入redis缓存
            redisTemplate.opsForValue().set("listMuseum_"+page+"_"+type,
listmuseum, 30, TimeUnit.SECONDS);
        }
        return listmuseum;
    }

    @Override
    public Museum getMuseum(String id) {
        //为提升系统性能和用户体验
        //首先在Redis缓存中查询，如果有，直接使用；如果没有，去数据库查询并放入redis缓
存
        Museum  museum = (Museum)
redisTemplate.opsForValue().get("museum_"+id);
        if (null == museum) {
            //去数据库查询
            museum = museumMapper.getMuseum(id);
            //并放入redis缓存
            redisTemplate.opsForValue().set("museum_"+id, museum, 30,
TimeUnit.SECONDS);
        }
        return museum;
    }

    @Override
    public int insertMuseum(Museum museum) {
        return museumMapper.insertMuseum(museum);
    }



}
```

RecipeService.java

```java
package com.example.demo.service;

import com.example.demo.pojo.Recipe;

import java.util.List;

public interface RecipeService {
    /**
     * 功能:显示部分信息
     *
     * @param page 页数
     * @return 返回每张页面的信息
     */
    List<Recipe> listRecipe(int page, int limit);

    /**
     * 功能:根据类型显示部分信息
     *
     * @param page 页数
     * @param type 类型
     * @return 返回每张页面的信息
     */
    List<Recipe> listTypeRecipe(int page, int limit, String type);

    /**
     * 显示具体信息
     * @param id 标识符
     * @return 返回页面的具体信息
     */
    Recipe getRecipe(String id);

    /**
     * 插入具体菜谱
     *
     * @param find 查找内容
     * @return 返回插入成功的数组
     */
    List<Recipe> searchRecipe(String find,int page, int limit);

    /**
     * 插入具体菜谱
     *
     * @param recipe 实体类
```

```
     * @return 返回插入成功的列
     */
    int insertRecipe(Recipe recipe);


}
```

RecipeServiceImpl.java

```
package com.example.demo.service;

import com.example.demo.dao.MallMapper;
import com.example.demo.dao.RecipeMapper;
import com.example.demo.pojo.Mall;
import com.example.demo.pojo.Recipe;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.redis.core.RedisTemplate;
import org.springframework.stereotype.Service;

import java.util.ArrayList;
import java.util.List;
import java.util.concurrent.TimeUnit;

/**
 * @author czh
 */@Service
public class RecipeServiceImpl implements RecipeService {
    @Autowired
    RecipeMapper recipeMapper;

    @Autowired
    MallMapper mallMapper;

    @Autowired
    private RedisTemplate<Object,Object> redisTemplate;

    @Override
    public List<Recipe> listRecipe(int page, int limit) {
        //为提升系统性能和用户体验
        //首先在Redis缓存中查询，如果有，直接使用；如果没有，去数据库查询并放入redis缓
存
        List<Recipe>  listRecipe = (List<Recipe>)
redisTemplate.opsForValue().get("listRecipe_"+page);

        if (null == listRecipe) {
            //去数据库查询
```

```
            //{(page -1) * limit},#{limit};")
            int first = (page - 1) * limit;
            int second = limit;
            listRecipe = recipeMapper.listRecipe(first, second);
            //只返回第一张图片
            for( Recipe t : listRecipe ){
                String picture = t.getPicture();
                int f = picture.indexOf("|");
                if( f != -1 ) {
                    t.setPicture(picture.substring(0, f));
                }
            }
            //并放入redis缓存
            redisTemplate.opsForValue().set("listRecipe_"+page, listRecipe,
30, TimeUnit.SECONDS);
        }

        return listRecipe;
    }

    @Override
    public List<Recipe> listTypeRecipe(int page, int limit, String type) {
        //为提升系统性能和用户体验
        //首先在Redis缓存中查询，如果有，直接使用；如果没有，去数据库查询并放入redis缓
存
        List<Recipe>  listRecipe = (List<Recipe>)
redisTemplate.opsForValue().get("listRecipe_"+page+"_"+type);

        if (null == listRecipe) {
            //去数据库查询
            int first = (page - 1) * limit;
            int second = limit;
            listRecipe = recipeMapper.listTypeRecipe(type, first, second);

            for( Recipe t : listRecipe ){
                //只返回第一张图片
                String picture = t.getPicture();
                int f = picture.indexOf("|");
                if( f != -1 ) {
                    t.setPicture(picture.substring(0, f));
                }
            }


            //并放入redis缓存
            redisTemplate.opsForValue().set("listRecipe_"+page+"_"+type,
listRecipe, 30, TimeUnit.SECONDS);
        }
```

```
        return listRecipe;
    }

    @Override
    public Recipe getRecipe(String id) {
        //为提升系统性能和用户体验
        //首先在Redis缓存中查询，如果有，直接使用；如果没有，去数据库查询并放入redis缓
存
        Recipe  recipe = (Recipe)
redisTemplate.opsForValue().get("Recipe_"+id);
        if (null == recipe) {
            //去数据库查询
            recipe = recipeMapper.getRecipe(id);

            //只返回第一张图片
            String picture = recipe.getPicture();
            int f = picture.indexOf("|");
            if( f != -1 ) {
                recipe.setPicture(picture.substring(0, f));

recipe.setPracticePicture(picture.substring(f+1,picture.length()));
            }

            //分割原材料
            List<Mall> tmp = new ArrayList<>();
            String materials = recipe.getMaterials();
            String[] material = materials.split("|");
            for(int i = 0 ; i < material.length ; i++ ){
                List<Mall> mall = mallMapper.searchSepicMall(material[i]);
                if( mall != null && mall.size() > 0)
                {
                    tmp.add(mall.get(0));
                }
            }
            recipe.setMall(tmp);




            //并放入redis缓存
            redisTemplate.opsForValue().set("Recipe_"+id, recipe, 30,
TimeUnit.SECONDS);

        }
        return recipe;
    }
```

```
    @Override
    public int insertRecipe(Recipe recipe) {
        return recipeMapper.insertRecipe(recipe);
    }

    @Override
    public List<Recipe> searchRecipe(String find, int page, int limit) {
        //为提升系统性能和用户体验
        //首先在Redis缓存中查询，如果有，直接使用；如果没有，去数据库查询并放入redis缓
存
        List<Recipe>  searchRecipe = (List<Recipe>)
redisTemplate.opsForValue().get("searchRecipe_"+find);

        if (null == searchRecipe) {
            //去数据库查询
            int first = (page - 1) * limit;
            int second = limit;
            searchRecipe = recipeMapper.searchRecipe(find,first,second);
            //只返回第一张图片
            for( Recipe t : searchRecipe ){
                String picture = t.getPicture();
                int f = picture.indexOf("|");
                if( f != -1 ) {
                    t.setPicture(picture.substring(0, f));
                }
            }
            //并放入redis缓存
            redisTemplate.opsForValue().set("searchRecipe_"+find,
searchRecipe, 30, TimeUnit.SECONDS);
        }
        return searchRecipe;
    }

}
```

UserService.java

```
package com.example.demo.service;

import com.example.demo.pojo.User;

public interface UserService {
    int isUser(String openid);

    int insertUser(User user);
}
```

UserServiceImpl.java

```
package com.example.demo.service;

import com.example.demo.dao.UserMapper;
import com.example.demo.pojo.User;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

@Service
public class UserServiceImpl implements UserService {
    @Autowired
    UserMapper userMapper;

    @Override
    public int isUser(String openid) {
        return userMapper.isUser(openid);
    }

    @Override
    public int insertUser(User user) {
        return userMapper.insertUser(user);
    }
}
```

utils层
DemoApplication.java

```
package com.example.demo;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.boot.builder.SpringApplicationBuilder;
import
org.springframework.boot.web.servlet.support.SpringBootServletInitializer;

@SpringBootApplication

public class DemoApplication extends SpringBootServletInitializer {
    @Override
    protected SpringApplicationBuilder configure(SpringApplicationBuilder
builder) {
        return builder.sources(DemoApplication.class);
```

```
    }

    public static void main(String[] args) {
        SpringApplication.run(DemoApplication.class, args);
    }

}
```

pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.6.6</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>
    <groupId>com.example</groupId>
    <artifactId>demo</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>demo</name>
    <packaging>war</packaging>
    <description>Demo project for Spring Boot</description>
    <properties>        <java.version>1.8</java.version>
    </properties>
    <dependencies>        <!--导入 spring-boot-starter-web：能够为提供 Web 开发
场景所需要的几乎所有依赖-->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
            <exclusions>                <exclusion>
<groupId>org.springframework.boot</groupId>
                    <artifactId>spring-boot-starter-tomcat</artifactId>
                </exclusion>                <exclusion>
<groupId>org.springframework.boot</groupId>
                    <artifactId>spring-boot-starter-logging</artifactId>
                </exclusion>            </exclusions>        </dependency>
<dependency>            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-tomcat</artifactId>
            <version>2.5.2</version>
            <scope>provided</scope>
        </dependency>        <dependency>
```

```
<groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>        <dependency>
<groupId>org.mybatis.spring.boot</groupId>
            <artifactId>mybatis-spring-boot-starter</artifactId>
            <version>1.3.2</version>
        </dependency>
        <!-- https://mvnrepository.com/artifact/mysql/mysql-connector-java --
>
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <version>8.0.16</version>
        </dependency>
        <dependency>            <groupId>io.springfox</groupId>
            <artifactId>springfox-swagger2</artifactId>
            <version>2.9.2</version>
        </dependency>
        <!-- https://mvnrepository.com/artifact/io.jsonwebtoken/jjwt -->
        <dependency>
            <groupId>io.jsonwebtoken</groupId>
            <artifactId>jjwt</artifactId>
            <version>0.9.1</version>
        </dependency>
        <!--
https://mvnrepository.com/artifact/org.springframework.boot/spring-boot-
starter-validation -->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-validation</artifactId>
            <version>2.6.6</version>
        </dependency>
        <dependency>            <groupId>org.mybatis</groupId>
            <artifactId>mybatis</artifactId>
            <version>3.5.9</version>
        </dependency>
        <!-- https://mvnrepository.com/artifact/com.alibaba/fastjson -->
        <dependency>
            <groupId>com.alibaba</groupId>
            <artifactId>fastjson</artifactId>
            <version>1.2.68</version>
        </dependency>
        <!-- https://mvnrepository.com/artifact/com.google.code.gson/gson -->
        <dependency>
            <groupId>com.google.code.gson</groupId>
            <artifactId>gson</artifactId>
            <version>2.8.5</version>
```

```
        </dependency>        <!-- 引用注解Data -->
        <dependency>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
            <version>1.18.18</version>
            <scope>provided</scope>
        </dependency>        <!-- 添加如下依赖，配置为开发模式，代码做了修改，不用重
新运行 -->
        <!--
https://mvnrepository.com/artifact/org.springframework/springloaded -->
<dependency>
            <groupId>org.springframework</groupId>
            <artifactId>springloaded</artifactId>
            <version>1.2.8.RELEASE</version>
        </dependency>
        <dependency>            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-devtools</artifactId>
        </dependency>
        <!-- springboot集成Redis的依赖 -->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-redis</artifactId>
            <version>2.6.4</version>
        </dependency>
        <dependency>            <groupId>javax.mail</groupId>
            <artifactId>mail</artifactId>
            <version>1.5.0-b01</version>
        </dependency>
        <!-- 接口限流 -->
        <dependency>
            <groupId>com.google.guava</groupId>
            <artifactId>guava</artifactId>
            <version>30.1-jre</version>
        </dependency>


        <!-- https://mvnrepository.com/artifact/org.aspectj/aspectjweaver -->
        <dependency>
            <groupId>org.aspectj</groupId>
            <artifactId>aspectjweaver</artifactId>
            <version>1.9.8</version>
            <scope>runtime</scope>
        </dependency>
        <dependency>            <groupId>org.aspectj</groupId>
            <artifactId>aspectjrt</artifactId>
            <version>1.9.8</version>
        </dependency>
        <!--
```

```
https://mvnrepository.com/artifact/org.springframework.boot/spring-boot-
starter-log4j -->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-log4j</artifactId>
            <version>1.3.8.RELEASE</version>
        </dependency>

    </dependencies>
    <build>        <plugins>            <plugin>
<groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>

            </plugin>            <plugin>
<groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-war-plugin</artifactId>
                <configuration>                    <warName>recipe</warName>
                </configuration>            </plugin>
        </plugins>    </build>
</project>
```

数据库代码(最终版)

```
# 创建收藏数据库
create table collection(id int not null AUTO_INCREMENT,itemid varchar(50) not
null,openid varchar(50) not null,type varchar(50),primary key(id));
# 创建商店数据库
create table mall(goodsId int not null AUTO_INCREMENT,goodsName varchar(50)
not null,goodsPlace varchar(80),goodsDescribe longtext,goodsType varchar(50)
not null,goodsPicture longtext,goodsPrice varchar(50),goodsDetails
longtext,primary key (goodsId));
# 创建博物馆数据库
create table museum(picture longtext,body longtext,title varchar(50),id int
not null AUTO_INCREMENT,type varchar(50) ,primary key (id));
# 创建菜谱数据库
create table food(id int not null AUTO_INCREMENT,dishes varchar(50) not
null,regional longtext,culture longtext,efficacy longtext,materials
longtext,practice longtext,type varchar(30) not null,picture longtext,primary
key(id));
# 创建用户数据库
create table users(openid varchar(70),primary key (openid));
```

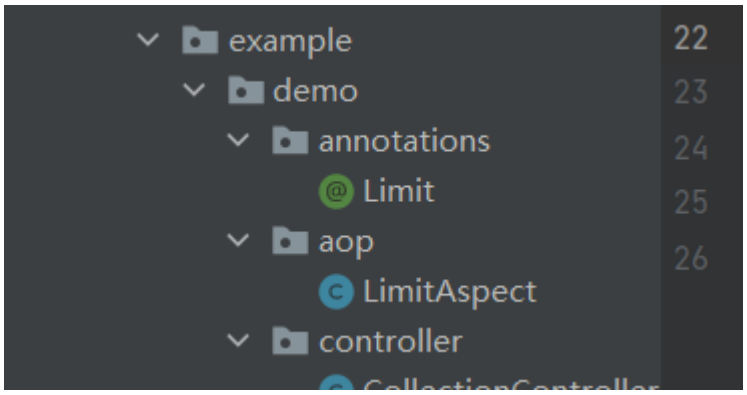
修改数据

```
update mall set goodsPicture=concat('\\files\\mall\\',mall.goodsName,'.png')
    WHERE mall.goodsId > 15 ;

update museum set
picture=concat('\\files\\museum\\',LEFT(museum.title,5),'.png')
    WHERE museum.id> 0 ;

update food set
picture=concat('\\files\\recipe\\',food.dishes,'.png')
    WHERE food.id > 0 ;
```

（2022.5.26）

限流小例子(令牌限流)

项目结构



annotations层

```
package com.example.demo.annotations;

import java.lang.annotation.*;
import java.util.concurrent.TimeUnit;

// 学习网站：https://blog.csdn.net/qq_34217386/article/details/122100904

/**
 * @author czh
 */
@Retention(RetentionPolicy.RUNTIME)
@Target({ElementType.METHOD})
@Documented
public @interface Limit {
```

```
    // 资源key
    String key() default "";

    // 最多访问次数
    double permitsPerSecond();

    // 时间
    long timeout();

    // 时间类型
    TimeUnit timeunit() default TimeUnit.MILLISECONDS;

    // 提示信息
    String msg() default "请求过于频繁,请稍后再试";

}
```

aop层
LimitAspect.java

```
package com.example.demo.aop;

import com.alibaba.fastjson.JSONObject;
import com.example.demo.annotations.Limit;

import com.example.demo.utils.StatusCode;
import com.fasterxml.jackson.core.JsonProcessingException;
import com.fasterxml.jackson.databind.ObjectMapper;
import com.google.common.collect.Maps;
import com.google.common.util.concurrent.RateLimiter;
import lombok.extern.slf4j.Slf4j;
import org.aspectj.lang.ProceedingJoinPoint;
import org.aspectj.lang.annotation.Around;
import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.reflect.MethodSignature;
import org.springframework.stereotype.Component;
import org.springframework.web.context.request.RequestContextHolder;
import org.springframework.web.context.request.ServletRequestAttributes;
import org.springframework.web.util.WebUtils;

import javax.servlet.http.HttpServletResponse;
import java.io.IOException;
import java.io.PrintWriter;
import java.lang.reflect.Method;
import java.util.Map;
```

```
@Slf4j
@Aspect
@Component

public class LimitAspect {
    private final Map<String, RateLimiter> limitMap =
Maps.newConcurrentMap();

    @Around("@annotation(com.example.demo.annotations.Limit)")
    public Object around(ProceedingJoinPoint pjp) throws Throwable {
        MethodSignature signature = (MethodSignature)pjp.getSignature();
        Method method = signature.getMethod();
        //拿limit的注解
        Limit limit = method.getAnnotation(Limit.class);
        if (limit != null) {
            //key作用：不同的接口，不同的流量控制
            String key=limit.key();
            RateLimiter rateLimiter;
            //验证缓存是否有命中key
            if (!limitMap.containsKey(key)) {
                // 创建令牌桶
                rateLimiter = RateLimiter.create(limit.permitsPerSecond());
                limitMap.put(key, rateLimiter);
                log.info("新建了令牌桶={}，容量=
{}",key,limit.permitsPerSecond());
            }
            rateLimiter = limitMap.get(key);
            // 拿令牌
            boolean acquire = rateLimiter.tryAcquire(limit.timeout(),
limit.timeunit());
            // 拿不到命令，直接返回异常提示
            if (!acquire) {
                log.debug("令牌桶={}，获取令牌失败",key);
                //throw new Exception(limit.msg());
                responseFail(limit.msg());
            }
        }
        return pjp.proceed();
    }

    /**
     * 直接向前端抛出异常
     * @param msg 提示信息
     */
    private void responseFail(String msg) throws IOException {
        HttpServletResponse response=((ServletRequestAttributes)
RequestContextHolder.getRequestAttributes()).getResponse();
```

```
        Map<String, Object> map = StatusCode.error(5001,msg);


        response.setCharacterEncoding("UTF-8");
        response.setContentType("application/json; charset=utf-8");
        PrintWriter out = response.getWriter();
        JSONObject json = new JSONObject(map);
        out.append(json.toString());
        out.close();

    }
}
```

controller层

```
   @Limit(key = "cs", permitsPerSecond = 1, timeout = 500, msg = "请求过于频繁,
请稍后再试！")
    @GetMapping("/cs")
    public Map<String, Object> cs() {
        Map<String, Object> map = new HashMap<>();
        map.put("msg", "helloworld");
        return map;
    }
```

pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.6.6</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>
    <groupId>com.example</groupId>
    <artifactId>demo</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>demo</name>
    <packaging>war</packaging>
```

```
    <description>Demo project for Spring Boot</description>
    <properties>
        <java.version>1.8</java.version>
    </properties>
    <dependencies>
        <!--导入 spring-boot-starter-web：能够为提供 Web 开发场景所需要的几乎所有依
赖-->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
            <exclusions>
                <exclusion>
                    <groupId>org.springframework.boot</groupId>
                    <artifactId>spring-boot-starter-tomcat</artifactId>
                </exclusion>
                <exclusion>
                    <groupId>org.springframework.boot</groupId>
                    <artifactId>spring-boot-starter-logging</artifactId>
                </exclusion>
            </exclusions>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-tomcat</artifactId>
            <version>2.5.2</version>
            <scope>provided</scope>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.mybatis.spring.boot</groupId>
            <artifactId>mybatis-spring-boot-starter</artifactId>
            <version>1.3.2</version>
        </dependency>

        <!-- https://mvnrepository.com/artifact/mysql/mysql-connector-java --
>
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <version>8.0.16</version>
        </dependency>

        <dependency>
            <groupId>io.springfox</groupId>
```

```
            <artifactId>springfox-swagger2</artifactId>
            <version>2.9.2</version>
        </dependency>

        <!-- https://mvnrepository.com/artifact/io.jsonwebtoken/jjwt -->
        <dependency>
            <groupId>io.jsonwebtoken</groupId>
            <artifactId>jjwt</artifactId>
            <version>0.9.1</version>
        </dependency>

