期末实训总结报告

Message即使通讯软件

项 目 名 称: Message即时通讯软件

学 校 名 称:重庆师范大学

项 目 导 师: 冉瑞生

姓 名: 杨云博 杜江鑫

学 号: 2021051615135 2021051615113

实 训 日 期: 2024年6月16日-6月30日

**1 引言**

1.1项目概述

项目目的：

开发一种应用程序，旨在实现用户之间的实时通信和交流。该项目的主要目标是提供快速、便捷的沟通方式，使用户能够即时发送和接收消息。

项目背景;

随着技术的不断发展和普及，人们对即时通信的需求也不断增加。传统的邮件和电话通话已经无法满足人们迅速交流的需求，因此即时聊天软件项目应运而生，以提供更快速、便捷的通信方式。

项目意义：

提供一种快速、即时的沟通方式，使用户能够实时发送和接收消息，与其他人进行实时交流。这种实时性促进了信息的快速传递和即时反馈，提高了沟通的效率和便捷性。

1.2 运行环境

|  |  |  |
| --- | --- | --- |
| 计算机硬件 | 操作系统：Windows 10/macOS 10.14+/Ubuntu 18.04+或类似的操作系统。  处理器：Intel Core i5或AMD Ryzen 5以上的四核处理器，或类似性能的处理器。  内存：8GB或更高的内存。  存储空间：至少5GB的可用存储空间。 | 操作系统：Windows 7+/macOS 10.12+/Ubuntu 16.04+或类似的操作系统。  处理器：双核处理器，如Intel Core i3或AMD Ryzen 3，或类似性能的处理器。  内存：4GB的内存。  存储空间：至少2GB的可用存储空间。 |
| 软件 | Message全部组件即可 | Message全部组件即可 |
| 网络通信 | 网络协议：使用TCP/IP协议栈进行网络通信。 | 网络协议：使用TCP协议进行基本的网络通信。 |

**2 需求分析**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **序号** | **大分类（模块）** | **小分类（子模块）** | **详细说明** | **担当者** |
| 1 | 客户端 | 局域网连接模块 | 使客户端应用程序能够在局域网中进行网络通信、发现其他设备、交换数据、错误处理和日志记录等操作。 | 黄思睿、傅怀毅 |
| 2 | 客户端 | 文件传输模块 | 实现文件选择和浏览，文件传输协议，上传和下载，传输进度和状态显等操作。 | 黄思睿、胡宇航、傅怀毅 |
| 3 | 客户端 | 用户数据库 | 数据库连接，数据库查询和检索，数据插入、更新和删除，事物模块，数据验证和约束，错误处理和日志记录。 | 夏诗航、张正伦 |
| 4 | 客户端 | 分组管理模块 | 实现用户分类和分组管理，以及实现权限控制，分组操作等。 | 黄思睿、胡宇航、傅怀毅 |
| 5 | 客户端 | 客户端登录模块 | 实现账户注册和用户登录以及用户验证和密码重置以及个人信息管理。 | 黄思睿、胡宇航、夏诗航、张正伦、傅怀毅、张钦赫 |
| 6 | 客户端 | 聊天和信息处理模块 | 实时通信，消息发送与接受，群聊功能消息提醒和通知，消息状态和已读确认，多媒体消息支持等 | 黄思睿、胡宇航、夏诗航、张正伦、傅怀毅、张钦赫 |
| 7 | 服务端 | 局域网连接模块 | 使服务端能够接收各客户端发送的网络通信数据，或是作为中转站连接起各客户端的网络通信 | 黄思睿、傅怀毅 |
| 8 | 服务端 | 文件传输模块 | 针对非聊天数据的数据进行客户端与服务端之间的传输，实现安全、高效的局域网数据传递 | 黄思睿、胡宇航、傅怀毅 |
| 9 | 服务端 | 检查维护模块 | 设置检查点机制，能够对服务端接收的数据进行一定时间的存储，以便对异常情况的处理，保障客户端、服务端信息不丢失即能够恢复 | 夏诗航、张正伦、傅怀毅 |
| 10 | 服务端 | 用户数据数据库模块 | 对用户的各类通信数据记录进行存储 | 夏诗航、张正伦 |
| 11 | 服务端 | 错误处理模块 | 建立针对不同异常状态的错误处理反应机制，尽可能保障软件系统的正常运行，能够更大限度地减弱异常状态对软件系统各部分的消极影响 | 黄思睿、胡宇航、张正伦、傅怀毅 |
| 12 | 服务端 | 安全处理模块 | 对客户端、服务端访问记录进行甄别，防止恶意访问等情况下数据泄漏、丢失等危险事件的发生 | 黄思睿、胡宇航、张正伦、傅怀毅 |
| 13 | 服务端 | 管理操作模块 | 便于管理人员对软件系统的维护更新等操作，减少管理过程中可能的失误带来的影响 | 黄思睿、胡宇航、夏诗航、张正伦、傅怀毅、张钦赫 |

**3 概要设计**

本项目的系统设计概要如下：

用户管理系统方面，实现用户注册和登录功能，用户信息存储和管理以及用户身份验证和安全性。

通讯协议方面，选择UDP协议实现消息传输和接收的协议规范。

消息传输方面，实现实时消息传输，包括文本、图片、音频和视频。

联系人管理方面，用户可以添加、删除、编辑联系人，可以实时同步联系人列表。

平台支持方面，支持linux操作系统。

消息存储和历史记录方面，存储用户消息历史记录，以便在需要时进行检索。

安全性方面，通过数据传输的加密以及用户身份验证和授权，防止恶意攻击。

性能和可伸缩性方面，高并发支持，可以处理较多用户同时在线。

用户界面设计方面，直观简洁，美观实用。

监控和分析方面，设计日志文件功能实时记录用户的操作，以跟踪应用程序性能和问题，收集分析数据以改进用户体验。

合规性方面，遵守数据隐私法规，保障用户数据的隐私性。

本项目的系统设计概要如下：

本项目使用QSQLLITE数据库，建立了people,friend,chat三个表分别记录用户信息，好友信息和文件信息，聊天信息，表结构如下：

1. People Table

|  |  |  |  |
| --- | --- | --- | --- |
| 属性名 | 属性中文名 | 类型 | 说明 |
| id | 用户i*d* | INTEGER | 主键  NOT NULL |
| name | 用户名 | TEXT | NOT NULL |
| password | 用户密码 | TEXT | NOT NULL |
| ip | 用户ip | TEXT | NOT NULL |
| islogin | 用户登录状态 | INTEGER | NOT NULL(1为已登录，0为未登录) |

1. friend Table

|  |  |  |  |
| --- | --- | --- | --- |
| 属性名 | 属性中文名 | 类型 | 说明 |
| id | 好友id | INTEGER | 主键 |
| name | 好友用户名 | TEXT |  |
| sendmassage | 是否准备向该好友发送消息 | INTEGER | 为1说明准备向该好友发送消息（已执行void wantsendmessage();还未执行void chat\_send() ;）  其余状态下为0 |
| sendfile | 是否准备向该好友发送文件 | INTEGER | 为1说明准备向该好友发送文件（已执行 void want\_send\_file() ; 还未执行void send\_file\_ok() ; 或 void send\_file\_miss() ;）  其余状态下为0 |

