易助 技术设计文档

Title	Description
作者	王宜锋
创建日期	2017-07-06
内容	易助系统后台技术设计及实现细节

编辑历史

Time	Version	Description
2017-07-06	v1.0	初始版本

0. 核心

框架

1.主体框架使用 Spring；
2.MVC 框架使用 SpringMVC+Thymeleaf；
3.ORM 持久层框架使用 Hibernate。

功能实现

1.使用 JSON 进行数据交互；
2.使用 MySQL 作为后端数据库。

1. 目录结构

```
yizhu
    - src
        - main
            - java
                - com.sysu.yizhu
                    - business              # 软件业务包
                        - entities          # 实体
                            - repositories  # 实体持久层仓库
                        - services          # 服务层
                    - util                  # 工具
                    - web                   # web层
                        - controller        # 接口控制器路由
                        - interceptor       # 拦截器
            - resources                     # 配置资源文件
            - webapp
                - WEB-INF                   # 配置文件
        - test
            - java
                - test
    - pom.xml                               # 项目管理
```

2. 外部系统

```
database
    mysql
        user: root
        pass: 123456
```

3. 控制器路由

3.1 实现

```
# 位于com.sysu.yizhu.web.controller包中
    - UserController.java        # 实现用户类接口
    - QuestionController.java    # 实现提问、回答类接口
    - SOSController.java         # 实现一键求救接口
    - HelpController.java        # 实现求助功能接口
```

所有接口通过获取HTTP请求GET、POST中的路径参数或请求参数，调用服务层方法，并返回结果内容。

3.2 返回值

```
# 位于com.sysu.yizhu.util包中
# 继承自LinkedHashMap<String, Object>，通过Spring返回值格式化为JSON格式
    - ReturnMsg.java
```

3.3 拦截器

```
# 位于com.sysu.yizhu.web.interceptor包中
    - LoginInterceptor.java
```

通过拦截用户请求，区分请求方法，POST方法需要登录。
将用户记录于session中。

4. 业务层

4.1 向上接口实现

```
# 位于com.sysu.yizhu.business.services包中
    - UserService.java
    - QuestionService.java
    - SOSService.java
    - HelpService.java
```

介于数据库持久层与控制器层中间，集成数据库操作并向控制器提供业务接口。

4.2 实体

```
# 位于com.sysu.yizhu.business.entities包中
# 实体设计
-  User
-  Comment
-  Question
-  Answer
-  AgreeAnswer
-  SOS
-  SOSResponse
-  Help
-  HelpResponse


# 持久层设计
- repositories
```

详情见数据库设计
https://github.com/SYSU-yizhu/yizhu/blob/master/docs/database_design.png

框架

使用Hibernate持久层框架，将实体映射为数据库的表结构。
通过JPA注解设计数据库，如SOSResponse实体

```
// 定义实体
@Entity
// 定义表名
@Table(name="sos_response")
public class SOSResponse {
    // 定义各列，面向对象设计
    private Integer sosResponseId;
    private SOS sos;
    private User sosResponseUser;

    public SOSResponse() {
    }

    // Id列，可设置自动增长，并设置列名
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    @Column(name = "sos_response_id")
```

```
    public Integer getSosResponseId() {
        return sosResponseId;
    }
    public void setSosResponseId(Integer sosResponseId) {
        this.sosResponseId = sosResponseId;
    }

    // 多对一，建立外键，以及设置懒加载
    @ManyToOne(cascade = {CascadeType.MERGE}, fetch = FetchType.LAZY)
    @JoinColumn(name = "sos_id")
    public SOS getSos() {
        return sos;
    }
    public void setSos(SOS sos) {
        this.sos = sos;
    }

    @ManyToOne(cascade = {CascadeType.MERGE}, fetch = FetchType.LAZY)
    @JoinColumn(name = "user_id")
    public User getSosResponseUser() {
        return sosResponseUser;
    }
    public void setSosResponseUser(User sosResponseUser) {
        this.sosResponseUser = sosResponseUser;
    }
}
```

4.3 数据库方法接口

使用Hibernate提供的JPA CRUD类模板，如HelpResponseRepository

```
// 注解仓库
@Repository
public interface HelpResponseRepository extends CrudRepository<HelpResponse, Integer> {
    // 使用查询语句注解，模板将自动将参数传入执行SQL并将结果写入对象返回
    @Query("select r.helpResponseUser.userId from HelpResponse r where r.help.helpId = ?1")
    List<String> findAllUserIdByHelpId(Integer sosId);

    // 使用模板方法名，By关键字后为属性名，定义传入属性，自动生成并执行SQl，获得结果返回
    List<HelpResponse> findByHelpAndHelpResponseUser(Help help, User user);
}
```