        <!--
https://mvnrepository.com/artifact/org.springframework.boot/spring-boot-
starter-validation -->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-validation</artifactId>
            <version>2.6.6</version>
        </dependency>

        <dependency>
            <groupId>org.mybatis</groupId>
            <artifactId>mybatis</artifactId>
            <version>3.5.9</version>
        </dependency>

        <!-- https://mvnrepository.com/artifact/com.alibaba/fastjson -->
        <dependency>
            <groupId>com.alibaba</groupId>
            <artifactId>fastjson</artifactId>
            <version>1.2.68</version>
        </dependency>

        <!-- https://mvnrepository.com/artifact/com.google.code.gson/gson -->
        <dependency>
            <groupId>com.google.code.gson</groupId>
            <artifactId>gson</artifactId>
            <version>2.8.5</version>
        </dependency>
        <!-- 引用注解Data -->
        <dependency>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
            <version>1.18.18</version>
            <scope>provided</scope>
        </dependency>
        <!-- 添加如下依赖，配置为开发模式，代码做了修改，不用重新运行 -->
        <!--
```

```
https://mvnrepository.com/artifact/org.springframework/springloaded -->
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>springloaded</artifactId>
            <version>1.2.8.RELEASE</version>
        </dependency>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-devtools</artifactId>
        </dependency>

        <!-- springboot集成Redis的依赖 -->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-redis</artifactId>
            <version>2.6.4</version>
        </dependency>

        <!-- 接口限流 -->
        <dependency>
            <groupId>com.google.guava</groupId>
            <artifactId>guava</artifactId>
            <version>30.1-jre</version>
        </dependency>

        <!-- https://mvnrepository.com/artifact/org.aspectj/aspectjweaver -->
        <dependency>
            <groupId>org.aspectj</groupId>
            <artifactId>aspectjweaver</artifactId>
            <version>1.9.8</version>
            <scope>runtime</scope>
        </dependency>

        <!-- https://mvnrepository.com/artifact/org.aspectj/aspectjrt -->
        <dependency>
            <groupId>org.aspectj</groupId>
            <artifactId>aspectjrt</artifactId>
            <version>1.9.9.1</version>
            <scope>runtime</scope>
        </dependency>
        <dependency>
            <groupId>org.aspectj</groupId>
            <artifactId>aspectjrt</artifactId>
            <version>1.9.8</version>
        </dependency>

        <!--
```

```
https://mvnrepository.com/artifact/org.springframework.boot/spring-boot-
starter-log4j -->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-log4j</artifactId>
            <version>1.3.8.RELEASE</version>
        </dependency>


    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>

            </plugin>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-war-plugin</artifactId>
                <configuration>
                    <warName>recipe</warName>
                </configuration>
            </plugin>

        </plugins>
    </build>

</project>
```

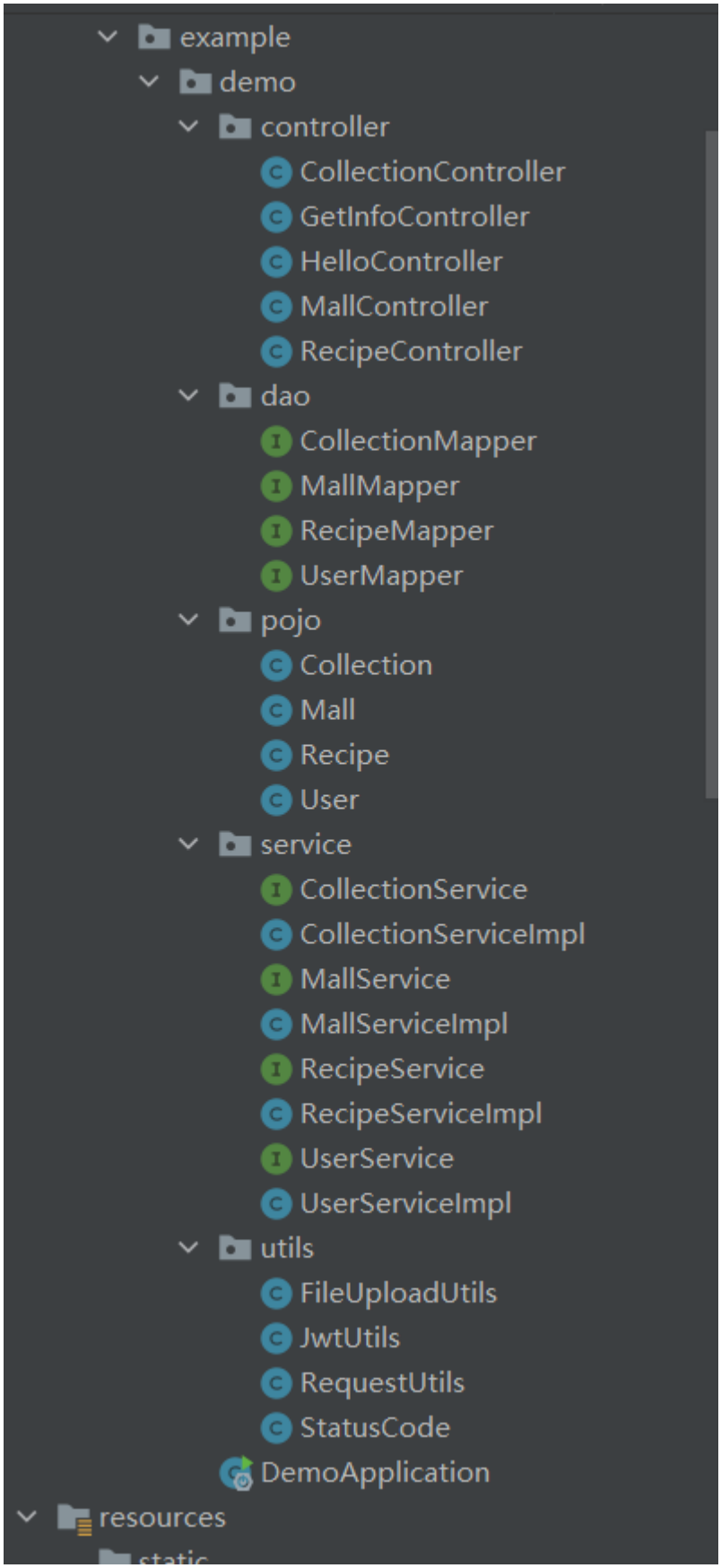
（2022.5.24）

菜谱2.0版本（功能基本完善版）

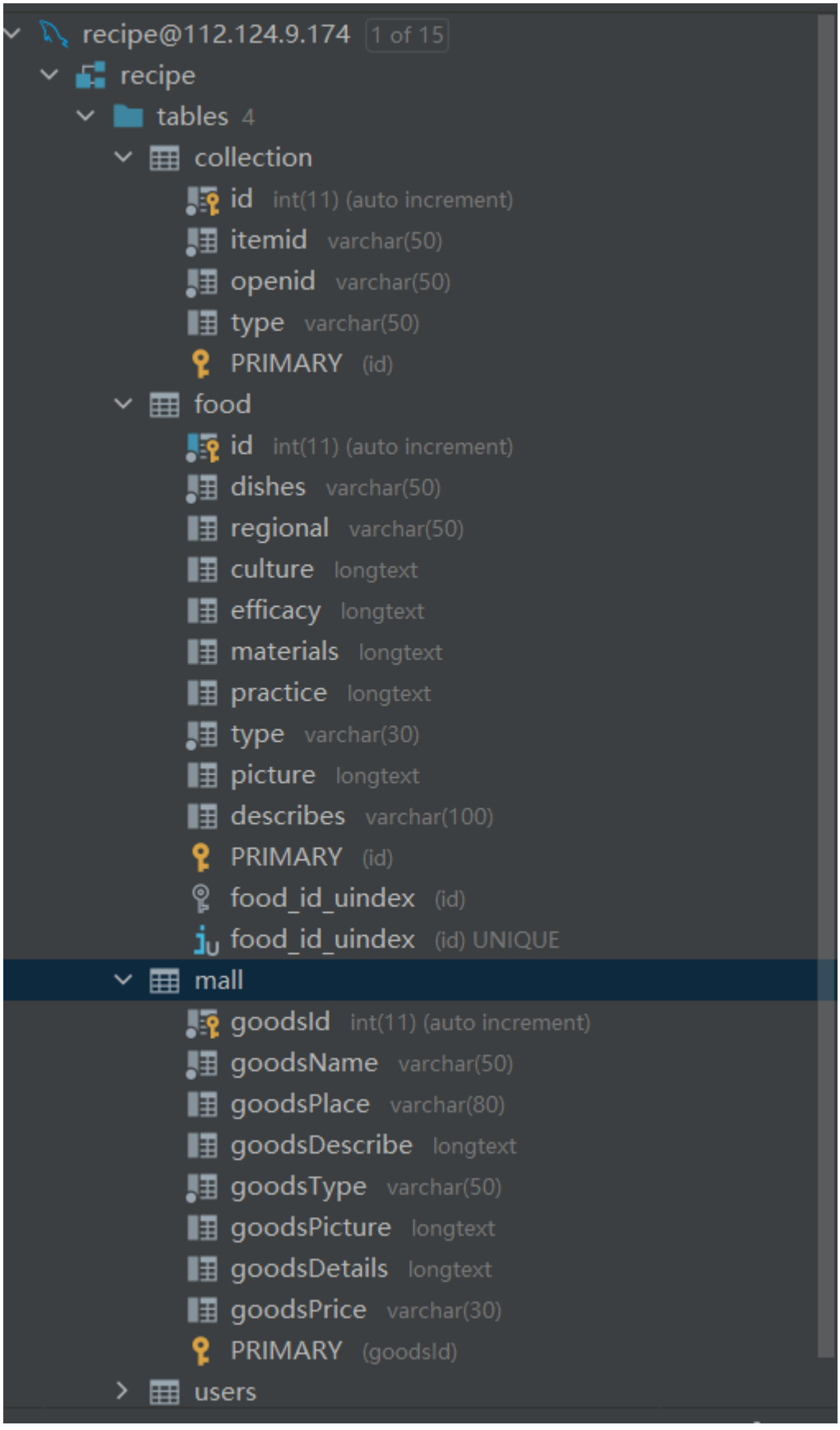
版本特点

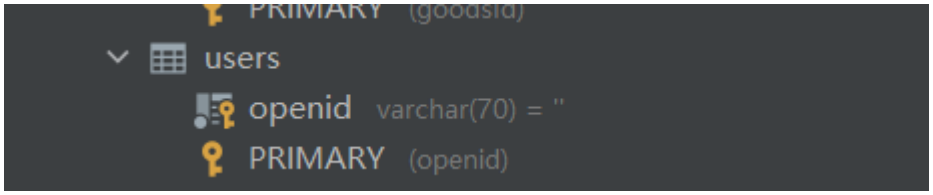
1.新增了收藏功能
2.添加了Redis来进行数据库信息的缓存

项目结构


example
demo
controller
CollectionController
GetInfoController
HelloController
MallController
RecipeController
dao
CollectionMapper
MallMapper
RecipeMapper
UserMapper
pojo
Collection
Mall
Recipe
User
service
CollectionService
CollectionServiceImpl
MallService
MallServiceImpl
RecipeService
RecipeServiceImpl
UserService
UserServiceImpl
utils
FileUploadUtils
JwtUtils
RequestUtils
StatusCode
DemoApplication
resources

数据库结构


recipe@112.124.9.174 1 of 15
recipe
tables 4
collection
id int(11) (auto increment)
itemid varchar(50)
openid varchar(50)
type varchar(50)
PRIMARY (id)
food
id int(11) (auto increment)
dishes varchar(50)
regional varchar(50)
culture longtext
efficacy longtext
materials longtext
practice longtext
type varchar(30)
picture longtext
describes varchar(100)
PRIMARY (id)
food_id_uindex (id)
food_id_uindex (id) UNIQUE
mall
goodsId int(11) (auto increment)
goodsName varchar(50)
goodsPlace varchar(80)
goodsDescribe longtext
goodsType varchar(50)
goodsPicture longtext
goodsDetails longtext
goodsPrice varchar(30)
PRIMARY (goodsId)
users


users
openid varchar(70) = ''
PRIMARY (openid)

代码

controller层
CollectionController.java

```
package com.example.demo.controller;

import com.example.demo.dao.CollectionMapper;
import com.example.demo.pojo.Collection;
import com.example.demo.service.CollectionService;
import com.example.demo.utils.JwtUtils;
import com.example.demo.utils.StatusCode;
import io.jsonwebtoken.Claims;
import io.swagger.annotations.ApiOperation;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;


import javax.validation.Valid;
import java.util.List;
import java.util.Map;

@RestController          //注解可以使结果以Json字符串的形式返回给客户端
@RequestMapping(value = "/api/collection")          //使链接还有一个 /api/
public class CollectionController {
    @Autowired
    CollectionService collectionService;


    @PostMapping("/type")
    @ApiOperation(value = "返回收藏页面")
    public Map<String, Object> listTypeCollection(@RequestParam("type")
String type, @RequestParam("token") String token) { //获取前端传过来的code
        //获取请求时的token
        System.out.println(type);
        JwtUtils jwt = JwtUtils.getInstance();
        Claims claims = jwt.check(token);
        if (claims != null) {
            String openid = (String) claims.get("openid");
            try {
                List<Collection> tmp = null;
                if( "mall".equals(type) ){
                    tmp = collectionService.listCollectionMall(openid);
                    System.out.println(type);
                }else if( "recipe".equals(type) ){
                    tmp = collectionService.listCollectionRecipe(openid);
```

```
                }
                Map<String, Object> data = StatusCode.success(tmp);
                return data;
            } catch (Exception e) {
                e.printStackTrace();
                return StatusCode.error(3001, "服务器内部错误：" +
e.toString());
            }
        }else{
            //非法token
            return StatusCode.error(2001, "用户未登录");
        }
    }


    @PostMapping("/insert")
    @ApiOperation(value = "添加收藏信息")
    public Map<String, Object> insertCollection(@Valid Collection
collection,@RequestParam("token") String token) {

        //获取请求时的token
        JwtUtils jwt = JwtUtils.getInstance();
        Claims claims = jwt.check(token);
        if (claims != null) {
            try {
                String openid = (String) claims.get("openid");
                collection.setOpenid(openid);
                if( collectionService.insertCollection(collection) == true )
{
                    return StatusCode.success("插入成功");
                }else{
                    return StatusCode.success("插入失败，内容已存在");
                }

            } catch (Exception e) {
                e.printStackTrace();
                Map<String, Object> data = StatusCode.error(3001, "服务器内部错
误：" + e.toString());
                return data;
            }
        }else{
            //非法token
            return StatusCode.error(2001, "用户未登录");
        }
    }

    @PostMapping("/delete")
    @ApiOperation(value = "删除收藏信息")
```

```
    public Map<String, Object> deleteCollection(@RequestParam("id") String
id,@RequestParam("type") String type,@RequestParam("token") String token) {

        //获取请求时的token
        JwtUtils jwt = JwtUtils.getInstance();
        Claims claims = jwt.check(token);
        if (claims != null) {
            try {
                String openid = (String) claims.get("openid");
                if( collectionService.deleteCollection(id,type,openid) ==
true){
                    return StatusCode.success("删除成功");
                }else{
                    return StatusCode.success("删除失败，内容不存在");
                }

            } catch (Exception e) {
                e.printStackTrace();
                Map<String, Object> data = StatusCode.error(3001, "服务器内部错
误: " + e.toString());
                return data;
            }
        }else{
            //非法token
            return StatusCode.error(2001, "用户未登录");
        }
    }

    @PostMapping("/exist")
    @ApiOperation(value = "是否存在收藏信息")
    public Map<String, Object> existCollection(@RequestParam("id") String
id,@RequestParam("type") String type,@RequestParam("token") String token) {

        //获取请求时的token
        JwtUtils jwt = JwtUtils.getInstance();
        Claims claims = jwt.check(token);
        if (claims != null) {
            try {
                String openid = (String) claims.get("openid");
                if( collectionService.Collection(id,type,openid) == true){
                    return StatusCode.success("该收藏存在");
                }else{
                    return StatusCode.success("该收藏不存在");
                }

            } catch (Exception e) {
                e.printStackTrace();
                Map<String, Object> data = StatusCode.error(3001, "服务器内部错
```

```
误: " + e.toString());
                return data;
            }
        }else{
            //非法token
            return StatusCode.error(2001, "用户未登录");
        }
    }

}
```

GetInfoController.java

```
package com.example.demo.controller;

//获取信息

import com.alibaba.fastjson.JSONObject;

import com.example.demo.dao.UserMapper;
import com.example.demo.pojo.User;
import com.example.demo.service.UserService;
import com.example.demo.utils.JwtUtils;
import com.example.demo.utils.RequestUtils;

import com.example.demo.utils.StatusCode;
import io.swagger.annotations.ApiOperation;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

import java.util.HashMap;
import java.util.Map;
import java.util.UUID;

@RestController          //注解可以使结果以Json字符串的形式返回给客户端
@RequestMapping(value = "/api/getinfo")          //使链接还有一个 /api/
public class GetInfoController {

    /**
     *  小程序的测试号
     *  appid
     *  secret
     * */
    public static String appid = "wx023e3a4297441859";
    public static String secret = "e7056611f6888dfd25745ff9b9dae882" ;
```

```
    @Autowired
    UserService userService;

    @GetMapping("/cs")
    public Map<String, Object> cs(){
        Map<String, Object> map = new HashMap<>();
        map.put("msg","helloworld");
        return map;
    }

    /**
     * 获取电话号码
     * 请求：POST
     * 链接：地址/api/getinfo/getphone
     * 参数：code
     *  Content-Type: application/json;
     * 返回
     json {
     "msg":"ok"
     }
     * */
    @PostMapping("getphone")
    @ApiOperation(value="获取用户手机号码")
    public Map<String, Object> getPhone(@RequestParam("code") String code){
//获取前端传过来的code

        try{
            System.out.println(code);

            //获取获取小程序全局唯一后台接口调用凭据（access_token）
            String getTokenUrl = "https://api.weixin.qq.com/cgi-bin/token?
grant_type=client_credential&appid="+appid+"&secret="+secret;

            //调用get请求去访问微信小程序自带的链接，将返回结果存储到jsonStringa中
            String jsonStringa =  RequestUtils.doGet(getTokenUrl);

            //String转JSON
            JSONObject jsonObject = JSONObject.parseObject(jsonStringa);

            //获取JSON数据中的access_token
            String access_token = jsonObject.getString("access_token");
            //提交参数
            String getPhoneUrl =
"https://api.weixin.qq.com/wxa/business/getuserphonenumber?
access_token="+access_token;

            Map<String,Object> map =  new HashMap<String,Object>();
```

```
            //将code放到map中
            map.put("code",code);
            //Map格式转化成JSON格式
            JSONObject json = new JSONObject(map);

            //向微信小程序接口提交Post请求得到结果
            String jsonStringb = RequestUtils.doPostForm(getPhoneUrl,json);

            //String转JSON
            JSONObject jsonObject2 = JSONObject.parseObject(jsonStringb);
            HashMap hashMap =
JSONObject.parseObject(jsonObject2.toJSONString(), HashMap.class);
            //请求成功
            if( 0 == (int)hashMap.get("errcode") ){
                Map<String, String> data = new HashMap<>();
                JSONObject tmp2 = (JSONObject)hashMap.get("phone_info");
                data.put("phone",(String)tmp2.get("phoneNumber"));
                //将结果存储下来
                return StatusCode.success(data);
            }else{
                return StatusCode.error((int)hashMap.get("errcode"),"获取失
败");
            }
        }catch (Exception e){
            System.out.println(e);
            return StatusCode.error(3001,"服务器内部错误: "+e.toString());
        }

