**《网络程序设计实训》课程报告**

**题目：** **基于Mqtt的兼职系统设计**

**班级：**

**学号：**

**姓名：**

目录

[一、 功能需求 3](#_Toc50638867)

[1. 用户管理 3](#_Toc50638868)

[2. 兼职管理 3](#_Toc50638869)

[二、 项目结构简介 4](#_Toc50638870)

[1. 项目技术 4](#_Toc50638871)

[2．项目目录结构 4](#_Toc50638872)

[3．项目主要类 4](#_Toc50638873)

[三、 项目演示 5](#_Toc50638874)

[四、 项目代码 12](#_Toc50638875)

[五、 小结 48](#_Toc50638876)

# 功能需求

**表1.1. 功能需求列表**

|  |  |  |
| --- | --- | --- |
| 一级功能 | 二级功能 | 附注 |
| 1. 用户管理 | 1.1 用户注册 |  |
| 1.2 用户登录 |  |
| 1.3 用户退出 |  |
| 1. 兼职管理 | 2.1 创建兼职 | 只在这部进行了检查登录处理 |
| 2.2 浏览兼职 |  |
| 2.3加入兼职 |  |
| 2.8 离开兼职 |  |
| 2.9 发布信息 |  |

## 用户管理

负责用户基本信息的处理。

* 1. 用户注册

允许任意用户注册，提交用户基本信息。注册后，系统存储用户信息。

* 1. 用户登录

用户提供账号、密码进行登录。

* 1. 用户退出

用户退出

## 兼职管理

负责聊天室的正常运作、维护。

* 1. 创建兼职

任何用户均可创建聊天室，提供兼职的基础信息，需管理员审批通过。

* 1. 浏览兼职

任何用户均可浏览当前已经创建的兼职信息。

* 1. 加入聊天室

任何用户均可申请加入某个聊天室，直接加入。

* 1. 退出兼职

某个用户直接退出某个兼职。

* 1. 发布信息

兼职创建者发布相应信息。

# 项目结构简介

## 项目技术

该项目基于maven进行构建，基于Java语言的实现。在用户交互方面，使用了最原始的控制台输入输出进行数据的交互。使用到了Jar包分别为org.eclipse.paho.client.mqttv3和gson，由maven仓库进行统一的jar包管理。org.eclipse.paho.client.mqttv3主要提供了mqtt相关api的操作接口，gson用于规范化数据的传输。

## 2．项目目录结构

项目主要文件为dos、mosquitto、src、pom.xml。dos文件夹存放相应简易文档；mosquitto中存放Mqtt broker；src存放业务主要代码、pom.xml为maven的jar包依赖。

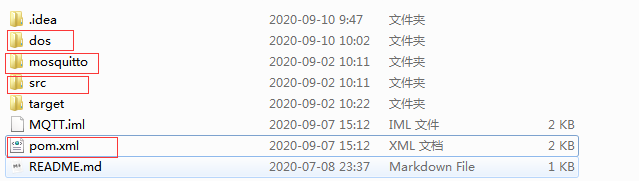


图2.1 项目结构目录

## 3．项目主要类

3.1．业务处理

mqtt基于发布订阅的操作，通过为mqttclient对象设置回调函数进行消息的获取。故在该系统中，callback包下的ClientCallBack与MessageCallBack两个类用于分别处理客户端收到的消息回调处理和服务端收到的消息回调处理。

3.2．数据存储

该系统数据存放于内存中，暂不与数据库进行交互。通过static静态变量来保存对应的数据。数据结构主要使用java.util.ArrayList动态数组进行存放，根据存放的数据不同，数据对象的类型也不同。对于保存客户端信息来说，类型使用String字符串进行存储；对于兼职信息和用户登录信息来说，设计到较多成员变量，故使用实体类进行存放。

数据存储类在db包下，MqttConnections维护两个ArrayList成员变量，分别用于存储登录队列和登陆账号用户名（clientId）队列。PartTimeManger维护一个ArrayList，存放兼职信息的队列。