1. chat \_Aid\_Bid Table(A,B为通讯双方且Aid < Bid)

|  |  |  |  |
| --- | --- | --- | --- |
| 属性名 | 属性中文名 | 类型 | 说明 |
| time | 消息时间 | datetime | 主键 |
| id | 发送者id | INTEGER |  |
| ip | 发送者ip | TEXT |  |
| Message | 消息内容 | TEXT |  |

本项目实现的主要功能包括用户的登录账号，注册账号，退出账号，消息发送，查看历史消息，添加好友，删除好友，查看好友列表，发送文件以及日志文件记录用户行为。

为实现上述核心功能，设计了13个函数void login(); void registered(); void wantsendmessage(); voidchat\_history(); void chat\_send() ; void logout(); void getfriendlist() ; void add\_friend() ; void delete\_friend() ; void want\_send\_file() ; void send\_file\_ok() ; void send\_file\_miss() ; void log(const QString &message);

其中，除了void log(const QString &message);函数在上述任意函数执行时随之执行，从而达到记录用户操作的目的外，其他12个函数均是根据用户指令被调用执行，实现相应功能。

**4 实现方案**

Message即时通讯软件的服务端接受用户指令，并根据指令调用相应函数实现用户的登录账号，注册账号，退出账号，消息发送，查看历史消息，添加好友，删除好友，查看好友列表，发送文件等功能。下文将详细介绍各个函数的核心代码以及日志文件的实现。

1.login()登录函数

***void server::login(){***

***db.setDatabaseName("./people.db");***

***db.open();***

***QSqlQuery sqlquery;***

***sqlquery.prepare("select \* from people where name = :name");***

***sqlquery.bindValue(":name",QString(buffer).section("##",1,1));***

***sqlquery.exec();***

***if(!sqlquery.next())***

***{//未查找到该用户***

***tcpSocket[0]->write(QString("login error##no\_user").toUtf8());***

***tcpSocket[0]->flush();***

***db.close();***

***}***

***else***

***{//用户存在***

***//用户存在***

***//查询用户是否已经登录***

***// 获取 "islogin" 字段的值***

***int isLoginValue = sqlquery.value(4).toInt();***

***// 根据查询结果进行处理***

***if (isLoginValue == 1) {***

***tcpSocket[0]->write(QString("login error##again\_user").toUtf8());***

***tcpSocket[0]->flush();***

***}***

***else{***

***int id = sqlquery.value(0).toInt();***

***QString pwd = sqlquery.value(2).toString();***

***if(pwd == QString(buffer).section("##",2,2))***

***{//登录成功***

***tcpSocket[0]->write(QString("login successed##%1").arg(id).toUtf8());***

***sqlquery.prepare("update people set ip=:ip, islogin=1 where name = :name");***

***sqlquery.bindValue(":ip",ip);***

***sqlquery.bindValue(":name",QString(buffer).section("##",1,1));***

***sqlquery.exec();***

***tcpSocket[0]->flush();***

***//更新服务器界面***

***ui->listWidget->clear();***

***sqlquery.prepare("select \* from people where islogin = 1");***

***sqlquery.exec();***

***if(sqlquery.next())***

***{***

***QString userid = sqlquery.value(0).toString();***

***QString username = sqlquery.value(1).toString();***

***QString userip = sqlquery.value(3).toString();***

***//qDebug()<<userid;***

***ui->listWidget->insertItem(0,"用户ID："+userid+",用户昵称:"+username+",用户IP:"+userip);***

***int rownum = 1;***

***while (sqlquery.next())***

***{***

***QString userid = sqlquery.value(0).toString();***

***QString username = sqlquery.value(1).toString();***

***QString userip = sqlquery.value(3).toString();***

***ui->listWidget->insertItem(rownum,"用户ID："+userid+",用户昵称:"+username+",用户IP:"+userip);***

***rownum++;***

***}***

***}***

***else***

***{***

***ui->listWidget->clear();***

***ui->listWidget->insertItem(0,tr("当前无在线用户"));***

***}***

***}***

***else***

***{//密码错误***

***tcpSocket[0]->write(QString("login error##errpwd").toUtf8());***

***tcpSocket[0]->flush();***

***db.close();***

***}***

***}***

***}***

***}***

这个函数的主要功能是处理用户登录请求，包括验证用户身份、检查是否已登录以及更新服务器界面上的用户列表。它使用了一个名为 "people.db" 的数据库，并根据输入数据中的用户名和密码执行相应的操作。

首先，它设置了一个数据库连接对象（db），并指定了数据库文件的路径为 "./people.db"。然后打开了数据库连接。接下来，创建了一个SQL查询对象 sqlquery，并使用 prepare 方法准备了一个SQL查询语句，用于从名为 "people" 的数据库表中查找符合特定用户名条件的记录。使用 bindValue 方法将查询语句中的参数 ":name" 绑定为从输入数据中提取的用户名。这是通过将 buffer 字符串使用 "##" 分隔后提取的。执行 SQL 查询语句，并使用 sqlquery.next() 来检查是否找到了匹配的记录。如果没有找到记录，则向客户端发送 "login error##no\_user" 消息，刷新缓冲区，然后关闭数据库连接。如果找到了匹配的记录，表示用户存在。继续处理登录操作。查询该用户的 "islogin" 字段的值，存储在 isLoginValue 中。

根据 isLoginValue 的值，如果为1，表示用户已经登录过，向客户端发送 "loginerror##again\_user" 消息，刷新缓冲区。如果 isLoginValue 不为1，表示用户未登录，继续验证密码。获取用户的ID和密码，分别存储在 id 和 pwd 中。

检查输入的密码是否与数据库中存储的密码匹配，如果匹配成功，向客户端发送 "login successed##<用户ID>" 消息，同时更新数据库中该用户的IP和登录状态为已登录。

向客户端刷新缓冲区，然后清空服务器界面上的用户列表。再次准备一个SQL查询，用于查找所有已登录的用户，并将他们的信息显示在服务器界面的列表中。循环遍历查询结果，将每个已登录用户的ID、用户名和IP添加到服务器界面的列表中。

1. registerIn()函数

***void server::registerIn(){***

***db.setDatabaseName("./people.db");***

***db.open();***

***QSqlQuery sqlquery;***

***//注册用户的时候需要进行判重***

***sqlquery.prepare("select \* from people where name = :name");***

***sqlquery.bindValue(":name",QString(buffer).section("##",1,1));***

***sqlquery.exec();***

***if(!sqlquery.next())***

***{//可以新建***

***sqlquery.clear();***

***sqlquery.prepare("insert into people values (null,:name,:password,null,0)");***

***sqlquery.bindValue(":name",QString(buffer).section("##",1,1));***

***sqlquery.bindValue(":password",QString(buffer).section("##",2,2));***

***sqlquery.exec();***

***sqlquery.clear();***

***sqlquery.prepare("select \* from people where name = :name");***

***sqlquery.bindValue(":name",QString(buffer).section("##",1,1));***

***sqlquery.exec();//获得新建的用户的id***

***sqlquery.next();***

***int newid = sqlquery.value(0).toInt();***

***sqlquery.exec("create table if not exists friend\_\_" + QString::number(newid) +"(id INTEGER unique, name TEXT,sendmassage INTEGER,sendfile INTEGER)");***

