

**课 程 实 验 报 告**

**课程名称： 物联网应用系统综合设计**

**专业班级： 物联网工程2101班**

**学 号： U202015370**

**姓 名： 李佳维**

**指导教师： 甘早斌**

**报告日期： 2024年10月19日**

**计算机科学与技术学院**

**目 录**

[酒店管理系统设计与实现 1](#_Toc18592)

[1.1 项目需求分析 1](#_Toc24381)

[1.2 项目概要设计](#_Toc26537) 3

[1.3 系统详细设计](#_Toc19726) 4

[1.4 系统测试与结果分析](#_Toc13971) 18

[1.5 总结](#_Toc11226) 22

自助式酒店客房管理系统设计与实现

1.1 项目需求分析

**1.1项目任务：自助式酒店客房管理系统设计与实现**

采用北京博创 RFID 实验箱模拟酒店的门禁读卡设备和房卡办理设备、支持 14443A 协议的 S50 卡(5 张)模拟门禁卡。 客人自助办理入住时，输入客人基本信息，系统自动绑定门禁卡与客人身份信息，根据客人输 入的入住日期、退房日期以及押金，自动写入门禁卡内。将卡与个人手机关联、姓名关联起来（采用 实名制，便于挂失）。

统一将某扇区 0 块作为电子钱包，用于存储客人入住时交付的押金，将某扇区 1 块、2 块保存个 人信息，将某扇区作为存储空间专门保存该客人入住记录（入住的起止日期、房间号以及缴纳的费用）。 一旦该卡丢失，马上通知管理员挂失，系统自动将该卡设置为未激活状态，锁定消费。当然， 每次使用该卡时，一定要判断该卡是否处于激活状态。 要求事先设置房间数、房间信息（面积、房间内设施、房费）等基础数据，并存入系统数据库

表中。系统测试时，要求至少有 5 个人客人的入住信息。假定读写器设备与上位机始终保持联系，上 位机与数据库服务器始终保持联系。

要求实现的主要功能包括：

(1) 客人入住自助办理管理（客人通过自助页面输入自己个人信息（身份证号、手机号、姓名、 性别等），输入入住日期、退房日期，押金，客人选择合适房间号后，系统自动绑定门禁卡与该客人）；

(2) 客人自助续住管理；

(3) 客人自助退房管理，清除卡内消费数据以及个人手机号，退还余款；

(4) 客人信息的写卡功能；

(5) 客人入住信息的自助查询功能；

(6) 酒店收入统计功能、客房入住信息统计功能、酒店客房使用率统计功能等等。

此功能模块需要用到的系统表结构：

(1) 客人基本信息表（卡号、姓名、手机号、…、是否激活）；

(2) 客人入住消费信息表（卡号、姓名、商品编号（房号）、购买时间、商品数量、扣款金额、…）；

(3) 客房资费信息表（客房编号、单价、单位、…）；

(4) 客人进出客房信息表(卡号、时间、进/出、...)；

(5) 商户信息表(商户编号、商户名称、单价、数量、总价、…)。

**1.2项目系统要求：**

采用云服务器。在云服务器上构建自己的 Web 服务器（建议 tomcat）、数据库服务器(建议 Mysql)、通信服务器（Mosquitto），数据采集终端与云端的数据传输采用 MQTT 协议，各个系统采集的数据以及 Web 服务系统都集中存放云服务器中。

**1.3项目需求分析：**

**项目目标：**

开发一套自助式酒店客房管理系统，实现客人自助入住、门禁控制、自动结算和数据统计分析等功能。

**用户群体：**

酒店管理人员：负责系统的日常管理和维护。

酒店客人：使用系统进行自助入住、续住和退房。

**具体需求功能：**

1.客人入住自助办理管理

功能描述：客人通过自助页面输入个人信息，选择房间号，系统自动绑定门禁卡与客人信息。

输入：客人基本信息（身份证号、手机号、姓名、性别等），入住日期、退房日期，押金。

输出：绑定成功的门禁卡，入住确认信息。

2.客人自助续住管理

功能描述：客人自助申请续住，系统更新入住记录。

输入：续住天数，续交押金。

输出：续住确认信息，更新后的门禁卡信息。

3.客人信息的记录功能

功能描述：将客人信息写入门禁卡的指定扇区。

输入：客人基本信息。

输出：写卡成功的确认信息。

4.客人入住信息的自助查询功能

功能描述：客人可以查询自己的入住记录和消费信息。

输入：客人身份验证信息。

输出：入住记录和消费信息。

5.酒店收入统计功能

功能描述：统计酒店收入，包括房费收入和其他消费收入。

输入：查询时间段。

输出：收入统计报表。

6.客房入住信息统计功能

功能描述：统计客房的入住情况。

输入：查询时间段。

输出：客房入住统计报表。

7.酒店客房使用率统计功能

功能描述：统计酒店客房的使用率。

输入：查询时间段。

输出：客房使用率统计报表。

**非功能需求**

系统性能

响应时间：系统操作响应时间不超过2秒。

并发用户：支持至少100个并发用户同时操作。

系统可靠性

系统稳定性：系统连续运行7x24小时无故障。

数据安全性：采用加密技术保护客人信息和交易数据。

系统可维护性

系统应提供日志记录功能，方便问题追踪和系统维护。

法律和标准遵从性

系统设计应符合相关法律法规，如数据保护法等。

**外部接口**

系统需与MQTT服务器保持通信，上传采集数据。

1.2 项目概要设计

项目主要分为本地系统和服务器系统两大部分，通过mosquitto代理实现mqtt通信。

本地端系统需要实现对RFID卡的读写操作，并将数据写入数据库中，并且通过mosquitto与服务器端进行通信；

服务器端系统需要搭建mqtt broker接收本地发送的mqtt消息，并通过python脚本对mqtt消息进行解析实现相应的sql操作，将数据存入服务器端数据库，然后构建一个web应用，实现与mysql数据库的交互。