    }

    //https://blog.csdn.net/qq_41432730/article/details/123617323
    //
    @PostMapping(value = "login")
    @ApiOperation(value="登录")
    public @ResponseBody Map<String,Object> login(@RequestParam("code")
String code/*,@RequestParam("avatarUrl") String
avatarUrl,@RequestParam("avatarUrl") String name*/) {
        Map<String, String> data = new HashMap<String, String>();
        try {
            //Get请求（登录凭证校验）
            String getAuthUrl =
"https://api.weixin.qq.com/sns/jscode2session?appid=" + appid + "&secret=" +
secret + "&js_code=" + code + "&grant_type=authorization_code";
            //进行get请求
            String jsonString = RequestUtils.doGet(getAuthUrl);
            //String转JSON，再json转为map
            JSONObject jsonObject = JSONObject.parseObject(jsonString);
            HashMap hashMap =
```

```
JSONObject.parseObject(jsonObject.toJSONString(), HashMap.class);

            //注意这里要加上 hashMap.get("errcode") == null
            if ( hashMap.get("errcode") == null || 0 == (int)
hashMap.get("errcode")) {                  //请求成功
                //得到openid和session_key去生成3rd_session
                //这个生成3rd_session的方式自己决定即可，比如使用SHA或Base64算法都可
以。例如：将session_key或openid+session_key作为SHA或Base64算法的输入，输出结果做为
3rd_session来使用，同时要将openid，session_key，3rd_session三者关联存储到数据库中，
方便下次拿3rd_session获取session_key或openid做其他处理。
                String openid = (String) hashMap.get("openid");
                String session_key = (String) hashMap.get("session_key");

                //判断是否注册过
                int tmp = userService.isUser(openid);
                if ( tmp == 0 ) {
                    //没有注册过
                    User user = new User(openid);
                    //插入数据库
                    userService.insertUser(user);
                }
                //生成token
                JwtUtils jwt = JwtUtils.getInstance();
                String token = jwt
                        .setClaim("openid",openid)
                        .generateToken();

                Map<String, String> tmp3 = new HashMap<>();
                tmp3.put("token",token);
                //将结果存储下来
                return StatusCode.success(tmp3);

            } else {
                return StatusCode.error((int) hashMap.get("errcode"),"获取失
败");
            }
        }
        catch(Exception e){
            e.printStackTrace();
            return StatusCode.error(3001,"服务器内部错误："+e.toString());
        }

    }
}
```

HelloController.java

```java
package com.example.demo.controller;

import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;

import java.util.HashMap;
import java.util.Map;

@RestController          //注解可以使结果以Json字符串的形式返回给客户端
public class HelloController {
    @GetMapping("/hello")
    public String hello() {
        return "hello SpringBoot";
    }

    @GetMapping("/cs")
    public Map<String, Object> cs() {
        Map<String, Object> map = new HashMap<>();
        map.put("msg", "helloworld");
        return map;
    }

    /*
    @GetMapping("/getappsecret")
    public Map<String, Object> getappsecret() {
        Map<String, Object> map = new HashMap<>();
        map.put("appsecret", "e7056611f6888dfd25745ff9b9dae882");
        return map;
    }*/

}
```

MallController.java

```java
package com.example.demo.controller;

import com.example.demo.pojo.Mall;
import com.example.demo.service.MallService;
import com.example.demo.utils.FileUploadUtils;
import com.example.demo.utils.StatusCode;
import io.swagger.annotations.ApiOperation;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;
import org.springframework.web.multipart.MultipartFile;
```

```java
import javax.servlet.http.HttpServletRequest;
import javax.validation.Valid;
import java.util.List;
import java.util.Map;

@RestController          //注解可以使结果以Json字符串的形式返回给客户端
@RequestMapping(value = "/api/mall")          //使链接还有一个 /api/
public class MallController {
    @Autowired
    MallService mallService;

    @PostMapping("/")
    @ApiOperation(value = "获取全部商品的关键信息")
    public Map<String, Object> listMall(@RequestParam("page") int page) { //
获取前端传过来的code
        try {
            List<Mall> tmp = mallService.listMall(page, 8);
            Map<String, Object> data = StatusCode.success(tmp);
            return data;
        } catch (Exception e) {
            e.printStackTrace();
            return StatusCode.error(3001, "服务器内部错误: " + e.toString());
        }
    }

    @PostMapping("/type")
    @ApiOperation(value = "获取指定类型商品的关键信息")
    public Map<String, Object> listTypeMall(@RequestParam("page") int
page,@RequestParam("type") String type) { //获取前端传过来的code
        try {
            List<Mall> tmp = mallService.listTypeMall(page, 8,type);
            Map<String, Object> data = StatusCode.success(tmp);
            return data;
        } catch (Exception e) {
            e.printStackTrace();
            return StatusCode.error(3001, "服务器内部错误: " + e.toString());
        }
    }

    @PostMapping("/{id}")
    @ApiOperation(value = "获取具体商品信息")
    public Map<String, Object> getMall(@PathVariable(name = "id") String id)
{
        //limit 为8
        try {
            Mall tmp = mallService.getMall(id);
            Map<String, Object> data = StatusCode.success(tmp);
            return data;
```

```
        } catch (Exception e) {
            e.printStackTrace();
            return StatusCode.error(3001, "服务器内部错误: " + e.toString());
        }
    }

    @PostMapping("/insert")
    @ApiOperation(value = "添加具体菜谱信息")
    public Map<String, Object> insertMall(@RequestParam MultipartFile file,
@Valid Mall mall, HttpServletRequest request) {
        try {
            //保存文件
            String filename = FileUploadUtils.SaveServer(file, request);
            //用UUID来生成唯一的id
            //String id = UUID.randomUUID().toString();
            //把实体类中的picture设置成文件路径
            mall.setGoodsPicture(filename);
            //Mall.setId(id);

            mallService.insertMall(mall);
            Map<String, Object> data = StatusCode.success("插入成功");
            return data;
        } catch (Exception e) {
            e.printStackTrace();
            Map<String, Object> data = StatusCode.error(3001, "服务器内部错
误: " + e.toString());
            return data;
        }
    }


    @PostMapping("/search")
    @ApiOperation(value = "搜索商品信息")
    public Map<String, Object> search(@RequestParam("find") String
find,@RequestParam("page") int page) {
        //limit 为8
        try {
            List<Mall> tmp = mallService.searchMall(find,page,8);
            Map<String, Object> data = StatusCode.success(tmp);
            return data;
        } catch (Exception e) {
            e.printStackTrace();
            Map<String, Object> data = StatusCode.error(3001, "服务器内部错
误: " + e.toString());
            return data;
        }
    }
```

```
}
```

RecipeController.java

```
package com.example.demo.controller;

import com.example.demo.pojo.Recipe;
import com.example.demo.service.RecipeService;
import com.example.demo.utils.FileUploadUtils;
import com.example.demo.utils.StatusCode;
import io.swagger.annotations.ApiOperation;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;
import org.springframework.web.multipart.MultipartFile;

import javax.servlet.http.HttpServletRequest;
import javax.validation.Valid;
import java.util.List;
import java.util.Map;

@RestController          //注解可以使结果以Json字符串的形式返回给客户端
@RequestMapping(value = "/api/recipes")          //使链接还有一个 /api/
public class RecipeController {
    @Autowired
    RecipeService recipeService;

    @PostMapping("/")
    @ApiOperation(value = "获取全部菜谱的关键信息")
    public Map<String, Object> listRecipe(@RequestParam("page") int page) {
//获取前端传过来的code
        try {
            List<Recipe> tmp = recipeService.listRecipe(page, 8);
            Map<String, Object> data = StatusCode.success(tmp);
            return data;
        } catch (Exception e) {
            e.printStackTrace();
            Map<String, Object> data = StatusCode.error(3001, "服务器内部错
误: " + e.toString());
            return data;
        }
    }

    @PostMapping("/type")
    @ApiOperation(value = "获取指定类型菜谱的关键信息")
```

```
    public Map<String, Object> listTypeMall(@RequestParam("page") int
page,@RequestParam("type") String type) { //获取前端传过来的code
        try {
            //System.out.println(type);
            List<Recipe> tmp = recipeService.listTypeRecipe(page, 8,type);
            Map<String, Object> data = StatusCode.success(tmp);
            return data;
        } catch (Exception e) {
            e.printStackTrace();
            Map<String, Object> data = StatusCode.error(3001, "服务器内部错
误: " + e.toString());
            return data;
        }
    }

    @PostMapping("/{id}")
    @ApiOperation(value = "获取具体菜谱信息")
    public Map<String, Object> getRecipe(@PathVariable(name = "id") String
id) {
        //limit 为8
        try {
            Recipe tmp = recipeService.getRecipe(id);
            Map<String, Object> data = StatusCode.success(tmp);
            return data;
        } catch (Exception e) {
            e.printStackTrace();
            Map<String, Object> data = StatusCode.error(3001, "服务器内部错
误: " + e.toString());
            return data;
        }
    }

    @PostMapping("/insert")
    @ApiOperation(value = "添加具体菜谱信息")
    public Map<String, Object> insertRecipe(@RequestParam MultipartFile file,
@Valid Recipe recipe, HttpServletRequest request) {
        try {
            //保存文件
            String filename = FileUploadUtils.SaveServer(file, request);
            //用UUID来生成唯一的id
            //String id = UUID.randomUUID().toString();
            //把实体类中的picture设置成文件路径
            recipe.setPicture(filename);
            //recipe.setId(id);

            recipeService.insertRecipe(recipe);
            Map<String, Object> data = StatusCode.success("插入成功");
            return data;
```

```
        } catch (Exception e) {
            e.printStackTrace();
            Map<String, Object> data = StatusCode.error(3001, "服务器内部错
误: " + e.toString());
            return data;
        }
    }


    @PostMapping("/search")
    @ApiOperation(value = "搜索菜谱信息")
    public Map<String, Object> search(@RequestParam("find") String
find,@RequestParam("page") int page) {
        //limit 为8
        try {
            List<Recipe> tmp = recipeService.searchRecipe(find,page,8);
            Map<String, Object> data = StatusCode.success(tmp);
            return data;
        } catch (Exception e) {
            e.printStackTrace();
            Map<String, Object> data = StatusCode.error(3001, "服务器内部错
误: " + e.toString());
            return data;
        }
    }


}
```

dao层
CollectionMapper.java

```
package com.example.demo.dao;

import com.example.demo.pojo.Collection;
import com.example.demo.pojo.Mall;
import org.apache.ibatis.annotations.Delete;
import org.apache.ibatis.annotations.Insert;
import org.apache.ibatis.annotations.Mapper;
import org.apache.ibatis.annotations.Select;

import java.util.List;

//create table collection(id int not null AUTO_INCREMENT,itemid varchar(50)
not null,openid varchar(50) not null,type varchar(50),primary key(id));
@Mapper
```

```
public interface CollectionMapper {
    /**
     * 筛选商店收藏信息
     */
    @Select("select
m.goodsName,m.goodsDescribe,m.goodsId,m.goodsPicture,m.goodsPrice from mall m
inner join collection c on m.goodsId = c.itemid  where c.type='mall' and
c.openid=#{openid} ;")
    List<Collection> listCollectionMall(String openid);


    /**
     * 筛选菜谱收藏信息
     */
    @Select("select f.picture,f.dishes,f.id,f.describes from food f inner
join collection c on f.id = c.itemid  where c.type='recipe' and c.openid=#
{openid} ;")
    List<Collection> listCollectionRecipe(String openid);

    /**
     * 判断是否已存在
     */
    @Select("select count(*) from  collection where type=#{type} and openid=#
{openid} and itemid=#{itemid} ;")
    int Collection(String type,String openid,String itemid);

    /**
     * 添加新的收藏信息
     */
    @Insert("insert into collection(itemid,openid,type) values(#{itemid},#
{openid},#{itemtype})")
    int insertCollection(Collection collection);

    /**
     * 删除收藏信息
     */
    @Delete("delete from collection where itemid=#{id} and type=#{type} and
openid=#{openid}")
    void deleteCollection(String id,String type,String openid);

}
```

MallMapper.java

```
package com.example.demo.dao;
```

```java
import com.example.demo.pojo.Mall;
import org.apache.ibatis.annotations.Insert;
import org.apache.ibatis.annotations.Mapper;
import org.apache.ibatis.annotations.Select;

import java.util.List;

@Mapper
public interface MallMapper {
    /**
     * 页面只返回一张图片，名字，id
     */
    @Select("select goodsName,goodsDescribe,goodsId,goodsPicture,goodsPrice
from mall limit #{first},#{second};")
    List<Mall> listMall(int first, int second);

    /**
     * 页面返回部分信息
     */
    @Select("select goodsName,goodsDescribe,goodsId,goodsPicture,goodsPrice
from mall where goodsType=#{type} limit #{first},#{second};")
    List<Mall> listTypeMall(String type, int first, int second);

    /**
     * 页面返回具体信息
     */
    @Select("select * from mall where goodsId=#{id};")
    Mall getMall(String id);

    /**
     * 页面返回查找的信息（显示于页面上）
     */
    @Select("select goodsName,goodsDescribe,goodsId,goodsPicture,goodsPrice
from mall where goodsName like CONCAT('%',#{find},'%') limit #{first},#
{second};")
    List<Mall> searchMall(String find,int first,int second);

    /**
     * 添加新的商城信息
     */
    @Insert("insert into
mall(goodsName,goodsPlace,goodsDescribe,goodsType,goodsDetails,goodsPicture,g
oodsPrice) values(#{goodsName},#{goodsPlace},#{goodsDescribe},#{goodsType},#
{goodsDetails},#{goodsPicture},#{goodsPrice})")
    int insertMall(Mall mall);
```

```
}
```

RecipeMapper.java

```
package com.example.demo.dao;


import com.example.demo.pojo.Recipe;
import org.apache.ibatis.annotations.Insert;
import org.apache.ibatis.annotations.Mapper;
import org.apache.ibatis.annotations.Select;

import java.util.List;

@Mapper
public interface RecipeMapper {


    /**
     * 页面只返回一张图片，名字，id
     */
    @Select("select picture,dishes,id,describes from food limit #{first},#
{second};")
    List<Recipe> listRecipe(int first, int second);

    /**
     * 页面返回部分信息
     */
    @Select("select picture,dishes,id,describes from food where type=#{type}
limit #{first},#{second};")
    List<Recipe> listTypeRecipe(String type, int first, int second);

    /**
     * 页面返回具体信息
     */
    @Select("select * from food where id=#{id};")
    Recipe getRecipe(String id);

    /**
     * 页面返回查找的信息（显示于页面上）
     */
    @Select("select picture,dishes,id,describes from food where dishes like
CONCAT('%',#{find},'%') limit #{first},#{second};")
    List<Recipe> searchRecipe(String find,int first,int second);
```

```
    /**
     * 添加新的菜谱信息
     */
    @Insert("insert into
food(dishes,regional,culture,efficacy,materials,practice,type,id,picture,desc
ribes) values(#{dishes},#{regional},#{culture},#{efficacy},#{materials},#
{practice},#{type},#{id},#{picture},#{describes})")
    int insertRecipe(Recipe recipe);


}
```

UserMapper.java

```
package com.example.demo.dao;

import com.example.demo.pojo.User;
import org.apache.ibatis.annotations.Insert;
import org.apache.ibatis.annotations.Mapper;
import org.apache.ibatis.annotations.Select;

@Mapper
public interface UserMapper {
    /**
     * 判断用户是否存在
     */
    @Select("SELECT count(*) FROM users WHERE openid=#{openid}")
    int isUser(String openid);

    /**
     * 添加新的用户信息
     */
    @Insert("insert into users(openid) values(#{openid})0")
    int insertUser(User user);
    //@Insert("insert into users(avatarUrl,name,openid) values(#{dishes},#
{avatarUrl},#{name},#{openid})")
    //int insertUser(User user);

}
```

pojo层
Collection.java

```java
package com.example.demo.pojo;

import com.fasterxml.jackson.annotation.JsonInclude;
import lombok.Data;

import javax.validation.constraints.NotBlank;

/**
 * 收藏页面
 * itemid 物品id
 * openid 用户id
 * type 类型（菜谱还是商店）
*/
//create table collection(id int not null AUTO_INCREMENT,itemid varchar(50)
not null,openid varchar(50) not null,type varchar(50),primary key(id));
@Data       //使用这个注解可以省去代码中大量的get()、 set()、 toString()等方法;
@JsonInclude(JsonInclude.Include.NON_NULL)    // 忽略返回参时值为null的字段
public class Collection {
    String itemid;
    String openid;
    String itemtype;
    //菜谱
    private String picture;
    private String dishes;
    private String type;
    private String id;

    //商品
    private String goodsPicture;
    private String goodsName;
    private String goodsPlace;
    private String goodsId;
    private String goodsType;
    private String goodsPrice;

}
```

Mall.java

```java
package com.example.demo.pojo;

import com.fasterxml.jackson.annotation.JsonInclude;
import lombok.Data;
```

```
import java.io.Serializable;

/**
 * 商城实体类
 * 0.图片集
 * 1.商品名字
 * 2.商品产地
 * 3.商品描述
 * 4.商品详情
 * 5.id唯一标识符
 * @author czh
 */
//create table mall(goodsId int not null AUTO_INCREMENT,goodsName varchar(50)
not null,
//goodsPlace varchar(80),goodsDescribe longtext,goodsType varchar(50) not
null,goodsPicture longtext,goodsDetails longtext,primary key (goodsId));

@Data       //使用这个注解可以省去代码中大量的get()、 set()、 toString()等方法;
@JsonInclude(JsonInclude.Include.NON_NULL)   // 忽略返回参时值为null的字段
public class Mall implements Serializable {
    private String goodsPicture;
    private String goodsName;
    private String goodsPlace;
    private String goodsDescribe;
    private String goodsId;
    private String goodsType;
    private String goodsDetails;
    private String goodsPrice;
}
```

Recipe.java

```
package com.example.demo.pojo;

import com.fasterxml.jackson.annotation.JsonInclude;
import lombok.Data;

import javax.validation.constraints.NotBlank;
import java.io.Serializable;


//create table food()

//create table food(id int not null AUTO_INCREMENT,dishes varchar(50) not
null,regional varchar(50),culture longtext,efficacy longtext,
```

```
// efficacy longtext,materials longtext,practice longtext,type varchar(30)
not null,picture longtext,pirmary key(id));

/**
 * 菜谱实体类
 * 0.图片集
 * 1.菜名
 * 2.所属地域及地域特色
 * 3.菜品文化
 * 4.菜品功效
 * 5.菜品原材料（链接到商城）
 * 6.做法分享
 * 7.类型
 * 8.id唯一
 * 9.菜品描述
 */
@Data       //使用这个注解可以省去代码中大量的get()、 set()、 toString()等方法;
@JsonInclude(JsonInclude.Include.NON_NULL)   // 忽略返回参时值为null的字段
public class Recipe implements Serializable {
    // 如果是字符串类型的数据,使用 @NotBlank 比 @NoNull 更好,因为 @NotBlank 不仅会
校验 null 值,它还会校验空字符串
    private String picture;
    @NotBlank
    private String dishes;
    private String regional;
    private String culture;
    private String efficacy;
    private String materials;
    private String practice;
    @NotBlank
    private String type;
    private String id;
    private String describes;
}
```

User.java

```
package com.example.demo.pojo;

import lombok.Data;

//create table users(avatarUrl longtext not null,name varchar(50),openid
varchar(70),primary key (openid));

/**
 * 用户实体类
```

```
 * avatarUrl 头像链接
 * name 昵称
 */
@Data       //使用这个注解可以省去代码中大量的get()、 set()、 toString()等方法;
public class User {
    //private String avatarUrl;
    //private String name;
    private String openid;

    /*public User(String _avatarUrl, String _name, String _openid) {
        avatarUrl = _avatarUrl;
        name = _name;
        openid = _openid;
    }*/

    public User(String _openid) {
        openid = _openid;
    }
}
```

CollecionService.java

```
package com.example.demo.service;

import com.example.demo.pojo.Collection;
import com.example.demo.pojo.Mall;
import com.example.demo.pojo.Recipe;
import java.util.List;

public interface CollectionService {
    /**
     * 功能:显示商店部分信息
     * @return 返回每张页面的信息
     */
    List<Collection> listCollectionMall(String openid);

    /**
     * 功能:显示菜谱部分信息
     * @return 返回每张页面的信息
     */
    List<Collection> listCollectionRecipe(String openid);