3.3．实体类

在Java中可以通过实体类对信息进行简单的管理。位于entity包下。

3.4．系统启动

用于启动客户端和服务端的两个类。位于client和server包下。

3.5．工具类

用于接收到对应的主题进行数值转换方便判断的工具。位于utils包下。

# 项目演示

1. 登录

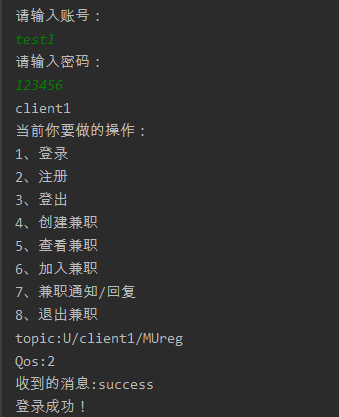


图3.1 客户端登陆

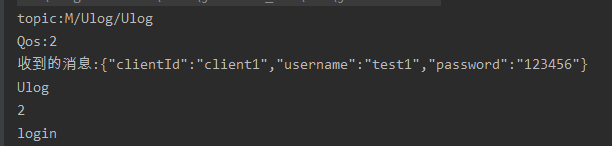


图3.2 服务端处理

1. 注册

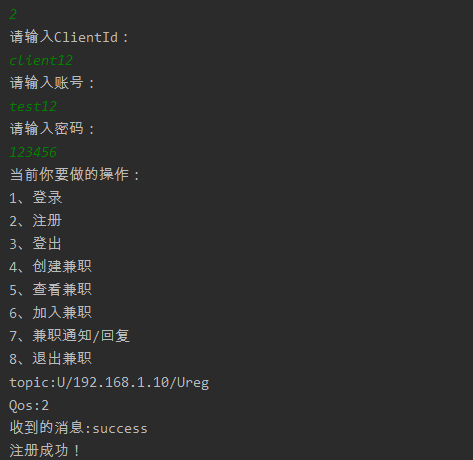


图3.3 客户端注册

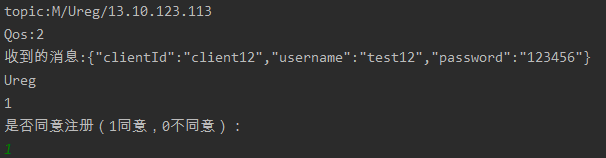


图3.4 服务端消息处理

1. 登出

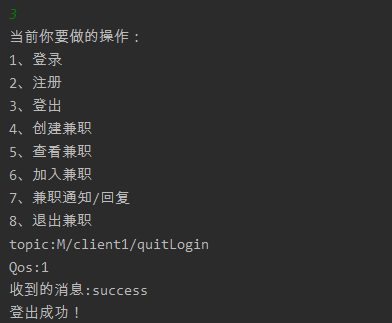


图3.5 客户端登出

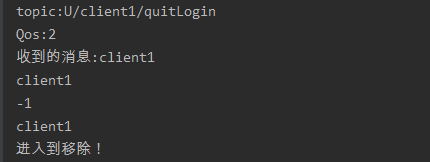


图3.6 服务端消息

1. 创建兼职

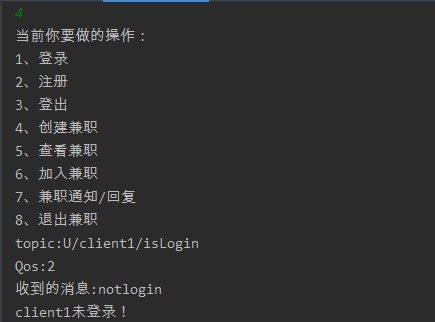


图3.7 客户端未登录创建兼职

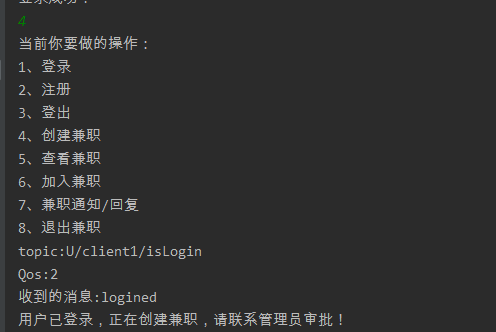


图3.8 客户端已登录创建兼职

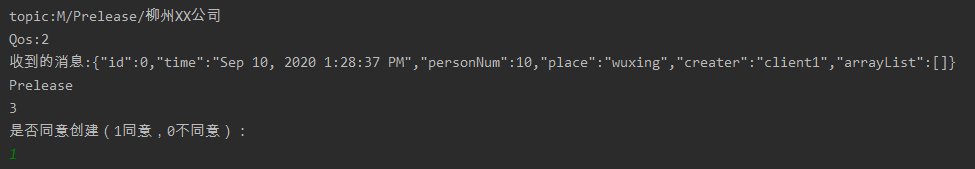


图3.9 管理员审批

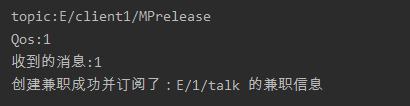


图3.10 管理员审批成功

1. 浏览兼职

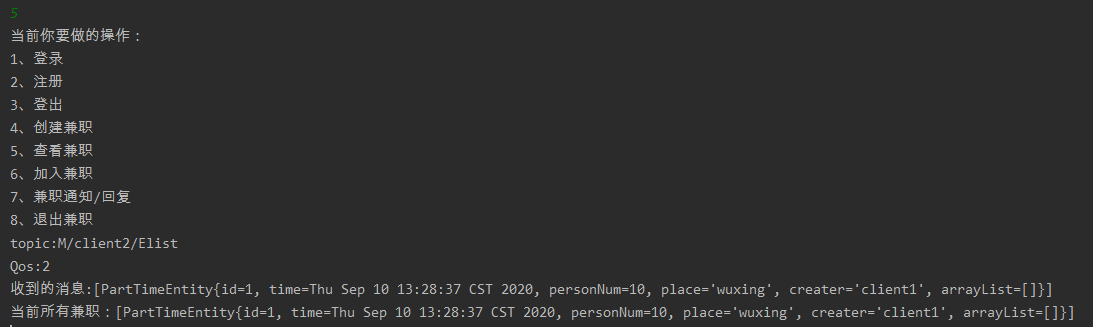


图3.11 浏览兼职

1. 加入兼职

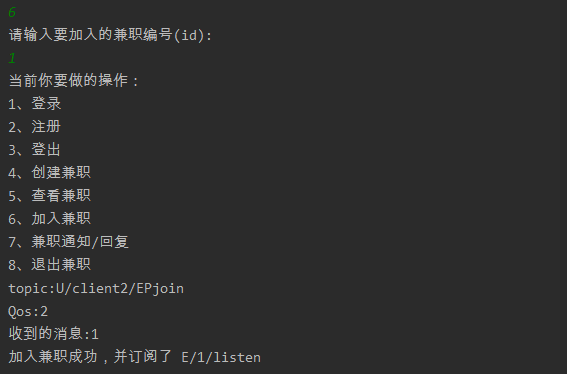


图3.12 加入兼职

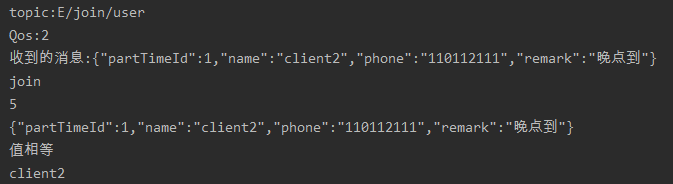


图3.12 服务端响应

1. 退出兼职

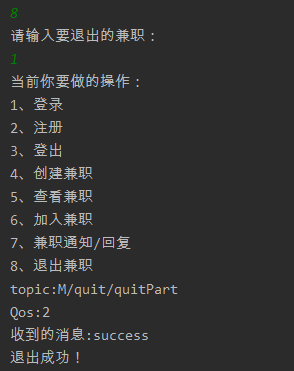


图3.13 退出兼职

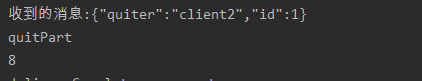


图3.14 服务端消息

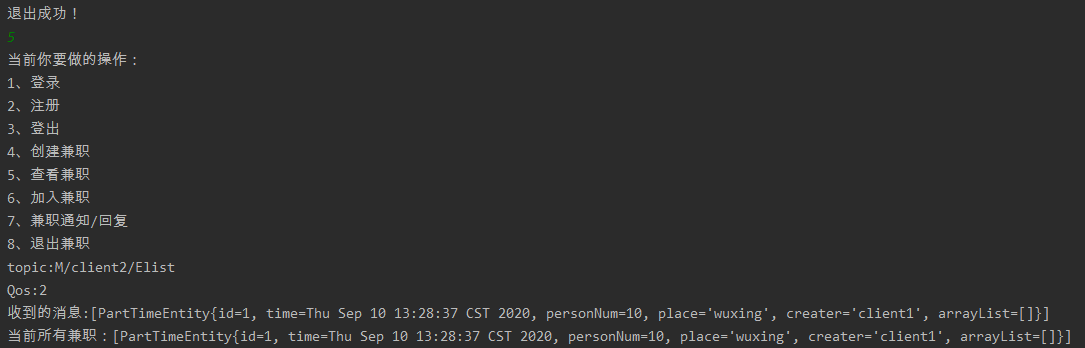


图3.15 再次浏览兼职

1. 信息发布



图3.16 留言内容

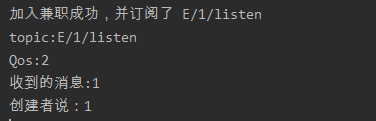


图3.17 进入兼职的客户端收到的消息

# 项目代码

1. callback.ClientCallBack

|  |
| --- |
| package mqtthomework.callback;  import com.google.gson.Gson; import mqtthomework.db.MqttConnections; import mqtthomework.entity.PartTimeEntity; import mqtthomework.utils.ClientCaoZuo; import org.eclipse.paho.client.mqttv3.\*;  import java.util.Date; import java.util.HashMap; import java.util.Scanner;  */\*\*  \** ***@PackageName*** *mqtthomework.db  \** ***@Classname*** *ClientCallBack  \** ***@Description*** *客户端回调函数，用于接收来自服务端的消息推送  \** ***@Author*** *flj  \** ***@Version*** *1.0.0  \*/* public class ClientCallBack implements MqttCallback {   private MqttClient mqttClient;   private String mqttClientName;   public ClientCallBack(MqttClient mqttClient, String mqttClientName){  this.mqttClient = mqttClient;  this.mqttClientName = mqttClientName;  }   @Override  public void connectionLost(Throwable cause) {  // 掉线就死了，暂时不处理  System.*out*.println("掉线了，请重新启动连接......");  }   @Override  public void messageArrived(String topic, MqttMessage message) throws Exception {  String content = new String(message.getPayload());  System.*out*.println("topic:"+topic);  System.*out*.println("Qos:"+message.getQos());  System.*out*.println("收到的消息:"+content);  doing(topic,content);  }     // 选择执行的操作  private void doing(String topic, final String message) throws MqttException {  String[] split = topic.split("/");  String whatToDo = split[2];  int code = ClientCaoZuo.*getCode*(whatToDo);  // 如果是登录返回的消息  if (code == 1) {  showLoginInfo(message);  } else if (code == 2) {  // 成功创建兼职  successCreatePartTime(message);  } else if (code == 3 && split[0].equals("U")) {  // 成功加入兼职  successJoinPartTime(message);  } else if (code == 4) {  // 检查是否处于登录状态  checkIsLogined(message);  } else if (code == 5) {  // 注册情况  registerRelation(message);  } else if (code == 6) {  // 检查该用户是否有创建兼职  checkCreatePartTime(message);  } else if(code == 7){  quitSuccess(message);  }else if(code == 8){  // 登出  loginOut(message);  }else if(code ==9){  // 查看兼职列表  partTimeList(message);  }else if(code == 10){  listen(message);  }   }   // 收到监听的消息  private void listen(String message) {  System.