5. 工具类

```
# 位于com.sysu.yizhu.util包中
```

5.1 推送、短信服务

```
# 使用leancloud提供的服务
- LCConfig   # 读取leancloud配置数据，如app-key
- LCUtil     # 自定义封装leancloud服务，便于调用
```

封装了：短信验证服务；消息推送服务。
通过包装LeanCloud接口服务，给易助前端提供更简单的调用方式。
如消息推送服务：

```
// LCUtil类内方法
// 字符串模板，包装了一个leancloud请求的语句
private static final String REQ_PUSH_SOS_TEMPLATE =
        "{\n" +
        "    \"where\": {\n" +
        "          \"location\": {\n" +
        "            \"$nearSphere\": {\n" +
        "              \"__type\": \"GeoPoint\",\n" +
        "              \"latitude\": {latitude},\n" +
        "              \"longitude\": {longitude}\n" +
        "            },\n" +
```

```
        "            \"$maxDistanceInMiles\": {maxDistance}\n" +
        "          }\n" +
        "    },\n" +
        "    \"data\": {\n" +
        "        \"alert\": \"{alertContent}\"\n" +
        "    }\n" +
        "}";

// 自定义方法，通过替代占位符达到传参的目的
private String getPushSOSJSON(Double latitude, Double longitude, Double maxDistance, String alertContent) {
    return REQ_PUSH_SOS_TEMPLATE.replace("{latitude}", latitude.toString())
            .replace("{longitude}", longitude.toString())
            .replace("{maxDistance}", maxDistance.toString())
            .replace("{alertContent}", alertContent);
}

// 对外接口方法，在Http头中封装了信息，直接发送即可
public boolean pushHelp(Double latitude, Double longitude) {
    try {
        HttpEntity<String> reqEntity = new HttpEntity<String>(
                getPushHelpJSON(latitude, longitude, 100.0, "一条新的求助消息！"),
                headers);
        String res = restTemplate.postForObject(URL_PUSH, reqEntity, String.class);
        LOG.info("pushSOS() res: " + res);
        return true;
    } catch (HttpClientErrorException e) {
        LOG.error("pushSOS() res: " + e.getMessage());
        LOG.error("pushSOS() res: " + e.getResponseBodyAsString());
        return false;
    }
}
```

5.2 MD5计算

```
- MD5Parser
    - static getMD5(String)    # 将传入字符串计算MD5值并返回
```

5.3 数据库MySQL自定义连接配置类

```
- MySQL5Dialect   # 配置了innoDB和建表编码utf-8
```

5.4 手机号码检验

```
- PhoneNumUtil
    - static isPhone()   # 检验传入手机号是否为有效格式的手机号
```

检验使用了正则表达式。

5.5 返回消息

见控制器层叙述。

6. 配置资源文件

```
# 位于yizhu/src/main/resources中
```

6.1 LeanCloud配置文件

```
- leancloud.properties    # 需要X-LC-Id，X-LC-Key字段，由leancloud官网提供
```

6.2 log4j日志系统配置

```
- log4j.properties        # 配置log4j插件的具体项
```

使用AOP的编程模式。

6.3 各运行环境配置

```
- settings-development.properties   # 开发环境配置
- settings-jenkins.properties       # 集成测试环境配置
- settings-production.properties    # 正式环境配置

# 均有字段
db.user
db.password
db.jdbcUrl
```

7. 系统运行部署

见 《部署文档》

8. 对外接口

8.1 用户类

8.1.1 注册前发送短信验证码

Code	Content	Description
200	OK	请求成功
400	FAILED	手机号已存在
403	FORBIDDEN	手机号（用户名）无效

URI:

```
GET /user/sendSms/{userId}
```

GET参数

字段	描述	类型
userId	用户名（手机号）	string

成功例子：

```
{
    "userId":"11111111111"
}
```

8.1.2 注册

Code	Content	Description
200	OK	请求成功
400	FAILED	用户名已存在
403	FORBIDDEN	用户名不是手机号或参数格式错误
450	MISS	验证码错误

URI:

```
POST /user/register
```

POST参数

字段	描述	类型
userId	用户名（手机号）	string
password	密码（未处理）	string
code	短信验证码	string
name	姓名	string
gender	性别	string - "male"/"female"
birthDate	出生日期	string - YYYY-MM-DD
location	常住地	string

成功例子：

```
{
    "userId":"11111111111"
}
```

8.1.3 登录

Code	Content	Description
200	OK	请求成功
403	FORBIDDEN	用户名不是手机号或参数格式错误
400	NOT FOUND	用户名或密码错误