***tcpSocket[0]->write(QString("register successed").toUtf8());***

***tcpSocket[0]->flush();***

***db.close();***

***}***

***else***

***{//有重名***

***tcpSocket[0]->write(QString("register error##same\_name").toUtf8());***

***tcpSocket[0]->flush();***

***db.close();***

***}***

***}***

这个函数的主要功能是处理用户注册请求，包括检查是否已存在相同用户名的用户，如果不存在则创建新用户，并为新用户创建一个好友表格。函数的最终结果是向客户端发送相应的注册成功或失败消息。

首先，它设置了一个数据库连接对象（db），并指定了数据库文件的路径为 "./people.db"。然后打开了数据库连接。创建了一个SQL查询对象 sqlquery，并使用 prepare 方法准备了一个SQL查询语句，用于检查数据库中是否已存在具有相同用户名的用户记录。使用 bindValue 方法将查询语句中的参数 ":name" 绑定为从输入数据中提取的用户名。这是通过将 buffer 字符串使用 "##" 分隔后提取的。执行 SQL 查询语句，并使用 sqlquery.next() 来检查是否找到了匹配的记录。如果没有找到记录，则表示可以创建新用户。

在新用户可以创建的情况下，首先清空 sqlquery，然后准备一个插入用户信息的SQL查询语句，将新用户的用户名和密码插入到数据库中，同时设置其他字段的默认值。执行插入操作，将新用户的信息插入到数据库中。再次清空 sqlquery，然后准备一个查询语句，以获取刚刚插入的新用户的ID。执行这个查询语句，并使用 sqlquery.next() 获取新用户的ID。使用新用户的ID创建一个名为 "friend\_\_<新用户ID>" 的表格，用于存储用户的好友信息。向客户端发送 "register successed" 消息，表示注册成功，刷新缓冲区。最后，关闭数据库连接。

如果找到了具有相同用户名的用户记录，表示已经存在具有相同用户名的用户，向客户端发送 "register error##same\_name" 消息，表示注册失败，刷新缓冲区。

1. wantsendmessage（）函数

***void server::wantsendmessage(){***

***//想发信息，校验有没有这个人***

***db.setDatabaseName("./people.db");***

***db.open();***

***QSqlQuery sqlquery;***

***sqlquery.prepare("select \* from people where name = :name");***

***sqlquery.bindValue(":name",QString(buffer).section("##",2,2));***

***sqlquery.exec();***

***if(sqlquery.next())***

***{//有这个人在，可以发消息***

***int otherid = sqlquery.value(0).toInt();***

***tcpSocket[0]->write(QString("wantsendmessage\_ok##%1").arg(otherid).toUtf8());***

***tcpSocket[0]->flush();***

***//发消息前把数据库准备好***

***int thisid = QString(buffer).section("##",1,1).toInt();***

***if(thisid < otherid)***

***{***

***sqlquery.exec("CREATE TABLE if not exists chat\_\_" + QString::number(thisid) + "\_\_" + QString::number(otherid) + "(time datetime NOT NULL UNIQUE,id INTEGER,ip TEXT,message TEXT, PRIMARY KEY(time))");***

***}***

***else***

***{***

***sqlquery.exec("CREATE TABLE if not exists chat\_\_" + QString::number(otherid) + "\_\_" + QString::number(thisid) + "(time datetime NOT NULL UNIQUE,id INTEGER,ip TEXT,message TEXT, PRIMARY KEY(time))");***

***}***

***db.close();***

***}***

***else***

***{//查无此人，无法对话***

***tcpSocket[0]->write(QString("wantsendmessage\_error").toUtf8());***

***tcpSocket[0]->flush();***

***db.close();***

***}***

这个函数的主要功能是处理用户发起消息请求，包括检查是否存在指定的目标用户，如果存在则准备好数据库表格以存储消息，并向客户端发送相应的消息以指示是否可以开始发送消息。

首先，它设置了一个数据库连接对象（db），并指定了数据库文件的路径为 "./people.db"。然后打开了数据库连接。

创建了一个SQL查询对象 sqlquery，并使用 prepare 方法准备了一个SQL查询语句，用于检查数据库中是否存在具有指定用户名的用户记录。

使用 bindValue 方法将查询语句中的参数 ":name" 绑定为从输入数据中提取的目标用户名。这是通过将 buffer 字符串使用 "##" 分隔后提取的。

执行 SQL 查询语句，并使用 sqlquery.next() 来检查是否找到了匹配的记录。如果找到了记录，表示有这个人存在，可以继续发消息。

如果存在该用户，获取该用户的ID（otherid），然后向客户端发送 "wantsendmessage\_ok##<用户ID>" 消息，表示可以开始发送消息，同时刷新缓冲区。

在准备发送消息之前，根据当前用户ID和目标用户ID（thisid 和 otherid）创建一个用于存储消息的数据库表格。如果 thisid 小于 otherid，则创建名为 "chat\_\_<thisid><otherid>" 的表格，否则创建名为 "chat<otherid>\_\_<thisid>" 的表格。

最后，关闭数据库连接。

如果未找到匹配的记录，表示数据库中不存在指定的用户，向客户端发送"wantsendmessage\_error" 消息，表示无法发起对话，同时刷新缓冲区。

1. chat\_history（）函数

***void server::chat\_history(){***

***//idone是发信息的，idtwo是收信息的，所以one已经阅览了two的所有消息***

***int idone = QString(buffer).section("##",1,1).toInt();***

***int idtwo = QString(buffer).section("##",2,2).toInt();***

***db.setDatabaseName("./people.db");***

***db.open();***

***QSqlQuery sqlquery;***

***QString sqlstring = "";***

***if(idone < idtwo)***

***{***

***sqlstring = "select \* from chat\_\_" + QString::number(idone) + "\_\_" + QString::number(idtwo) + " order by time desc limit 20";***

***}***

***else***

***{***

***sqlstring = "select \* from chat\_\_" + QString::number(idtwo) + "\_\_" + QString::number(idone) + " order by time desc limit 20";***

***}***

***qDebug()<<sqlstring;***

***sqlquery.exec(sqlstring);***

***if(sqlquery.next())***

***{***

***QString history = "##" + sqlquery.value(0).toDateTime().toString("yyyy-MM-dd hh:mm:ss.zzz")+ "##" + sqlquery.value(1).toString()+ "##" + sqlquery.value(2).toString()+"##"+sqlquery.value(3).toString();***