*out*.println("创建者说："+message);  }   // 兼职列表  private void partTimeList(String message) throws MqttException {  System.*out*.println("当前所有兼职："+message);  }   //登出  private void loginOut(String message) throws MqttException {  if("success".equals(message)){  System.*out*.println("登出成功！");  this.mqttClient.unsubscribe("M/quit/quitPart");  this.mqttClient.unsubscribe("E/#");  }else{  System.*out*.println("登出失败，原因"+message);  }  }    // 退出兼职成功  private void quitSuccess(String message) {  if("success".equals(message)){  System.*out*.println("退出成功！");  }  }   // 检查是否有创建兼职  private void checkCreatePartTime(String message) throws MqttException {  if ("falie".equals(message)) {  System.*out*.println("该用户为创建任何兼职");  } else {  while(true) {  System.*out*.println("请输入要留言的话：");  Scanner scanner = new Scanner(System.*in*);  String next = scanner.next();  if(next.equals("exit")){  break;  }  System.*out*.println("留言内容：" + next);  MqttMessage mqttMessage = new MqttMessage(next.getBytes());  mqttMessage.setQos(2);  this.mqttClient.publish("E/" + message + "/listen", mqttMessage);  }  this.mqttClient.unsubscribe("M/"+this.mqttClientName+"/check");  }  }   // 注册情况  private void registerRelation(String message) throws MqttException {  if ("success".equals(message)) {  System.*out*.println("注册成功！");  } else {  System.*out*.println("注册失败，原因：" + message);  }  // 不管成功与否都取消订阅  this.mqttClient.unsubscribe("U/#");  }     // 检查是否处于登录  private void checkIsLogined(String message) throws MqttException {  if ("logined".equals(message)) {  // 封装成JSON  System.*out*.println("用户已登录，正在创建兼职，请联系管理员审批！");  Gson gson = new Gson();  final PartTimeEntity partTimeEntity = new PartTimeEntity(new Date(), 10, "wuxing", this.mqttClientName);  String partTimeJson = gson.toJson(partTimeEntity);  // 封装成MQTT可发送的执行流  MqttMessage sendMessage = new MqttMessage(partTimeJson.getBytes());  sendMessage.setQos(2);  mqttClient.publish("M/Prelease/柳州XX公司", sendMessage);  // 开辟一个线程去订阅信息，等待数据  new Thread(new Runnable() {  public void run() {  try {  mqttClient.subscribe("E/"+partTimeEntity.getCreater()+"/MPrelease", 2);  } catch (MqttException e) {  e.printStackTrace();  }  }  }).start();  }else{  System.*out*.println(this.mqttClientName+"未登录！");  }  }   // 成功加入兼职的操作  private void successJoinPartTime(final String message) {  if (!"falie".equals(message)) {  new Thread(new Runnable() {  public void run() {  try {  mqttClient.subscribe("E/" + message + "/listen", 2);  } catch (MqttException e) {  e.printStackTrace();  }  }  }).start();  System.*out*.println("加入兼职成功，并订阅了 E/" + message +"/listen");  } else {  System.*out*.println("加入兼职失败!");  }  }   // 成功创建兼职的操作  private void successCreatePartTime(final String message) throws MqttException {  // 直接退订  mqttClient.unsubscribe("E/柳州XX公司/#");  new Thread(new Runnable() {  public void run() {  try {  // 订阅对应兼职号的ID  mqttClient.subscribe("E/" + message + "/talk", 2);  } catch (MqttException e) {  e.printStackTrace();  }  }  }).start();  System.*out*.println("创建兼职成功并订阅了：E/" + message + "/talk 的兼职信息");  }   //展示登录信息  private void showLoginInfo(String message) {  if ("success".equals(message)) {  System.*out*.println("登录成功！");  } else {  System.*out*.println("登录失败，原因：" + message);  }  }   // 不处理  @Override  public void deliveryComplete(IMqttDeliveryToken token) { } } |

