东软教育科技集团

项目实训总结报告

项 目 名 称:智慧医疗管理系统

学 校 名 称:北京理工大学

项 目 讲 师:夏军

姓 名:谢宝玛

学 号:1120233506

实 训 日 期: 2025年8月25日-9月3日

**1 引言**

1.1项目概述

本次智慧医疗管理系统实训项目的主要目标是通过构建一个集医疗信息交互、数据存储、客户端界面和后端服务为一体的综合性管理平台，实现患者端与医生端的高效通信、病历数据的安全存储以及智能化的信息管理。  
背景方面，随着“健康中国”战略与“智慧医疗”理念的推进，传统医疗模式逐渐向数字化、智能化转型，本系统的设计与实现旨在为未来智慧医院建设提供参考价值。  
项目意义在于：掌握网络通信与数据库结合的工程实现方法；训练跨平台开发与前后端协作的能力；探索智慧医疗应用场景下的技术方案与实现难点；培养解决复杂工程问题的实践经验。

参考资料包括 Qt 官方文档、SQLite 官方文档、《UNIX网络编程》、以及开源项目 nlohmann/json 的相关说明与示例。

1.2 运行环境

1.2.1硬件环境：

开发机：vmware 虚拟机 Linux系统

服务器：Ubuntu 24.04 64位，2 核（vCPU）2 GiB软件环境

1.2.2 软件环境：

开发语言：C++17

客户端框架：Qt 6.5

数据库：SQLite3

开发工具：Qt Creator, g++

主要库与技术：Qt Socket 库、nlohmann/json

**2 需求分析**

|  |  |
| --- | --- |
| **模块** | **细化需求** |
| **用户信息验证** | 注册和登录界面的信息输入，信息确认按钮，角色切换等 |
| 注册和登录用户信息的云端上传 |
| 重置密码界面的信息输入等 |
| 用户信息验证界面用户信息与云端的传输 |
| 云端数据库存储用户相关信息 |
| **患者端** | 患者端主页面，各个功能模块的跳转与页面UI |
| 患者端相关查询操作指令与数据与云端的传输（查询医生信息，修改个人信息，查询病历等） |
| 患者端相关数据的云端增删改查操作（查询医生信息，修改个人信息，查询病历等） |
| 云端挂号队列的配置与查询等操作 |
| **医生端** | 医生端主页面，各个功能模块的跳转与页面UI |
| 患者端相关查询操作指令与数据与云端的传输（修改个人信息，打卡，提供医嘱等） |
| 医生端相关数据的云端增删改查操作（修改个人信息，打卡，提供医嘱等） |
| 医生端接诊功能的云端实现 |
| **医患沟通平台** | 医患沟通平台主页面，验证页面等的前端UI |
| 医患沟通平台与云端的数据传输，指令传输等 |
| 医患沟通平台的云端实现（配置信息，链接等） |
| **统筹** | 总体项目的分工与统筹 |
| 项目前端的分工与统筹 |
| 项目后端（云端+通信）的分工与统筹 |
| **项目部署** | 代码评审 |
| Linux虚拟机配置与代码部署 |
| 云服务器部署 |
| **文档** | 前期评审相关文档 |
| 中期评审相关文档 |
| 结项相关文档 |
| 模块概要设计说明书 |

**3 概要设计**

系统采用 C/S 架构，整体分为客户端、服务器端与数据库三部分：客户端（Qt）：主要用于患者/医生交互，功能包括登录注册、病历查看、远程消息传输等，界面友好且跨平台。