***int hisnum = 1;***

***while(sqlquery.next())***

***{***

***hisnum++;***

***QDateTime time = sqlquery.value(0).toDateTime();***

***QString timestr = time.toString("yyyy-MM-dd hh:mm:ss.zzz");***

***history = history + "##" + timestr;//时间***

***history = history + "##" + sqlquery.value(1).toString();//谁发的***

***history = history + "##" + sqlquery.value(2).toString();//发送者ip***

***history = history + "##" + sqlquery.value(3).toString();//内容***

***}***

***history = "chat\_history\_ok##" + QString::number(hisnum) + history;***

***tcpSocket[0]->write(history.toUtf8());***

***tcpSocket[0]->flush();***

***QString sqlstring = "update friend\_\_" + QString::number(idone) + " set sendmassage = 0 where id = :id";***

***db.setDatabaseName("./people.db");***

***db.open();***

***QSqlQuery sqlquery;***

***sqlquery.prepare(sqlstring);***

***sqlquery.bindValue(":id", idtwo);***

***sqlquery.exec();***

***db.close();***

***}***

***else***

***{//无历史记录***

***tcpSocket[0]->write(QString("chat\_history\_error").toUtf8());***

***tcpSocket[0]->flush();***

***db.close();***

***}***

***}***

这个函数的主要功能是获取两个用户之间的聊天历史记录，并向客户端返回相应的历史记录或错误消息。以下是该函数的详细步骤：

首先，从输入数据中提取了两个用户的ID，分别存储在 idone 和 idtwo 中，这些ID表示聊天的两个参与者。

然后，设置了一个数据库连接对象（db），并指定了数据库文件的路径为 "./people.db"。然后打开了数据库连接。

创建了一个SQL查询对象 sqlquery。

根据 idone 和 idtwo 的大小关系，构建了一个SQL查询字符串 sqlstring，以选择正确的聊天历史记录表格，并按时间降序排列，最多选择20条记录。

执行 SQL 查询语句，用于获取聊天历史记录。

如果查询结果中有历史记录，就开始构建历史记录的字符串 history。

遍历查询结果，提取每一条历史记录的时间、发送者、发送者IP和内容，并将它们添加到 history 字符串中。

构建包含历史记录数量和历史记录内容的消息字符串，并加上前缀 "chat\_history\_ok"，然后向客户端发送该消息，刷新缓冲区。

更新数据库中的 "friend\_\_<idone>" 表格，将与 idtwo 用户的聊天消息状态设置为未发送（sendmassage = 0）。

最后，如果查询结果中没有历史记录，向客户端发送 "chat\_history\_error" 消息，表示没有找到历史记录。

1. chat\_send（）函数

***void server::chat\_send(){***

***//one发消息的，two收消息的***

***QDateTime nowstr = QDateTime::fromString(QString(buffer).section("##",1,1), "yyyy-MM-dd hh:mm:ss.zzz");***

***int idone = QString(buffer).section("##",2,2).toInt();***

***int idtwo = QString(buffer).section("##",3,3).toInt();***

***QString ip =QString(buffer).section("##",4,4);***

***db.setDatabaseName("./people.db");***

***db.open();***

***QSqlQuery sqlquery;***

***QString sqlstring = "";***

***if(idone < idtwo)***

***{***

***sqlstring = "insert into chat\_\_" + QString::number(idone) + "\_\_" + QString::number(idtwo) + " values(:time,:id,:ip,:message)";***

***}***

***else***

***{***

***sqlstring = "insert into chat\_\_" + QString::number(idtwo) + "\_\_" + QString::number(idone) + " values(:time,:id,:ip,:message)";***

***}***

***qDebug()<<sqlstring;***

***sqlquery.prepare(sqlstring);***

***sqlquery.bindValue(":time",nowstr);***

***sqlquery.bindValue(":id",idone);***

***sqlquery.bindValue(":ip",ip);***

***sqlquery.bindValue(":message",QString(buffer).section("##",5,5));***

***sqlquery.exec();***

***sqlstring = "update friend\_\_" + QString::number(idtwo) + " set sendmassage = 1 where id = :id";***

***db.setDatabaseName("./people.db");***

***db.open();***

***sqlquery.clear();***

***sqlquery.prepare(sqlstring);***

***sqlquery.bindValue(":id", idone);***

***sqlquery.exec();***

***db.close();***

***}***

这个函数的主要功能是将用户发送的消息插入到相应的聊天历史记录中，并更新接收消息的用户的消息状态，以表示有新消息到达。

首先，从输入数据中提取了消息的时间、发送者ID（idone）、接收者ID（idtwo）、发送者IP和消息内容。

然后，设置了一个数据库连接对象（db），并指定了数据库文件的路径为 "./people.db"。然后打开了数据库连接。

创建了一个SQL查询对象 sqlquery。

根据 idone 和 idtwo 的大小关系，构建了一个SQL插入语句 sqlstring，以将消息插入到正确的聊天历史记录表格中。

使用 prepare 方法准备了插入语句，并绑定了消息的时间、发送者ID、发送者IP和消息内容。

执行 SQL 插入语句，将消息添加到聊天历史记录中。

构建了一个SQL更新语句 sqlstring，用于将与接收消息的用户关联的 "friend\_\_<idtwo>" 表格中的 sendmassage 字段设置为1，表示有新消息。

使用 prepare 方法准备了更新语句，并绑定了接收消息的用户的ID。

执行 SQL 更新语句，更新 "friend\_\_<idtwo>" 表格中的 sendmassage 字段。

最后，关闭数据库连接。

1. getfriendlist（）函数

***void server::getfriendlist(){***

***db.setDatabaseName("./people.db");***

***db.open();***

***QSqlQuery sqlquery;***

***sqlquery.exec("select \* from friend\_\_" + QString(buffer).section("##",1,1) + " desc");***

***if(sqlquery.next())***

***{***

***QList <QString> friendlist;***

***QList <QString> friendsendfilelist;***

***QList <QString> friendsendmassagelist;***

***friendlist.append(sqlquery.value(1).toString());***

***int sendmassagenum = 0;***

***if(sqlquery.value(2).toString() == '1')***

***{***

***sendmassagenum++;***

***}***

***friendsendmassagelist.append(sqlquery.value(2).toString());***

***int sendfilenum = 0;***

***if(sqlquery.value(3).toString() == '1')***

***{***

***sendfilenum++;***

***}***

***friendsendfilelist.append(sqlquery.value(3).toString());***

***QString friends = "";***

***while(sqlquery.next())***

***{***

***friendlist.append(sqlquery.value(1).toString());***

***if(sqlquery.value(2).toString() == '1')***

***{***

***sendmassagenum++;***

***}***

***friendsendmassagelist.append(sqlquery.value(2).toString());***

***if(sqlquery.value(3).toString() == '1')***

***{***

***sendfilenum++;***

***}***

***friendsendfilelist.append(sqlquery.value(3).toString());***

***}***

***int onlinefriendnum = 0;***

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

***{***

***sqlquery.prepare("select \* from people where name = :name");***

***sqlquery.bindValue(":name",friendlist.at(i));***

***sqlquery.exec();***

***sqlquery.next();***

***QString peopleip = sqlquery.value(3).toString();***

***if(sqlquery.value(4).toInt() == 1)***

***{***

***onlinefriendnum++;***

***friends = "##" + friendlist.at(i) + "##1##" + peopleip +"##"+friendsendmassagelist.at(i)+"##"+ friendsendfilelist.at(i)+ friends;***