1. callback.MessageCallBack

|  |
| --- |
| package mqtthomework.callback;  import com.google.gson.Gson; import mqtthomework.db.MqttConnections; import mqtthomework.db.PartTimeManger; import mqtthomework.entity.\*; import mqtthomework.server.ManServer; import mqtthomework.utils.CaoZuo; import org.eclipse.paho.client.mqttv3.\*;  import java.sql.SQLException; import java.util.Scanner;  */\*\*  \** ***@PackageName*** *mqtthomework.db  \** ***@Classname*** *MessageCallBack  \** ***@Description*** *服务端回调函数，用于接收来自服务端的消息推送  \** ***@Author*** *flj  \** ***@Version*** *1.0.0  \*/* public class MessageCallBack implements MqttCallback {   @Override  public void connectionLost(Throwable cause) {  // 掉线就死了，暂时不处理  System.*out*.println("掉线了，请重新启动连接......");  }   // 收到推送消息执行  @Override  public void messageArrived(String topic, MqttMessage message) throws Exception {  String content = new String(message.getPayload());  System.*out*.println("topic:"+topic);  System.*out*.println("Qos:"+message.getQos());  System.*out*.println("收到的消息:"+content);  // 根据主题和信息执行下一步的操作  *doing*(topic,content);  }   // 推送完毕执行  @Override  public void deliveryComplete(IMqttDeliveryToken token) {  System.*out*.println("deliveryComplete---------"+ token.isComplete());  }   // 解析传入的主题进行分割获取  private static void doing(String topic,String content) throws SQLException, MqttException {  String[] split = topic.split("/");  System.*out*.println(split[1]);  // 传入主题进行id的转换  int code = CaoZuo.*getCode*(split[1]);  System.*out*.println(code);  // 根据返回id处理信息  if(code == 1){  *register*(content);  }else if(code == 2){  *login*(content);  }else if(code == 3){  *createPartTime*(content);  }else if(code == 4){  *lookPartTime*(content);  }else if(code == 5){  *joinPartTime*(content);  }else if(code ==6){  *isLogin*(content);  }else if(code ==7 && split[0].equals("E")){  *check*(content);  }else if(code == 8 && split[0].equals("E")){  *quitPartTime*(content);  }else if(split[0].equals("U") && split[2].equals("quitLogin")){  *loginout*(content);  }  }   // 登出  private static void loginout(String content) throws MqttException {  MqttClient mqttClientEntity = ManServer.*getMqttClientEntity*();  // 直接拿来对不，找不到就没有登录  for (int i = 0; i < MqttConnections.*MqttLoginedList*.size() ; i++) {  String loginClientId = MqttConnections.*MqttLoginedList*.get(i);  System.*out*.println(loginClientId);  if(loginClientId.equals(content)){  System.*out*.println("进入到移除！");  MqttConnections.*MqttLoginedList*.remove(i);  MqttMessage message = new MqttMessage("success".getBytes());  mqttClientEntity.publish("M/"+content+"/quitLogin",message);  return;  }  }  MqttMessage message = new MqttMessage("falie！".getBytes());  message.setQos(2);  mqttClientEntity.publish("M/"+content+"/quitLogin",message);  }   // 退出兼职  private static void quitPartTime(String content) throws MqttException {  MqttClient mqttClientEntity = ManServer.*getMqttClientEntity*();  Gson gson = new Gson();  QuiterAndId quiterAndId = gson.fromJson(content, QuiterAndId.class);  // 找到要退出的那一个兼职  for (int i = 0; i <PartTimeManger.*PartTimeList*.size() ; i++) {  // 获取兼职信息  PartTimeEntity partTimeEntity = PartTimeManger.*PartTimeList*.get(i);  // 如果兼职id是我们要退的哪一个就进去if分支  if(partTimeEntity.getId() == quiterAndId.getId()) {  for (int j = 0; j < partTimeEntity.getArrayList().size(); j++) {  // 获取兼职人员信息  JoinInfo joinInfo = partTimeEntity.getArrayList().get(j);  // 如果是我们本人就从队列中移除这个信息  if(joinInfo.getName().equals(quiterAndId.getQuiter())){  partTimeEntity.getArrayList().remove(j);  MqttMessage message = new MqttMessage("success".getBytes());  message.setQos(2);  mqttClientEntity.publish("M/quit/quitPart",message);  return;  }  }  }  }  MqttMessage message = new MqttMessage("falie".getBytes());  message.setQos(2);  mqttClientEntity.publish("M/quit/quitPart",message);  }   // 检查clientId创建的兼职  private static void check(String content) throws MqttException {  MqttClient mqttClientEntity = ManServer.*getMqttClientEntity*();  Gson gson = new Gson();  CreaterAndId createrAndId = gson.fromJson(content, CreaterAndId.class);  // 遍历当前存在的所有兼职信息  for (int i = 0; i < PartTimeManger.*PartTimeList*.size(); i++) {  PartTimeEntity partTimeEntity = PartTimeManger.*PartTimeList*.get(i);  if(partTimeEntity.getCreater().equals(createrAndId.getCreater()) && partTimeEntity.getId() == createrAndId.getId()){  MqttMessage message = new MqttMessage(String.*valueOf*(partTimeEntity.getId()).getBytes());  message.setQos(2);  mqttClientEntity.publish("M/"+createrAndId.getCreater()+"/check",message);  return;  }  }  MqttMessage message = new MqttMessage("falie".getBytes());  message.setQos(2);  mqttClientEntity.publish("M/"+createrAndId.getCreater()+"/check",message);  }   // 判断用户是否登录  private static void isLogin(String content) throws MqttException {  MqttClient mqttClientEntity = ManServer.*getMqttClientEntity*();  for (int i = 0; i < MqttConnections.*MqttLoginedList*.size(); i++) {  String loginedList = MqttConnections.*MqttLoginedList*.get(i);  System.