3.1 服务器端（C++）：核心负责处理客户端请求，管理数据通信、解析 JSON 消息、与数据库交互。

3.2 数据库（SQLite）：用于存储用户基本信息、病历资料、消息记录等。

关键技术点：

3.3 Socket 通信：基于 TCP 协议，保证数据传输的可靠性。

IO复用（epoll）：支持高并发连接，避免阻塞，提高系统吞吐量。

3.4 JSON 数据传输：统一数据格式，保证跨平台可解析性；使用 nlohmann/json 进行序列化与反序列化。

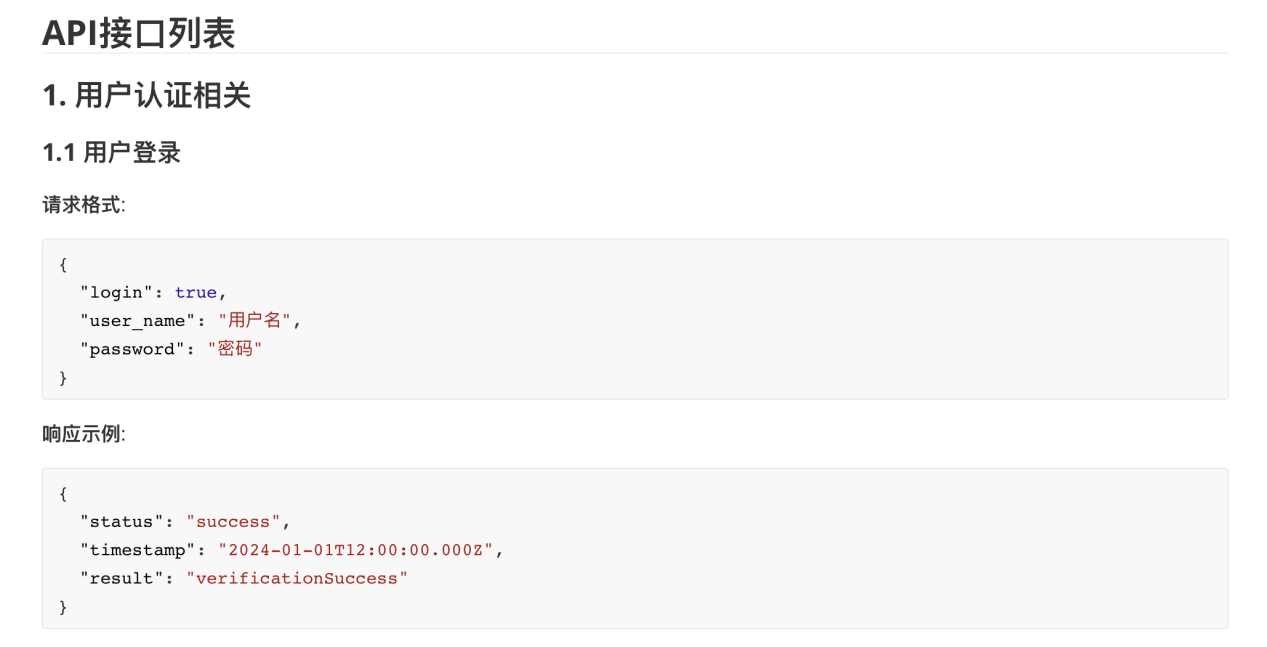
3.5 Qt 客户端界面：利用 Qt 的信号与槽机制，实现实时交互。

**4 实现方案**

4.1网络通信模块

使用 socket API 实现服务器监听与客户端连接。

数据交互统一采用 JSON 格式，包含请求类型（登录、查询、上传等）与数据内容。



4.2数据库模块（SQLite）

设计用户表（User）、病历表（Record）、消息表（Message）等结构。

通过 C++ API 操作 SQLite，支持增删改查。

结合事务机制保证数据一致性与安全性。

4.3JSON数据库服务器

这是一个整合的JSON数据库服务器程序，能够接收客户端的JSON请求，处理数据库操作，并返回JSON格式的结果。

功能特性

4.3.1网络通信: 基于TCP Socket的客户端-服务器通信

4.3.2JSON处理: 自动解析JSON数据并根据内容执行相应操作

4.3.3数据库操作: 支持SQLite数据库的增删改查操作

4.3.4多线程支持: 每个客户端连接使用独立线程处理

4.3.5完整日志: 详细的操作日志记录

4.3.6错误处理: 完善的异常处理和错误响应机制

4.3.7守护进程: 支持后台守护进程模式运行

4.3.8进程管理: 完整的启动/停止/重启/状态查看功能

4.3.9信号处理: 优雅的关闭和重启机制

4.3.10PID管理: 防止重复启动，支持进程监控

4.4文件说明

4.4.1核心文件

integrated\_server.py - 整合的服务端程序（支持守护进程模式）

test\_client.py - 测试客户端程序

check\_database.py - 数据库查看和管理工具

server\_control.sh - 服务器控制脚本（便于管理）

json-db-server.service - systemd服务配置文件

4.4.2原始文件（已整合）

execute\_json.py - JSON处理和数据库操作功能

recv\_json.py - 网络接收功能

send\_json.py - 网络发送功能

4.5支持的操作类型

4.5.1. 用户密码重置

{  
     "reset\_password": true,  
     "user\_name": "13800138000",  // 手机号或工号  
     "new\_password": "new\_password\_hash"  
 }

4.5.2. 患者信息更新

{  
     "reset\_patient\_information": true,  
     "old\_phone": "13800138000",  
     "new\_name": "新姓名",  
     "new\_birth\_date": "1990-01-01",  
     "new\_id\_card": "新身份证号",  
     "new\_phone": "新手机号",  
     "new\_email": "new@example.com"  
 }

4.5.3. 医生信息更新

{  
     "reset\_doctor\_information": true,  
     "old\_employee\_id": "DOC001",  
