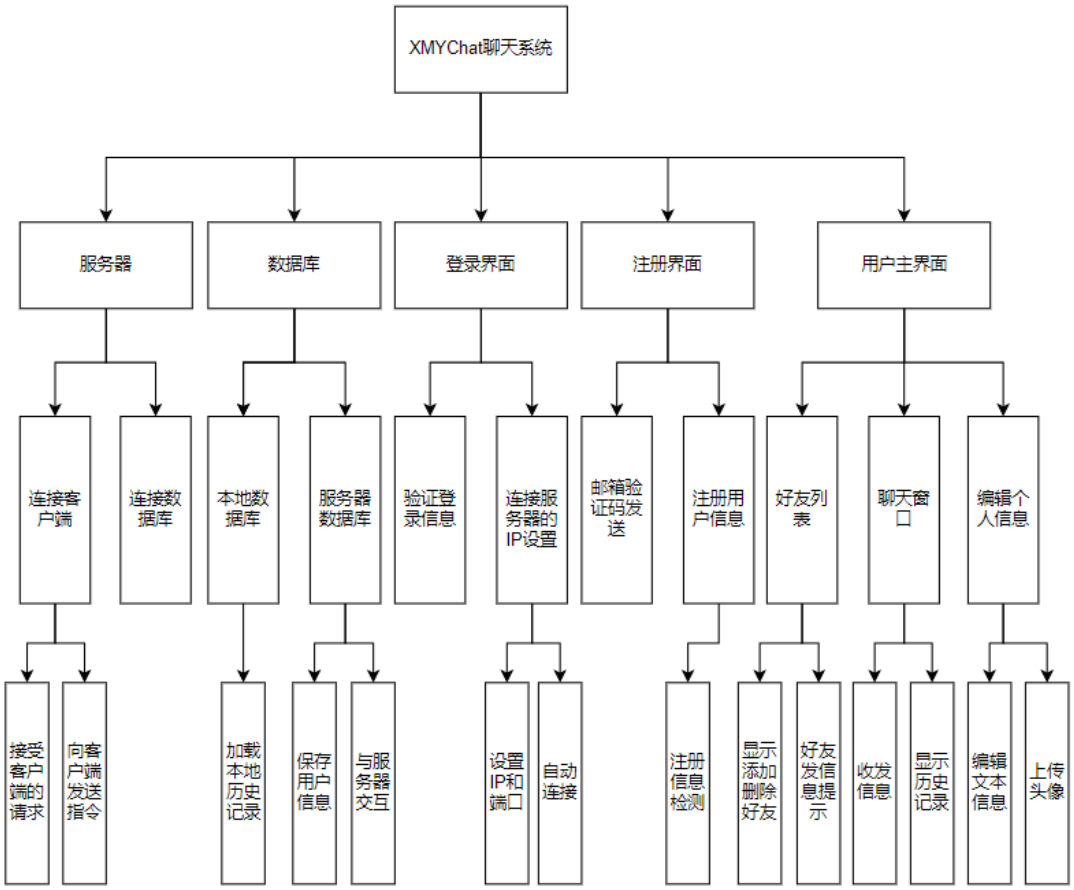
XMYChat项目设计文档

项目成员：王成龙

范南岳

林均驰

一、 设计结构图：



二、 模块概述:

1、模块功能定义

序号	模块	功能点	功能点详细内容
1	连接服务器	将客户端连接到服务器	根据输入的 IP 地址和端口号尝试连接服务器，并显示是否连接成功
2	注册模块	将用户输入的用户名和密码加密后提交到服务	将用户填写的用户名和密码经过哈希加密后提交到服务器