*out*.println(loginedList);  if(loginedList.equals(content)){  MqttMessage message = new MqttMessage("logined".getBytes());  message.setQos(2);  mqttClientEntity.publish("U/"+content+"/isLogin",message);  return;  }  }  MqttMessage message = new MqttMessage("notlogin".getBytes());  message.setQos(2);  mqttClientEntity.publish("U/"+content+"/isLogin",message);  }   // 加入兼职  private static void joinPartTime(String content) throws MqttException {  MqttClient mqttClientEntity = ManServer.*getMqttClientEntity*();  System.*out*.println(content);  Gson gson = new Gson();  JoinInfo joinInfo = gson.fromJson(content, JoinInfo.class);  // 判断用户要加入哪一个兼职  for (int i = 0; i < PartTimeManger.*PartTimeList*.size() ; i++) {  PartTimeEntity partTimeEntity = PartTimeManger.*PartTimeList*.get(i);  if(partTimeEntity.getId() == joinInfo.getPartTimeId()){  System.*out*.println("值相等");  System.*out*.println(joinInfo.getName());  partTimeEntity.setArrayList(joinInfo);  MqttMessage message = new MqttMessage(String.*valueOf*(partTimeEntity.getId()).getBytes());  message.setQos(2);  mqttClientEntity.publish("U/"+joinInfo.getName()+"/EPjoin",message);  break;  }  // 如果还找到就返回错误信息  if(i == PartTimeManger.*PartTimeList*.size() -1){  MqttMessage message = new MqttMessage("falie".getBytes());  message.setQos(2);  mqttClientEntity.publish("U/"+joinInfo.getName()+"/EPjoin",message);  }  }  }   // 查看兼职  private static void lookPartTime(String content) throws MqttException {  MqttClient mqttClientEntity = ManServer.*getMqttClientEntity*();  MqttMessage message = new MqttMessage(PartTimeManger.*PartTimeList*.toString().getBytes());  message.setQos(2);  mqttClientEntity.publish("M/"+content+"/Elist",message);  }   */\*\*  \** ***@Description*** *: 创建兼职流程  \** ***@author*** *: flj  \** ***@param*** *: content 兼职信息  \** ***@return*** *: 空  \** ***@exception*** *: MqttException  \** ***@date*** *: 2020/9/6 12:07  \*/* private static void createPartTime(String content) throws MqttException{  MqttClient mqttClientEntity = ManServer.*getMqttClientEntity*();  Gson gson = new Gson();  PartTimeEntity partTimeEntity = gson.fromJson(content, PartTimeEntity.class);  System.*out*.println("是否同意创建（1同意，0不同意）：");  Scanner scanner = new Scanner(System.*in*);  int i = scanner.nextInt();  if(i == 0){  // 告诉创建者消息  MqttMessage message = new MqttMessage("管理员不同意".getBytes());  message.setQos(2);  mqttClientEntity.publish("E/"+partTimeEntity.getCreater()+"/MPrelease",message);  }else {  String partTime = PartTimeManger.*createPartTime*(partTimeEntity);  if(partTime !=null) {  MqttMessage message = new MqttMessage(String.*valueOf*(PartTimeManger.*PartTimeList*.get(Integer.*parseInt*(partTime)-1).getId()).getBytes());  mqttClientEntity.publish("E/" + partTimeEntity.getCreater() + "/MPrelease", message);  }  }  }   // 执行登录流程  private static void login(String content) throws MqttException {  System.*out*.println("login");  Gson gson = new Gson();  LoginEntity loginEntity = gson.fromJson(content, LoginEntity.class);  // 静态类  MqttClient mqttClientEntity = ManServer.*getMqttClientEntity*();  // 先检查是否已经登录  for (int i = 0; i < MqttConnections.*MqttLoginedList*.size() ; i++) {  String isLoginedMap = MqttConnections.*MqttLoginedList*.get(i);  if(isLoginedMap.equals(loginEntity.getClientId())){  MqttMessage message = new MqttMessage((loginEntity.getClientId()+"已经登录，请勿重复登录").getBytes());  mqttClientEntity.publish("U/"+loginEntity.getClientId()+"/MUreg",message);  return;  }  }  // 如果未登录则查看存不存在  if(MqttConnections.*checkIsLoginInfo*(loginEntity)){  MqttMessage message = new MqttMessage("success".getBytes());  message.setQos(2);  mqttClientEntity.publish("U/"+loginEntity.getClientId()+"/MUreg",message);  // 获取对象并存入到登录集合队列中  MqttConnections.*MqttLoginedList*.add(loginEntity.getClientId());  }else {  MqttMessage message = new MqttMessage("账号密码错误或用户不存在".getBytes());  message.setQos(2);  mqttClientEntity.publish("U/"+loginEntity.getClientId()+"/MUreg", message);  }  }   // 执行注册的流程  private static void register(String content) throws SQLException, MqttException {  MqttClient mqttClientEntity = ManServer.*getMqttClientEntity*();  System.*out*.println("是否同意注册（1同意，0不同意）：");  Scanner scanner = new Scanner(System.*in*);  int i = scanner.nextInt();   if(i == 0){  MqttMessage message = new MqttMessage("管理员不同意".getBytes());  message.setQos(2);  mqttClientEntity.publish("U/192.168.1.10/Ureg",message);  }else {  Gson gson = new Gson();  LoginEntity loginEntity = gson.fromJson(content, LoginEntity.class);  // 如果账号不存在的情况  if (MqttConnections.*checkIsExist*(loginEntity.getUsername(),loginEntity.getClientId()) == null) {  MqttConnections.*register*(loginEntity);  MqttMessage message = new MqttMessage("success".getBytes());  message.setQos(2);  mqttClientEntity.publish("U/192.168.1.10/Ureg", message);  } else {  // 如果存在的话就提醒失败  MqttMessage message = new MqttMessage("账号已存在".getBytes());  mqttClientEntity.publish("U/192.168.1.10/Ureg", message);  }  }  } } |