     "new\_name": "医生姓名",  
     "new\_employee\_id": "新工号",  
     "new\_department": "科室",  
     "new\_max\_patients": 40,  
     "new\_fee": 80.0,  
     "new\_work\_schedule": "{\"monday\": \"9:00-17:00\"}",  
     "new\_is\_available": true,  
     "new\_photo\_path": "/path/to/photo.jpg"  
 }

4.5.44. SQL查询执行

{  
     "sql\_query": "SELECT p.name, p.phone, d.name as doctor\_name FROM patients p JOIN appointments a ON p.patient\_id = a.patient\_id JOIN doctors d ON a.doctor\_id = d.doctor\_id"  
 }

常用查询示例：

查询所有用户信息：

{  
    "sql\_query": "SELECT user\_id, username, role, created\_at FROM users"  
}

查询患者详细信息：

{  
    "sql\_query": "SELECT p.name, p.phone, p.email, u.username FROM patients p JOIN users u ON p.patient\_id = u.user\_id"  
}

查询医生及其预约情况：

{  
    "sql\_query": "SELECT d.name, d.department, COUNT(a.appointment\_id) as appointment\_count FROM doctors d LEFT JOIN appointments a ON d.doctor\_id = a.doctor\_id GROUP BY d.doctor\_id"  
}

查询病历信息：

{  
    "sql\_query": "SELECT p.name as patient\_name, d.name as doctor\_name, m.diagnosis, m.visit\_time FROM medical\_records m JOIN patients p ON m.patient\_id = p.patient\_id JOIN doctors d ON m.doctor\_id = d.doctor\_id"  
}

4.5.5. 数据插入

{  
    "table\_name": "students",  
    "name": "张三",  
    "age": 20,  
    "student\_id": 12345  
}

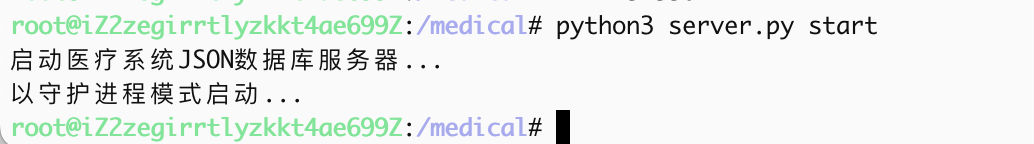
4.6使用方法

启动服务器

现在服务器支持多种启动方式：

4.6.1. 后台守护进程模式（推荐）

启动服务器  
python integrated\_server.py start

  
4.6.2. 前台调试模式

前台启动（用于调试，按Ctrl+C停止）  
python integrated\_server.py start --foreground

  
​  
4.6.4. 自定义配置启动

指定自定义配置  
python integrated\_server.py start --host 127.0.0.1 --port 8080 --db-path ./my.db --log-file ./my.log