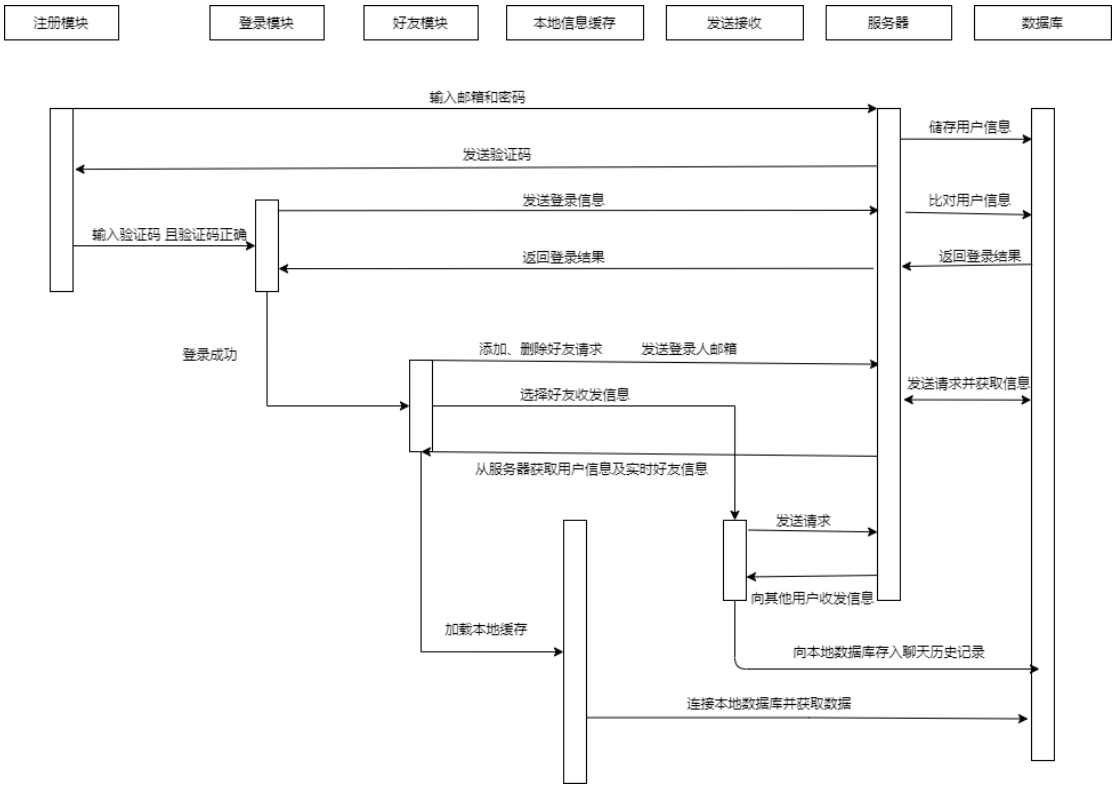
		器	
		检测用户注册邮箱是否合法	检测用户注册邮箱是否合法
		检测用户用户名	检测是否已被注册
		向注册邮箱发送验证码	通过 SMTP 协议实现验证码邮件的自动发送
		数据存入数据库	如果注册成功则将数据存储至数据库
		返回注册结果	显示注册成功或注册失败情况
3	登录模块	将用户输入的用户名和密码提交到服务器	将用户输入的用户名和密码提交到服务器
		传送数据到服务端	传送数据到服务端
		检验用户名和密码情况	检测用户名是否存在，用户名和密码是否正确，用户是否被禁用
		返回登录结果	根据检验情况返回用户登录结果
		更新信息	更新好友数，从服务端传回好友总数、 每一个好友名字、好友头像
4	添加、删除好友	将用户输入的用户名提交到服务器	将用户输入的用户名提交到服务器
		服务器进行检索	判断该用户名是否不存在或重复添加
		返回添加好友结果	根据服务器检索结果返回添加好友结果
		服务器信息更新	更新好友列表
		删除好友	删除好友
5	收发模块	发送信息	直接发送信息
		接收信息	由服务器接收发送给自己的信息
		发送文件	发送文件
		接收文件	接收文件
6	数据库	服务器数据库	与服务器交互、辅助注册、登录等功能
		个人本地数据库	储存历史记录，缓冲图片

2、模块结构

模块名称	模块类型	概要说明
登陆注册界面	内部模块	将账户邮箱和密码发送到服务端
邮箱注册界面	外部模块	向注册邮箱发送验证码
设置 ip 和端口界面	内部模块	客户端选择服务端的 ip 和端口号
添加好友模块	内部模块	
删除好友模块	内部模块	
发送信息模块	内部模块	
修改个人信息模块	内部模块	
上传头像模块	内部模块	
登出模块	内部模块	

Hash 处理模块	外部模块	
Socket 模块	外部模块	数据传输函数设计
数据库模块	内部模块	数据库底层函数设计
Jsonobject 模块	外部模块	实现消息打包发送