***}***

***else***

***{***

***friends = "##" + friendlist.at(i) + "##0##" + peopleip +"##"+friendsendmassagelist.at(i)+"##"+ friendsendfilelist.at(i)+ friends;***

***}***

***}***

***friends = "getfriendlist\_ok##" + QString::number(friendlist.length()) +"##"+QString::number(onlinefriendnum)+"##"+QString::number(sendmassagenum)+"##"+QString::number(sendfilenum)+ friends;***

***qDebug()<<friends;***

***tcpSocket[0]->write(friends.toUtf8());***

***tcpSocket[0]->flush();***

***db.close();***

***}***

***else***

***{//无朋友***

***tcpSocket[0]->write(QString("getfriendlist\_error").toUtf8());***

***tcpSocket[0]->flush();***

***db.close();***

***}***

***}***

这个函数的主要功能是从数据库中获取用户的好友列表信息，包括好友的在线状态、IP地址以及是否可以接收消息和文件，然后将这些信息构建成消息字符串并发送给客户端。

以下是该函数的详细步骤：

首先，它设置了一个数据库连接对象（db），并指定了数据库文件的路径为 "./people.db"。然后打开了数据库连接。

创建了一个SQL查询对象 sqlquery，并执行一个SQL查询语句，用于从与用户关联的 "friend\_\_<用户ID>" 表格中获取好友列表。

如果查询结果中有数据，开始构建好友列表。

创建了三个列表：friendlist 用于存储好友的用户名，friendsendmassagelist 用于存储好友是否可以接收消息的状态，friendsendfilelist 用于存储好友是否可以接收文件的状态。然后将第一位好友的信息添加到这些列表中。

统计第一个好友的消息发送和文件发送状态，如果为 '1' 表示可以发送，分别增加 sendmassagenum 和 sendfilenum。

使用 while 循环遍历查询结果，将每位好友的信息添加到相应的列表中，并统计发送消息和文件状态。

创建一个变量 onlinefriendnum 用于统计在线好友数量。

使用for 循环遍历 friendlist，并查询数据库中每个好友的在线状态和IP地址。

如果好友在线（在线状态为1），增加 onlinefriendnum 计数，然后构建好友信息字符串 friends，包括好友用户名、在线状态、IP地址、是否可以接收消息和是否可以接收文件，并将这些信息逐渐添加到 friends 字符串中。

最后，将好友列表信息构建成消息字符串，以 "##" 作为分隔符，并向客户端发送该消息字符串，刷新缓冲区。

1. add\_friend（）函数

***void server::add\_friend(){***

***int whowantadd\_id = QString(buffer).section("##",1,1).toInt();***

***QString friend\_name = QString(buffer).section("##",2,2);***

***qDebug() << whowantadd\_id << friend\_name;***

***db.setDatabaseName("./people.db");***

***db.open();***

***QSqlQuery sqlquery;***

***sqlquery.prepare("select \* from people where name = :name");***

***sqlquery.bindValue(":name",friend\_name);***

***sqlquery.exec();***

***if(sqlquery.next())***

***{***

***int friend\_id = sqlquery.value(0).toInt();***

***qDebug() <<friend\_id;***

***sqlquery.clear();***

***QString sqlstring = "insert into friend\_\_" + QString::number(whowantadd\_id) + " values(:id,:name,0,0)";***

***qDebug()<<sqlstring;***

***sqlquery.prepare(sqlstring);***

***sqlquery.bindValue(":id",friend\_id);***

***sqlquery.bindValue(":name",friend\_name);***

***sqlquery.exec();***

***qDebug()<<sqlquery.lastError();***

***tcpSocket[0]->write(QString("add\_friend\_ok").toUtf8());***

***tcpSocket[0]->flush();***

***db.close();***

***}***

***else***

***{//没这人***

***qDebug() <<"e";***

***tcpSocket[0]->write(QString("add\_friend\_error").toUtf8());***

***tcpSocket[0]->flush();***

***db.close();***

***}***

***}***

这个函数的主要功能是处理用户添加好友的请求，包括检查好友是否存在、将好友添加到用户的好友列表，并向客户端发送相应的成功或失败消息。以下是该函数的详细步骤：

首先，从输入数据中提取了两个重要的信息：发起添加好友请求的用户的ID（whowantadd\_id）和要添加的好友的用户名（friend\_name）。

然后，设置了一个数据库连接对象（db），并指定了数据库文件的路径为 "./people.db"。然后打开了数据库连接。

创建了一个SQL查询对象 sqlquery，并执行一个SQL查询语句，用于检查数据库中是否存在具有指定用户名的用户记录。

如果查询结果中存在匹配的用户记录，表示要添加的好友存在。

从查询结果中获取好友的用户ID（friend\_id）。

清空 sqlquery，然后构建一个SQL插入语句 sqlstring，以将好友添加到与用户关联的 "friend\_\_<用户ID>" 表格中，同时设置好友的消息发送和文件发送状态为0（表示不能发送）。

使用 prepare 方法准备了插入语句，并绑定了好友的ID和用户名。

执行 SQL 插入语句，将好友添加到好友列表中。

如果添加成功，向客户端发送 "add\_friend\_ok" 消息，表示好友添加成功，刷新缓冲区。

最后，关闭数据库连接。

如果在步骤4中未找到匹配的用户记录，表示要添加的好友不存在，向客户端发送 "add\_friend\_error" 消息，表示添加好友失败，刷新缓冲区。

1. delete\_friend（）函数

***void server::delete\_friend(){***

***int whowantdelete\_id = QString(buffer).section("##",1,1).toInt();***

***QString friend\_name = QString(buffer).section("##",2,2);***

***db.setDatabaseName("./people.db");***

***db.open();***

***QSqlQuery sqlquery;***

***sqlquery.prepare("delete from friend\_\_"+QString::number(whowantdelete\_id)+" where name = :name");***

***sqlquery.bindValue(":name",friend\_name);***

***if(sqlquery.exec())***

***{***

***tcpSocket[0]->write(QString("delete\_friend\_ok").toUtf8());***

***tcpSocket[0]->flush();***

***db.close();***

***}***

***else***

***{***

***tcpSocket[0]->write(QString("delete\_friend\_error").toUtf8());***

***tcpSocket[0]->flush();***

***db.close();***

***}***

***}***

这个函数的主要功能是处理用户删除好友的请求，包括检查好友是否存在、将好友移除到用户的好友列表，并向客户端发送相应的成功或失败消息。

1. want\_send\_file（）函数

***void server::want\_send\_file(){***

***int userid = QString(buffer).section("##",1,1).toInt();***

***qDebug()<<buffer;***

***db.setDatabaseName("./people.db");***

***db.open();***

***QSqlQuery sqlquery;***

***sqlquery.prepare("select \* from people where name = :name");***

***sqlquery.bindValue(":name",QString(buffer).section("##",2,2));***

***sqlquery.exec();***

***sqlquery.next();***

***int otheruserid = sqlquery.value(0).toInt();***

***QString sqlstring = "update friend\_\_" + QString::number(otheruserid) + " set sendfile = 1 where id = :id";***

***sqlquery.prepare(sqlstring);***

***sqlquery.bindValue(":id",userid);***

***sqlquery.exec();***

***}***

这个函数的主要功能是处理用户发起发送文件请求，包括检查是否存在指定的目标用户，如果存在则准备好数据库表格以存储消息，并向客户端发送相应的消息以指示是否可以开始发送文件。