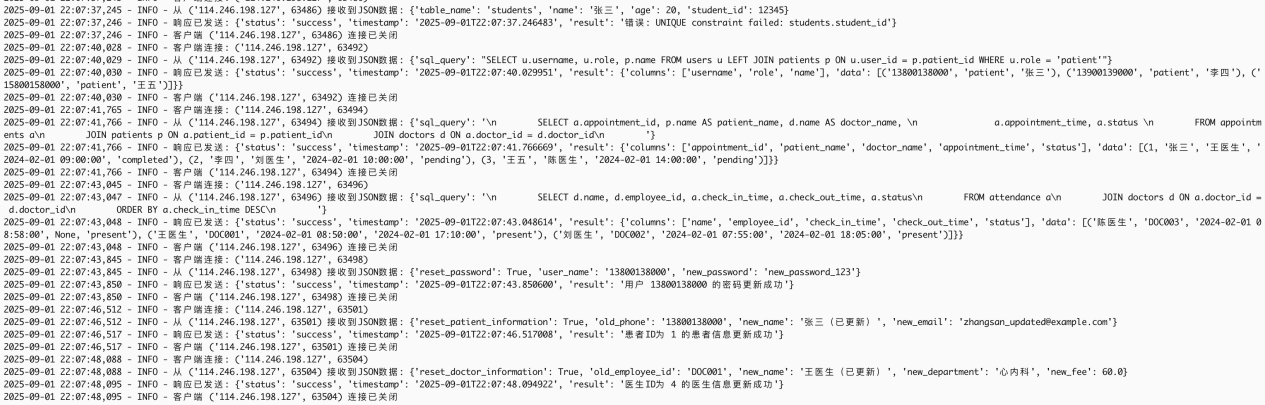
服务器默认在 0.0.0.0:55000 上启动并等待客户端连接。

运行测试客户端

python test\_client.py

测试客户端将执行各种类型的操作测试。

服务器响应格式



成功响应

{  
    "status": "success",  
    "timestamp": "2024-01-01T12:00:00",  
    "result": "操作结果"  
}

错误响应

{  
    "status": "error",  
    "timestamp": "2024-01-01T12:00:00",  
    "error": "错误信息"  
}