    /**
     * 功能:判断当前是否收藏
     * @return 返回每张页面的信息
     */
```

```
    boolean Collection(String id,String type,String openid);

    /**
     * 添加新的收藏信息
     */
    boolean insertCollection(Collection collection);

    /**
     * 删除收藏信息
     */
    boolean deleteCollection(String id,String type,String openid);


}
```

CollectionServiceImpl.java

```
package com.example.demo.service;

import com.example.demo.dao.CollectionMapper;
import com.example.demo.pojo.Collection;
import com.example.demo.pojo.Mall;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.util.List;
import java.util.concurrent.TimeUnit;

@Service
public class CollectionServiceImpl implements CollectionService {
    @Autowired
    CollectionMapper collectionMapper;


    /**
     * 功能:显示部分信息
     * @return 返回每张页面的信息
     */
    @Override
    public List<Collection> listCollectionMall(String openid) {
        List<Collection> listMall =
collectionMapper.listCollectionMall(openid);
        //只返回第一张图片
        for( Collection t : listMall ){
            String picture = t.getGoodsPicture();
            int f = picture.indexOf("|");
```

```
            if( f != -1 ) {
                t.setGoodsPicture(picture.substring(0, f));
            }
        }
        return listMall;
    }

    /**
     * 功能:显示部分信息
     * @return 返回每张页面的信息
     */
    @Override
    public List<Collection> listCollectionRecipe(String openid) {
        return collectionMapper.listCollectionRecipe(openid);
    }

    /**
     * 添加新的收藏信息
     */
    @Override
    public boolean insertCollection(Collection collection) {
        if(
collectionMapper.Collection(collection.getItemtype(),collection.getOpenid(),c
ollection.getItemid()) <= 0 ) {
            collectionMapper.insertCollection(collection);
            return true;
        }else{
            return false;
        }
    }

    /**
     * 删除收藏信息
     */
    @Override
    public boolean deleteCollection(String id,String type,String openid) {
        if( collectionMapper.Collection(type,openid,id) <= 0 ){
            //不存在
            return false;
        }else{
            collectionMapper.deleteCollection(id,type,openid);
            return true;
        }
    }

    @Override
    public boolean Collection(String id,String type,String openid){
        if( collectionMapper.Collection(type,openid,id) <= 0 ){
```

```
            return false;
        }else{
            return true;
        }
    }
}
```

MallService.java

```
package com.example.demo.service;

import com.example.demo.pojo.Mall;

import java.util.List;

public interface MallService {
    /**
     * 功能:显示部分信息
     *
     * @param page 页数
     * @return 返回每张页面的信息
     */
    List<Mall> listMall(int page, int limit);

    /**
     * 功能:根据类型显示部分信息
     *
     * @param page 页数
     * @param type 类型
     * @return 返回每张页面的信息
     */
    List<Mall> listTypeMall(int page, int limit, String type);

    /**
     * 显示具体信息
     *
     * @param id 标识符
     * @return 返回页面的具体信息
     */
    Mall getMall(String id);

    /**
     * 插入具体菜谱
     *
     * @param find 查找内容
     * @return 返回插入成功的数组
```

```
     */
    List<Mall> searchMall(String find,int page, int limit);

    /**
     * 插入具体菜谱
     *
     * @param recipe 实体类
     * @return 返回插入成功的列
     */
    int insertMall(Mall recipe);


}
```

MallServiceImpl.java

```
package com.example.demo.service;

import com.example.demo.dao.MallMapper;
import com.example.demo.pojo.Mall;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.redis.core.RedisTemplate;
import org.springframework.stereotype.Service;

import java.util.List;
import java.util.concurrent.TimeUnit;

@Service
public class MallServiceImpl implements MallService {
    @Autowired
    MallMapper mallMapper;
    @Autowired
    private RedisTemplate<Object,Object> redisTemplate;

    @Override
    public List<Mall> listMall(int page, int limit) {

        //为提升系统性能和用户体验
        //首先在Redis缓存中查询，如果有，直接使用；如果没有，去数据库查询并放入redis缓
存
        List<Mall> listMall = (List<Mall>)
redisTemplate.opsForValue().get("listMall_"+page);

        if (null == listMall) {
            //去数据库查询
            //{(page -1) * limit},#{limit};")
```

```
            int first = (page - 1) * limit;
            int second = limit;
            listMall = mallMapper.listMall(first, second);
            //只返回第一张图片
            for( Mall t : listMall ){
                String picture = t.getGoodsPicture();
                int f = picture.indexOf("|");
                if( f != -1 ) {
                    t.setGoodsPicture(picture.substring(0, f));
                }
            }
            //并放入redis缓存
            redisTemplate.opsForValue().set("listMall_"+page, listMall, 30,
TimeUnit.SECONDS);
        }
        return listMall;
    }

    @Override
    public List<Mall> listTypeMall(int page, int limit, String type) {
        //为提升系统性能和用户体验
        //首先在Redis缓存中查询，如果有，直接使用；如果没有，去数据库查询并放入redis缓
存
        List<Mall> listMall = (List<Mall>)
redisTemplate.opsForValue().get("listMall_"+page+"_"+type);

        if (null == listMall) {
            //去数据库查询
            //{(page -1) * limit},#{limit};")
            int first = (page - 1) * limit;
            int second = limit;
            listMall =  mallMapper.listTypeMall(type, first, second);
            //只返回第一张图片
            for( Mall t : listMall ){
                String picture = t.getGoodsPicture();
                int f = picture.indexOf("|");
                if( f != -1 ) {
                    t.setGoodsPicture(picture.substring(0, f));
                }
            }
            //并放入redis缓存
            redisTemplate.opsForValue().set("listMall_"+page+"_"+type,
listMall, 30, TimeUnit.SECONDS);
        }
        return listMall;
    }

    @Override
```

```
    public Mall getMall(String id) {

        //为提升系统性能和用户体验
        //首先在Redis缓存中查询，如果有，直接使用；如果没有，去数据库查询并放入redis缓
存
        Mall mall = (Mall) redisTemplate.opsForValue().get("Mall_"+id);
        if (null == mall) {
            //去数据库查询
            mall = mallMapper.getMall(id);
            //并放入redis缓存
            redisTemplate.opsForValue().set("Mall_"+id, mall, 30,
TimeUnit.SECONDS);
        }
        return mall;
    }

    @Override
    public int insertMall(Mall Mall) {
        return mallMapper.insertMall(Mall);
    }

    @Override
    public List<Mall> searchMall(String find,int page, int limit) {
        //为提升系统性能和用户体验
        //首先在Redis缓存中查询，如果有，直接使用；如果没有，去数据库查询并放入redis缓
存
        List<Mall>  searchMall = (List<Mall>)
redisTemplate.opsForValue().get("searchMall_"+find);

        if (null == searchMall) {
            //{(page -1) * limit},#{limit};")
            int first = (page - 1) * limit;
            int second = limit;
            //去数据库查询
            searchMall = mallMapper.searchMall(find,first,second);
            //只返回第一张图片
            for( Mall t : searchMall ){
                String picture = t.getGoodsPicture();
                int f = picture.indexOf("|");
                if( f != -1 ) {
                    t.setGoodsPicture(picture.substring(0, f));
                }
            }
            //并放入redis缓存
            redisTemplate.opsForValue().set("searchMall_"+find, searchMall,
30, TimeUnit.SECONDS);
        }
        return searchMall;
```

```
    }

}
```

RecipeService.java

```
package com.example.demo.service;

import com.example.demo.pojo.Recipe;

import java.util.List;

public interface RecipeService {
    /**
     * 功能:显示部分信息
     *
     * @param page 页数
     * @return 返回每张页面的信息
     */
    List<Recipe> listRecipe(int page, int limit);

    /**
     * 功能:根据类型显示部分信息
     *
     * @param page 页数
     * @param type 类型
     * @return 返回每张页面的信息
     */
    List<Recipe> listTypeRecipe(int page, int limit, String type);

    /**
     * 显示具体信息
     * @param id 标识符
     * @return 返回页面的具体信息
     */
    Recipe getRecipe(String id);

    /**
     * 插入具体菜谱
     *
     * @param find 查找内容
     * @return 返回插入成功的数组
     */
    List<Recipe> searchRecipe(String find,int page, int limit);

    /**
```

```
     * 插入具体菜谱
     *
     * @param recipe 实体类
     * @return 返回插入成功的列
     */
    int insertRecipe(Recipe recipe);


}
```

RecipeServiceImpl.java

```
package com.example.demo.service;

import com.example.demo.dao.RecipeMapper;
import com.example.demo.pojo.Recipe;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.redis.core.RedisTemplate;
import org.springframework.stereotype.Service;

import java.util.List;
import java.util.concurrent.TimeUnit;

@Service
public class RecipeServiceImpl implements RecipeService {
    @Autowired
    RecipeMapper recipeMapper;

    @Autowired
    private RedisTemplate<Object,Object> redisTemplate;

    @Override
    public List<Recipe> listRecipe(int page, int limit) {
        //为提升系统性能和用户体验
        //首先在Redis缓存中查询，如果有，直接使用；如果没有，去数据库查询并放入redis缓
存
        List<Recipe>  listRecipe = (List<Recipe>)
redisTemplate.opsForValue().get("listRecipe_"+page);

        if (null == listRecipe) {
            //去数据库查询
            //{(page -1) * limit},#{limit};")
            int first = (page - 1) * limit;
            int second = limit;
            listRecipe = recipeMapper.listRecipe(first, second);
            //并放入redis缓存
```

```
            redisTemplate.opsForValue().set("listRecipe_"+page, listRecipe,
30, TimeUnit.SECONDS);
        }

        return listRecipe;
    }

    @Override
    public List<Recipe> listTypeRecipe(int page, int limit, String type) {
        //为提升系统性能和用户体验
        //首先在Redis缓存中查询，如果有，直接使用；如果没有，去数据库查询并放入redis缓
存
        List<Recipe>  listRecipe = (List<Recipe>)
redisTemplate.opsForValue().get("listRecipe_"+page+"_"+type);

        if (null == listRecipe) {
            //去数据库查询
            int first = (page - 1) * limit;
            int second = limit;
            listRecipe = recipeMapper.listTypeRecipe(type, first, second);
            //并放入redis缓存
            redisTemplate.opsForValue().set("listRecipe_"+page+"_"+type,
listRecipe, 30, TimeUnit.SECONDS);
        }

        return listRecipe;
    }

    @Override
    public Recipe getRecipe(String id) {
        //为提升系统性能和用户体验
        //首先在Redis缓存中查询，如果有，直接使用；如果没有，去数据库查询并放入redis缓
存
        Recipe  recipe = (Recipe)
redisTemplate.opsForValue().get("Recipe_"+id);
        if (null == recipe) {
            //去数据库查询
            recipe = recipeMapper.getRecipe(id);
            //并放入redis缓存
            redisTemplate.opsForValue().set("Recipe_"+id, recipe, 30,
TimeUnit.SECONDS);
        }
        return recipe;
    }

    @Override
    public int insertRecipe(Recipe recipe) {
        return recipeMapper.insertRecipe(recipe);
```

```
    }

    @Override
    public List<Recipe> searchRecipe(String find, int page, int limit) {
        //为提升系统性能和用户体验
        //首先在Redis缓存中查询，如果有，直接使用；如果没有，去数据库查询并放入redis缓
存
        List<Recipe>  searchRecipe = (List<Recipe>)
redisTemplate.opsForValue().get("searchRecipe_"+find);

        if (null == searchRecipe) {
            //去数据库查询
            int first = (page - 1) * limit;
            int second = limit;
            searchRecipe = recipeMapper.searchRecipe(find,first,second);
            //并放入redis缓存
            redisTemplate.opsForValue().set("searchRecipe_"+find,
searchRecipe, 30, TimeUnit.SECONDS);
        }
        return searchRecipe;
    }

}
```

UserService.java

```
package com.example.demo.service;

import com.example.demo.pojo.User;

public interface UserService {
    int isUser(String openid);

    int insertUser(User user);
}
```

UserServiceImpl.java

```
package com.example.demo.service;

import com.example.demo.dao.UserMapper;
import com.example.demo.pojo.User;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
```

```
@Service
public class UserServiceImpl implements UserService {
    @Autowired
    UserMapper userMapper;

    @Override
    public int isUser(String openid) {
        return userMapper.isUser(openid);
    }

    @Override
    public int insertUser(User user) {
        return userMapper.insertUser(user);
    }
}
```

utils层
FileUploadUtils.java

```
package com.example.demo.utils;

import org.springframework.web.multipart.MultipartFile;

import javax.servlet.http.HttpServletRequest;
import java.io.File;
import java.util.UUID;

//文件上传相关的工具包
public class FileUploadUtils {

    // UUID可以避免重命名
    // 传入原始文件名可以得到  uuid+"_"+文件的原始名称 的返回值
    public static String getUuidFileName(String fileName) {
        //文件名以：uuid+"_"+文件的原始名称
        return UUID.randomUUID().toString() + "_" + fileName;
    }

    //保存到服务器，返回存储的位置
    public static String SaveServer(MultipartFile file, HttpServletRequest
request) {
        System.out.println("正在上传文件");
        //得到上传文件的保存目录，将上传的文件存放于WEB-INF目录下，不允许外界直接访问，
保证上传文件的安全
        //String realpath = request.getServletContext().getRealPath("/WEB-
INF/files");
```

```
        //暂时就不保存在WEB-INF那里，先放在files
        //这里保存在根目录中的file文件中
        String realpath = request.getServletContext().getRealPath("files");

        //获取文件名字，进行UUID重命名
        String fileName = getUuidFileName(file.getOriginalFilename());

        //文件上传
        File targetFile = new File(realpath, fileName);

        //如果不存在，创建文件
        if (!targetFile.exists()) {
            targetFile.mkdirs();
        }
        // 上传
        try {
            file.transferTo(targetFile);            //保存下来
            System.out.println("上传成功");
            //return realpath + '\\' + fileName;
            //返回相对路径就行
            return "\\files\\" + fileName;
        } catch (Exception e) {
            e.printStackTrace();
            return null;                            //说明保存不成功
        }
    }

}
```

JwtUtils.java

```
package com.example.demo.utils;

import io.jsonwebtoken.*;

import java.util.Date;

//学习网站：https://www.jianshu.com/p/578a7b2f3e8d

/**
 * jwt工具类
 */
public class JwtUtils {
    /**
     * 实例
```

```
     */
    private static JwtUtils instance;

    /**
     * 发行者
     */
    private String subObject = "wechatbyorall";

    /**
     * 过期时间，默认1天
     */
    private long expired = 1000 * 60 * 60 * 24 * 1;

    /**
     * jwt构造
     */
    private static JwtBuilder jwtBuilder;

    /**
     * 密钥
     */
    private String secret = "UYRL@sdjow114da5#KXKRWF$1124gki";// 密钥

    /**
     * 获取实例
     *
     * @return
     */
    public static JwtUtils getInstance() {
        if (instance == null) {
            instance = new JwtUtils();
        }
        jwtBuilder = Jwts.builder();
        return instance;
    }

    /**
     * 荷载信息(通常是一个User信息，还包括一些其他的元数据)
     *
     * @param key
     * @param val
     * @return
     */
    public JwtUtils setClaim(String key, Object val) {
        jwtBuilder.claim(key, val);
        return this;
    }
```

```
    /**
     * 生成 jwt token
     *
     * @return
     */
    public String generateToken() {
        String token = jwtBuilder
                .setSubject(subObject) // 发行者
                //.claim("id","121") // 参数
                .setIssuedAt(new Date()) // 发行时间
                .setExpiration(new Date(System.currentTimeMillis() +
expired))
                .signWith(SignatureAlgorithm.HS256, secret) // 签名类型 与 密钥
                .compressWith(CompressionCodecs.DEFLATE)// 对载荷进行压缩
                .compact(); // 压缩一下
        return token;
    }

    /**
     * 解析 token
     *
     * @param token
     * @return
     */
    public Claims check(String token) {
        try {
            final Claims claims = Jwts.parser()
                    .setSigningKey(secret)
                    .parseClaimsJws(token)
                    .getBody();
            return claims;
        } catch (Exception e) {
            System.out.println(e);
        }
        return null;
    }

    public String getSubObject() {
        return subObject;
    }

    /**
     * 设置发行者
     *
     * @param subObject
     * @return
     */
    public JwtUtils setSubObject(String subObject) {
```

```
        this.subObject = subObject;
        return this;
    }

    public long getExpired() {
        return expired;
    }

    /**
     * 设置过期时间
     *
     * @param expired
     * @return
     */
    public JwtUtils setExpired(long expired) {
        this.expired = expired;
        return this;
    }

    public String getSecret() {
        return secret;
    }

    /**
     * 设置密钥
     *
     * @param secret
     * @return
     */
    public JwtUtils setSecret(String secret) {
        this.secret = secret;
        return this;
    }
}
```

RequestUtils.java

```
package com.example.demo.utils;

import com.alibaba.fastjson.JSONObject;
import com.google.gson.Gson;

import java.io.*;
import java.net.HttpURLConnection;
import java.net.MalformedURLException;
```

```java
import java.net.URL;
import java.util.*;

//https://www.cnblogs.com/mufengforward/p/10510337.html

public class RequestUtils {
    /**
     * 向指定URL发送GET方法的请求
     *
     * @param httpurl 请求参数用?拼接在url后边，请求参数应该是
name1=value1&name2=value2 的形式。
     * @return result 所代表远程资源的响应结果
     */
    //发送get请求
    public static String doGet(String httpurl) {
        HttpURLConnection connection = null;
        InputStream is = null;
        BufferedReader br = null;
        // 返回结果字符串
        String result = null;
        try {
            // 创建远程url连接对象
            URL url = new URL(httpurl);
            // 通过远程url连接对象打开一个连接，强转成httpURLConnection类
            connection = (HttpURLConnection) url.openConnection();
            // 设置连接方式：get
            connection.setRequestMethod("GET");
            // 设置连接主机服务器的超时时间：15000毫秒
            connection.setConnectTimeout(15000);
            // 设置读取远程返回的数据时间：60000毫秒
            connection.setReadTimeout(60000);
            // 发送请求
            connection.connect();
            // 通过connection连接，获取输入流
            if (connection.getResponseCode() == 200) {
                is = connection.getInputStream();
                // 封装输入流is，并指定字符集
                br = new BufferedReader(new InputStreamReader(is, "UTF-8"));
                // 存放数据
                StringBuffer sbf = new StringBuffer();
                String temp = null;
                while ((temp = br.readLine()) != null) {
                    sbf.append(temp);
                    sbf.append("\r\n");
                }
                result = sbf.toString();
            }
        } catch (MalformedURLException e) {
```

```
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            // 关闭资源
            if (null != br) {
                try {
                    br.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }

            if (null != is) {
                try {
                    is.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }

            connection.disconnect();// 关闭远程连接
        }

        return result;
    }

    /**
     * @param httpUrl 请求的url
     * @param param   form表单的参数（key,value形式）
     * @return
     */
    //发送post请求，注意请求微信小程序接口的格式是"Content-Type":"application/json"
    public static String doPostForm(String httpUrl, Map param) {

        HttpURLConnection connection = null;
        InputStream is = null;
        OutputStream os = null;
        BufferedReader br = null;
        String result = null;
        try {
            URL url = new URL(httpUrl);
            // 通过远程url连接对象打开连接
            connection = (HttpURLConnection) url.openConnection();
            // 设置连接请求方式
            connection.setRequestMethod("POST");
            // 设置连接主机服务器超时时间：15000毫秒
            connection.setConnectTimeout(15000);
```

```
            // 设置读取主机服务器返回数据超时时间：60000毫秒
            connection.setReadTimeout(60000);