数据库实现：

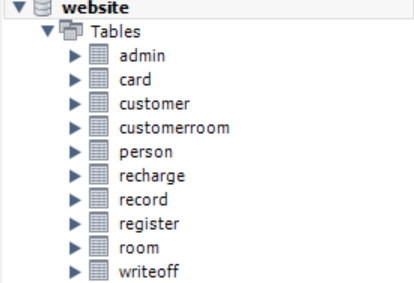
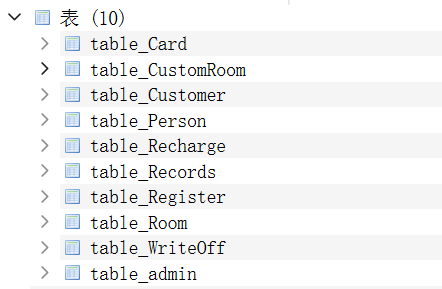


图1-1 本地和云端数据库表

注意：这里本地端系统是可以没有数据库存储数据的，但是之前做的酒店管理系统已经构建好了数据库，所以我没有删掉。再考虑到进行数据库操作的时候会有一些逻辑判断，放在本地处理的话会更便捷一点，就不用在服务器端进行逻辑判断，并返回mqtt消息。也就是说如果本地也有一个数据库，那么两个数据库的数据是同步的，本地系统只需要publish消息，服务器端系统只需要subscribe消息。

以下是项目流程图：

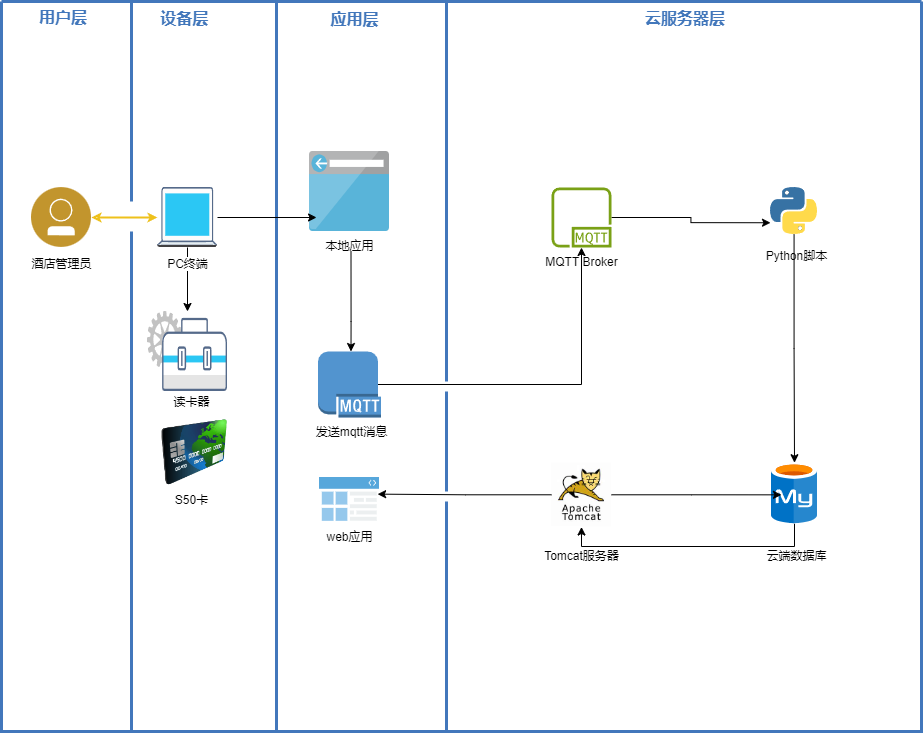


图1-2酒店管理系统项目流程图

1.3 系统详细设计

本地端酒店管理应用设计

操作系统：windows 11；

开发工具：Qt 5.9.1/QT Creator 4.3.1 (Community)；

Mqtt代理：mosquitto-2.0.18；

数据库：SQLite3(QT自带)；

**项目名称：Hotel\_Management\_System**

**主要架构： 实现MainWindow类(连接其他页面和各个槽函数）**

**实现Mqttclienthandler类(处理mqtt代理连接和发送消息功能）**

**实现SerialPortThread类(对连接的串口进行相关处理）**

**实现DataBase的各个TableModel类(将卡数据插入到表中，并发送mqtt消息)**

**MainWindow类**

Qt应用程序的主窗口类 MainWindow 的实现，它继承自 QMainWindow。这个类负责应用程序的主界面，包括初始化界面、处理用户交互、管理子页面以及与串口和MQTT客户端进行通信。以下是代码的主要组成部分和功能：

构造函数：初始化用户界面，创建串口和MQTT客户端对象，设置窗口标题，并调用 handConnect 方法连接信号和槽。

析构函数：清理动态分配的资源。

添加子页面：addWidgets 方法向主窗口的堆栈式 widget ui->stackedWidget 添加子页面，如欢迎页面、注册页面、表格信息页面和消费页面。