4.7工作流程

4.7.1

接收连接: 服务器监听端口，接受客户端连接

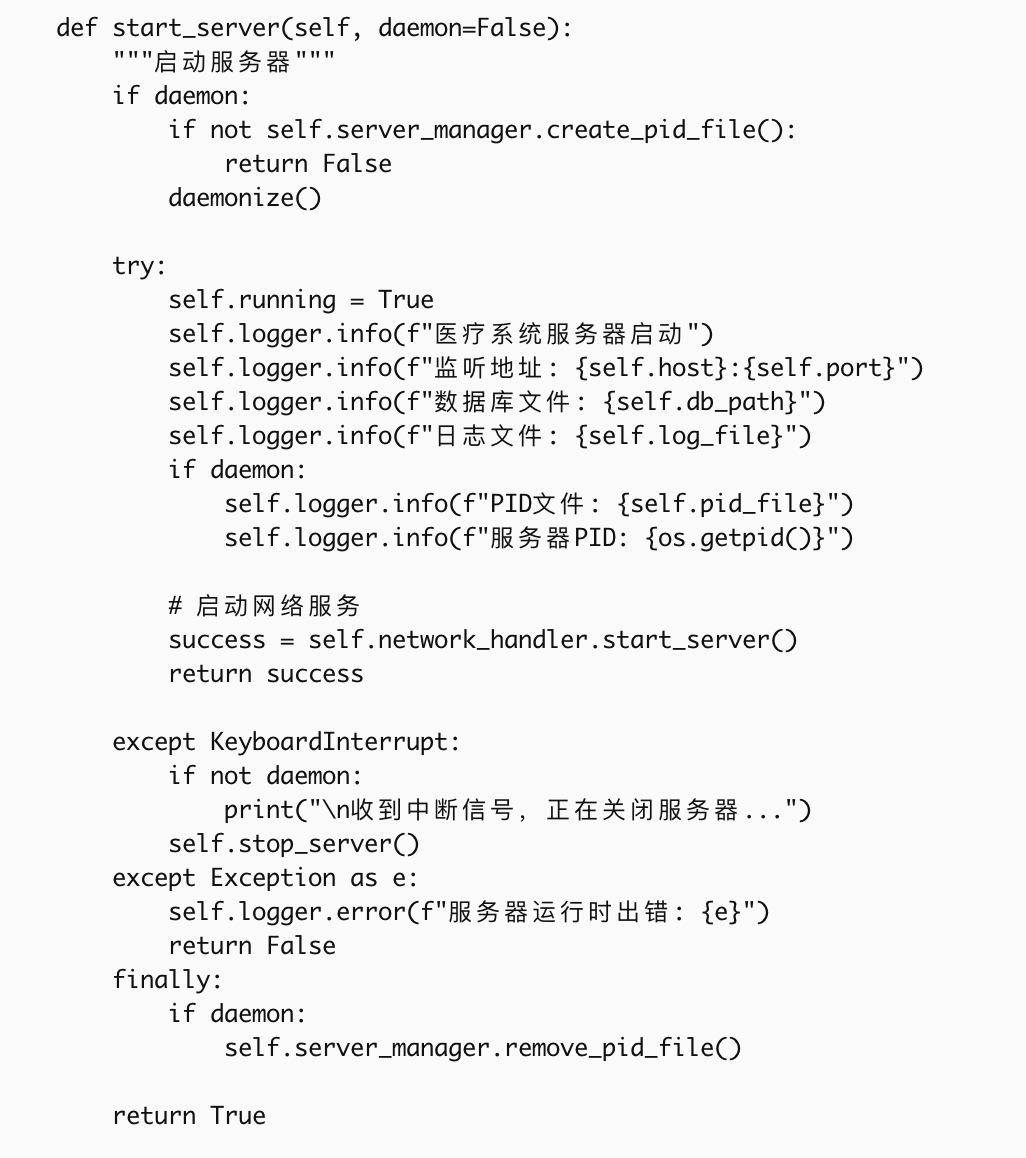
接收JSON数据: 按照协议接收客户端发送的JSON数据

数据处理: 根据JSON内容判断操作类型并执行相应的数据库操作

返回结果: 将处理结果封装为JSON格式返回给客户端

循环处理: 服务器持续运行，处理多个客户端请求

核心代码：



4.7.2网络协议

客户端发送格式

文件名长度 (4字节, 大端序)

文件名 (UTF-8编码)

文件大小 (4字节, 大端序)

JSON内容 (UTF-8编码)



4.7.3服务端响应格式

响应长度 (4字节, 大端序)

JSON响应内容 (UTF-8编码)