1. client.UClient

|  |
| --- |
| package mqtthomework.client;  import com.google.gson.Gson; import mqtthomework.callback.ClientCallBack; import mqtthomework.entity.\*; import org.eclipse.paho.client.mqttv3.MqttClient; import org.eclipse.paho.client.mqttv3.MqttConnectOptions; import org.eclipse.paho.client.mqttv3.MqttException; import org.eclipse.paho.client.mqttv3.MqttMessage; import org.eclipse.paho.client.mqttv3.persist.MemoryPersistence; import java.net.InetAddress; import java.net.UnknownHostException; import java.util.Scanner;  public class UClient {   private String UClientName;   // 昵称作为该mqtt客户端的name  public UClient(String UClientName){  this.UClientName = UClientName;  }   public String getUClientName() {  return UClientName;  }   // 创建一个MQTT连接  public MqttClient getMqttClient(String connectionClientId) throws MqttException {  // broker的地址  String broker = "tcp://127.0.0.1:1883";  // 先用固定值，不同的客户端这个属性也不同(这里默认使用clientId,由用户输入，跟MqttClientName同值)  String clientId = connectionClientId;  MqttClient mqttClient = new MqttClient(broker, clientId, new MemoryPersistence());  // 创建链接参数  MqttConnectOptions connOpts = new MqttConnectOptions();  // 在重新启动和重新连接时记住状态  connOpts.setCleanSession(false);  // 建立连接  mqttClient.connect(connOpts);  return mqttClient;  }   public static void showMenu(MqttClient mqttClient,String clientId) throws Exception {  while(true){  System.*out*.println("当前你要做的操作：");  System.*out*.println("1、登录");  System.*out*.println("2、注册");  System.*out*.println("3、登出");  System.*out*.println("4、创建兼职");  System.*out*.println("5、查看兼职");  System.*out*.println("6、加入兼职");  System.*out*.println("7、兼职通知/回复");  System.*out*.println("8、退出兼职");  Scanner scanner = new Scanner(System.*in*);  int choose = scanner.nextInt();  switch (choose){  case 1: *login*(mqttClient,clientId);break;  case 2: *register*(mqttClient);break;  case 3: *quitLogin*(mqttClient,clientId);break;  case 4: *createPartTime*(mqttClient,clientId);break;  case 5: *lookPartTime*(mqttClient,clientId); break;  case 6: *joinPartTime*(mqttClient,clientId);break;  case 7: *resever*(mqttClient,clientId);break;  case 8: *quitPartTime*(mqttClient,clientId);break;  default:break;  }  }  }   // 登出  private static void quitLogin(final MqttClient mqttClient, final String clientId) throws MqttException {  MqttMessage message = new MqttMessage(clientId.getBytes());  message.setQos(2);  mqttClient.publish("U/"+clientId+"/quitLogin",message);  new Thread(new Runnable() {  public void run() {  try {  mqttClient.subscribe("M/"+clientId+"/#", 2);  } catch (MqttException e) {  e.printStackTrace();  }  }  }).start();  }   // 退出兼职  private static void quitPartTime(final MqttClient mqttClient, String clientId) throws MqttException {  System.*out*.println("请输入要退出的兼职：");  Scanner scanner = new Scanner(System.*in*);  int quitId = scanner.nextInt();  QuiterAndId quiterAndId = new QuiterAndId(clientId,quitId);  Gson gson = new Gson();  String quiterAndIdJson = gson.toJson(quiterAndId);  MqttMessage message = new MqttMessage(quiterAndIdJson.getBytes());  message.setQos(2);  mqttClient.publish("E/quitPart/quitPartTime",message);  new Thread(new Runnable() {  public void run() {  try {  mqttClient.subscribe("M/quit/#", 2);  } catch (MqttException e) {  e.printStackTrace();  }  }  }).start();    }   // 信息发布和回复  private static void resever(final MqttClient mqttClient, final String clientId) throws MqttException {  // 先传入clientId判断他创建的兼职是哪一个  System.*out*.println("请输入要传话的兼职ID：");  Scanner scanner = new Scanner(System.*in*);  int i = scanner.nextInt();  CreaterAndId createrAndId = new CreaterAndId(clientId,i);  Gson gson = new Gson();  String createrAndIdJson = gson.toJson(createrAndId);  MqttMessage message = new MqttMessage(createrAndIdJson.getBytes());  message.setQos(2);  mqttClient.publish("E/checkPartTime/checking",message);  mqttClient.subscribe("M/"+clientId+"/#", 2);  /\*new Thread(new Runnable() {  public void run() {  try {  mqttClient.subscribe("M/"+clientId+"/#", 2);  } catch (MqttException e) {  e.printStackTrace();  }  }  }).start();\*/  }   // 加入兼职  private static void joinPartTime(final MqttClient mqttClient, final String clientId) throws MqttException {  System.*out*.println("请输入要加入的兼职编号(id):");  Scanner scanner = new Scanner(System.*in*);  int choosePartTime = scanner.nextInt();  JoinInfo joinInfo = new JoinInfo(choosePartTime,clientId,"110112111","晚点到");  Gson gson = new Gson();  String jonInfoJson = gson.toJson(joinInfo);  MqttMessage message = new MqttMessage(jonInfoJson.getBytes());  message.setQos(2);  mqttClient.publish("E/join/user", message);  new Thread(new Runnable() {  public void run() {  try {  mqttClient.subscribe("U/"+clientId+"/EPjoin", 2);  } catch (MqttException e) {  e.printStackTrace();  }  }  }).start();  }   // 查看兼职  private static void lookPartTime(final MqttClient mqttClient, final String clientId) throws MqttException {  MqttMessage message = new MqttMessage(clientId.getBytes());  message.setQos(2);  mqttClient.publish("E/list/user", message);  new Thread(new Runnable() {  public void run() {  try {  mqttClient.subscribe("M/"+clientId+"/Elist", 2);  } catch (MqttException e) {  e.printStackTrace();  }  }  }).start();  }    // 创建兼职（先判断该client是否已经登录，根据回调结果看是否具备创建兼职的条件）  private static void createPartTime(final MqttClient mqttClient, final String clientId) throws MqttException {  // 封装成MQTT可发送的执行流  MqttMessage message = new MqttMessage(clientId.getBytes());  message.setQos(2);  mqttClient.publish("M/isLogin/Login", message);  // 开辟一个线程去订阅信息，等待数据  new Thread(new Runnable() {  public void run() {  try {  mqttClient.subscribe("U/"+clientId+"/isLogin", 2);  } catch (MqttException e) {  e.printStackTrace();  }  }  }).start();   }    // 登录  private static void login(final MqttClient mqttClient, final String clientId) throws MqttException {  // 封装成JSON  Gson gson = new Gson();  String username = *getInfo*("请输入账号：");  String password = *getInfo*("请输入密码：");  LoginEntity loginEntity = new LoginEntity(clientId,username,password);  String loginJson = gson.toJson(loginEntity);  // 封装成MQTT可发送的执行流  MqttMessage message = new MqttMessage(loginJson.getBytes());  message.setQos(2);  mqttClient.publish("M/Ulog/Ulog",message);  System.*out*.println(clientId);  // 开辟一个线程去订阅信息，等待数据  new Thread(new Runnable() {  public void run() {  try {  // 订阅对应ID  mqttClient.subscribe("U/"+clientId + "/MUreg",2);  } catch (MqttException e) {  e.printStackTrace();  }  }  }).start();  }    // 获取本机Ip  private static String getLocalIp() throws UnknownHostException {  InetAddress localHost = InetAddress.*getLocalHost*();  String[] split = localHost.toString().split("/");  return split[1];  }   // 封装输入数据  private static String getInfo(String message){  System.*out*.println(message);  Scanner scanner = new Scanner(System.*in*);  String info = scanner.next();  return info;  }   // 用户注册  private static void register(final MqttClient mqttClient) throws Exception {  String localIp = *getLocalIp*();  String info = "M/Ureg/"+localIp;  // 封装成JSON  Gson gson = new Gson();  String clientId = *getInfo*("请输入ClientId：");  String username = *getInfo*("请输入账号：");  String password = *getInfo*("请输入密码：");  LoginEntity loginEntity = new LoginEntity(clientId,username,password);  String loginJson = gson.toJson(loginEntity);  // 封装成MQTT可发送的执行流  MqttMessage message = new MqttMessage(loginJson.getBytes());  message.setQos(2);  mqttClient.publish(info,message);  // 开辟一个线程去订阅信息，等待数据  new Thread(new Runnable() {  public void run() {  try {  mqttClient.subscribe("U/#",2);  } catch (MqttException e) {  e.printStackTrace();  }  }  }).start();  }   public static void main(String[] args) throws Exception {  String clientId = *getInfo*("请输入账号昵称（ClientId）:");  UClient uClient = new UClient(clientId);  MqttClient mqttClientEntity = uClient.getMqttClient(clientId);  mqttClientEntity.setCallback(new ClientCallBack(mqttClientEntity,uClient.getUClientName()));  *showMenu*(mqttClientEntity,uClient.getUClientName());  mqttClientEntity.disconnect(); // 断开连接  mqttClientEntity.close(); // 关闭客户端  }  } |

1. client.UClientOne

|  |
| --- |
| package mqtthomework.client;   import mqtthomework.callback.ClientCallBack; import org.eclipse.paho.client.mqttv3.MqttClient;  import java.util.Scanner;  public class UClientOne {   // 新启一个客户端  public static void main(String[] args) throws Exception {  System.*out*.println("请输入账号昵称（ClientId）：");  Scanner scanner = new Scanner(System.*in*);  String clientId = scanner.next();  UClient uClient = new UClient(clientId);  System.*out*.println(uClient.hashCode());  MqttClient mqttClientEntity = uClient.getMqttClient(clientId);  mqttClientEntity.setCallback(new ClientCallBack(mqttClientEntity,uClient.getUClientName()));  UClient.*showMenu*(mqttClientEntity,uClient.getUClientName());  mqttClientEntity.disconnect(); // 断开连接  mqttClientEntity.close(); // 关闭客户端  } } |