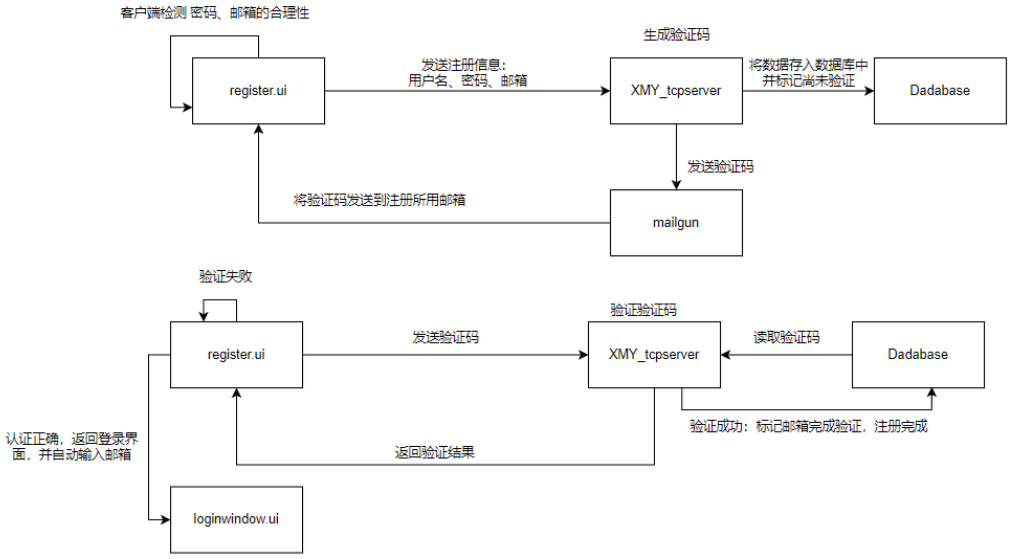
3、模块动作时序：



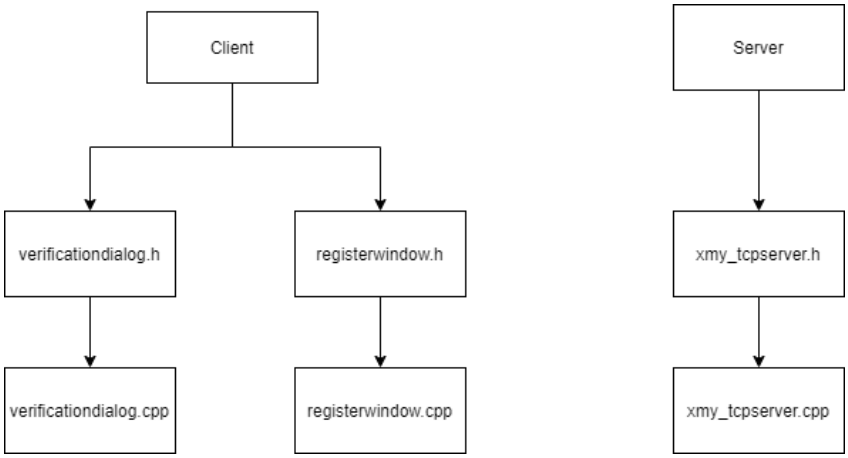
三、 各个模块设计结构及程序流程图：

1、 用户注册模块：

主要分为两步，第一步是客户端向服务器发出注册申请，并且将用户名、密码和邮箱，发送到服务器中生成验证码；第二步客户端将验证码发送给服务器验证身份后返回验证结果完成注册，用户注册的流程图如下：