连接信号和槽：handConnect 方法连接了各种用户交互事件（如菜单项点击）到相应的槽函数。

窗口关闭事件：closeEvent 方法处理窗口关闭事件，弹出确认对话框，并在确认后断开串口和MQTT连接。

检查登录状态：CheckLogin 方法检查用户是否已经登录，如果没有登录，则显示警告消息。

MQTT测试：MqttTest 方法用于测试MQTT连接和消息发布。

关于对话框：About 方法显示关于对话框，包含软件版本、作者和描述信息。

修改密码：ChangePasswd 方法弹出修改密码对话框。

数据导出：ExportTable 方法弹出数据导出对话框。

登录：Login 方法弹出登录对话框，并在登录成功后更新登录状态。

查看消费场景：ViewConsume 方法切换到消费场景页面。

查看主页面：viewMainPage 方法切换回主页面。

充值：Recharge 方法弹出充值对话框。

查看表格：ViewTables 方法根据用户选择的菜单项，切换到相应的表格信息页面。

注册：Register 方法切换到注册页面。

注销用户：WriteOff 方法弹出注销用户对话框。

连接和断开串口：Connect 和 Disconnect 方法处理串口连接和断开。

更新串口参数：updateConnect 方法根据用户在设置对话框中输入的参数更新串口配置，并尝试重新连接。

退出应用程序：ExitApplication 方法关闭应用程序。

串口操作错误处理：onOperationError 方法显示串口操作错误消息。

串口消息接收处理：on\_serialMsgreceived 方法处理从串口接收到的消息，如寻卡、寻卡结果和卡选择。

这个类的设计体现了Qt的信号和槽机制，通过连接信号和槽来响应用户交互和串口事件。此外，它还使用了单例模式来管理MQTT客户端连接。代码中使用了多个自定义的对话框类，如 SettingsDialog、LoginPage、ChangePassword、ExportDatas、RechargeDialog 等，

**MQTTClientHandler类**

程序与mqtt通信通过单例类 MQTTClientHandler 实现，它封装了与MQTT代理（broker）的连接、消息发送和接收的功能。以下是代码的主要组成部分和功能：

单例模式实现：m\_instance 是一个静态指针成员变量，用于持有单例实例。

私有构造函数确保外部代码不能直接创建类的实例。公共的静态方法 instance() 用于获取单例实例。如果实例不存在，则创建一个新实例。

MQTT客户端初始化：在构造函数中，创建了一个 QMqttClient 对象，并连接了几个信号到相应的槽函数，以处理连接、断开连接和接收消息的事件。

连接和断开连接的处理：

connectToMqttBroker() 方法设置了MQTT代理的主机名和端口，并尝试连接。

onConnected() 和 onDisconnected() 分别处理连接成功和断开连接的信号。

onConnected() 方法中还包含了在连接成功后自动发送消息的逻辑。

消息发送：sendMessage() 方法检查MQTT客户端是否已连接，如果是，则发送消息到指定的主题。如果客户端未连接，显示一个警告消息框。

消息接收：onMessageReceived() 方法是一个槽函数，用于处理接收到的消息。当前实现为空，需要根据具体需求实现消息处理逻辑。

资源清理：在析构函数中，如果MQTT客户端还处于连接状态，则断开连接，并删除 QMqttClient 对象。

设置消息参数：setTopic() 和 setMessage() 方法用于设置要发送的消息的主题和内容。

检查连接状态：isConnected() 方法用于检查MQTT客户端是否已连接到代理。

这个类的设计模式是单例模式，确保整个应用程序中只有一个 MQTTClientHandler 实例。这有助于管理MQTT连接，并确保连接状态的一致性。

**SerialPortThread类**

代码定义了一个名为 SerialPortThread 的类，它继承自 QThread，用于在单独的线程中处理串口通信。这个类提供了串口读写操作的接口，并处理与串口相关的事件。以下是代码的主要组成部分和功能：

构造函数：初始化串口状态为未打开，设置一个定时器来处理错误，并连接信号槽。

析构函数：目前为空，用于在对象销毁时进行清理。

stopTimer：停止定时器。

onError：当串口发生错误时，发出错误信号，并停止定时器。

stopThread：停止线程，包括停止定时器和设置停止标志。

setRetryTimes：设置重试次数。

writeData：发送数据到串口，如果设置了回显，则发出发送数据的信号。

serialPortIsOpen：判断串口是否打开。

run：线程的主体函数，循环检查串口是否有数据可读，并处理这些数据。

setSerialPort：设置串口对象。

关于如何读卡的具体实现：

/\*\*

\* @brief RegistorWidget::on\_btn\_Inventory\_clicked

\* 识别按钮点击事件

\*/

void RegistorWidget::on\_btn\_Inventory\_clicked()

{

if(!serialThread->serialPortIsOpen())

{

QMessageBox::warning(this,tr("温馨提示"),tr("请先连接读卡器后再试！"),QMessageBox::Yes);

return;

}

uint16 frameLen;

quint8 buffer[1];

uint8 \*p;

memset(buffer, 0, 1);

buffer[0] = RC632\_14443\_ALL;

p = m1356dll->RC632\_SendCmdReq(RC632\_CMD\_REQUEST\_A,buffer,1);

frameLen = BUILD\_UINT16(p[0], p[1]);

serialThread->writeData((char \*)(p + 2 ),frameLen);

}

识别按钮点击后，串口读取s50卡数据

void SerialPortThread::*run*()

{

int count = retryTimes;

QByteArray bytes;

//控制线程循环

while (!Stop) {

if(serialPort->isOpen())

{

if(serialPort->*bytesAvailable*() >= 4)

{

emit hasResponse();

bytes = serialPort->readAll();

if(bytes.at(0) != (char) 0xAA && bytes.at(1) != (char)0xBB)

continue;

qint16 waitforReadLen = (qint16)((bytes.at(2) & 0x00FF) + ((bytes.at(3) & 0x00FF) << 8)) + 4;

while ((waitforReadLen - bytes.length()) > 0 && count -- > 0)

{

if(serialPort->*bytesAvailable*() == 0)

QThread::usleep(10);

else

{

bytes += serialPort->readAll();

count = retryTimes;

}

}

count = retryTimes;

for(int i = 0 ; i < bytes.length() ; i ++)

{

if(bytes.at(i) == (char)0xAA && bytes.at(i+1) == (char)0x00)

{

bytes.remove(i + 1,1);

}

}

qDebug() << "RecMessage: "<< CharStringtoHexString(tr(" "),bytes.data(),bytes.length());;

emit receivedMsg(bytes);