            // 默认值为：false，当向远程服务器传送数据/写数据时，需要设置为true
            connection.setDoOutput(true);
            // 默认值为：true，当前向远程服务读取数据时，设置为true，该参数可有可无
            connection.setDoInput(true);
            // 设置传入参数的格式:请求参数应该是 name1=value1&name2=value2 的形式。
            connection.setRequestProperty("Content-Type",
"application/json");
            // 设置鉴权信息：Authorization: Bearer da3efcbf-0845-4fe3-8aba-
ee040be542c0
            //connection.setRequestProperty("Authorization", "Bearer
da3efcbf-0845-4fe3-8aba-ee040be542c0");
            // 通过连接对象获取一个输出流
            os = connection.getOutputStream();
            // 通过输出流对象将参数写出去/传输出去,它是通过字节数组写出的(form表单形式
的参数实质也是key,value值的拼接，类似于get请求参数的拼接)
            JSONObject json = new JSONObject(param);
            os.write(createLinkString(param).getBytes());
            // 通过连接对象获取一个输入流，向远程读取
            if (connection.getResponseCode() == 200) {

                is = connection.getInputStream();
                // 对输入流对象进行包装:charset根据工作项目组的要求来设置
                br = new BufferedReader(new InputStreamReader(is, "UTF-8"));

                StringBuffer sbf = new StringBuffer();
                String temp = null;
                // 循环遍历一行一行读取数据
                while ((temp = br.readLine()) != null) {
                    sbf.append(temp);
                    sbf.append("\r\n");
                }
                result = sbf.toString();
            }
        } catch (MalformedURLException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            // 关闭资源
            if (null != br) {
                try {
                    br.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
```

```
            }
            if (null != os) {
                try {
                    os.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
            if (null != is) {
                try {
                    is.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
            // 断开与远程地址url的连接
            connection.disconnect();
        }
        return result;
    }

    //发送"application/json"格式的POST请求
    public static String doPostForm(String httpUrl, JSONObject param) {

        HttpURLConnection connection = null;
        InputStream is = null;
        OutputStream os = null;
        BufferedReader br = null;
        String result = null;
        try {
            URL url = new URL(httpUrl);
            // 通过远程url连接对象打开连接
            connection = (HttpURLConnection) url.openConnection();
            // 设置连接请求方式
            connection.setRequestMethod("POST");
            // 设置连接主机服务器超时时间：15000毫秒
            connection.setConnectTimeout(15000);
            // 设置读取主机服务器返回数据超时时间：60000毫秒
            connection.setReadTimeout(60000);

            // 默认值为：false，当向远程服务器传送数据/写数据时，需要设置为true
            connection.setDoOutput(true);
            // 默认值为：true，当前向远程服务读取数据时，设置为true，该参数可有可无
            connection.setDoInput(true);
            // 设置传入参数的格式:请求参数应该是 name1=value1&name2=value2 的形式。
            connection.setRequestProperty("Content-Type",
"application/json");
            // 设置鉴权信息：Authorization: Bearer da3efcbf-0845-4fe3-8aba-
```

```
ee040be542c0
            //connection.setRequestProperty("Authorization", "Bearer
da3efcbf-0845-4fe3-8aba-ee040be542c0");
            // 通过连接对象获取一个输出流
            os = connection.getOutputStream();
            // 通过输出流对象将参数写出去/传输出去,它是通过字节数组写出的(form表单形式
的参数实质也是key,value值的拼接，类似于get请求参数的拼接)

            os.write(param.toString().getBytes());
            // 通过连接对象获取一个输入流，向远程读取
            if (connection.getResponseCode() == 200) {

                is = connection.getInputStream();
                // 对输入流对象进行包装:charset根据工作项目组的要求来设置
                br = new BufferedReader(new InputStreamReader(is, "UTF-8"));

                StringBuffer sbf = new StringBuffer();
                String temp = null;
                // 循环遍历一行一行读取数据
                while ((temp = br.readLine()) != null) {
                    sbf.append(temp);
                    sbf.append("\r\n");
                }
                result = sbf.toString();
            }
        } catch (MalformedURLException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            // 关闭资源
            if (null != br) {
                try {
                    br.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
            if (null != os) {
                try {
                    os.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
            if (null != is) {
                try {
                    is.close();
```

```
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
            // 断开与远程地址url的连接
            connection.disconnect();
        }
        return result;
    }


    /**
     * 把数组所有元素排序，并按照“参数=参数值”的模式用“&”字符拼接成字符串
     *
     * @param params 需要排序并参与字符拼接的参数组
     * @return 拼接后字符串
     */
    public static String createLinkString(Map<String, String> params) {

        List<String> keys = new ArrayList<String>(params.keySet());
        Collections.sort(keys);

        StringBuilder prestr = new StringBuilder();
        for (int i = 0; i < keys.size(); i++) {
            String key = keys.get(i);
            String value = params.get(key);
            if (i == keys.size() - 1) {// 拼接时，不包括最后一个&字符
                prestr.append(key).append("=").append(value);
            } else {
                prestr.append(key).append("=").append(value).append("&");
            }
        }

        return prestr.toString();
    }

    //String 转 Map
    public static Map<String, Object> strToJson(String jsonString) {
        Gson gson = new Gson();
        Map<String, Object> map = new HashMap<String, Object>();
        map = gson.fromJson(jsonString, map.getClass());
        return map;
    }


}
```

StatusCode.java

```java
package com.example.demo.utils;

import lombok.Data;

import java.util.HashMap;
import java.util.Map;

/**
 * status 状态码 0-表示失败，1-表示成功
 * error_code 错误码，一般在设计时定义
 * error_des 错误描述，一般在设计时定义
 * data 成功数据
 * msg 请求成功 / 请求失败
 *
 * @author czh
 */
@Data
public class StatusCode {
    /**
     * int status;
     * String error_code;
     * String error_des;
     * * Object data;
     */
    public static Map<String, Object> success(Object data) {
        Map<String, Object> tmp = new HashMap<>();
        tmp.put("status", 1);
        tmp.put("data", data);
        tmp.put("msg", "请求成功");
        return tmp;
    }

    public static Map<String, Object> error(int error_code) {
        Map<String, Object> tmp = new HashMap<>();
        tmp.put("status", 0);
        tmp.put("msg", "请求失败");
        tmp.put("error_code", error_code);
        String error_des = "";
        // 以下待完善
        if (error_code == 1001) {
            error_des = "参数无效";
        } else if (error_code == 1002) {
            error_des = "参数缺失";
        } else if (error_code == 2001) {
            error_des = "用户未登录";
```

```
        } else if (error_code == 3001) {
            error_des = "服务器错误";
        }
        tmp.put("error_des", error_des);
        return tmp;
    }

    public static Map<String, Object> error(int error_code, String error_des)
{
        Map<String, Object> tmp = new HashMap<>();
        tmp.put("status", 0);
        tmp.put("msg", "请求失败");
        tmp.put("error_code", error_code);
        tmp.put("error_des", error_des);
        return tmp;
    }

}
```

DemoApplication.java

```
package com.example.demo;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.boot.builder.SpringApplicationBuilder;
import
org.springframework.boot.web.servlet.support.SpringBootServletInitializer;

@SpringBootApplication

public class DemoApplication extends SpringBootServletInitializer {
    @Override
    protected SpringApplicationBuilder configure(SpringApplicationBuilder
builder) {
        return builder.sources(DemoApplication.class);
    }

    public static void main(String[] args) {
        SpringApplication.run(DemoApplication.class, args);
    }

}
```

pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.6.6</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>
    <groupId>com.example</groupId>
    <artifactId>demo</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>demo</name>
    <packaging>war</packaging>
    <description>Demo project for Spring Boot</description>
    <properties>
        <java.version>1.8</java.version>
    </properties>
    <dependencies>
        <!--导入 spring-boot-starter-web：能够为提供 Web 开发场景所需要的几乎所有依
赖-->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
            <exclusions>
                <exclusion>
                    <groupId>org.springframework.boot</groupId>
                    <artifactId>spring-boot-starter-tomcat</artifactId>
                </exclusion>
            </exclusions>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-tomcat</artifactId>
            <version>2.5.2</version>
            <scope>provided</scope>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
```

```
        <dependency>
            <groupId>org.mybatis.spring.boot</groupId>
            <artifactId>mybatis-spring-boot-starter</artifactId>
            <version>1.3.2</version>
        </dependency>

        <!-- https://mvnrepository.com/artifact/mysql/mysql-connector-java --
>
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <version>8.0.16</version>
        </dependency>

        <dependency>
            <groupId>io.springfox</groupId>
            <artifactId>springfox-swagger2</artifactId>
            <version>2.9.2</version>
        </dependency>

        <!-- https://mvnrepository.com/artifact/io.jsonwebtoken/jjwt -->
        <dependency>
            <groupId>io.jsonwebtoken</groupId>
            <artifactId>jjwt</artifactId>
            <version>0.9.1</version>
        </dependency>

        <!--
https://mvnrepository.com/artifact/org.springframework.boot/spring-boot-
starter-validation -->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-validation</artifactId>
            <version>2.6.6</version>
        </dependency>

        <dependency>
            <groupId>org.mybatis</groupId>
            <artifactId>mybatis</artifactId>
            <version>3.5.9</version>
        </dependency>

        <!-- https://mvnrepository.com/artifact/com.alibaba/fastjson -->
        <dependency>
            <groupId>com.alibaba</groupId>
            <artifactId>fastjson</artifactId>
            <version>1.2.68</version>
        </dependency>
```

```
        <!-- https://mvnrepository.com/artifact/com.google.code.gson/gson -->
        <dependency>
            <groupId>com.google.code.gson</groupId>
            <artifactId>gson</artifactId>
            <version>2.8.5</version>
        </dependency>
        <!-- 引用注解Data -->
        <dependency>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
            <version>1.18.18</version>
            <scope>provided</scope>
        </dependency>
        <!-- 添加如下依赖，配置为开发模式，代码做了修改，不用重新运行-->
        <!--
https://mvnrepository.com/artifact/org.springframework/springloaded -->
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>springloaded</artifactId>
            <version>1.2.8.RELEASE</version>
        </dependency>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-devtools</artifactId>
        </dependency>
        <!--        springboot集成Redis的依赖-->

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-redis</artifactId>
            <version>2.6.4</version>
        </dependency>

    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>

            </plugin>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-war-plugin</artifactId>
                <configuration>
```

```
                    <warName>recipe</warName>
                </configuration>
            </plugin>

        </plugins>
    </build>

</project>
```

application.properties

```
# 数据库
spring.datasource.url=jdbc:mysql://****:3306/****
spring.datasource.username=****
spring.datasource.password=****
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

# 上传文件
spring.servlet.multipart.max-file-size= 50MB
spring.servlet.multipart.max-request-size= 50MB

# 设置Redis配置
spring.redis.host=****
spring.redis.port=6379
spring.redis.password=****
```

（2022.5.23）

redis缓存小例子

MallServiceImpl.java

```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.redis.core.RedisTemplate;

 @Autowired
    private RedisTemplate<Object,Object> redisTemplate;

  @Override
    public List<Mall> listMall(int page, int limit) {

        //为提升系统性能和用户体验
```

```
        //首先在Redis缓存中查询，如果有，直接使用；如果没有，去数据库查询并放入redis缓
存
        List<Mall> listMall = (List<Mall>)
redisTemplate.opsForValue().get("listMall_"+page);

        if (null == listMall) {
            //去数据库查询
            //{(page -1) * limit},#{limit};")
            int first = (page - 1) * limit;
            int second = limit;
            listMall = mallMapper.listMall(first, second);
            //只返回第一张图片
            for( Mall t : listMall ){
                String picture = t.getGoodsPicture();
                int f = picture.indexOf("|");
                if( f != -1 ) {
                    t.setGoodsPicture(picture.substring(0, f));
                }
            }
            //并放入redis缓存
            redisTemplate.opsForValue().set("listMall_"+page, listMall, 30,
TimeUnit.SECONDS);
        }
        return listMall;
    }
```

pom.xml

```
    <!-- springboot集成Redis的依赖 -->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-redis</artifactId>
            <version>2.6.4</version>
        </dependency>
```

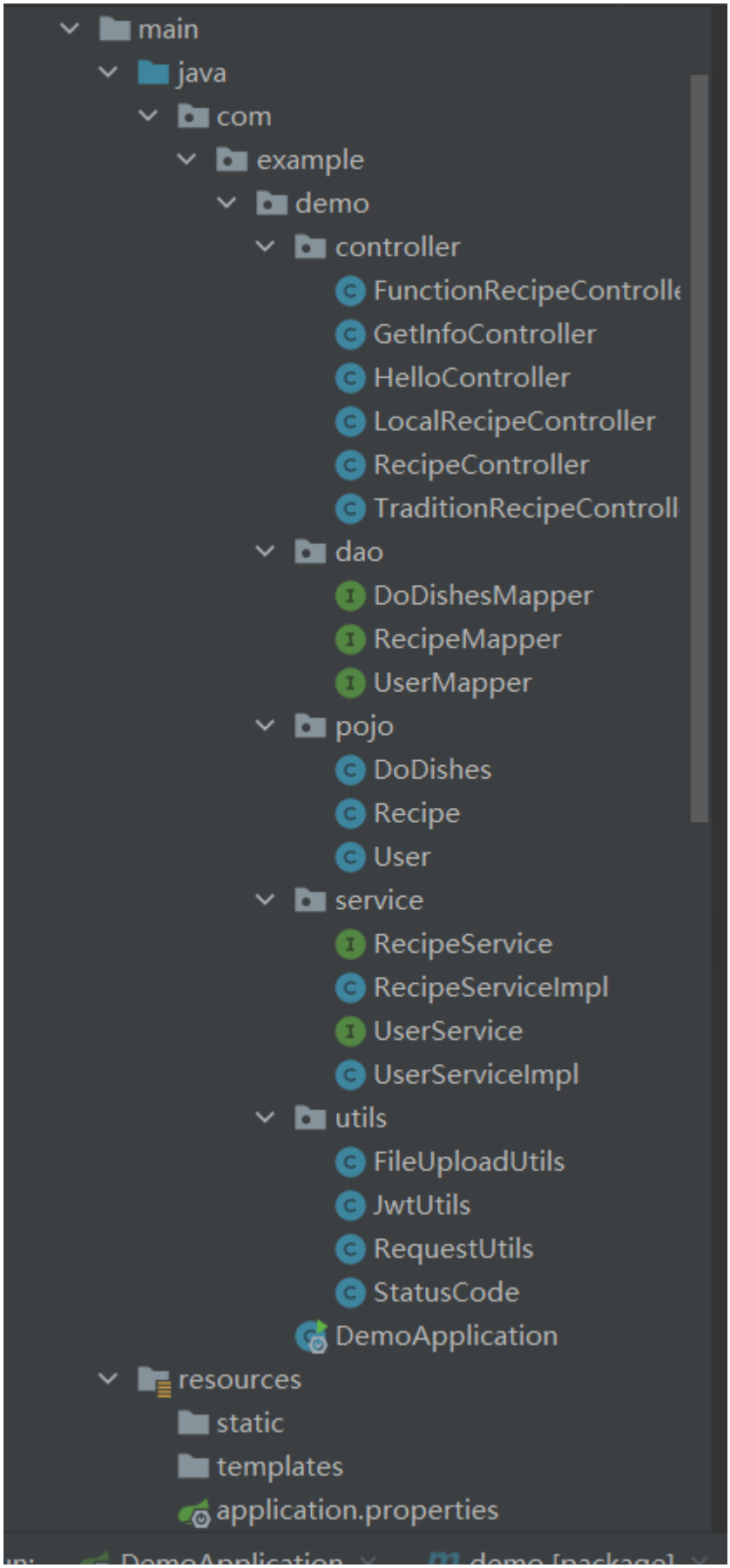
（2022.5.17）

```
create table mall(goodsId int not null AUTO_INCREMENT,goodsName varchar(50)
not null,
goodsPlace varchar(80),goodsDescribe longtext,goodsType varchar(50) not
null,goodsPicture longtext,goodsDetails longtext,primary key (goodsId));
```

(2022.5.17)

菜谱端基本接口1.0

项目结构


main
java
com
example
demo
controller
FunctionRecipeControlle
GetInfoController
HelloController
LocalRecipeController
RecipeController
TraditionRecipeControll
dao
DoDishesMapper
RecipeMapper
UserMapper
pojo
DoDishes
Recipe
User
service
RecipeService
RecipeServiceImpl
UserService
UserServiceImpl
utils
FileUploadUtils
JwtUtils
RequestUtils
StatusCode
DemoApplication
resources
static
templates
application.properties

代码

controller层
FuctionRecipeController.java

```java
package com.example.demo.controller;

import com.example.demo.pojo.Recipe;
import com.example.demo.service.RecipeService;
import com.example.demo.utils.StatusCode;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

import java.util.List;
import java.util.Map;


/**
 * 功能菜谱
 * */
@RestController          //注解可以使结果以Json字符串的形式返回给客户
@RequestMapping(value = "/api/recipes/function/")          //使链接还有一个
/api/
public class FunctionRecipeController {          //功能菜谱
    @Autowired
    RecipeService recipeService;

    @PostMapping("/")
    public Map<String, Object> listRecipe(@RequestParam("page") int page) {
//获取前端传过来的code
        try{
            List<Recipe> tmp = recipeService.listTypeRecipe(page,8,"功能菜
谱");
            Map<String, Object> data = StatusCode.success(tmp);
            return data;
        }catch (Exception e){
            System.out.println(e);
            Map<String, Object> data = StatusCode.error(3001);
            return data;
        }
    }

    @PostMapping("/{id}")
    public Map<String, Object> getRecipe(@PathVariable(name="id") String id)
{ //获取前端传过来的code
        //limit 为8
        try{
            Recipe tmp = recipeService.getRecipe(id);
```

```
            Map<String, Object> data = StatusCode.success(tmp);
            return data;
        }catch (Exception e){
            Map<String, Object> data = StatusCode.error(3001);
            return data;
        }
    }
}
```