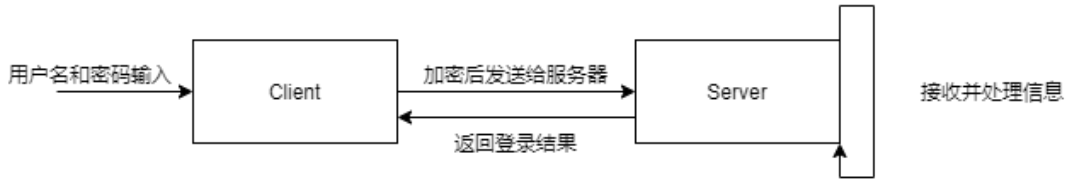


其中主要实现的文件如下图所示：



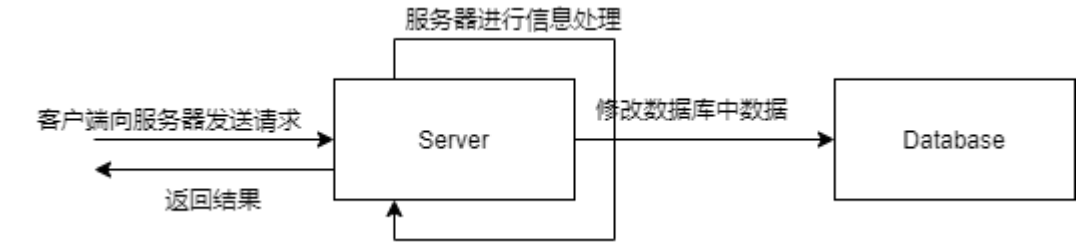
2、 登录模块：

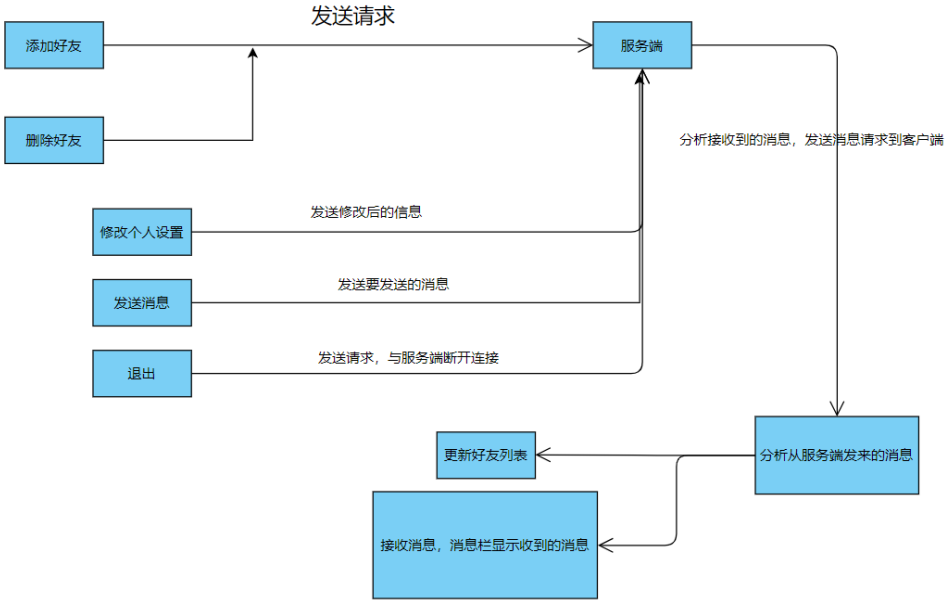
登录模块客户端将用户在前端输入的账号和密码进行密码加密，然后将其发送到服务器，服务器经过验证，向客户端发送验证结果，完成登录，基本流程图如下：



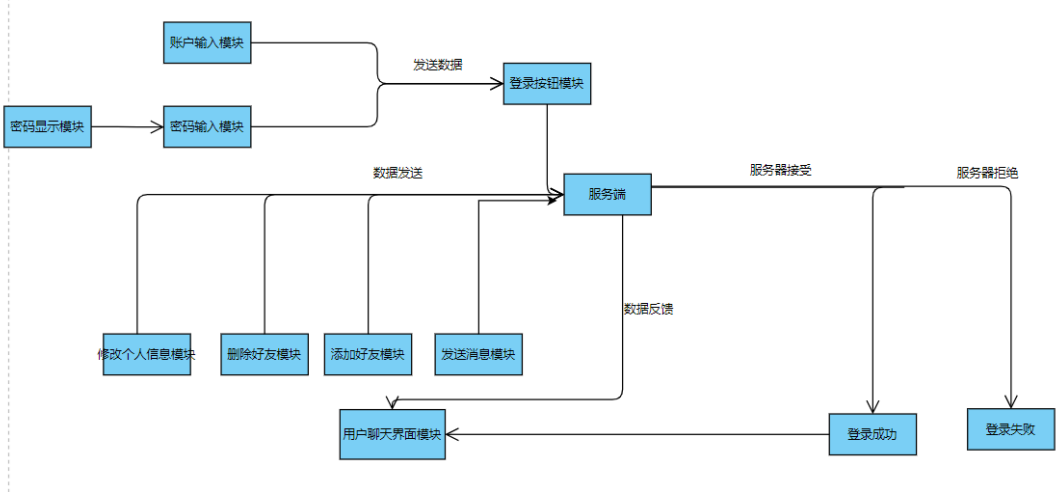
这部分的实现主要在文件 loginwindow.cpp 和 loginwindow.h 中实现。