1. db.MqttConnections

|  |
| --- |
| package mqtthomework.db;  import mqtthomework.entity.LoginEntity; import org.eclipse.paho.client.mqttv3.MqttClient;  import java.util.ArrayList; import java.util.HashMap; import java.util.Map;  */\*\*  \** ***@PackageName*** *mqtthomework.db  \** ***@Classname*** *MqttConnections  \** ***@Description*** *用于存储MQTT登录队列 和 用户账号  \** ***@Date*** *2020/9/5 19:48  \** ***@Author*** *flj  \** ***@Version*** *1.0.0  \*/* public class MqttConnections {   // 存放登录队列（clientId队列）  public static ArrayList<String> *MqttLoginedList* = new ArrayList<String>();   // 存放登录账号密码用户名（clientId）的队列  public static ArrayList<HashMap<String, String>> *UserList* = new ArrayList<HashMap<String, String>>();    static {  // 创建10个基础账号  for (int i = 0; i <10 ; i++) {  HashMap map = new HashMap();  map.put("username","test"+i);  map.put("password","123456");  map.put("clientId","client"+i);  *UserList*.add(map);  }  }    // 校验账号和clientId是否存在  public static HashMap checkIsExist(String username,String clientId){  for (int i = 0; i < *UserList*.size() ; i++) {  HashMap<String, String> UserLoginMap = *UserList*.get(i);  if(UserLoginMap.get("username").equals(username) && UserLoginMap.get("clientId").equals(clientId)){  return UserLoginMap;  }  }  return null;  }   // 注册  public static void register(LoginEntity loginEntity){  HashMap map = new HashMap();  map.put("username",loginEntity.getUsername());  map.put("password",loginEntity.getPassword());  map.put("clientId",loginEntity.getClientId());  *UserList*.add(map);  }   // 校验账号是否存在  public static HashMap checkIsExist(String username){  for (int i = 0; i < *UserList*.size() ; i++) {  HashMap<String, String> UserLoginMap = *UserList*.get(i);  if(UserLoginMap.get("username").equals(username)){  return UserLoginMap;  }  }  return null;  }    // 检查账号密码是否正确  public static boolean checkIsLoginInfo(LoginEntity loginEntity){  HashMap UserLoginMap = *checkIsExist*(loginEntity.getUsername());  if(UserLoginMap == null){  return false;  }   // 获取该账号对应的密码  String DBPassword = (String)UserLoginMap.get("password");  String DBClientId = (String)UserLoginMap.get("clientId");  if(DBPassword.equals(loginEntity.getPassword())&& DBClientId.equals(loginEntity.getClientId())){  return true;  }else {  return false;  }  }   public static void main(String[] args) {  for (int i = 0; i <*UserList*.size(); i++) {  System.*out*.println(*UserList*.get(i));  }   }   } |

1. db.PartTimeManger

|  |
| --- |
| package mqtthomework.db;  import mqtthomework.entity.PartTimeEntity;  import java.util.ArrayList; import java.util.HashMap;  */\*\*  \** ***@PackageName*** *mqtthomework.db  \** ***@Classname*** *PartTimeManger  \** ***@Description*** *兼职管理DB  \** ***@Date*** *2020/9/6 11:34  \** ***@Author*** *flj  \** ***@Version*** *1.0.0  \*/* public class PartTimeManger {   // 存放兼职信息的队列  public static ArrayList<PartTimeEntity> *PartTimeList* = new ArrayList<PartTimeEntity>();   // 新建兼职  public static String createPartTime(PartTimeEntity partTimeEntity){  partTimeEntity.setId(*PartTimeList*.size()+1);  *PartTimeList*.add(partTimeEntity);  return String.*valueOf*(partTimeEntity.getId());  }  } |

1. entity.CreaterAndId

|  |
| --- |
| package mqtthomework.entity;  */\*\*  \** ***@PackageName*** *mqtthomework.entity  \** ***@Classname*** *CreaterAndId  \** ***@Description*** *兼职创建者和对应id  \** ***@Date*** *2020/9/6 23:25  \** ***@Author*** *flj  \** ***@Version*** *1.0.0  \*/* public class CreaterAndId {   private String creater;  private int id;   public CreaterAndId(String creater, int id) {  this.creater = creater;  this.id = id;  }   public String getCreater() {  return creater;  }   public void setCreater(String creater) {  this.creater = creater;  }   public int getId() {  return id;  }   public void setId(int id) {  this.id = id;  } } |

1. entity.JoinInfo

|  |
| --- |
| package mqtthomework.entity;  */\*\*  \** ***@PackageName*** *mqtthomework.db  \** ***@Classname*** *JoinInfo  \** ***@Description*** *人员信息（加入兼职人员）实体类  \** ***@Date*** *2020/9/6 19:25  \** ***@Author*** *flj  \** ***@Version*** *1.0.0  \*/* public class JoinInfo {  private int partTimeId;  private String name;  private String phone;  private String remark;   public int getPartTimeId() {  return partTimeId;  }   public void setPartTimeId(int partTimeId) {  this.partTimeId = partTimeId;  }   public String getName() {  return name;  }   public void setName(String name) {  this.name = name;  }   public String getPhone() {  return phone;  }   public void setPhone(String phone) {  this.phone = phone;  }   public String getRemark() {  return remark;  }   public void setRemark(String remark) {  this.remark = remark;  }   public JoinInfo(int partTimeId, String name, String phone, String remark) {  this.partTimeId = partTimeId;  this.name = name;  this.phone = phone;  this.remark = remark;  }   @Override  public String toString() {  return "JoinInfo{" +  "partTimeId=" + partTimeId +  ", name='" + name + '\'' +  ", phone='" + phone + '\'' +  ", remark='" + remark + '\'' +  '}';  } } |

1. entity.LoginEntity

|  |
| --- |
| package mqtthomework.entity;    */\*\*  \** ***@PackageName*** *mqtthomework.db  \** ***@Classname*** *LoginEntity  \** ***@Description*** *登录信息实体类  \** ***@Date*** *2020/9/6 19:25  \** ***@Author*** *flj  \** ***@Version*** *1.0.0  \*/* public class LoginEntity {   private String clientId;  private String username;  private String password;      public String getUsername() {  return username;  }   public void setUsername(String username) {  this.username = username;  }   public String getPassword() {  return password;  }   public void setPassword(String password) {  this.password = password;  }   public LoginEntity(String username, String password) {  this.username = username;  this.password = password;  }   public String getClientId() {  return clientId;  }   public LoginEntity(String clientId, String username, String password) {  this.clientId = clientId;  this.username = username;  this.password = password;  }   public void setClientId(String clientId) {  this.clientId = clientId;  }   public LoginEntity(){   } } |