GetInfoController.java

```
package com.example.demo.controller;

//获取信息

import com.alibaba.fastjson.JSONObject;

import com.example.demo.dao.UserMapper;
import com.example.demo.pojo.User;
import com.example.demo.service.UserService;
import com.example.demo.utils.JwtUtils;
import com.example.demo.utils.RequestUtils;

import com.example.demo.utils.StatusCode;
import io.swagger.annotations.ApiOperation;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

import java.util.HashMap;
import java.util.Map;
import java.util.UUID;

@RestController          //注解可以使结果以Json字符串的形式返回给客户端
@RequestMapping(value = "/api/getinfo")          //使链接还有一个 /api/
public class GetInfoController {

    /**
     *  小程序的测试号
     *  appid
     *  secret
     * */
    public static String appid = "wx023e3a4297441859";
    public static String secret = "e7056611f6888dfd25745ff9b9dae882" ;

    @Autowired
    UserService userService;
```

```
    @GetMapping("/cs")
    public Map<String, Object> cs(){
        Map<String, Object> map = new HashMap<>();
        map.put("msg","helloworld");
        return map;
    }

    /**
     * 获取电话号码
     * 请求：POST
     * 链接：地址/api/getinfo/getphone
     * 参数：code
     *  Content-Type: application/json;
     * 返回
        json {
            "msg":"ok"
        }
     * */
    @PostMapping("getphone")
    @ApiOperation(value="获取用户手机号码")
    public Map<String, Object> getPhone(@RequestParam("code") String code){
//获取前端传过来的code

        try{
            System.out.println(code);

            //获取获取小程序全局唯一后台接口调用凭据（access_token）
            String getTokenUrl = "https://api.weixin.qq.com/cgi-bin/token?
grant_type=client_credential&appid="+appid+"&secret="+secret;

            //调用get请求去访问微信小程序自带的链接，将返回结果存储到jsonStringa中
            String jsonStringa =  RequestUtils.doGet(getTokenUrl);

            //String转JSON
            JSONObject jsonObject = JSONObject.parseObject(jsonStringa);

            //获取JSON数据中的access_token
            String access_token = jsonObject.getString("access_token");
            //提交参数
            String getPhoneUrl =
"https://api.weixin.qq.com/wxa/business/getuserphonenumber?
access_token="+access_token;

            Map<String,Object> map =  new HashMap<String,Object>();

            //将code放到map中
            map.put("code",code);
```

```
            //Map格式转化成JSON格式
            JSONObject json = new JSONObject(map);

            //向微信小程序接口提交Post请求得到结果
            String jsonStringb = RequestUtils.doPostForm(getPhoneUrl,json);

            //String转JSON
            JSONObject jsonObject2 = JSONObject.parseObject(jsonStringb);
            HashMap hashMap =
JSONObject.parseObject(jsonObject2.toJSONString(), HashMap.class);
            //请求成功
            if( 0 == (int)hashMap.get("errcode") ){
                Map<String, String> data = new HashMap<>();
                JSONObject tmp2 = (JSONObject)hashMap.get("phone_info");
                data.put("phone",(String)tmp2.get("phoneNumber"));
                //将结果存储下来
                return StatusCode.success(data);
            }else{
                return StatusCode.error((int)hashMap.get("errcode"),"获取失
败");
            }
        }catch (Exception e){
            System.out.println(e);
            return StatusCode.error(3001,"服务器内部错误: "+e.toString());
        }

    }

    //https://blog.csdn.net/qq_41432730/article/details/123617323
    //
    @PostMapping(value = "login")
    @ApiOperation(value="登录")
    public @ResponseBody Map<String,Object> login(@RequestParam("code")
String code,@RequestParam("avatarUrl") String
avatarUrl,@RequestParam("avatarUrl") String name) {
        Map<String, String> data = new HashMap<String, String>();
        try {
            //Get请求（登录凭证校验）
            String getAuthUrl =
"https://api.weixin.qq.com/sns/jscode2session?appid=" + appid + "&secret=" +
secret + "&js_code=" + code + "&grant_type=authorization_code";
            //进行get请求
            String jsonString = RequestUtils.doGet(getAuthUrl);
            //String转JSON，再json转为map
            JSONObject jsonObject = JSONObject.parseObject(jsonString);
            HashMap hashMap =
JSONObject.parseObject(jsonObject.toJSONString(), HashMap.class);
```

```
            //注意这里要加上 hashMap.get("errcode") == null
            if ( hashMap.get("errcode") == null || 0 == (int)
hashMap.get("errcode")) {                //请求成功
                //得到openid和session_key去生成3rd_session
                //这个生成3rd_session的方式自己决定即可，比如使用SHA或Base64算法都可
以。例如：将session_key或openid+session_key作为SHA或Base64算法的输入，输出结果做为
3rd_session来使用，同时要将openid，session_key，3rd_session三者关联存储到数据库中，
方便下次拿3rd_session获取session_key或openid做其他处理。
                String openid = (String) hashMap.get("openid");
                String session_key = (String) hashMap.get("session_key");

                //判断是否注册过
                int tmp = userService.isUser(openid);
                if ( tmp == 0 ) {
                    //没有注册过
                    User user = new User(avatarUrl,name,openid);
                    //插入数据库
                    userService.insertUser(user);
                }
                //生成token
                JwtUtils jwt = JwtUtils.getInstance();
                String token = jwt
                        .setClaim("openid",openid)
                        .setClaim("name",name)
                        .generateToken();

                Map<String, String> tmp3 = new HashMap<>();
                tmp3.put("token",token);
                //将结果存储下来
                return StatusCode.success(tmp3);

            } else {
                return StatusCode.error((int) hashMap.get("errcode"),"获取失
败");
            }
        }
        catch(Exception e){
            e.printStackTrace();
            return StatusCode.error(3001,"服务器内部错误："+e.toString());
        }

    }
}
```

HelloController.java

```java
package com.example.demo.controller;

import com.example.demo.pojo.User;
import com.example.demo.service.UserService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.ResponseBody;
import org.springframework.web.bind.annotation.RestController;

import java.util.HashMap;
import java.util.List;
import java.util.Map;

@RestController          //注解可以使结果以Json字符串的形式返回给客户端
public class HelloController {
    @GetMapping("/hello")
    public String hello(){
        return "hello SpringBoot";
    }

    @GetMapping("/cs")
    public Map<String, Object> cs(){
        Map<String, Object> map = new HashMap<>();
        map.put("msg","helloworld");
        return map;
    }

}
```

LocalRecipeController.java

```java
package com.example.demo.controller;

import com.example.demo.pojo.Recipe;
import com.example.demo.service.RecipeService;
import com.example.demo.utils.StatusCode;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

import java.util.List;
import java.util.Map;

/**
```

```
 * 地域特色菜谱
 * */
@RestController          //注解可以使结果以Json字符串的形式返回给客户端
@RequestMapping(value = "/api/recipes/localfeature")          //使链接还有一个
/api/
public class LocalRecipeController {
    @Autowired
    RecipeService recipeService;

    @PostMapping("/")
    public Map<String, Object> listRecipe(@RequestParam("page") int page) {
//获取前端传过来的code
        try{
            List<Recipe> tmp = recipeService.listTypeRecipe(page,8,"地域特色菜
谱");
            Map<String, Object> data = StatusCode.success(tmp);
            return data;
        }catch (Exception e){
            System.out.println(e);
            Map<String, Object> data = StatusCode.error(3001);
            return data;
        }
    }

    @PostMapping("/{id}")
    public Map<String, Object> getRecipe(@PathVariable(name="id") String id)
{ //获取前端传过来的code
        //limit 为8
        try{
            Recipe tmp = recipeService.getRecipe(id);
            Map<String, Object> data = StatusCode.success(tmp);
            return data;
        }catch (Exception e){
            Map<String, Object> data = StatusCode.error(3001);
            return data;
        }
    }

}
```

RecipeController.java

```
package com.example.demo.controller;

import com.example.demo.pojo.Recipe;
import com.example.demo.service.RecipeService;
```

```java
import com.example.demo.utils.FileUploadUtils;
import com.example.demo.utils.StatusCode;
import com.fasterxml.jackson.annotation.JsonInclude;
import io.swagger.annotations.ApiOperation;
import org.apache.tomcat.util.http.fileupload.FileUpload;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;
import org.springframework.web.multipart.MultipartFile;

import javax.servlet.http.HttpServletRequest;
import javax.validation.Valid;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.UUID;

@RestController          //注解可以使结果以Json字符串的形式返回给客户端
@RequestMapping(value = "/api/recipes")          //使链接还有一个 /api/
public class RecipeController {
    @Autowired
    RecipeService recipeService;

    @PostMapping("/")
    @ApiOperation(value="获取全部菜谱的关键信息")
    public Map<String, Object> listRecipe(@RequestParam("page") int page) {
//获取前端传过来的code
        try{
            List<Recipe> tmp = recipeService.listRecipe(page,8);
            Map<String, Object> data = StatusCode.success(tmp);
            return data;
        }catch (Exception e){
            System.out.println(e);
            Map<String, Object> data = StatusCode.error(3001);
            return data;
        }
    }

    @PostMapping("/{id}")
    @ApiOperation(value="获取具体菜谱信息")
    public Map<String, Object> getRecipe(@PathVariable(name="id") String id)
{
        //limit 为8
        try{
            Recipe tmp = recipeService.getRecipe(id);
            Map<String, Object> data = StatusCode.success(tmp);
            return data;
        }catch (Exception e){
            Map<String, Object> data = StatusCode.error(3001);
```

```
            return data;
        }
    }

    @PostMapping("/insert")
    @ApiOperation(value="添加具体菜谱信息")
    public Map<String, Object> insertRecipe(@RequestParam MultipartFile file,
@Valid Recipe recipe, HttpServletRequest request) {
        try{
            //保存文件
            String filename = FileUploadUtils.SaveServer(file,request);
            //用UUID来生成唯一的id
            //String id = UUID.randomUUID().toString();
            //把实体类中的picture设置成文件路径
            recipe.setPicture(filename);
            //recipe.setId(id);

            recipeService.insertRecipe(recipe);
            Map<String, Object> data = StatusCode.success("插入成功");
            return data;
        }catch (Exception e){
            e.printStackTrace();
            Map<String, Object> data = StatusCode.error(3001);
            return data;
        }
    }


    @PostMapping("/search")
    @ApiOperation(value="搜索菜谱信息")
    public Map<String, Object> search(@RequestParam("find") String find) {
        //limit 为8
        try{
            List<Recipe> tmp = recipeService.searchRecipe(find);
            Map<String, Object> data = StatusCode.success(tmp);
            return data;
        }catch (Exception e){
            e.printStackTrace();
            Map<String, Object> data = StatusCode.error(3001,"服务器内部错
误: "+e.toString());
            return data;
        }
    }
}
```

TraditionRecipeController.java

```java
package com.example.demo.controller;

import com.example.demo.pojo.Recipe;
import com.example.demo.service.RecipeService;
import com.example.demo.utils.StatusCode;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

import java.util.List;
import java.util.Map;

/**
 * 传统文化菜谱
 * */
@RestController          //注解可以使结果以Json字符串的形式返回给客户端
@RequestMapping(value = "/api/recipes/traditionculture")          //使链接还有一个 /api/
public class TraditionRecipeController {
    @Autowired
    RecipeService recipeService;

    @PostMapping("/")
    public Map<String, Object> listRecipe(@RequestParam("page") int page) {
//获取前端传过来的code
        try{
            List<Recipe> tmp = recipeService.listTypeRecipe(page,8,"传统文化菜谱");
            Map<String, Object> data = StatusCode.success(tmp);
            return data;
        }catch (Exception e){
            System.out.println(e);
            Map<String, Object> data = StatusCode.error(3001);
            return data;
        }
    }

    @PostMapping("/{id}")
    public Map<String, Object> getRecipe(@PathVariable(name="id") String id) { //获取前端传过来的code
        //limit 为8
        try{
            Recipe tmp = recipeService.getRecipe(id);
            Map<String, Object> data = StatusCode.success(tmp);
            return data;
        }catch (Exception e){
            Map<String, Object> data = StatusCode.error(3001);
            return data;
```

```
        }
    }
}
```

dao层
DoDishesMapper.java

```
package com.example.demo.dao;


import com.example.demo.pojo.Recipe;
import org.apache.ibatis.annotations.Mapper;
import org.apache.ibatis.annotations.Select;

import java.util.List;

@Mapper
public interface DoDishesMapper {


    @Select("select picture,dishes,id from food limit #{first},#{second};")
    List<Recipe> listRecipe(int first,int second);

    @Select("select picture,dishes,id from food where type=#{type} limit #
{first},#{second};")
    List<Recipe> listTypeRecipe(String type,int first,int second);

    @Select("select * from food where id=#{id};")
    Recipe getRecipe(String id);


}
```

RecipeMapper.java

```
package com.example.demo.dao;


import com.example.demo.pojo.Recipe;
import com.example.demo.pojo.User;
import org.apache.ibatis.annotations.Insert;
import org.apache.ibatis.annotations.Mapper;
import org.apache.ibatis.annotations.Select;
```

```
import java.util.List;

@Mapper
public interface RecipeMapper {


    /**
     * 页面只返回一张图片，名字，id
     * */
    @Select("select picture,dishes,id from food limit #{first},#{second};")
    List<Recipe> listRecipe(int first,int second);

    /**
     * 页面返回部分信息
     * */
    @Select("select picture,dishes,id from food where type=#{type} limit #
{first},#{second};")
    List<Recipe> listTypeRecipe(String type,int first,int second);

    /**
     * 页面返回具体信息
     * */
    @Select("select * from food where id=#{id};")
    Recipe getRecipe(String id);

    /**
     * 页面返回查找的信息（显示于页面上）
     * */
    @Select("select picture,dishes,id from food where dishes like
CONCAT('%',#{find},'%')  ")
    List<Recipe> searchRecipe(String find);

    /**
     * 添加新的菜谱信息
     * */
    @Insert("insert into
food(dishes,regional,culture,efficacy,materials,practice,type,id,picture)
values(#{dishes},#{regional},#{culture},#{efficacy},#{materials},#
{practice},#{type},#{id},#{picture})")
    int insertRecipe(Recipe recipe);
}
```

UserMapper.java

```
package com.example.demo.dao;
```

```
import com.example.demo.pojo.User;
import org.apache.ibatis.annotations.Insert;
import org.apache.ibatis.annotations.Mapper;
import org.apache.ibatis.annotations.Select;

import java.util.List;

@Mapper
public interface UserMapper {
     /**
      * 判断用户是否存在
      * */
     @Select("SELECT count(*) FROM users WHERE openid=#{openid}")
     int isUser(String openid);

     /**
      * 添加新的用户信息
      * */
     @Insert("insert into food(avatarUrl,name,openid) values(#{dishes},#
{avatarUrl},#{name},#{openid})")
     int insertUser(User user);

}
```

pojo层
DoDishes.java

```
package com.example.demo.pojo;

import com.fasterxml.jackson.annotation.JsonInclude;
import lombok.Data;

/**
 * 做菜实体类
 * 1.蔬菜
 * 2.肉类
 * 3.调料 seasoning
 * */
//create table food()

//create table food(id varchar(50) not null,dishes varchar(50) not
null,regional varchar(50),culture longtext,efficacy longtext,
// efficacy longtext,materials longtext,practice longtext,type varchar(30)
not null,picture longtext,pirmary key(id));

@Data       //使用这个注解可以省去代码中大量的get()、 set()、 toString()等方法;
```

```
@JsonInclude(JsonInclude.Include.NON_NULL)    // 忽略返回参时值为null的字段
public class DoDishes {
    String vegetable;
    String meat;
    String seasoning;

}
```

Recipe.java

```
package com.example.demo.pojo;

import com.fasterxml.jackson.annotation.JsonInclude;
import lombok.Data;

import javax.validation.constraints.NotBlank;

/**
 * 菜谱实体类
 * 0.图片集
 * 1.菜名
 * 2.所属地域及地域特色
 * 3.菜品文化
 * 4.菜品功效
 * 5.菜品原材料（链接到商城）
 * 6.做法分享
 * 7.类型
 * 8.id唯一
 * */
//create table food()

//create table food(id int not null AUTO_INCREMENT,dishes varchar(50) not
null,regional varchar(50),culture longtext,efficacy longtext,
// efficacy longtext,materials longtext,practice longtext,type varchar(30)
not null,picture longtext,pirmary key(id));

@Data       //使用这个注解可以省去代码中大量的get()、 set()、 toString()等方法;
@JsonInclude(JsonInclude.Include.NON_NULL)    // 忽略返回参时值为null的字段
public class Recipe {
    // 如果是字符串类型的数据,使用 @NotBlank 比 @NoNull 更好,因为 @NotBlank 不仅会
校验 null 值,它还会校验空字符串


    private String picture;
    @NotBlank
```

```
    private String dishes;
    private String regional;
    private String culture;
    private String efficacy;
    private String materials;
    private String practice;
    @NotBlank
    private String type;
    private String id;
}
```

User.java

```
package com.example.demo.pojo;
import lombok.Data;

//create table users(avatarUrl longtext not null,name varchar(50),openid
varchar(70),primary key (openid));
/**
 * 用户实体类
 * avatarUrl 头像链接
 * name 昵称
 * */
@Data       //使用这个注解可以省去代码中大量的get()、 set()、 toString()等方法;
public class User {
    private String avatarUrl;
    private String name;
    private String openid;
    public User(String _avatarUrl,String _name,String _openid){
        avatarUrl = _avatarUrl;
        name = _name;
        openid = _openid;
    }
}
```

service层
RecipeService.java

```
package com.example.demo.service;

import com.example.demo.pojo.Recipe;

import java.util.List;
```

```
public interface RecipeService {
    /**
     * 功能:显示部分信息
     * @param page 页数
     * @return 返回每张页面的信息
     * */
    List<Recipe> listRecipe(int page,int limit);

    /**
     * 功能:根据类型显示部分信息
     * @param page 页数
     * @param type 类型
     * @return 返回每张页面的信息
     * */
    List<Recipe> listTypeRecipe(int page,int limit,String type);

    /**
     * 显示具体信息
     * @param id 标识符
     * @return 返回页面的具体信息
     * */
    Recipe getRecipe(String id);

    /**
     * 插入具体菜谱
     * @param find 查找内容
     * @return 返回插入成功的数组
     * */
    List<Recipe> searchRecipe(String find);

    /**
     * 插入具体菜谱
     * @param recipe 实体类
     * @return 返回插入成功的列
     * */
    int insertRecipe(Recipe recipe);


}
```

RecipeServiceImpl.java

```
package com.example.demo.service;

import com.example.demo.dao.RecipeMapper;
import com.example.demo.dao.UserMapper;
```

```java
import com.example.demo.pojo.Recipe;
import com.example.demo.pojo.User;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.util.List;

@Service
public class RecipeServiceImpl implements RecipeService {
    @Autowired
    RecipeMapper recipeMapper;

    @Override
    public List<Recipe> listRecipe(int page, int limit){
        //{(page -1) * limit},#{limit};")
        int first = (page-1)*limit;
        int second = limit;
        return recipeMapper.listRecipe(first, second);
    }

    @Override
    public List<Recipe> listTypeRecipe(int page, int limit,String type){
        //{(page -1) * limit},#{limit};")
        int first = (page-1)*limit;
        int second = limit;
        return recipeMapper.listTypeRecipe(type,first,second);
    }

    @Override
    public Recipe getRecipe(String id){
        return recipeMapper.getRecipe(id);
    }

    @Override
    public int insertRecipe(Recipe recipe){
        return recipeMapper.insertRecipe(recipe);
    }

    @Override
    public List<Recipe> searchRecipe(String find){
        return recipeMapper.searchRecipe(find);
    }

}
```

UserService.java

```java
package com.example.demo.service;

import com.example.demo.pojo.User;

import java.util.List;

public interface UserService {
    int isUser(String openid);
    int insertUser(User user);
}
```

UserServiceImpl.java

```java
package com.example.demo.service;

import com.example.demo.dao.UserMapper;
import com.example.demo.pojo.User;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

@Service
public class UserServiceImpl implements UserService {
    @Autowired
    UserMapper userMapper;

    @Override
    public int isUser(String openid) {
        return userMapper.isUser(openid);
    }

    @Override
    public int insertUser(User user) {
        return userMapper.insertUser(user);
    }
}
```

utils层
FileUploadUtils.java

```java
package com.example.demo.utils;

import org.springframework.web.multipart.MultipartFile;
```

```java
import javax.servlet.http.HttpServletRequest;
import java.io.File;
import java.util.UUID;

//文件上传相关的工具包
public class FileUploadUtils {

    // UUID可以避免重命名
    // 传入原始文件名可以得到  uuid+"_"+文件的原始名称 的返回值
    public static String getUuidFileName(String fileName) {
        //文件名以：uuid+"_"+文件的原始名称
        return UUID.randomUUID().toString() + "_" + fileName;
    }

    //保存到服务器，返回存储的位置
    public static String SaveServer(MultipartFile file, HttpServletRequest
request) {
        System.out.println("正在上传文件");
        //得到上传文件的保存目录，将上传的文件存放于WEB-INF目录下，不允许外界直接访问，
保证上传文件的安全
        //String realpath = request.getServletContext().getRealPath("/WEB-
INF/files");
        //暂时就不保存在WEB-INF那里，先放在files
        //这里保存在根目录中的file文件中
        String realpath = request.getServletContext().getRealPath("files");

        //获取文件名字，进行UUID重命名
        String fileName = getUuidFileName(file.getOriginalFilename());

        //文件上传
        File targetFile = new File(realpath, fileName);

        //如果不存在，创建文件
        if (!targetFile.exists()) {
            targetFile.mkdirs();
        }
        // 上传
        try {
            file.transferTo(targetFile);            //保存下来
            System.out.println("上传成功");
            //return realpath + '\\' + fileName;
            //返回相对路径就行
            return "\\files\\" + fileName;
        } catch (Exception e) {
            e.printStackTrace();
            return null;                        //说明保存不成功
        }
    }
```

```
}
```