URI:

```
POST /user/login
```

POST参数

字段	描述	类型
userId	用户名（手机号）	string
password	密码（未处理）	String

成功例子：

```
{
    "userId":"11111111111"
}
```

8.1.4 修改个人信息

Code	Content	Description
200	OK	请求成功
401	FAILED	未登录
403	FORBIDDEN	性别或日期参数格式不正确

URI:

```
POST /user/modifyInfo
```

POST参数

字段	描述	类型
name	姓名	string
gender	性别	string - "male"/"female"
birthDate	出生日期	string - YYYY-MM-DD
location	常住地	string

成功例子：

```
{
    "userId":"11111111111"
}
```

8.1.5 获取个人信息

Code	Content	Description
200	OK	请求成功
401	FAILED	未登录

URI:

```
POST /user/info
```

POST参数：无

成功例子：

```
{
    "userId":"11111111111",
    "name":"张三",
    "gender":"male",
    "birthDate":"1995-12-02",
    "location":"中山大学"
}
```

8.1.6 （推送）更新用户objectId（leancloud SDK获取）

该接口需要登录

Code	Content	Description

200	OK	请求成功
401	FAILED	未登录
404	MISS	objectId不存在

URI:

```
POST /user/updateObjectId
```

POST参数

字段	描述	类型
objectId	安装Id	string

成功例子：

```
{
    "userId":"11111111111"
}
```

8.1.7 （推送）更新用户定位位置

该接口需要登录

Code	Content	Description
200	OK	请求成功
401	FAILED	未登录
403	FORBIDDEN	纬度不在-90~90或经度不在-180~180间
450	MISS	用户未记录安装Id
500	Error	服务器错误

URI:

```
POST /user/updateLocation
```

POST参数

字段	描述	类型
latitude	纬度	double
longitude	经度	double

成功例子：

```
{
    "userId":"11111111111"
}
```

8.2 提问类

8.2.1 提问

Code	Content	Description
200	OK	请求成功
401	FAILED	未登录

URI:

```
POST /question/ask
```

POST参数

字段	描述	类型
title	标题	string
content	提问内容	string

成功例子：

```
{
    "questionId":1
}
```

8.2.2 回答

Code	Content	Description
200	OK	请求成功
401	FAILED	未登录
450	MISS	该提问id不存在

URI:

```
POST /question/answer
```

POST参数

字段	描述	类型
questionId	问题id	int
content	回答内容	string

成功例子：

```
{
    "answerId":1
}
```

8.2.3 赞同

Code	Content	Description

200	OK	请求成功
401	FAILED	未登录
450	MISS	该回答id不存在

URI:

```
POST /question/agreeAnswer
```

POST参数

字段	描述	类型
answerId	回答id	int
agreeOrNot	是否赞同	bool - true赞同/false不赞同

成功例子：

```
{
    "answerAgreeId":1
}
```

8.2.4 获取所有问题Id

Code	Content	Description
200	OK	请求成功

URI:

```
GET /question/getAllId
```

成功例子：

```
{
    "count":2,
    "data":[1,2]
}
```

8.2.5 根据问题Id获取问题摘要

Code	Content	Description
200	OK	请求成功
404	NOT FOUND	questionId不存在

URI:

```
GET /question/digest/{questionId}
```

成功例子：

```
{
    "questionId":1,
    "userId":"133133123456",
```

```
    "userName":"张三",
    "title":"扶老奶奶过马路是一种怎样的体验？",
    "createDate":"2017-05-17"
}
```

8.2.6 根据问题Id获取完整问题内容

Code	Content	Description
200	OK	请求成功
404	NOT FOUND	questionId不存在

URI:

```
GET /question/detail/{questionId}
```

成功例子：

```
{
    "questionId":1,
    "userId":"133133123456",
    "userName":"张三",
    "title":"扶老奶奶过马路是一种怎样的体验？",
    "content":"bla bla blabla",
    "createDate":"2017-05-17"
}
```

8.2.7 根据问题Id获取回答所有Id

Code	Content	Description
200	OK	请求成功
404	NOT FOUND	questionId不存在

URI:

```
GET /question/getAnswerIds/{questionId}
```

成功例子：

```
{
    "count":2,
    "data":[1,2]
}
```

8.2.8 根据回答Id获取回答内容

Code	Content	Description
200	OK	请求成功
404	NOT FOUND	answerId不存在

URI:

```
GET /question/getAnswer/{answerId}
```

成功例子：

```
{
    "answerId":1,
    "userId":"133133123456",
    "userName":"张三",
    "content":"无可奉告",
    "createDate":"2017-05-17",
    "good":5,
    "bad":30
}
```