10.send\_file\_ok（）函数

***void server::send\_file\_ok(){***

***//1给2发,1是id,2是name;在2的表中显示1的信息***

***int userid = QString(buffer).section("##",1,1).toInt();***

***qDebug()<<buffer;***

***QString sqlstring = "update friend\_\_" + QString::number(userid) + " set sendfile = 0 where name = :name";***

***qDebug()<<sqlstring;***

***db.setDatabaseName("./people.db");***

***db.open();***

***QSqlQuery sqlquery;***

***sqlquery.prepare(sqlstring);***

***sqlquery.bindValue(":name", QString(buffer).section("##",2,2));***

***sqlquery.exec();***

***}***

这个函数的主要功能是成功发送文件后，取消掉friend表中准备发送文件状态（friend表的sendfile属性取1是准备发送文件状态，本函数将其取值改为0）。

11.send\_file\_miss（）函数

***void server::send\_file\_miss(){***

***int userid = QString(buffer).section("##",1,1).toInt();***

***qDebug()<<buffer;***

***db.setDatabaseName("./people.db");***

***db.open();***

***QSqlQuery sqlquery;***

***sqlquery.prepare("select \* from people where name = :name");***

***sqlquery.bindValue(":name",QString(buffer).section("##",2,2));***

***sqlquery.exec();***

***sqlquery.next();***

***int otheruserid = sqlquery.value(0).toInt();***

***QString sqlstring = "update friend\_\_" + QString::number(otheruserid) + " set sendfile = 0 where id = :id";***

***sqlquery.prepare(sqlstring);***

***sqlquery.bindValue(":id",userid);***

***sqlquery.exec();***

***}***

这个函数的主要功能是成功发送文件后，取消掉friend表中准备发送文件状态（friend表的sendfile属性取1是准备发送文件状态，本函数将其取值改为0）。

12.logout()函数

***void server::logout(){***

***db.setDatabaseName("./people.db");***

***db.open();***

***QSqlQuery sqlquery;***

***sqlquery.prepare("update people set islogin=0 where id = :id");***

***sqlquery.bindValue(":id",QString(buffer).section("##",1,1));***

***sqlquery.exec();***

***//更新服务器界面***

***ui->listWidget->clear();***

***sqlquery.prepare("select \* from people where islogin = 1");***

***sqlquery.exec();***

***if(sqlquery.next())***

***{***

***QString userid = sqlquery.value(0).toString();***

***QString username = sqlquery.value(1).toString();***

***QString userip = sqlquery.value(3).toString();***

***//qDebug()<<userid;***

***ui->listWidget->insertItem(0,"用户ID："+userid+",用户昵称:"+username+",用户IP:"+userip);***

***int rownum = 1;***

***while (sqlquery.next())***

***{***

***QString userid = sqlquery.value(0).toString();***

***QString username = sqlquery.value(1).toString();***

***QString userip = sqlquery.value(3).toString();***

***ui->listWidget->insertItem(rownum,"用户ID："+userid+",用户昵称:"+username+",用户IP:"+userip);***

***rownum++;***

***}***

***}***

***else***

***{***

***ui->listWidget->clear();***

***ui->listWidget->insertItem(0,tr("当前无在线用户"));***

***}***

***}***

这个函数的主要功能是退出登录。一方面修改数据库poeple表中islogin属性（islogin属性为1表示已登录，这里改为0），另一方面更新服务端在线用户名单。