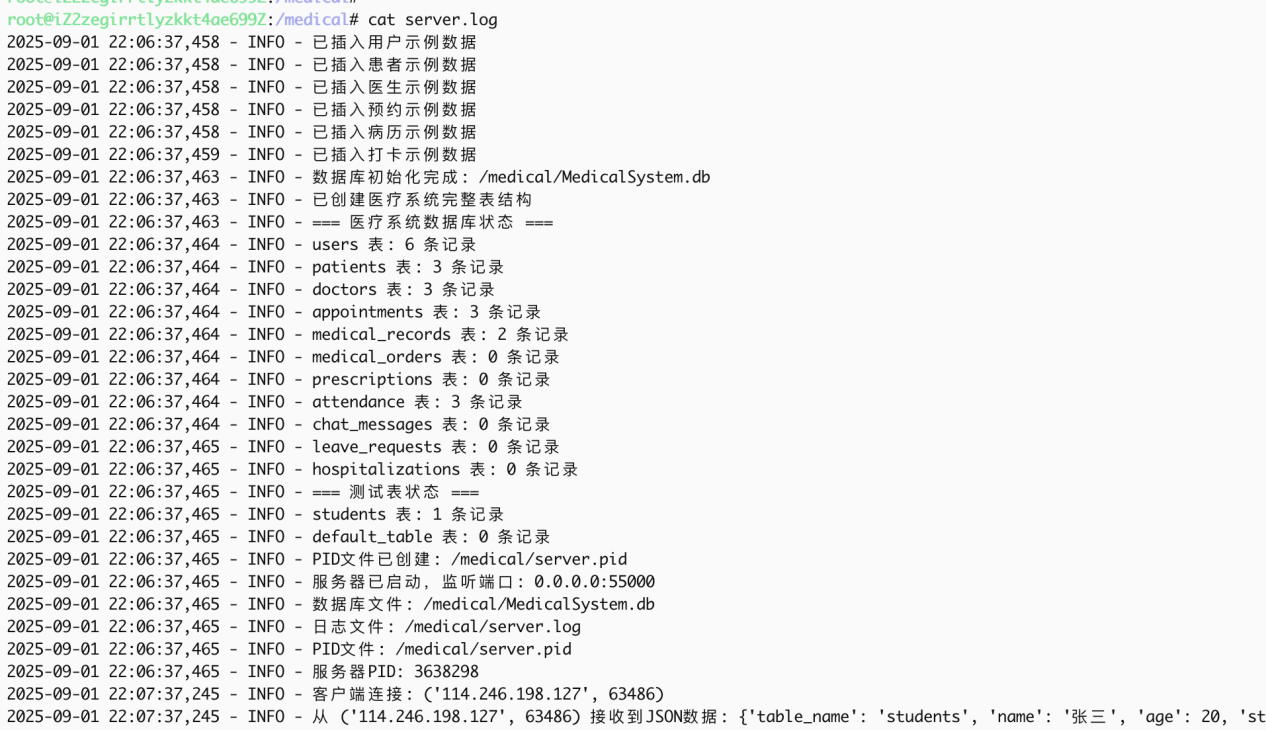
4.7.4日志记录和监控

日志文件

服务器默认将日志记录到 /medical/server.log：

# 查看实时日志  
 tail -f /medical/server.log  
​ # 或使用控制脚本  
 ./server\_control.sh logs

日志内容



日志包含：

服务器启动/停止信息

客户端连接信息

请求和响应内容

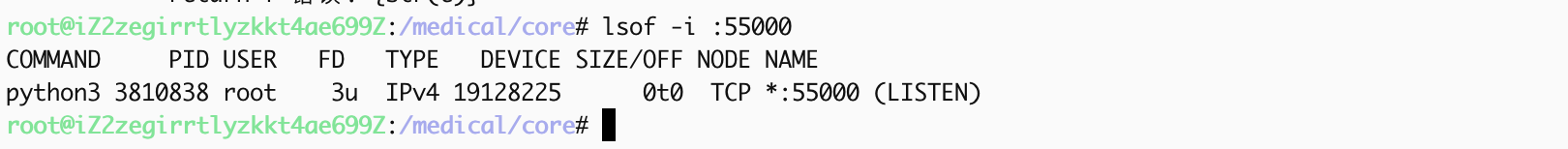
数据库操作记录

错误信息和堆栈跟踪

操作时间戳

4.7.5进程监控

# 查看服务器状态  
 python integrated\_server.py status  
​  
 # 查看进程信息  
 ps aux | grep integrated\_server  
​  
 # 查看端口使用情况  
 netstat -tulpn | grep :55000  
​  
 # 或使用ss命令  
 ss -tulpn | grep :55000