8.3 一键求救（推送、导航、评价）

8.3.1 发起求救

该接口需要登录

Code	Content	Description
200	OK	请求成功
401	FAILED	未登录
403	FORBIDDEN	纬度不在-90~90或经度不在-180~180间
450	MISS	用户未记录安装Id
500	Error	服务器错误

URI:

```
POST /sos/push
```

POST参数

字段	描述	类型
latitude	纬度	double
longitude	经度	double

成功例子：

```
{
    "sosId":1
}
```

8.3.2 响应求救

该接口需要登录

Code	Content	Description
200	OK	请求成功
401	FAILED	未登录
403	FORBIDDEN	已响应，不能重复响应
404	NOT FOUND	SOS id 不存在或已完成

450	MISS	用户未记录安装Id

URI:

```
POST /sos/response
```

POST参数

字段	描述	类型
sosId	求救Id	string

成功例子：

```
{
    "userId":"11111111111"
}
```

8.3.3 查询所有有效求救id

Code	Content	Description
200	OK	请求成功

URI:

```
GET /sos/allValidId
```

成功例子：

```
{
    "count":2,
    "data":[1,2]
}
```

8.3.4 根据sos id获取sos内容

Code	Content	Description
200	OK	请求成功
404	NOT FOUND	SOS id 不存在

URI:

```
GET /sos/get/{sosId}
```

成功例子：

```
{
    "sosId": 2,
    "latitude": 0.0,
    "longitude": 5.0,
    "createTime":"2017-06-27 10:15",
    "finished":false,
    "pushUserId": "11111111111"
}
```

8.3.5 根据sos id获取所有响应者id

Code	Content	Description
200	OK	请求成功
404	NOT FOUND	SOS id 不存在或已完成

URI:

```
GET /sos/response/{sosId}
```

成功例子：

```
{
    "count":2,
    "data":["1234568911", "12345678922"]
}
```

8.3.6 结束求救

该接口需要登录

Code	Content	Description
200	OK	请求成功
401	FAILED	未登录
404	NOT FOUND	SOS id 不存在或已完成
450	MISS	非发起用户无法结束该求救

URI:

```
POST /sos/finish
```

POST参数

字段	描述	类型
sosId	sos id	integer

成功例子：

```
{
    "sosId":2
}
```

8.4 求助 （图片、文字、评价）

8.4.1 发起求助

该接口需要登录

Code	Content	Description
200	OK	请求成功

401	FAILED	未登录
402	WRONG	需求人数无效，应在1-10人之间
403	FORBIDDEN	纬度不在-90~90或经度不在-180~180间
450	MISS	用户未记录安装Id
500	Error	服务器错误

URI:

```
POST /help/push
```

POST参数

字段	描述	类型
latitude	纬度	double
longitude	经度	double
title	事件标题	string
detail	事件信息	string
needs	需要多少帮助者	integer

成功例子：

```
{
    "helpId":1
}
```

8.4.2 响应求助

该接口需要登录

Code	Content	Description
200	OK	请求成功
401	FAILED	未登录
402	WRONG	该求助人数已满
403	FORBIDDEN	已响应，不能重复响应
404	NOT FOUND	Help id 不存在或已完成
450	MISS	用户未记录安装Id

URI:

```
POST /help/response
```

POST参数

字段	描述	类型

helpId	求助Id	string

成功例子：

```
{
    "userId":"11111111111"
}
```

8.4.3 查询所有有效求助id

Code	Content	Description
200	OK	请求成功

URI:

```
GET /help/allValidId
```

成功例子：

```
{
    "count":2,
    "data":[1,2]
}
```

8.4.4 根据help id获取求助内容

Code	Content	Description
200	OK	请求成功
404	NOT FOUND	Help id 不存在

URI:

```
GET /help/get/{helpId}
```

成功例子：

```
{
    "helpId": 2,
    "latitude": 0.0,
    "longitude": 5.0,
    "finished":false,
    "title":"帮忙抬米",
    "detail":"抬三袋米上五楼",
    "needs":3,
    "responseNum":2,
    "pushUserId": "11111111111"
}
```

8.4.5 根据help id获取所有响应者id

Code	Content	Description
200	OK	请求成功
404	NOT FOUND	Help id 不存在或已完成

URI:

```
GET /help/response/{helpId}
```

成功例子：

```
{
    "count":2,
    "data":["1234568911", "12345678922"]
}
```

8.4.5 结束求助

该接口需要登录

Code	Content	Description
200	OK	请求成功
401	FAILED	未登录
404	NOT FOUND	Help id 不存在或已完成
450	MISS	非发起用户无法结束该求救

URI:

```
POST /help/finish
```

POST参数

字段	描述	类型
helpId	help id	integer

成功例子：

```
{
    "helpId":2
}
```