}

}

QThread::msleep(20);

}

}

线程发送receivedMsg(bytes)信号，转到槽函数on\_serialMsgreceived(QByteArray)

connect(serialPortThread,SIGNAL(receivedMsg(QByteArray)),this,SLOT(on\_serialMsgreceived(QByteArray)));

\* @brief MainWindow::on\_serialMsgreceived

\* @param bytes 接收到的数据字节数组

\* 串口接收消息后的处理槽函数

\*/

void MainWindow::on\_serialMsgreceived(QByteArray bytes){

M1356\_RspFrame\_t frame = m1356dll->M1356\_RspFrameConstructor(bytes);

if(frame.status.left(2) == "00")

{

if(frame.cmd.remove(" ") == "0102")//寻卡

{

uint16 frameLen;

quint8 buffer[1];

uint8 \*p;

memset(buffer, 0, 1);

buffer[0] = 0x04;

p = m1356dll->RC632\_SendCmdReq(RC632\_CMD\_ANTICOLL, buffer, 1);

frameLen = BUILD\_UINT16(p[0], p[1]);

serialPortThread->writeData((char \*)(p + 2 ),frameLen);

}

else if(frame.cmd.remove(" ") == "0202")//寻卡结果

{

uint16 frameLen;

quint8 buffer[4];

uint8 \*p;

memset(buffer, 0, 4);

QSTRING\_TO\_HEX(frame.vdata.remove(" "),buffer,4); // 卡号

p = m1356dll->RC632\_SendCmdReq(RC632\_CMD\_SELECT,buffer,4);

frameLen = BUILD\_UINT16(p[0], p[1]);

serialPortThread->writeData((char \*)(p + 2 ),frameLen);

tagId = frame.vdata.remove(" ");

}

else if(frame.cmd.remove(" ") == "0302")

{

emit sendCardId(tagId);

}

}

else

{

if(frame.cmd.remove(" ") == "0102")//寻卡

{

QMessageBox::warning(this,tr("温馨提示"),tr("寻卡失败，请调整卡与读卡器的距离后再试！"),QMessageBox::Yes);

}

else if(frame.cmd.remove(" ") == "0202")//寻卡结果

{

QMessageBox::warning(this,tr("温馨提示"),tr("A卡防冲撞失败，请调整卡与读卡器的距离后再试！"),QMessageBox::Yes);

}

}

}

槽函数处理后得到卡号，保存卡号，发送信号emit sendCardId(tagId)，转到槽函数显示卡号

connect(this,SIGNAL(sendCardId(QString)),registWidget,SLOT(on\_tagIdReceived(QString)));

**TableModel类(以 CardTableModel为例）**

CardTableModel 类，它继承自 QSqlTableModel，用于管理一个名为 tableName 的数据库表。这个类提供了创建表、插入记录、删除记录和查找记录的功能。以下是代码的主要组成部分和功能：

构造函数：初始化表名和表头。

createTable：创建一个新表，如果表已存在，则不会重复创建。这个方法构建了一个SQL语句来创建表，并执行这个语句。

bindTable：绑定到现有的数据库表，并执行查询。

findRecordByCardId 和 findRecordByRoomId：根据卡号或房间号查找特定的记录。

insertRecords：向数据库表中插入新的记录。这个方法构建了一个插入语句，并执行这个语句。此外，它还创建了一个JSON对象，并通过MQTT协议发送这个消息。

deleteRecords：删除指定行的记录。

findRoomIdByTagId 和 findCheckOutTimeByTagId：根据卡号查找对应的房间号或退房时间。

insertRecords 方法中包含了MQTT协议代码，MQTTClientHandler::instance() 调用假设有一个单例模式的MQTT客户端处理器，这个处理器负责处理MQTT通信。这部分代码需要确保MQTT客户端已经正确配置并且连接到了MQTT服务器。这意味着当插入记录时，不仅会更新数据库，还会通过MQTT协议发送消息。当消息到达云端后，云端再解析json格式的message。

为了避免mqtt连接和发送消息时的异步性，即当连接后立即发送消息，可能消息发送不到的情况，我在mainwindow窗口加了一个mqtttest函数，用来手动连接mqtt，避免第一次启动mqtt实例时，发送消息失败的情况。

服务器端设计

开发工具：IntelliJ IDEA Community Edition 2024.2.3；

PyCharm Community Edition 2024.2.3；

Mqtt代理：mosquitto-2.0.19；

Tomcat版本：Apache Tomcat/9.0.96；

mysql服务器版本: 5.7.43-log MySQL Community Server (GPL)；

Eclipse Mosquitto MQTT v5/v3.1.1 broker；

操作系统：Microsoft Windows Server 2022 Datacenter；

**Python脚本**

服务器端首先需要解决的问题是如何将收到的mqtt消息解析成sql语句，实现数据的插入。先写一个python脚本：需要下载库文件mysql.connector paho.mqtt.client as mqtt json

import mysql.connector

import paho.mqtt.client as mqtt

import json

# MySQL 配置

mysql\_config = {

    'user': 'root',

    'password': 'password',

    'host': 'localhost',

    'database': 'website'

}

# MQTT 配置

mqtt\_broker = '121.199.31.186'  # 替换为你的 MQTT 代理地址

mqtt\_topic = 'website'  # 替换为你的主题

# 连接到 MySQL 数据库

def connect\_to\_mysql():

    try:

        connection = mysql.connector.connect(\*\*mysql\_config)

        return connection

    except mysql.connector.Error as err:

        print(f"Error: {err}")

        return None

# MQTT 消息回调

def on\_message(client, userdata, message):

    print(f"Received message: {message.payload.decode()}")