JwtUtils.java

```
package com.example.demo.utils;

import io.jsonwebtoken.*;

import java.util.Date;

//学习网站：https://www.jianshu.com/p/578a7b2f3e8d

/**
 * jwt工具类
 */
public class JwtUtils {
    /**
     * 实例
     */
    private static JwtUtils instance;

    /**
     * 发行者
     */
    private String subObject = "wechatbyorall";

    /**
     * 过期时间，默认1天
     */
    private long expired = 1000 * 60 * 60 * 24 * 1;

    /**
     * jwt构造
     */
    private static JwtBuilder jwtBuilder;

    /**
     * 密钥
     */
    private String secret = "UYRL@sdjow114da5#KXKRWF$1124gki";// 密钥

    /**
     * 获取实例
     *
```

```
     * @return
     */
    public static JwtUtils getInstance() {
        if (instance == null) {
            instance = new JwtUtils();
        }
        jwtBuilder = Jwts.builder();
        return instance;
    }

    /**
     * 荷载信息(通常是一个User信息，还包括一些其他的元数据)
     *
     * @param key
     * @param val
     * @return
     */
    public JwtUtils setClaim(String key, Object val) {
        jwtBuilder.claim(key, val);
        return this;
    }

    /**
     * 生成 jwt token
     *
     * @return
     */
    public String generateToken() {
        String token = jwtBuilder
                .setSubject(subObject) // 发行者
                //.claim("id","121") // 参数
                .setIssuedAt(new Date()) // 发行时间
                .setExpiration(new Date(System.currentTimeMillis() +
expired))
                .signWith(SignatureAlgorithm.HS256, secret) // 签名类型 与 密钥
                .compressWith(CompressionCodecs.DEFLATE)// 对载荷进行压缩
                .compact(); // 压缩一下
        return token;
    }

    /**
     * 解析 token
     *
     * @param token
     * @return
     */
    public Claims check(String token) {
        try {
```

```
            final Claims claims = Jwts.parser()
                    .setSigningKey(secret)
                    .parseClaimsJws(token)
                    .getBody();
            return claims;
        } catch (Exception e) {
            System.out.println(e);
        }
        return null;
    }

    public String getSubObject() {
        return subObject;
    }

    /**
     * 设置发行者
     *
     * @param subObject
     * @return
     */
    public JwtUtils setSubObject(String subObject) {
        this.subObject = subObject;
        return this;
    }

    public long getExpired() {
        return expired;
    }

    /**
     * 设置过期时间
     *
     * @param expired
     * @return
     */
    public JwtUtils setExpired(long expired) {
        this.expired = expired;
        return this;
    }

    public String getSecret() {
        return secret;
    }

    /**
     * 设置密钥
     *
```

```
     * @param secret
     * @return
     */
    public JwtUtils setSecret(String secret) {
        this.secret = secret;
        return this;
    }
}
```

RequestsUtils.java

```
package com.example.demo.utils;

import com.alibaba.fastjson.JSONObject;
import com.google.gson.Gson;

import java.io.*;
import java.net.HttpURLConnection;
import java.net.MalformedURLException;
import java.net.URL;
import java.util.*;

//https://www.cnblogs.com/mufengforward/p/10510337.html

public class RequestUtils {
    /**
     * 向指定URL发送GET方法的请求
     *
     * @param httpurl 请求参数用?拼接在url后边，请求参数应该是
name1=value1&name2=value2 的形式。
     * @return result 所代表远程资源的响应结果
     */
    //发送get请求
    public static String doGet(String httpurl) {
        HttpURLConnection connection = null;
        InputStream is = null;
        BufferedReader br = null;
        // 返回结果字符串
        String result = null;
        try {
            // 创建远程url连接对象
            URL url = new URL(httpurl);
            // 通过远程url连接对象打开一个连接，强转成httpURLConnection类
            connection = (HttpURLConnection) url.openConnection();
            // 设置连接方式：get
            connection.setRequestMethod("GET");
```

```
            // 设置连接主机服务器的超时时间：15000毫秒
            connection.setConnectTimeout(15000);
            // 设置读取远程返回的数据时间：60000毫秒
            connection.setReadTimeout(60000);
            // 发送请求
            connection.connect();
            // 通过connection连接，获取输入流
            if (connection.getResponseCode() == 200) {
                is = connection.getInputStream();
                // 封装输入流is，并指定字符集
                br = new BufferedReader(new InputStreamReader(is, "UTF-8"));
                // 存放数据
                StringBuffer sbf = new StringBuffer();
                String temp = null;
                while ((temp = br.readLine()) != null) {
                    sbf.append(temp);
                    sbf.append("\r\n");
                }
                result = sbf.toString();
            }
        } catch (MalformedURLException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            // 关闭资源
            if (null != br) {
                try {
                    br.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }

            if (null != is) {
                try {
                    is.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }

            connection.disconnect();// 关闭远程连接
        }

        return result;
    }
```

```
    /**
     * @param httpUrl 请求的url
     * @param param   form表单的参数（key,value形式）
     * @return
     */
    //发送post请求，注意请求微信小程序接口的格式是"Content-Type":"application/json"
    public static String doPostForm(String httpUrl, Map param) {

        HttpURLConnection connection = null;
        InputStream is = null;
        OutputStream os = null;
        BufferedReader br = null;
        String result = null;
        try {
            URL url = new URL(httpUrl);
            // 通过远程url连接对象打开连接
            connection = (HttpURLConnection) url.openConnection();
            // 设置连接请求方式
            connection.setRequestMethod("POST");
            // 设置连接主机服务器超时时间：15000毫秒
            connection.setConnectTimeout(15000);
            // 设置读取主机服务器返回数据超时时间：60000毫秒
            connection.setReadTimeout(60000);

            // 默认值为：false，当向远程服务器传送数据/写数据时，需要设置为true
            connection.setDoOutput(true);
            // 默认值为：true，当前向远程服务读取数据时，设置为true，该参数可有可无
            connection.setDoInput(true);
            // 设置传入参数的格式:请求参数应该是 name1=value1&name2=value2 的形式。
            connection.setRequestProperty("Content-Type",
"application/json");
            // 设置鉴权信息：Authorization: Bearer da3efcbf-0845-4fe3-8aba-
ee040be542c0
            //connection.setRequestProperty("Authorization", "Bearer
da3efcbf-0845-4fe3-8aba-ee040be542c0");
            // 通过连接对象获取一个输出流
            os = connection.getOutputStream();
            // 通过输出流对象将参数写出去/传输出去,它是通过字节数组写出的(form表单形式
的参数实质也是key,value值的拼接，类似于get请求参数的拼接)
            JSONObject json = new JSONObject(param);
            os.write(createLinkString(param).getBytes());
            // 通过连接对象获取一个输入流，向远程读取
            if (connection.getResponseCode() == 200) {

                is = connection.getInputStream();
                // 对输入流对象进行包装:charset根据工作项目组的要求来设置
                br = new BufferedReader(new InputStreamReader(is, "UTF-8"));
```

```
                StringBuffer sbf = new StringBuffer();
                String temp = null;
                // 循环遍历一行一行读取数据
                while ((temp = br.readLine()) != null) {
                    sbf.append(temp);
                    sbf.append("\r\n");
                }
                result = sbf.toString();
            }
        } catch (MalformedURLException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            // 关闭资源
            if (null != br) {
                try {
                    br.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
            if (null != os) {
                try {
                    os.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
            if (null != is) {
                try {
                    is.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
            // 断开与远程地址url的连接
            connection.disconnect();
        }
        return result;
    }

    //发送"application/json"格式的POST请求
    public static String doPostForm(String httpUrl, JSONObject param) {

        HttpURLConnection connection = null;
        InputStream is = null;
        OutputStream os = null;
```

```
        BufferedReader br = null;
        String result = null;
        try {
            URL url = new URL(httpUrl);
            // 通过远程url连接对象打开连接
            connection = (HttpURLConnection) url.openConnection();
            // 设置连接请求方式
            connection.setRequestMethod("POST");
            // 设置连接主机服务器超时时间：15000毫秒
            connection.setConnectTimeout(15000);
            // 设置读取主机服务器返回数据超时时间：60000毫秒
            connection.setReadTimeout(60000);

            // 默认值为：false，当向远程服务器传送数据/写数据时，需要设置为true
            connection.setDoOutput(true);
            // 默认值为：true，当前向远程服务读取数据时，设置为true，该参数可有可无
            connection.setDoInput(true);
            // 设置传入参数的格式:请求参数应该是 name1=value1&name2=value2 的形式。
            connection.setRequestProperty("Content-Type",
"application/json");
            // 设置鉴权信息：Authorization: Bearer da3efcbf-0845-4fe3-8aba-
ee040be542c0
            //connection.setRequestProperty("Authorization", "Bearer
da3efcbf-0845-4fe3-8aba-ee040be542c0");
            // 通过连接对象获取一个输出流
            os = connection.getOutputStream();
            // 通过输出流对象将参数写出去/传输出去,它是通过字节数组写出的(form表单形式
的参数实质也是key,value值的拼接，类似于get请求参数的拼接)

            os.write(param.toString().getBytes());
            // 通过连接对象获取一个输入流，向远程读取
            if (connection.getResponseCode() == 200) {

                is = connection.getInputStream();
                // 对输入流对象进行包装:charset根据工作项目组的要求来设置
                br = new BufferedReader(new InputStreamReader(is, "UTF-8"));

                StringBuffer sbf = new StringBuffer();
                String temp = null;
                // 循环遍历一行一行读取数据
                while ((temp = br.readLine()) != null) {
                    sbf.append(temp);
                    sbf.append("\r\n");
                }
                result = sbf.toString();
            }
        } catch (MalformedURLException e) {
            e.printStackTrace();
```

```
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            // 关闭资源
            if (null != br) {
                try {
                    br.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
            if (null != os) {
                try {
                    os.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
            if (null != is) {
                try {
                    is.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
            // 断开与远程地址url的连接
            connection.disconnect();
        }
        return result;
    }


    /**
     * 把数组所有元素排序，并按照“参数=参数值”的模式用“&”字符拼接成字符串
     *
     * @param params 需要排序并参与字符拼接的参数组
     * @return 拼接后字符串
     */
    public static String createLinkString(Map<String, String> params) {

        List<String> keys = new ArrayList<String>(params.keySet());
        Collections.sort(keys);

        StringBuilder prestr = new StringBuilder();
        for (int i = 0; i < keys.size(); i++) {
            String key = keys.get(i);
            String value = params.get(key);
            if (i == keys.size() - 1) {// 拼接时，不包括最后一个&字符
```

```
                prestr.append(key).append("=").append(value);
            } else {
                prestr.append(key).append("=").append(value).append("&");
            }
        }

        return prestr.toString();
    }

    //String 转 Map
    public static Map<String, Object> strToJson(String jsonString) {
        Gson gson = new Gson();
        Map<String, Object> map = new HashMap<String, Object>();
        map = gson.fromJson(jsonString, map.getClass());
        return map;
    }


}
```

StatusCode.java

```
package com.example.demo.utils;

import lombok.Data;

import java.util.HashMap;
import java.util.Map;

/**
 * status 状态码 0-表示失败，1-表示成功
 * error_code 错误码，一般在设计时定义
 * error_des 错误描述，一般在设计时定义
 * data 成功数据
 * msg 请求成功 / 请求失败
 *
 * @author czh
 */
@Data
public class StatusCode {
    /**
     * int status;
     * String error_code;
     * String error_des;
     * * Object data;
```

```
     */
    public static Map<String, Object> success(Object data) {
        Map<String, Object> tmp = new HashMap<>();
        tmp.put("status", 1);
        tmp.put("data", data);
        tmp.put("msg", "请求成功");
        return tmp;
    }

    public static Map<String, Object> error(int error_code) {
        Map<String, Object> tmp = new HashMap<>();
        tmp.put("status", 0);
        tmp.put("msg", "请求失败");
        tmp.put("error_code", error_code);
        String error_des = "";
        // 以下待完善
        if (error_code == 1001) {
            error_des = "参数无效";
        } else if (error_code == 1002) {
            error_des = "参数缺失";
        } else if (error_code == 2001) {
            error_des = "用户未登录";
        } else if (error_code == 3001) {
            error_des = "服务器错误";
        }
        tmp.put("error_des", error_des);
        return tmp;
    }

    public static Map<String, Object> error(int error_code, String error_des)
{
        Map<String, Object> tmp = new HashMap<>();
        tmp.put("status", 0);
        tmp.put("msg", "请求失败");
        tmp.put("error_code", error_code);
        tmp.put("error_des", error_des);
        return tmp;
    }

}
```

DemoApplication.java

```
package com.example.demo;
```

```
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.boot.builder.SpringApplicationBuilder;
import
org.springframework.boot.web.servlet.support.SpringBootServletInitializer;

@SpringBootApplication

public class DemoApplication extends SpringBootServletInitializer {
    @Override
    protected SpringApplicationBuilder configure(SpringApplicationBuilder
builder) {
        return builder.sources(DemoApplication.class);
    }

    public static void main(String[] args) {
        SpringApplication.run(DemoApplication.class, args);
    }

}
```

application.properties

```
spring.datasource.url=jdbc:mysql://****:3306/****
spring.datasource.username=****
spring.datasource.password=****
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver


spring.servlet.multipart.max-file-size= 50MB
spring.servlet.multipart.max-request-size= 50MB
```

pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>org.springframework.boot</groupId>
```

```
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.6.6</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>
    <groupId>com.example</groupId>
    <artifactId>demo</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>demo</name>
    <packaging>war</packaging>
    <description>Demo project for Spring Boot</description>
    <properties>
        <java.version>1.8</java.version>
    </properties>
    <dependencies>
        <!--导入 spring-boot-starter-web：能够为提供 Web 开发场景所需要的几乎所有依
赖-->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
            <exclusions>
                <exclusion>
                    <groupId>org.springframework.boot</groupId>
                    <artifactId>spring-boot-starter-tomcat</artifactId>
                </exclusion>
            </exclusions>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-tomcat</artifactId>
            <version>2.5.2</version>
            <scope>provided</scope>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.mybatis.spring.boot</groupId>
            <artifactId>mybatis-spring-boot-starter</artifactId>
            <version>1.3.2</version>
        </dependency>

        <!-- https://mvnrepository.com/artifact/mysql/mysql-connector-java --
>
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
```

```
            <version>8.0.16</version>
        </dependency>

        <dependency>
            <groupId>io.springfox</groupId>
            <artifactId>springfox-swagger2</artifactId>
            <version>2.9.2</version>
        </dependency>

        <!-- https://mvnrepository.com/artifact/io.jsonwebtoken/jjwt -->
        <dependency>
            <groupId>io.jsonwebtoken</groupId>
            <artifactId>jjwt</artifactId>
            <version>0.9.1</version>
        </dependency>

        <!--
https://mvnrepository.com/artifact/org.springframework.boot/spring-boot-
starter-validation -->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-validation</artifactId>
            <version>2.6.6</version>
        </dependency>

        <dependency>
            <groupId>org.mybatis</groupId>
            <artifactId>mybatis</artifactId>
            <version>3.5.9</version>
        </dependency>

        <!-- https://mvnrepository.com/artifact/com.alibaba/fastjson -->
        <dependency>
            <groupId>com.alibaba</groupId>
            <artifactId>fastjson</artifactId>
            <version>1.2.68</version>
        </dependency>

        <!-- https://mvnrepository.com/artifact/com.google.code.gson/gson -->
        <dependency>
            <groupId>com.google.code.gson</groupId>
            <artifactId>gson</artifactId>
            <version>2.8.5</version>
        </dependency>
        <!-- 引用注解Data -->
        <dependency>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
```

```
            <version>1.18.18</version>
            <scope>provided</scope>
        </dependency>
    </dependencies>
    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-war-plugin</artifactId>
                <configuration>
                    <warName>recipe</warName>
                </configuration>
            </plugin>

        </plugins>
    </build>

</project>
```

（2022.5.16）

基本雏形

数据库代码

创建数据库

```
mysql> create database recipe;
mysql> create user '******'@'%' identified by  '******';
mysql> grant all privileges on *.* to 'wc'@'%' with grant option;
mysql> flush privileges;
```

创建表

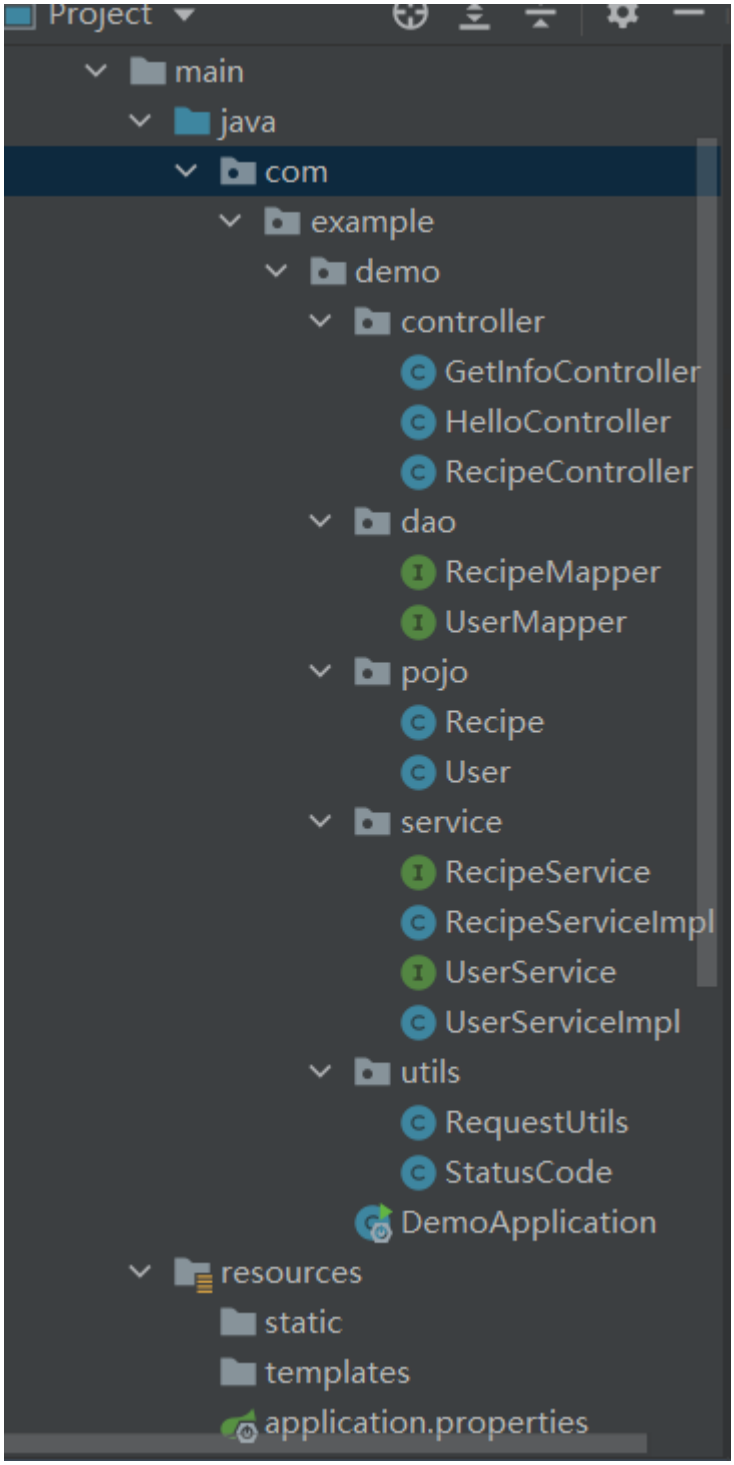
```
create table food(id int not null AUTO_INCREMENT,dishes varchar(50) not
null,regional varchar(50),culture longtext,efficacy longtext,materials
longtext,practice longtext,type varchar(30) not null,picture longtext,primary
key(id) );
```

```
create table users(avatarUrl longtext not null,name varchar(50),openid
varchar(70),primary key (openid));
```

代码

项目结构

```
├─src
│  ├─main
│  │  ├─java
│  │  │  └─com
│  │  │      ├─body
│  │  │      └─example
│  │  │          └─demo
│  │  │              ├─controller
│  │  │              ├─dao
│  │  │              ├─pojo
│  │  │              ├─service
│  │  │              └─utils
│  │  └─resources
│  │      ├─static
│  │      └─templates
│  └─test
│      └─java
│          └─com
│              └─example
│                  └─demo
```



GetInfoController.java

```
package com.example.demo.controller;

//获取信息

import com.alibaba.fastjson.JSONObject;

import com.example.demo.utils.RequestUtils;
```

```java
import org.springframework.web.bind.annotation.*;

import java.util.HashMap;
import java.util.Map;
import java.util.UUID;