4.8数据库要求

服务器使用SQLite数据库，默认文件名为 database.db。

⚠️ 数据库自动初始化: 服务器启动时会自动创建所需的数据库表和示例数据，无需手动创建！

4.8.1医疗系统完整表结构：

核心用户表

users 表: user\_id, username, password\_hash, role, created\_at

医疗人员信息表

patients 表: patient\_id, name, birth\_date, id\_card, phone, email, created\_at

doctors 表: doctor\_id, name, employee\_id, department, photo\_path, max\_patients, fee, work\_schedule, is\_available

医疗业务表

appointments 表: appointment\_id, patient\_id, doctor\_id, appointment\_time, status, fee\_paid, queue\_number, created\_at

medical\_records 表: record\_id, patient\_id, doctor\_id, diagnosis, symptoms, visit\_time, created\_at

medical\_orders 表: order\_id, record\_id, doctor\_id, content, created\_at

prescriptions 表: prescription\_id, record\_id, doctor\_id, content, created\_at

hospitalizations 表: hospitalization\_id, patient\_id, doctor\_id, ward\_number, bed\_number, admission\_date, discharge\_date, status, created\_at, updated\_at

管理功能表

attendance 表: attendance\_id, doctor\_id, check\_in\_time, check\_out\_time, status

leave\_requests 表: leave\_id, doctor\_id, start\_date, end\_date, reason, status, created\_at

chat\_messages 表: message\_id, sender\_id, receiver\_id, content, sent\_at, is\_read