    # 将消息解析为 JSON

    try:

        msg\_json = json.loads(message.payload.decode())

        table\_name = msg\_json['tableName']

        data = msg\_json['data']

    except (json.JSONDecodeError, KeyError) as e:

        print(f"Failed to parse message: {e}")

        return

    # 将数据保存到指定的 MySQL 数据库表

    connection = connect\_to\_mysql()

    if connection:

        try:

            with connection.cursor() as cursor:

                # 构建插入查询

                columns = ', '.join(data.keys())

                placeholders = ', '.join(['%s'] \* len(data))

                insert\_query = f"INSERT INTO {table\_name} ({columns}) VALUES ({placeholders})"

                cursor.execute(insert\_query, tuple(data.values()))

                connection.commit()

                print(f"Message inserted into {table\_name} successfully")

        except mysql.connector.Error as err:

            print(f"Failed to insert message: {err}")

        finally:

            connection.close()

# 设置 MQTT 客户端

client = mqtt.Client()

client.on\_message = on\_message

# 连接到 MQTT 代理并订阅主题

client.connect(mqtt\_broker)

client.subscribe(mqtt\_topic)

# 开始循环

client.loop\_start()

try:

    print("Waiting for messages...")

    while True:

        pass

except KeyboardInterrupt:

    print("Exiting...")

finally:

    client.loop\_stop()

    client.disconnect()

它连接到一个MySQL数据库，并且订阅了一个MQTT主题以接收消息。当接收到MQTT消息时，它会将消息内容解析为JSON，并将这些数据插入到MySQL数据库中。以下是代码的主要组成部分和功能：

MySQL配置：定义了连接MySQL数据库所需的配置参数。

MQTT配置：定义了MQTT代理服务器的地址和要订阅的主题。

connect\_to\_mysql函数：尝试使用提供的配置连接到MySQL数据库，并返回数据库连接对象。如果连接失败，捕获异常并打印错误信息。

on\_message函数：这是一个回调函数，当客户端接收到MQTT消息时被调用。它解析消息内容为JSON格式，并将数据插入到MySQL数据库中。

构建插入查询：使用接收到的数据构建一个插入SQL查询，并执行它。

设置MQTT客户端：创建一个MQTT客户端实例，并设置on\_message回调函数。

连接和订阅：客户端连接到MQTT代理，并订阅配置中指定的主题。

开始循环：启动MQTT客户端的循环，开始接收消息。

无限循环等待消息：使用一个无限循环等待接收MQTT消息。

异常处理：捕获键盘中断异常，以便在按下Ctrl+C时优雅地退出脚本。

清理：在退出脚本之前，停止MQTT客户端循环并断开连接。

**Web应用**

实现对mysql/website数据库监控，展示card表，room表，writeoff表，customer表，并做相关的查询更新操作。

以Room表为例，给出实现的网页jsp文件：

<%@ page import="java.sql.\*" %>

<%@ page contentType="text/html;charset=UTF-8" language="java" %>

<html>

<head>

    <title>房间信息表</title>

    <style>

        table, th, td {

            border: 1px solid black;

            border-collapse: collapse;

        }

        th, td {

            padding: 8px;

            text-align: left;

        }

        th {

            background-color: #f2f2f2;

        }

    </style>

</head>

<body>

    <h2>房间信息表</h2>

    <a href="welcome.jsp" class="button">回到欢迎页面</a>

    <%

        String url = "jdbc:mysql://localhost:3306/website?useSSL=false&serverTimezone=UTC";

        String user = "root";

        String password = "password";

        String queryRoom = "SELECT \* FROM room";

        String queryCard = "SELECT \* FROM card WHERE isUse = '未退房'";

        String queryPrice = "SELECT SUM(r.price) AS totalPrice FROM room r INNER JOIN card c ON r.roomId = c.roomId WHERE c.isUse = '未退房'";

        try {

            Class.forName("com.mysql.cj.jdbc.Driver");

            try (Connection conn = DriverManager.getConnection(url, user, password);

                 Statement stmt = conn.createStatement()) {

                // Query room table

                try (ResultSet rs = stmt.executeQuery(queryRoom)) {

                    out.println("<table>");

                    out.println("<tr>");

                    out.println("<th>房间号</th>");

                    out.println("<th>面积/m\*2</th>");

                    out.println("<th>房间类型</th>");

                    out.println("<th>房价￥/天</th>");

                    out.println("</tr>");

                    while (rs.next()) {

                        out.println("<tr>");

                        out.println("<td>" + rs.getInt("roomId") + "</td>");

                        out.println("<td>" + rs.getDouble("square") + "</td>");

                        out.println("<td>" + rs.getString("roomType") + "</td>");

                        out.println("<td>" + rs.getString("price") + "</td>");

                        out.println("</tr>");

                    }

                    out.println("</table>");

                }

                // Query card table to count records

                int x = 0;

                try (ResultSet rs2 = stmt.executeQuery(queryCard)) {

                    while (rs2.next()) {

                        x++;

                    }

                }

                out.println("<p>房间利用率: " + x \* 10 + "%</p>");

                    try (ResultSet rs3 = stmt.executeQuery(queryPrice)) {

                                     if (rs3.next()) {

                                           double totalPrice = rs3.getDouble("totalPrice");

                                           out.println("<p>当天总房价: " + totalPrice + "￥" + "</p>");

                                       }

                            }

            }

        } catch (Exception e) {

            out.println("<p>Error: " + e.getMessage() + "</p>");