1. log(const QString &message)函数

***void server::log(const QString &message){***

***QDateTime currentTime = QDateTime::currentDateTime();***

***QString s = QString(buffer).section("##",1,1);***

***s=s+" : "+message;***

***QString logMessage = QString("(%1) %2").arg(currentTime.toString("MM-dd hh:mm:ss")).arg(s);***

***QString currentText = ui->textEditOut->toPlainText();***

***currentText += "\n" + logMessage;***

***ui->textEditOut->setText(currentText);***

***}***

这个函数的主要功能是生成一个日志文件，记录用户的操作，记录内容包括：操作时间，用户ip以及用户操作。

**5 系统测试**

5.1注册测试

首先在登录界面，点击注册按钮即可进入注册界面



若两次密码输入不一致或用户名、密码为空则弹出提示框



如下图为正确的注册案例

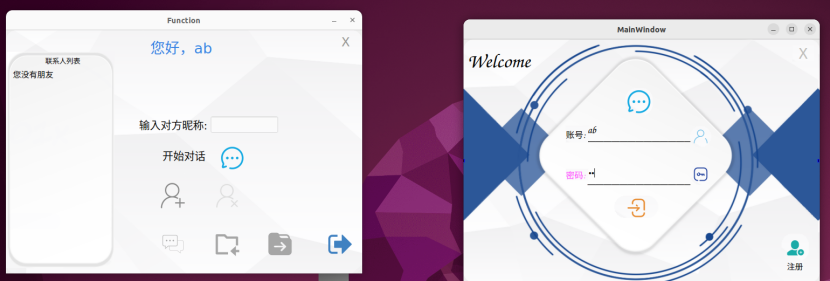


5.2登录/下线测试

登录成功后能够成功进入客户端界面

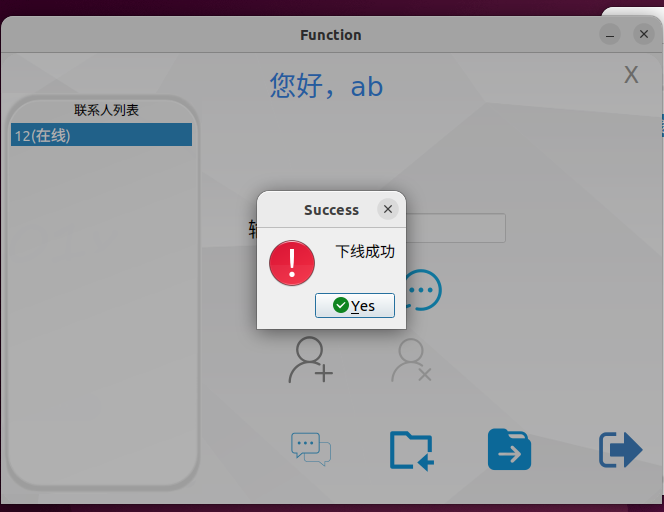


若重复登录也会弹出提示窗口





再进行下线即可



5.3添加联系人测试

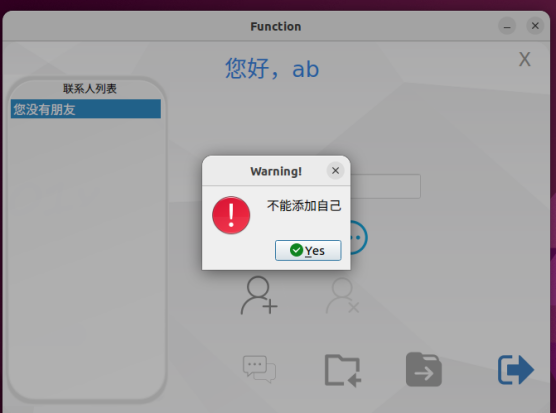


在客户端中输入对方昵称



若输入自身名称或错误名称都会添加失败，弹出提示窗口





在输入正确昵称后可以成功添加对方到联系人列表





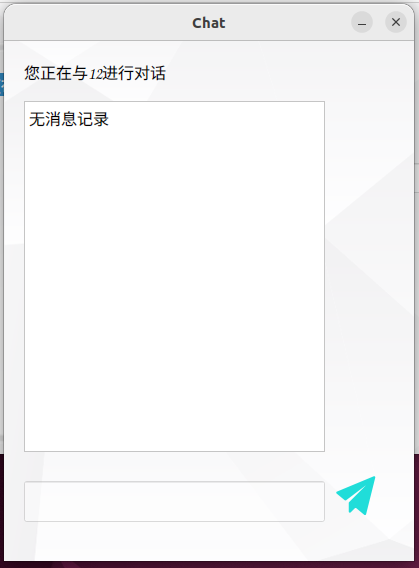
5.4聊天测试

在联系人列表中选中联系人可以开始对话

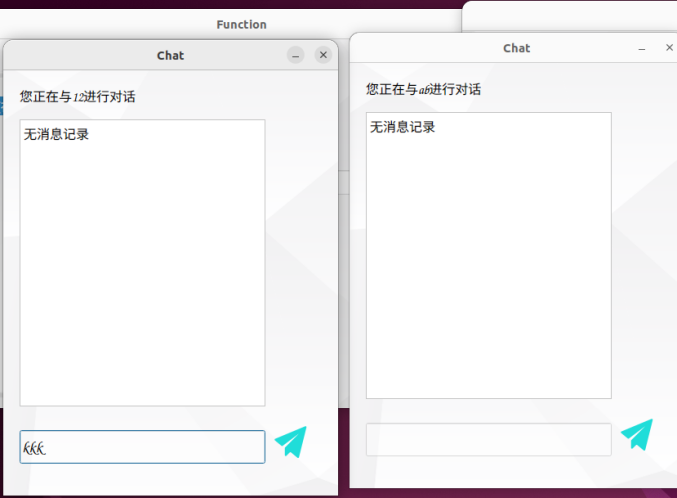


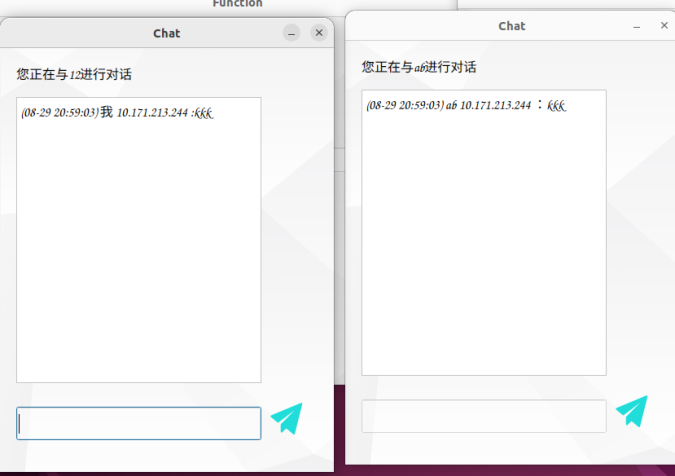
或是直接打开对话框





发送信息后对方也会有信息提示





发送消息的时候会出现明显卡顿的bug：

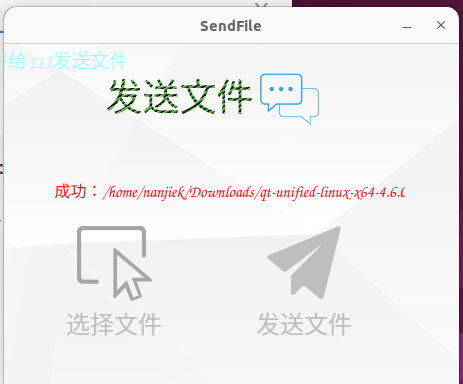
单线程在500ms内不断获取历史记录从而实现实时更新解决卡顿问题

发送信息的时候输出格式错误的bug：

对ui界面对应的代码进行语法更正修改

5.5文件传输测试

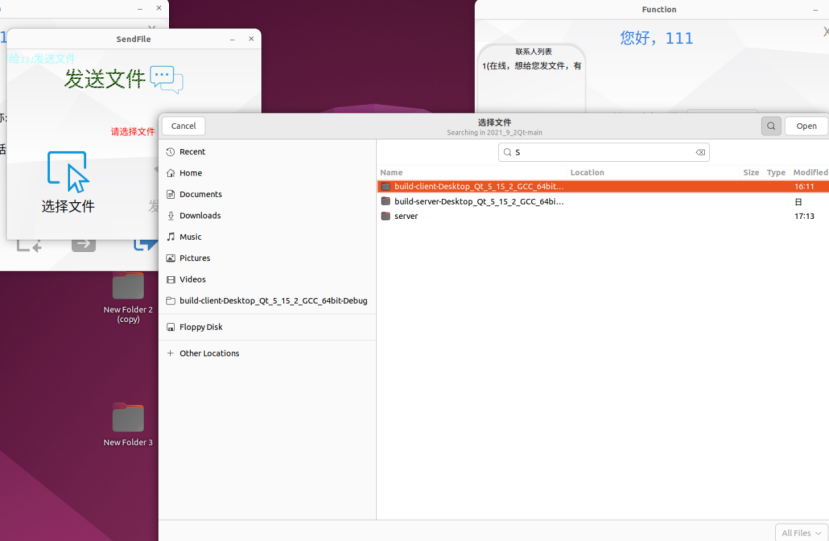
在客户端界面点击发送文件按钮



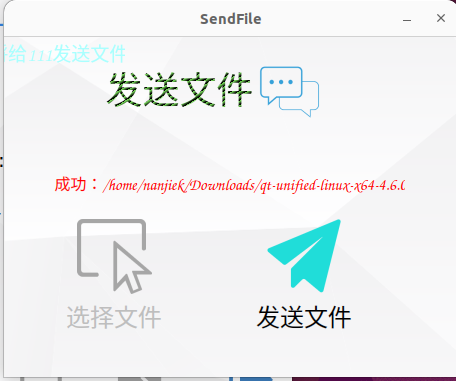
再点击选择文件



开始选择文件



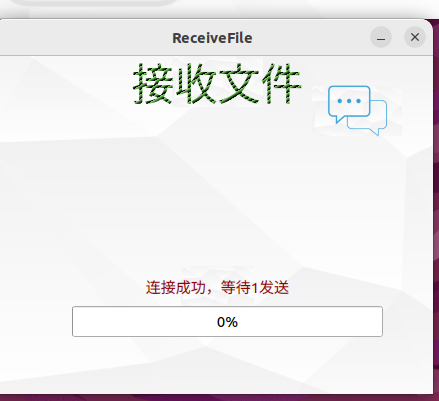
选择成功后再点击发送文件



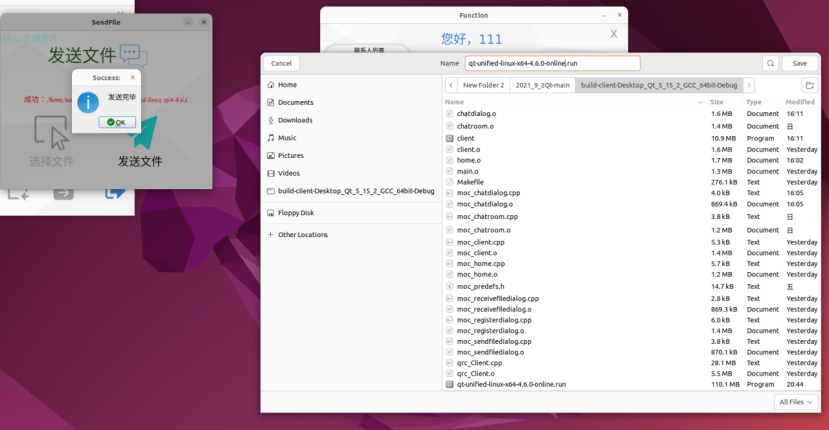
对方联系人列表会进行提示



点击接收文件按钮



待文件传输完成后都会有提示



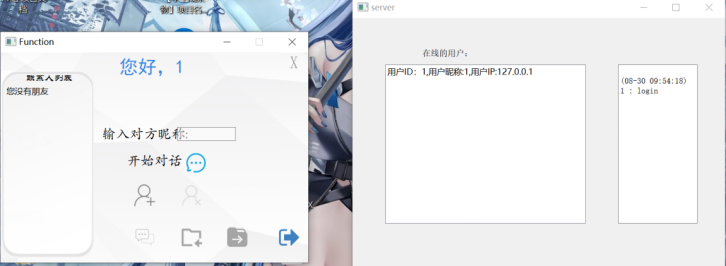
传输文件时会出现黏包问题，文件传输不完整、错误：

对收发数据双方事先定义一个数据包头，发送数据格式为数据包头+数据，用Qt封装的readyRead作为接收网络数据接口，可以关联一个函数响应槽函数后对数据进行拆包，利用QTcpSocket::read()以固定字节读取数据，在重复验证数据完整性后再接收完整数据

5.6后台监听测试