@RestController          //注解可以使结果以Json字符串的形式返回给客户端
@RequestMapping(value = "/api/getinfo")          //使链接还有一个 /api/
public class GetInfoController {

    /**
     *  小程序的测试号
     *  appid
     *  secret
     * */
    public static String appid = "wx023e3a4297441859";
    public static String secret = "e7056611f6888dfd25745ff9b9dae882" ;

    @GetMapping("/cs")
    public Map<String, Object> cs(){
        Map<String, Object> map = new HashMap<>();
        map.put("msg","helloworld");
        return map;
    }

    /**
     * 获取电话号码
     * 请求：POST
     * 链接：地址/api/getinfo/getphone
     * 参数：code
     *  Content-Type: application/json;
     * 返回
        json {
            "msg":"ok"
        }
     * */
    @PostMapping("getphone")
    public Map<String, String> getPhone(@RequestBody String code){ //获取前端传过来的code
        Map<String,String> data = new HashMap<String,String>();
        try{
            System.out.println(code);

            //获取获取小程序全局唯一后台接口调用凭据（access_token）
            String getTokenUrl = "https://api.weixin.qq.com/cgi-bin/token?grant_type=client_credential&appid="+appid+"&secret="+secret;

            //调用get请求去访问微信小程序自带的链接，将返回结果存储到jsonStringa中
```

```
            String jsonStringa =  RequestUtils.doGet(getTokenUrl);

            //String转JSON
            JSONObject jsonObject = JSONObject.parseObject(jsonStringa);

            //获取JSON数据中的access_token
            String access_token = jsonObject.getString("access_token");
            //提交参数
            String getPhoneUrl =
"https://api.weixin.qq.com/wxa/business/getuserphonenumber?
access_token="+access_token;

            Map<String,Object> map =  new HashMap<String,Object>();

            //将code放到map中
            map.put("code",code);
            //Map格式转化成JSON格式
            JSONObject json = new JSONObject(map);

            //向微信小程序接口提交Post请求得到结果
            String jsonStringb = RequestUtils.doPostForm(getPhoneUrl,json);

            //String转JSON
            JSONObject jsonObject2 = JSONObject.parseObject(jsonStringb);
            HashMap hashMap =
JSONObject.parseObject(jsonObject2.toJSONString(), HashMap.class);
            //请求成功
            if( 0 == (int)hashMap.get("errcode") ){
                data.put("msg","ok" );
                JSONObject tmp2 = (JSONObject)hashMap.get("phone_info");
                //将结果存储下来
                data.put("phone",(String)tmp2.get("phoneNumber"));
            }else{
                data.put("msg","fail" );
                data.put("error",(int)hashMap.get("errcode")+"" );
            }
        }catch (Exception e){
            System.out.println(e);
            data.put("msg","fail" );
        }

        return data;
    }

    @PostMapping(value = "login")
    public @ResponseBody Map<String,String> login(@RequestBody String code) {
        Map<String, String> data = new HashMap<String, String>();
        try {
```

```
            //Get请求（登录凭证校验）
            String getAuthUrl =
"https://api.weixin.qq.com/sns/jscode2session?appid=" + appid + "&secret=" +
secret + "&js_code=" + code + "&grant_type=authorization_code";
            String jsonString = RequestUtils.doGet(getAuthUrl);          //进
行get请求
            //System.out.println(jsonString);
            //String转JSON，再json转为map
            JSONObject jsonObject = JSONObject.parseObject(jsonString);
            HashMap hashMap =
JSONObject.parseObject(jsonObject.toJSONString(), HashMap.class);

            //注意这里要加上 hashMap.get("errcode") == null
            if ( hashMap.get("errcode") == null || 0 == (int)
hashMap.get("errcode")) {                 //请求成功
                data.put("msg", "ok");
                //得到openid和session_key去生成3rd_session
                //这个生成3rd_session的方式自己决定即可，比如使用SHA或Base64算法都可
以。例如：将session_key或openid+session_key作为SHA或Base64算法的输入，输出结果做为
3rd_session来使用，同时要将openid，session_key，3rd_session三者关联存储到数据库中，
方便下次拿3rd_session获取session_key或openid做其他处理。
                String openid = (String) hashMap.get("openid");
                String session_key = (String) hashMap.get("session_key");

                //uuid生成唯一
key(https://blog.csdn.net/weixin_38169886/article/details/99820453?
utm_medium=distribute.pc_relevant.none-task-blog-
2~default~baidujs_baidulandingword~default-
5.pc_relevant_default&spm=1001.2101.3001.4242.4&utm_relevant_index=8)
                String skey = UUID.randomUUID().toString();         //用UUID来
生成唯一的skey

                //判断是否注册过
                /*
                boolean tmp = Login.checkUser(openid);
                if (tmp == false) {                 //没有注册过
                    Login.register(openid, skey);
                    data.put("msg", "ok");
                    data.put("skey", skey);
                } else {                            //注册过，更新新的skey
                    Login.updateskey(openid, skey);
                    data.put("skey", skey);
                }*/
            } else {
                data.put("msg", "fail");
                data.put("error", (int) hashMap.get("errcode") + "");
            }
        }
```

```
        catch(Exception e){
            e.printStackTrace();
            data.put("msg", "fail");
            data.put("error", e.toString());
        }
        return data;
    }
}
```

HelloController.java

```
package com.example.demo.controller;

import com.body.Login;
import com.example.demo.pojo.User;
import com.example.demo.service.UserService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.ResponseBody;
import org.springframework.web.bind.annotation.RestController;

import java.util.HashMap;
import java.util.List;
import java.util.Map;

@RestController          //注解可以使结果以Json字符串的形式返回给客户端
public class HelloController {
    @GetMapping("/hello")
    public String hello(){
        return "hello SpringBoot";
    }

    @GetMapping("/cs")
    public Map<String, Object> cs(){
        Map<String, Object> map = new HashMap<>();
        map.put("msg","helloworld");
        return map;
    }

    @Autowired
    UserService userService;
    @GetMapping("/show")
    public List<User> getUser(int age){
```

```
        Map<String, Object> map = new HashMap<>(3);
        return userService.getUser(age);
    }
}
```

RecipeController.java

```
package com.example.demo.controller;

import com.example.demo.pojo.Recipe;
import com.example.demo.service.RecipeService;
import com.example.demo.utils.StatusCode;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

import java.util.HashMap;
import java.util.List;
import java.util.Map;

@RestController          //注解可以使结果以Json字符串的形式返回给客户端
@RequestMapping(value = "/api/recipes")          //使链接还有一个 /api/
public class RecipeController {
    @Autowired
    RecipeService recipeService;


    @PostMapping("/re")
    public Map<String, Object> getPwone() { //获取前端传过来的code
        try{
            List<Recipe> tmp = recipeService.listRecipe();
            Map<String, Object> data = StatusCode.success(tmp);
            return data;
        }catch (Exception e){
            System.out.println(e);
            Map<String, Object> data = StatusCode.error(3001);
            return data;
        }
    }

    @PostMapping("/{id}")
    public Map<String, Object> getPhone(@PathVariable(name="id") String id) {
//获取前端传过来的code

        //limit 为8
        try{
            Recipe tmp = recipeService.getRecipe(id);
```

```
            Map<String, Object> data = StatusCode.success(tmp);
            return data;
        }catch (Exception e){
            Map<String, Object> data = StatusCode.error(3001);
            return data;
        }


    }

    @PostMapping("/id")
    public Map<String, Object> asd(@RequestBody String id) { //获取前端传过来的
code
        //limit 为8
        try{
            Recipe tmp = recipeService.getRecipe(id);
            Map<String, Object> data = StatusCode.success(tmp);
            return data;
        }catch (Exception e){
            Map<String, Object> data = StatusCode.error(3001);
            return data;
        }


    }

}
```

dao层
RecipeMapper.java

```
package com.example.demo.dao;


import com.example.demo.pojo.Recipe;
import com.example.demo.pojo.User;
import org.apache.ibatis.annotations.Mapper;
import org.apache.ibatis.annotations.Select;

import java.util.List;

@Mapper
public interface RecipeMapper {
     //页面只返回一张图片，名字，id
```

```
    @Select("select picture,dishes,id from food;")
    List<Recipe> listRecipe();

    @Select("select * from food where id=#{id};")
    Recipe getRecipe(String id);

    //insert into food('1','1','1','1','1','1','1','1','1');

}
```

pojo层

```
package com.example.demo.pojo;

import lombok.Data;

/**
 * 菜谱实体类
 * 0.图片集
 * 1.菜名
 * 2.所属地域及地域特色
 * 3.菜品文化
 * 4.菜品功效
 * 5.菜品原材料（链接到商城）
 * 6.做法分享
 * 7.类型
 * 8.id唯一
 * */
//create table food()

//create table food(id varchar(50) not null,dishes varchar(50) not
null,regional varchar(50),culture longtext,efficacy longtext,
// efficacy longtext,materials longtext,practice longtext,type varchar(30)
not null,picture longtext,pirmary key(id));

@Data       //使用这个注解可以省去代码中大量的get()、 set()、 toString()等方法;
public class Recipe {
    private String picture;
    private String dishes;
    private String regional;
    private String culture;
    private String efficacy;
    private String materials;
    private String practice;
    private String type;
```

```
    private String id;
}
```

service层
RecipeService.java

```
package com.example.demo.service;

import com.example.demo.pojo.Recipe;

import java.util.List;

public interface RecipeService {
    /**
     * 功能:显示部分信息
     * @return 返回每张页面的信息
     * */
    List<Recipe> listRecipe();
    /**
     * 显示具体信息
     * @param id 标识符
     * @return 返回页面的具体信息
     * */
    Recipe getRecipe(String id);
}
```

RecipeServiceImpl.java

```
package com.example.demo.service;

import com.example.demo.dao.RecipeMapper;
import com.example.demo.dao.UserMapper;
import com.example.demo.pojo.Recipe;
import com.example.demo.pojo.User;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.util.List;

@Service
public class RecipeServiceImpl implements RecipeService {
    @Autowired
    RecipeMapper recipeMapper;
```

```
    @Override
    public List<Recipe> listRecipe(){
        //{(page -1) * limit},#{limit};")
        return recipeMapper.listRecipe();
    }

    @Override
    public Recipe getRecipe(String id){
        System.out.println(id);
        System.out.println(recipeMapper.getRecipe(id));
        return recipeMapper.getRecipe(id);
    }


}
```

utils层
RequsetUtils.java

```
package com.example.demo.utils;

import com.alibaba.fastjson.JSONObject;
import com.google.gson.Gson;

import java.io.*;
import java.net.HttpURLConnection;
import java.net.MalformedURLException;
import java.net.URL;
import java.util.*;

//https://www.cnblogs.com/mufengforward/p/10510337.html

public class RequestUtils {
    /**
     * 向指定URL发送GET方法的请求
     *
     * @param httpurl
     *             请求参数用?拼接在url后边，请求参数应该是
name1=value1&name2=value2 的形式。
     * @return result 所代表远程资源的响应结果
     */
    //发送get请求
    public static String doGet(String httpurl) {
        HttpURLConnection connection = null;
        InputStream is = null;
        BufferedReader br = null;
```

```
        // 返回结果字符串
        String result = null;
        try {
            // 创建远程url连接对象
            URL url = new URL(httpurl);
            // 通过远程url连接对象打开一个连接，强转成httpURLConnection类
            connection = (HttpURLConnection) url.openConnection();
            // 设置连接方式：get
            connection.setRequestMethod("GET");
            // 设置连接主机服务器的超时时间：15000毫秒
            connection.setConnectTimeout(15000);
            // 设置读取远程返回的数据时间：60000毫秒
            connection.setReadTimeout(60000);
            // 发送请求
            connection.connect();
            // 通过connection连接，获取输入流
            if (connection.getResponseCode() == 200) {
                is = connection.getInputStream();
                // 封装输入流is，并指定字符集
                br = new BufferedReader(new InputStreamReader(is, "UTF-8"));
                // 存放数据
                StringBuffer sbf = new StringBuffer();
                String temp = null;
                while ((temp = br.readLine()) != null) {
                    sbf.append(temp);
                    sbf.append("\r\n");
                }
                result = sbf.toString();
            }
        } catch (MalformedURLException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            // 关闭资源
            if (null != br) {
                try {
                    br.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }

            if (null != is) {
                try {
                    is.close();
                } catch (IOException e) {
                    e.printStackTrace();
```

```
                }
            }

            connection.disconnect();// 关闭远程连接
        }

        return result;
    }

    /**
     *
     * @param httpUrl  请求的url
     * @param param  form表单的参数（key,value形式）
     * @return
     */
    //发送post请求，注意请求微信小程序接口的格式是"Content-Type":"application/json"
    public static String doPostForm(String httpUrl, Map param) {

        HttpURLConnection connection = null;
        InputStream is = null;
        OutputStream os = null;
        BufferedReader br = null;
        String result = null;
        try {
            URL url = new URL(httpUrl);
            // 通过远程url连接对象打开连接
            connection = (HttpURLConnection) url.openConnection();
            // 设置连接请求方式
            connection.setRequestMethod("POST");
            // 设置连接主机服务器超时时间：15000毫秒
            connection.setConnectTimeout(15000);
            // 设置读取主机服务器返回数据超时时间：60000毫秒
            connection.setReadTimeout(60000);

            // 默认值为：false，当向远程服务器传送数据/写数据时，需要设置为true
            connection.setDoOutput(true);
            // 默认值为：true，当前向远程服务读取数据时，设置为true，该参数可有可无
            connection.setDoInput(true);
            // 设置传入参数的格式:请求参数应该是 name1=value1&name2=value2 的形式。
            connection.setRequestProperty("Content-Type",
"application/json");
            // 设置鉴权信息：Authorization: Bearer da3efcbf-0845-4fe3-8aba-
ee040be542c0
            //connection.setRequestProperty("Authorization", "Bearer
da3efcbf-0845-4fe3-8aba-ee040be542c0");
            // 通过连接对象获取一个输出流
            os = connection.getOutputStream();
            // 通过输出流对象将参数写出去/传输出去,它是通过字节数组写出的(form表单形式
```

```
的参数实质也是key,value值的拼接，类似于get请求参数的拼接)
            JSONObject json = new JSONObject(param);
            os.write( createLinkString(param).getBytes() );
            // 通过连接对象获取一个输入流，向远程读取
            if (connection.getResponseCode() == 200) {

                is = connection.getInputStream();
                // 对输入流对象进行包装:charset根据工作项目组的要求来设置
                br = new BufferedReader(new InputStreamReader(is, "UTF-8"));

                StringBuffer sbf = new StringBuffer();
                String temp = null;
                // 循环遍历一行一行读取数据
                while ((temp = br.readLine()) != null) {
                    sbf.append(temp);
                    sbf.append("\r\n");
                }
                result = sbf.toString();
            }
        } catch (MalformedURLException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            // 关闭资源
            if (null != br) {
                try {
                    br.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
            if (null != os) {
                try {
                    os.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
            if (null != is) {
                try {
                    is.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
            // 断开与远程地址url的连接
            connection.disconnect();
```

```
        }
        return result;
    }

    //发送"application/json"格式的POST请求
    public static String doPostForm(String httpUrl, JSONObject param) {

        HttpURLConnection connection = null;
        InputStream is = null;
        OutputStream os = null;
        BufferedReader br = null;
        String result = null;
        try {
            URL url = new URL(httpUrl);
            // 通过远程url连接对象打开连接
            connection = (HttpURLConnection) url.openConnection();
            // 设置连接请求方式
            connection.setRequestMethod("POST");
            // 设置连接主机服务器超时时间：15000毫秒
            connection.setConnectTimeout(15000);
            // 设置读取主机服务器返回数据超时时间：60000毫秒
            connection.setReadTimeout(60000);

            // 默认值为：false，当向远程服务器传送数据/写数据时，需要设置为true
            connection.setDoOutput(true);
            // 默认值为：true，当前向远程服务读取数据时，设置为true，该参数可有可无
            connection.setDoInput(true);
            // 设置传入参数的格式:请求参数应该是 name1=value1&name2=value2 的形式。
            connection.setRequestProperty("Content-Type",
"application/json");
            // 设置鉴权信息：Authorization: Bearer da3efcbf-0845-4fe3-8aba-
ee040be542c0
            //connection.setRequestProperty("Authorization", "Bearer
da3efcbf-0845-4fe3-8aba-ee040be542c0");
            // 通过连接对象获取一个输出流
            os = connection.getOutputStream();
            // 通过输出流对象将参数写出去/传输出去,它是通过字节数组写出的(form表单形式
的参数实质也是key,value值的拼接，类似于get请求参数的拼接)

            os.write( param.toString().getBytes() );
            // 通过连接对象获取一个输入流，向远程读取
            if (connection.getResponseCode() == 200) {

                is = connection.getInputStream();
                // 对输入流对象进行包装:charset根据工作项目组的要求来设置
                br = new BufferedReader(new InputStreamReader(is, "UTF-8"));

                StringBuffer sbf = new StringBuffer();
```

```
                String temp = null;
                // 循环遍历一行一行读取数据
                while ((temp = br.readLine()) != null) {
                    sbf.append(temp);
                    sbf.append("\r\n");
                }
                result = sbf.toString();
            }
        } catch (MalformedURLException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            // 关闭资源
            if (null != br) {
                try {
                    br.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
            if (null != os) {
                try {
                    os.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
            if (null != is) {
                try {
                    is.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
            // 断开与远程地址url的连接
            connection.disconnect();
        }
        return result;
    }



    /**
     * 把数组所有元素排序，并按照“参数=参数值”的模式用“&”字符拼接成字符串
     * @param params 需要排序并参与字符拼接的参数组
     * @return 拼接后字符串
     */
```

```
    public static String createLinkString(Map<String, String> params) {

        List<String> keys = new ArrayList<String>(params.keySet());
        Collections.sort(keys);

        StringBuilder prestr = new StringBuilder();
        for (int i = 0; i < keys.size(); i++) {
            String key = keys.get(i);
            String value = params.get(key);
            if (i == keys.size() - 1) {// 拼接时，不包括最后一个&字符
                prestr.append(key).append("=").append(value);
            } else {
                prestr.append(key).append("=").append(value).append("&");
            }
        }

        return prestr.toString();
    }

    //String 转 Map
    public static Map<String,Object> strToJson(String jsonString) {
        Gson gson = new Gson();
        Map<String, Object> map = new HashMap<String, Object>();
        map = gson.fromJson(jsonString, map.getClass());
        return map;
    }


}
```

StatusCode.java

```
package com.example.demo.utils;

import lombok.Data;

import java.util.HashMap;
import java.util.Map;

/**
 * status 状态码 0-表示失败，1-表示成功
 * error_code 错误码，一般在设计时定义
 * error_des 错误描述，一般在设计时定义
 * data 成功数据
 * msg 请求成功 / 请求失败
```

```
 * @author czh
 * */
@Data
public class StatusCode {
    /**
    * int status;
    * String error_code;
    * String error_des;
    * * Object data;
    */
    public static Map<String,Object> success(Object data){
        Map<String,Object> tmp = new HashMap<>();
        tmp.put("status",1);
        tmp.put("data",data);
        tmp.put("msg","请求成功");
        return tmp;
    }
    public static Map<String, Object> error(int error_code){
        Map<String,Object> tmp = new HashMap<>();
        tmp.put("status",0);
        tmp.put("msg","请求失败");
        tmp.put("error_code",error_code);
        String error_des = "";
        // 以下待完善
        if( error_code == 1001 ){
            error_des = "参数无效";
        }else if( error_code == 1002 ){
            error_des = "参数缺失";
        }else if( error_code == 2001 ){
            error_des = "用户未登录";
        }else if( error_code == 3001 ){
            error_des = "服务器错误";
        }
        tmp.put("error_des",error_des);
        return tmp;
    }

}
```

DemoApplication.java

```
package com.example.demo;

import org.mybatis.spring.annotation.MapperScan;
import org.springframework.boot.SpringApplication;
```

```java
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.ComponentScan;

@SpringBootApplication

public class DemoApplication {
    public static void main(String[] args) {
        SpringApplication.run(DemoApplication.class, args);
    }

}
```