        }

    %>

</body>

</html>

这段代码生成了一个简单的HTML页面，显示了从MySQL数据库中检索到的房间信息表，并计算了房间的利用率和当天的总房价。以下是代码的主要组成部分和功能：

页面指令和标题：

使用<%@ page %>指令指定了页面的属性，如导入的包、内容类型和字符编码。

HTML头部包含了页面标题和内联CSS样式，用于设置表格的外观。

数据库连接和查询：

定义了数据库连接的URL、用户名和密码。

定义了三个SQL查询：一个用于检索所有房间信息，一个用于计算未退房的房间数量，一个用于计算当天的总房价。

执行数据库查询：

使用Class.forName加载MySQL JDBC驱动程序。

使用DriverManager.getConnection建立数据库连接，并创建Statement对象。

执行查询并使用ResultSet遍历结果集，动态生成HTML表格显示房间信息。

计算房间利用率：

通过查询未退房的房间数量，并假设有一个固定的房间总数（这里没有给出），计算房间利用率。

计算当天总房价：

通过查询计算当天的总房价，并在页面上显示。

错误处理：

使用try-catch块捕获并处理可能发生的异常，如数据库连接失败或SQL执行错误。

关闭资源：

使用try-with-resources语句自动关闭数据库资源，如Connection和ResultSet。

返回欢迎页面的链接：

提供了一个链接，允许用户返回到欢迎页面。

Web应用的链接为 <http://121.199.31.186:8080/myweb/login>

先进入登录页面，输入管理员名和密码，然后进入欢迎页面 [欢迎页面](http://121.199.31.186:8080/myweb/welcome.jsp) 可执行对以上四个表的相关操作。