1. entity.PartTimeEntity

|  |
| --- |
| package mqtthomework.entity;   import java.util.ArrayList; import java.util.Date;  */\*\*  \** ***@PackageName*** *mqtthomework.entity  \** ***@Classname*** *PartTimeEntity  \** ***@Description*** *兼职实体类  \** ***@Date*** *2020/9/4 22:14  \** ***@Author*** *flj  \** ***@Version*** *1.0.0  \*/* public class PartTimeEntity {  private int id;  private Date time;  private int personNum;  private String place;  private String creater;  private ArrayList<JoinInfo> arrayList = new ArrayList<JoinInfo>();   public Date getTime() {  return time;  }   public void setTime(Date time) {  this.time = time;  }   public int getPersonNum() {  return personNum;  }   public void setPersonNum(int personNum) {  this.personNum = personNum;  }   public String getPlace() {  return place;  }   public void setPlace(String place) {  this.place = place;  }   @Override  public String toString() {  return "PartTimeEntity{" +  "id=" + id +  ", time=" + time +  ", personNum=" + personNum +  ", place='" + place + '\'' +  ", creater='" + creater + '\'' +  ", arrayList=" + arrayList +  '}';  }   public String getCreater() {  return creater;  }   public void setCreater(String creater) {  this.creater = creater;  }   public void setArrayList(ArrayList<JoinInfo> arrayList) {  this.arrayList = arrayList;  }   public ArrayList<JoinInfo> getArrayList() {  return arrayList;  }   public void setArrayList(JoinInfo joinInfo) {  this.arrayList.add(joinInfo);  }   public int getId() {  return id;  }   public void setId(int id) {  this.id = id;  }   public PartTimeEntity(Date time, int personNum, String place) {  this.time = time;  this.personNum = personNum;  this.place = place;  }   public PartTimeEntity(int id,Date time, int personNum, String place) {  this.id = id;  this.time = time;  this.personNum = personNum;  this.place = place;  }   public PartTimeEntity(Date time, int personNum, String place, String creater) {  this.time = time;  this.personNum = personNum;  this.place = place;  this.creater = creater;  } } |

1. entity.QuiterAndId

|  |
| --- |
| package mqtthomework.entity;   */\*\*  \** ***@PackageName*** *mqtthomework.entity  \** ***@Classname*** *QuiterAndId  \** ***@Description*** *退出信息实体类  \** ***@Date*** *2020/9/4 22:14  \** ***@Author*** *flj  \** ***@Version*** *1.0.0  \*/* public class QuiterAndId {  private String quiter;  private int id;   public String getQuiter() {  return quiter;  }   public void setQuiter(String quiter) {  this.quiter = quiter;  }   public int getId() {  return id;  }   public void setId(int id) {  this.id = id;  }   public QuiterAndId(String quiter, int id) {  this.quiter = quiter;  this.id = id;  } } |

1. server.ManServer

|  |
| --- |
| package mqtthomework.server;  import mqtthomework.callback.MessageCallBack; import org.eclipse.paho.client.mqttv3.\*; import org.eclipse.paho.client.mqttv3.persist.MemoryPersistence;   // 服务端，先启动这个 public class ManServer {   private static MqttClient *mqttClient*;   public static MqttClient getMqttClientEntity(){  return *mqttClient*;  }   // 创建一个MQTT连接  public void getMqttClient() throws MqttException {  String broker = "tcp://127.0.0.1:1883";  String clientId = "server1";  MqttClient sampleClient = new MqttClient(broker, clientId, new MemoryPersistence()); // 创建客户端  MqttConnectOptions connOpts = new MqttConnectOptions(); // 创建链接参数  connOpts.setCleanSession(false); // 在重新启动和重新连接时记住状态  sampleClient.connect(connOpts); // 建立连接  *mqttClient* = sampleClient;  }    public static void main(String[] args) throws MqttException {  ManServer manServer = new ManServer();  manServer.getMqttClient();  // 设置回调函数  *mqttClient*.setCallback(new MessageCallBack());  // 先订阅所有消息  *mqttClient*.subscribe("#",2);  }  } |

1. utils.Caozuo

|  |
| --- |
| package mqtthomework.utils;  public enum CaoZuo {   // 注册  USER\_REGISTER("Ureg",1),  // 登录  USER\_LOGIN("Ulog",2),  // 创建兼职  USER\_CREATE("Prelease",3),  // 查看兼职  *USER\_LOOK*("list",4),  // 加入兼职  USER\_JOIN("join",5),  // 用户是否登录  USER\_ISLOGIN("isLogin",6),  // 检查clientId创建对应的兼职  USER\_CHECK("checkPartTime",7),  // 退出兼职  USER\_QUIT("quitPart",8),  // 本人退出  USER\_QUITLOGIN("quitUser",9);   private String desc;//文字描述  private Integer code; //对应的代码   */\*\*  \* 私有构造,防止被外部调用  \** ***@param*** *desc  \*/* private CaoZuo(String desc, Integer code){  this.desc=desc;  this.code=code;  }  */\*\*  \* 定义方法,返回描述,跟常规类的定义没区别  \** ***@return*** *\*/* public String getDesc(){  return desc;  }   public void setDesc(String desc) {  this.desc = desc;  }   public Integer getCode() {  return code;  }   public void setCode(Integer code) {  this.code = code;  }   */\*\*  \* 根据传入的操作返回对应的code值  \** ***@return*** *\*/* public static int getCode(String doing){  for (CaoZuo a: CaoZuo.values()) {  if(doing.equals(a.getDesc())){  return a.getCode();  }  }  return -1;  }    } |

1. utils.ClientCaozuo

|  |
| --- |
| package mqtthomework.utils;  public enum ClientCaoZuo {   // 登录成功  *USER\_REGISTER\_SUCCESS*("MUreg",1),  // 创建兼职成功  *USER\_CREATEPARTTIMESUCCESS*("MPrelease",2),  // 加入兼职  *USER\_JOINPARTTIMESUCCESS*("EPjoin",3),  // 是否登录  *USER\_ISNOTLOGIN*("isLogin",4),  // 注册  *USER\_REGISTERINFO*("Ureg",5),  // 检查是否创建兼职  *USER\_CHECKISNOTCREATEPARTTIME*("check",6),  // 退出兼职  *USER\_QUITPARTTIME*("quitPart",7),  // 登出  *USER\_LOGINOUT*("quitLogin",8),  // 查看兼职列表  *USER\_PARTTIMELIST*("Elist",9),  // 查看兼职列表  *USER\_LISTEN*("listen",10);   private String desc;//文字描述  private Integer code; //对应的代码   */\*\*  \* 私有构造,防止被外部调用  \** ***@param*** *desc  \*/* private ClientCaoZuo(String desc, Integer code){  this.desc=desc;  this.code=code;  }  */\*\*  \* 定义方法,返回描述,跟常规类的定义没区别  \** ***@return*** *\*/* public String getDesc(){  return desc;  }   public void setDesc(String desc) {  this.desc = desc;  }   public Integer getCode() {  return code;  }   public void setCode(Integer code) {  this.code = code;  }   */\*\*  \* 根据传入的操作返回对应的code值  \** ***@return*** *\*/* public static int getCode(String doing){  for (ClientCaoZuo a: ClientCaoZuo.*values*()) {  if(doing.equals(a.getDesc())){  return a.getCode();  }  }  return -1;  }    } |

# 小结