测试用表

students 表: id, name, age, student\_id, created\_at（用于测试）

default\_table 表: id, data, created\_at（通用插入测试）

4.8.2数据库特性

外键约束: 确保数据完整性

角色管理: 患者和医生角色区分

医生管理: 打卡、请假、排班管理

住院管理: 病房、床位管理

完整的医疗业务流程: 预约→就诊→病历→处方

4.8.3数据库管理工具

使用 check\_database.py 查看和管理数据库：

# 查看数据库内容和状态  
 python check\_database.py

4.8.4注意事项

(1)基本配置

数据库自动初始化: 首次启动服务器时会自动创建数据库和所有必要的表

网络安全: 服务器默认监听所有网卡 (0.0.0.0)，生产环境请注意防火墙设置

SQL安全: 使用参数化查询防止SQL注入攻击

并发支持: 服务器支持多客户端并发连接

事务处理: 所有数据库操作都有完整的事务和异常处理

(2)文件权限和目录

目录权限: 确保 /medical/ 目录存在且有写入权限

日志文件: 日志文件路径需要有写入权限

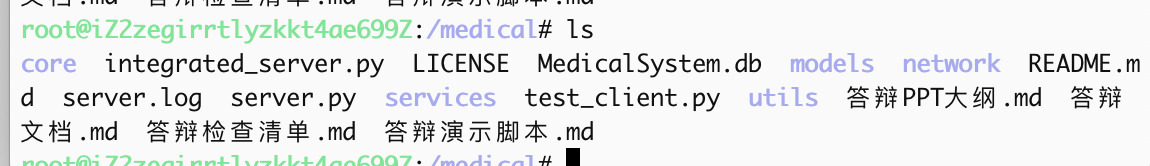
PID文件: PID文件路径需要有写入权限

(3)故障排除

重置数据库: 如果遇到数据库相关错误，可以使用 check\_database.py --reset 重置数据库

端口占用: 如果启动失败，检查端口是否被占用：lsof -i :55000

进程清理: 如果服务异常退出，可能需要手动删除PID文件：rm /medical/server.pid



(4)后台运行注意事项

守护进程: 后台模式下服务器会自动转为守护进程，脱离终端运行

日志输出: 后台模式下所有输出都会重定向到日志文件

信号处理: 支持 SIGTERM、SIGINT、SIGQUIT 信号优雅关闭

自动重启: 服务异常退出时，可配置systemd自动重启

(5)性能和资源

内存使用: 每个客户端连接会创建一个线程，注意监控内存使用

数据库锁: SQLite在高并发时可能出现锁等待，生产环境建议使用PostgreSQL

文件句柄: 大量并发连接时注意系统文件句柄限制

(6)开发和调试

修改 JSONDatabaseServer 类的构造参数可以改变监听地址、端口和数据库路径

日志级别可以通过修改 setup\_logging() 方法调整

添加新的操作类型需要在 process\_json\_data() 方法中添加相应判断逻辑

(7)示例使用

启动服务器后，可以使用任何支持TCP Socket的客户端发送JSON数据进行测试，或者使用提供的 test\_client.py 进行功能验证。

**5 系统测试**

测试方法：

功能测试：对登录注册、消息收发、病历上传与查询进行功能性验证。

压力测试：模拟 100+ 客户端同时接入，验证 epoll 模型下的系统稳定性。

数据库测试：检查 SQLite 在高并发下的读写正确性。



测试发现的问题及修复：

Bug1：客户端 JSON 格式错误导致解析异常 → 通过增加 JSON schema 校验解决。

Bug2：epoll 下部分长连接超时未清理 → 增加心跳包机制，定期检测连接存活。

Bug3：SQLite 在高并发写入下锁冲突 → 引入写操作排队机制，缓解竞争。

预期结果：  
系统在 100 个并发连接下能够稳定运行，消息延迟控制在 200ms 内，病历数据能够正确写入并查询，达到预期目标。

**6 项目经验及心得体会**

通过本次智慧医疗管理系统项目，我主要负责后端开发，收获非常丰富。首先在 **网络通信模块** 上，实际操作 socket API 与 Qt Socket 库让我对阻塞、非阻塞、IO复用等概念有了更深刻的理解。特别是在实现 epoll 事件驱动时，我第一次直观感受到高并发编程中状态管理与资源释放的重要性。

在 **数据传输** 层面，使用 JSON 格式是一次实践选择。最初我以为 JSON 的序列化/反序列化是简单操作，但在实际调试中发现，字段丢失、编码问题、客户端/服务器版本不一致等情况都会引发解析错误。通过 nlohmann/json 这个开源库的使用，我学会了如何优雅地处理异常输入，同时也认识到接口定义的重要性——后端必须与前端建立严格的协议文档。

在 **数据库交互** 上，SQLite 的轻量化特性非常适合本项目。但它在高并发写入时的锁冲突问题让我意识到：选择合适的数据库需要考虑应用规模与并发特性。通过事务与写队列的方式，我体会到工程实践中“权衡”的艺术。

在 **系统稳定性** 方面，我曾遇到过客户端断开连接后服务器未能及时清理资源的问题，导致内存泄漏。经过查阅文档与反复调试，我增加了心跳机制与连接回收逻辑，从而保障了长时间运行的稳定性。这次经历让我认识到软件开发不仅是功能实现，更重要的是可靠性保障。

个人最大的成长是 **团队协作** 与 **工程思维**。在与前端开发同学对接时，我们因为接口定义不清楚而出现过多次数据解析失败。后来我们决定先制定统一的 JSON 协议文档，再分别实现，效率与成功率大幅提升。这让我认识到工程项目必须有规范与文档，而不仅仅是代码。

此外，本次实训让我体会到智慧医疗的社会意义。通过软件技术能够减少患者排队、实现远程诊疗，不仅提升了医疗效率，也能让更多人享受公平的医疗资源。这与我所学的计算机科学结合紧密，也让我更加坚定未来从事技术研发的决心。

总之，本次项目让我在 **网络编程、数据库管理、跨平台开发与系统测试** 等方面获得了全面的训练。尽管过程中遇到了很多困难，但最终能够实现预期目标，并且深刻理解了智慧医疗系统对社会的价值。这将成为我今后学习与科研道路上的宝贵经验。