1.4 系统测试与结果分析

本地客户端

**主页面**

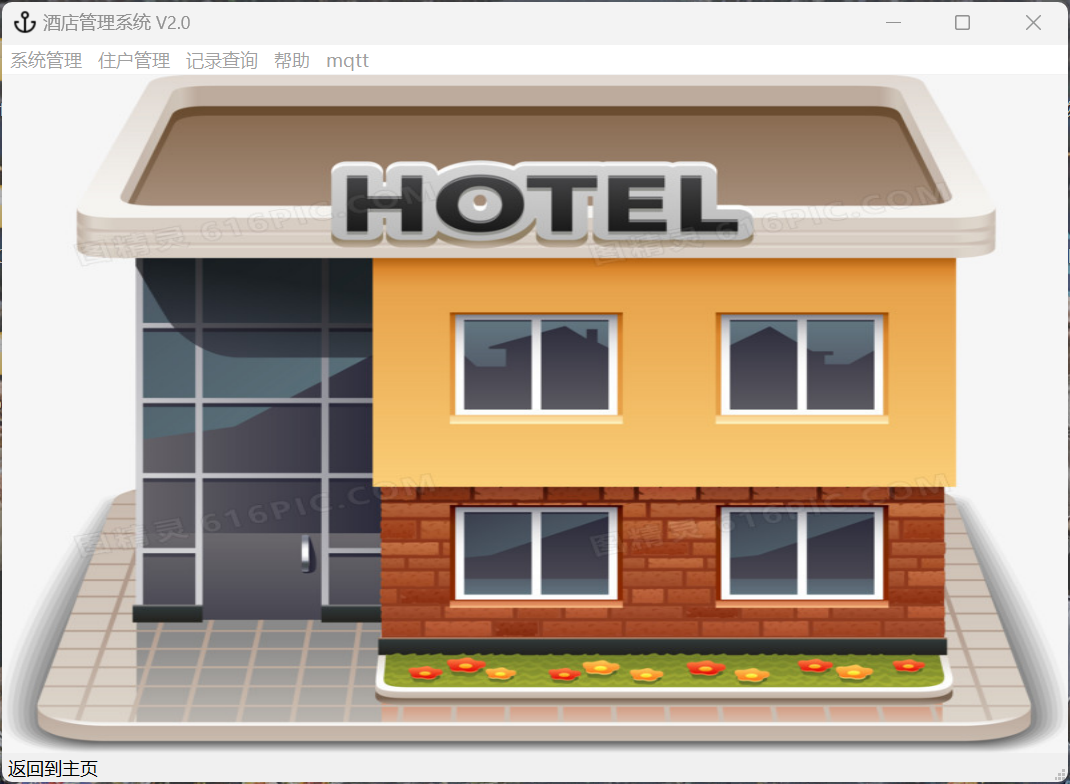


图1-3 系统主页面

为了确保mqtt在发送消息时已经于mqtt broker连接，先进行手动连接

mqtttest功能



图1-4

点击连接后，mqtt实例会自动连接，连接成功则会在连接状态上显示已连接。

识别注册功能

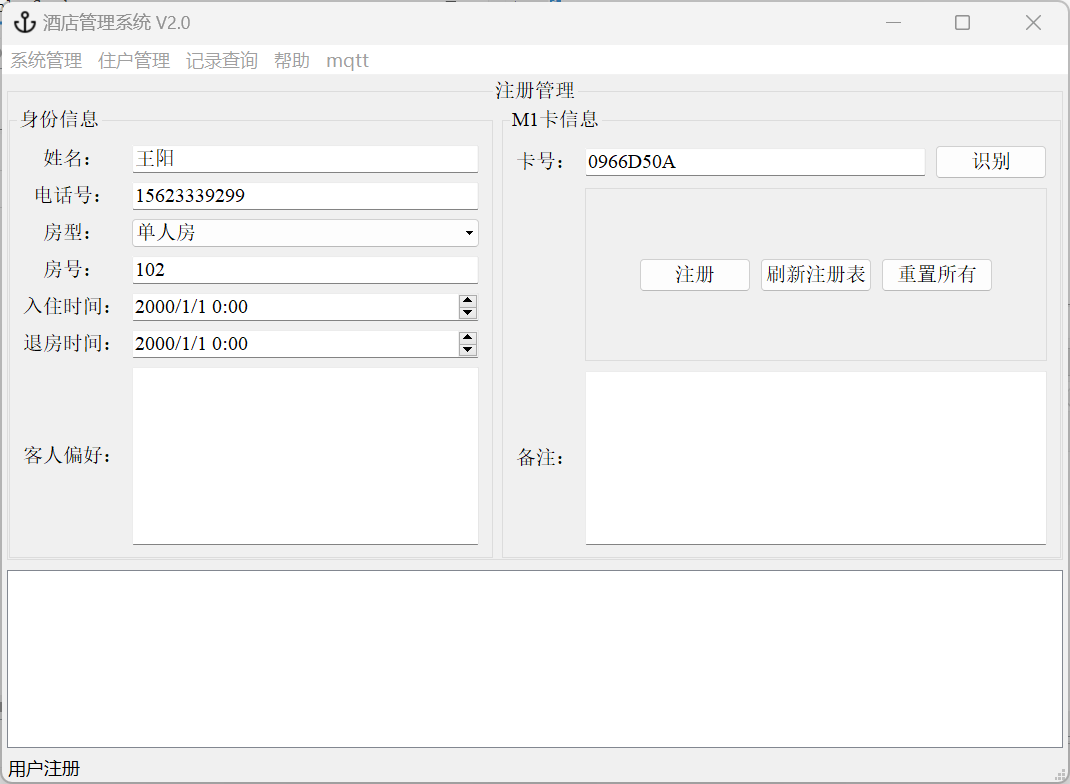


图1-5用户注册

点击注册后，mqtt实例向mqtt broker发送json格式的message。

**服务器端**

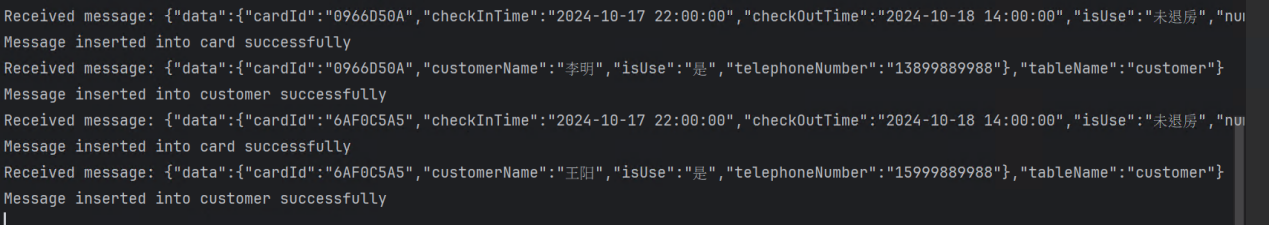
Python脚本输出如下：

图1-6 脚本运行

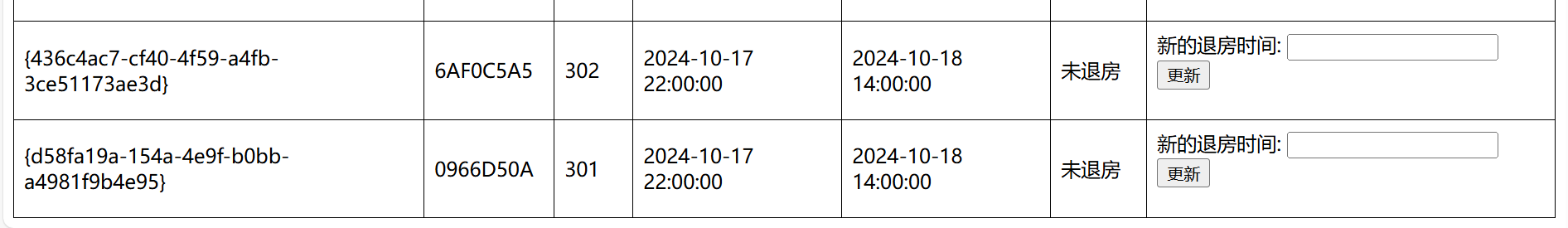
可在web应用上查看数据：查看card表

图1-7 card表新加数据

查看customer表：李明、王阳已添加。



图1-8 room表新加数据

另外可查看客房数据：

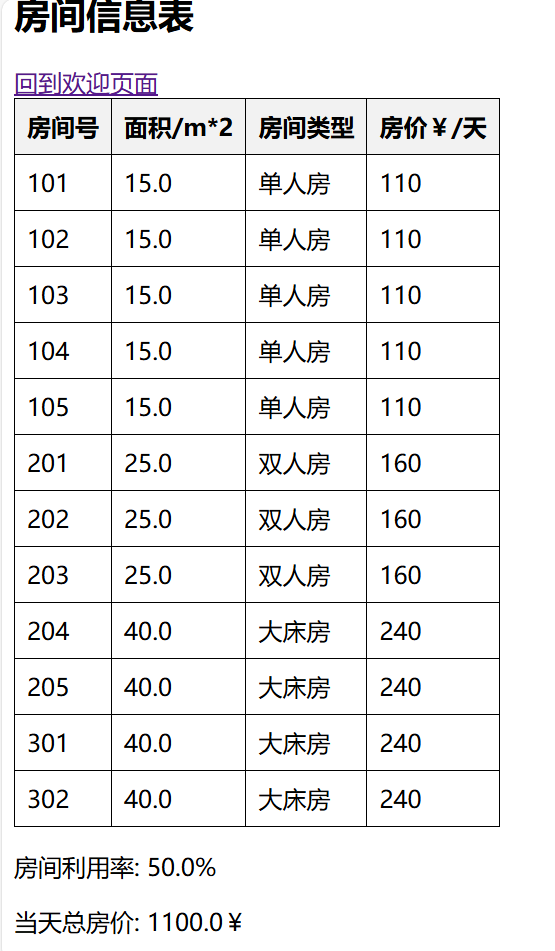


图 1-9 客房信息

显示了房间信息并计算了房间利用率和当天的总房价。

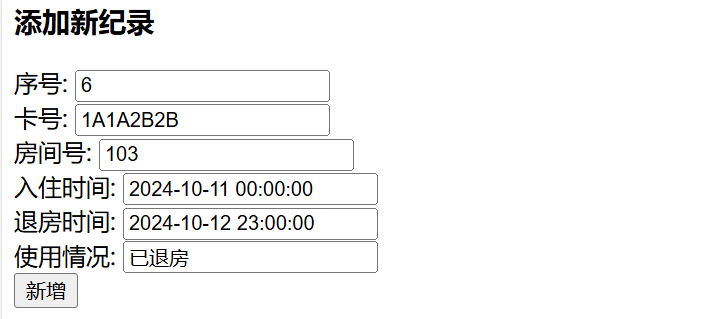
Card表添加数据操作：



图1-10 数据添加成功

1.5 总结

遇到的问题

1.一个遇到的问题是，tomcat和Java版本不匹配，导致网页打不开。

在配置tomcat的时候，没有准确配置jdk路径，并且jdk版本有问题，更新了jdk版本为jdk-17，显示准确。另外tomcat是用http协议的，之前用https协议一直登不上。

2.第二个问题是，在考虑通过mqtt代理实现消息传输时，不知道如何增加mqtt发送模块。

当时想构建一个mqtt代理类，每次发消息就实例化一个对象出来，然后连接发送，后来感觉生成太多代理对象不便于管理，也可能发送传输冲突，导致服务器接收不到消息。于是想到设置一个单例模式的MQTT客户端处理器，每次发送消息时只需要调用这个单例的sendMessage函数，不需要生成多个实例。

3.增加mqtt模块后，第一次发送数据时服务器总是收不到。

测试时发现，如果mqtt实例调用连接函数后立即发送消息，服务器是收不到的。查阅资料后得知，mqtt与broker连接和发送消息是异步的，第一次连接时，立即发送消息，mqtt可能还未正真建立连接，导致收不到消息。

解决办法是用槽函数，当mqtt代理返回了已连接信号后，调用槽函数发送mqtt消息，为确保第一次连接已经建立，我还设置了一个测试模块，来手动连接目mqtt。这样第一次发送消息时就不会遗漏了。

4.第四个问题是，web网页显示中文乱码。

原因是在创建mysql数据库表时没有指定字符集，在写jsp文件时也没有指定字符串集，导致mysql存储中文会出错，web网站也显示不了中文。修改字符集为utf-8后，显示准确。

5.最后一个问题，在重新构建qt项目是，发现生成的sqlite数据库没有重新构建。