在服务端可以获取用户登录记录





|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **项目名称** | Message即时通讯系统 | | **程序版本** |  | | |
| **功能模块名** | 即时通讯系统 | | | | | |
| **编制人** | 杨云博 | | **编制时间** | 2024/6/28 | | |
| **功能特性** | 用户注册和登录、好友管理、信息传输与接收 | | | | | |
| **测试目的** | 明确测试每个功能的目的和期望结果 | | | | | |
| **预置条件** | 用户注册如空白账户数据库、好友管理如已存在的用户账户、消息发送与接收如已存在的好友关系 | | | | | |
| **用例编号** | **用例说明** | **输入数据** | **预期结果** | **测试结果** | **缺陷编号** | **备注** |
| 1 | 用户注册输入密码不一致 | 用户名：user  密码1：123 密码2：456 | 报错，两次密码不一样 | 报错，两次密码不一样 | N/A |  |
| 2 | 重复注册 | 用户名：user  密码：123 | 报错，昵称有重复 | 报错，昵称有重复 | N/A |  |
| 3 | 重复登录 | 用户名：user  密码：123 | 报错，用户已登录 | 报错，用户已登录 | N/A |  |
| 4 | 账号不存在 | 用户名：test 密码：123 | 报错，用户不存在 | 报错，用户不存在 | N/A |  |
| 5 | 密码错误 | 用户名：user 密码：456 | 报错，密码错误 | 报错，密码错误 | N/A |  |
| 6 | 向user2文本传输 | hello | 即时接收 | 即时接收 | N/A |  |
| 7 | 向user2传输文件 | 自选文件 | 即时接收 | 即时接收 | N/A |  |
| 8 | 后台对于用户登录的信息记录 | 用户名：user  密码：123 | （time） 1 : login | （time） 1 : login | N/A |  |

**6 项目经验及心得体会**

开发即时通信软件项目Message并负责数据库部分是一项具有挑战性的任务。以下是我作为数据库部分负责人的一些心得体会：

数据库设计的关键性：数据库是整个即时通信系统的基础，良好的数据库设计是至关重要的。在开始编码之前，花时间仔细设计数据库架构，包括表格结构、数据关系和索引。

Qt的数据库模块：Qt提供了出色的数据库支持，特别是Qt SQL模块。使用Qt的数据库类来连接和管理SQLite数据库非常方便，它提供了高级的抽象，减少了重复的代码。

数据模型和视图：使用Qt的数据模型和视图来处理数据库数据。QSqlTableModel等模型可以简化数据的展示和更新，使得与数据库的交互更加容易。

SQL查询和性能优化：编写高效的SQL查询是提高应用性能的关键。了解如何使用索引、优化查询以及合理使用事务非常重要。Qt的数据库工具可以帮助分析和优化查询。

错误处理和鲁棒性：实现强大的错误处理机制，能够处理数据库操作中可能出现的错误。在记录和处理错误时，提供有用的反馈信息有助于快速排除问题。

测试和调试：为数据库功能编写全面的单元测试，确保其正确性。Qt的测试框架可帮助自动化测试。调试工具也是必不可少的，用于识别和解决潜在问题。

版本控制：使用版本控制系统来跟踪数据库模式的更改，以便团队成员可以协同工作，并能够回滚到以前的版本。

文档编写：编写清晰、详细的文档，描述数据库架构、表格结构和数据模型。这有助于其他团队成员了解和使用数据库。

与团队协作：与前端和后端开发人员密切合作，确保数据库与应用程序的其他部分协同工作。沟通和协作是项目成功的关键。

持续学习和改进：SQLite和Qt的数据库模块不断发展，持续学习和关注新特性和最佳实践是保持项目健康的关键。