原因是当qt项目重构时，并没有清理原来生成的db文件，而是保留了，要想重构时删除db文件，可以手动删除，也可以手动更改构建选项。

6.一个思考

是否可以不用python脚本，将mqtt接收集成到web应用中，用java写一个订阅mqtt消息并解析的脚本，这样可以简化工程的复杂性。

在Java中，可以使用Eclipse Paho MQTT客户端库，编写一个脚本嵌入到web应用中，这个方法也是可行的，实现起来应该也不难。

总结和收获

这次实验是在上学期RFID实验的基础上加以改进，使其上云。通过这次实验，我把上次实验时没弄明白的地方基本上都想清楚了，对qt项目的搭建也更熟悉了，还复习了数据库的相关操作。

另外这次试验因为时间有限，也没有实现很完备的功能，但是主要功能均已实现，技术啊上没有问题，要想增加功能的话需要花时间完善代码。

在做项目的过程中，我发现对qt槽函数的运用非常重要，Qt框架中的信号和槽机制提供了一种强大且灵活的方式来实现对象之间的通信。槽函数作为信号的目的地，具有以下优势：信号和槽机制使得发送信号的对象不需要知道任何关于接收对象的信息，反之亦然。这种解耦合促进了代码的模块化和重用性。槽函数可以在信号发出后异步执行，这意味着它们不会阻塞发送信号的代码。这对于提高应用程序的响应性和性能至关重要。一个信号可以连接到多个槽函数，使得多个对象可以响应同一个事件。Qt的信号和槽机制支持跨线程通信。槽函数可以在与信号发出不同的线程中执行，使得多线程编程更加容易。槽函数可以定义在任何继承自QObject的类中，这使得它们可以在应用程序的任何部分使用。

在本项目中，从串口读卡的操作和mqtt消息的发送都依赖于槽函数的实现。

关于数据库，为什么我要在本地保留数据库呢，因为我希望在本地保留相关数据，当数据库需要增删查改时的逻辑处理可以在本地先实现，然后进行相关数据库操作时再给服务器发送mqtt消息，这样避免了服务器还要从云上获取数据，然后给本地程序发送处理消息用来支持逻辑处理。另外当本地数据量较小时，选用qt自带的sqlite数据库就够用了，因为不用配相关环境，qt项目生成时便自动生成该db文件，管理方便。

使用mqtt的时候我意识到，MQTT的异步性是其核心特性之一，它为物联网和实时通信领域带来了显著的优势。MQTT异步性的优势在于： MQTT采用发布/订阅模式，允许消息的异步传递，发送者（发布者）和接收者（订阅者）之间不需要直接通信，从而实现了解耦，提高了系统的可扩展性和灵活性 。由于消息传递是异步的，发布者在发送消息后不需要等待接收者的响应，可以立即返回，这减少了等待时间，提高了系统的响应性。 MQTT的异步通信模型支持大规模的设备连接和实时数据传输，使得它非常适合于物联网场景，其中可能涉及成千上万的设备 。但同时，这种异步性可能带来问题，当mqtt连接并发送消息时，我们无法知道是否连接成功，如果还没有连接成功，此时发消息是收不到的。因此，我们需要在接受到相应的连接成功信号后，再进行发送消息(publish)操作。

通过这次的实践和学习，使我可以有效地将不同的技术（如RFID、MQTT、数据库和Qt开发）结合起来，解决实际问题。这种跨领域的技术整合能力对于成为一名优秀的软件工程师至关重要。