3、 客户端与服务器连接以及服务器功能模块：





4、 综合流程图：



功能核心代码展示：

```
void UserMainWindow::on_listWidget_friends_customContextMenuRequested(const QPoint &pos)
{
    QListWidgetItem *curItem = ui->listWidget_friends->itemAt(pos);
        if (curItem == NULL)
            return;

        QMenu *popMenu = new QMenu(this);
        QAction *deleteSeed = new QAction(tr("Delete"), this);
        popMenu->addAction(deleteSeed);
        connect(deleteSeed, SIGNAL(triggered()), this, SLOT(deleteSeedSlot()));
        popMenu->exec(QCursor::pos());
        delete popMenu;
        delete deleteSeed;
}
void UserMainWindow::deleteSeedSlot()
{
    QListWidgetItem *item = ui->listWidget_friends->currentItem();
    if (item == NULL)
        return;
    QString email = item->text(); //好友邮箱
    email = email.split('\n')[1];
    int ch = QMessageBox::warning(NULL, "Warning",
                                  tr("Are you sure to delete %1?").arg(email),
```

```
                                                QMessageBox::Yes | QMessageBox::No,
                                                QMessageBox::No);

    if (ch != QMessageBox::Yes)
        return;
    if(ui->label_email_friend->text()==email) {
        clear_friend_info();
    }
    session->delete_user(email);
    int curIndex = ui->listWidget_friends->row(item);
    ui->listWidget_friends->takeI tem(curIndex);
    delete item;
}
```

该部分实现右键删除好友

```
void loginsession::user_info_modify(QJsonObject data)
{
    socket->send_json(data);
}

void loginsession::get_user_info(QString email)
{
    if(email.isEmpty()) email=info["email"];
    QJsonObject data;
    data.insert("type",TYPE_GET_USER_INFO);
    data.insert("email",email);
    socket->send_json(data);
}
```

```
void loginsession::fetch_friend_list()
{
    QJsonObject data;
    data.insert("type",TYPE_FETCH_FRIEND_LIST);
    data.insert("email",info["email"]);
    qDebug()<<data;
    socket->send_json(data);
}

void loginsession::get_avatar(QString email, QString md5)
{
    if(email.isEmpty()) email=info["email"];
    QString filename=".cache\\"+XMY_Utilities::emailtomd5(email)+".png";
    QPixmap pic(filename);
    QString picmd5=XMY_Utilities::pixmaptomd5(pic);
    qDebug()<<email<<"cache md5: "<<picmd5<<" remote md5: "<<md5;
    if(picmd5==md5) return;
    QJsonObject data;
    data.insert("type",TYPE_GET_AVATAR);
    data.insert("email",email);
    data.insert("md5",picmd5);
    socket->send_json(data);
}

void loginsession::search_user(QString email)
{
    QJsonObject data;
    data.insert("type",TYPE_SEARCH_USER);
```

```
    data.insert("email",email);
    socket->send_json(data);
}

void loginsession::add_user(QString email)
{
    QJsonObject data;
    data.insert("type",TYPE_ADD_FRIEND);
    data.insert("email1",email);
    data.insert("email2",info.value("email"));
    socket->send_json(data);
}

void loginsession::delete_user(QString email)
{
    QJsonObject data;
    data.insert("type",TYPE_DELETE_FRIEND);
    data.insert("email1",email);
    data.insert("email2",info.value("email"));
    socket->send_json(data);
}

QList<chatMessage> loginsession::get_messages_by_email(QString email)
{
    QList<chatMessage> msg;
    db->get_messages_by_email(email,msg);
    return msg;
}
```

```
void loginsession::callback_process(QJsonObject data)
{
//      qDebug()<<data;
        switch (data.value("type").toInt()) {
        case TYPE_LOGIN: {
                if(data.value("result").toInt()==LOGIN_SUCCESS) {
                        info["username"]=data.value("username").toString();
                        db=new ClientDatabase(this,info.value("email"));
                }
                emit login_return(data.value("result").toInt());
                break;
        }
        case TYPE_REGISTER: {
                //emit register_return(data.value("result").toInt());
                break;
        }
        case TYPE_SEND_MESSAGE: {
                if(data.value("result").toBool()==false) emit send_message_failed();
                break;
        }
        case 0: break;
        case TYPE_VERIFY: break;
        case TYPE_SEND_EMAIL: break;
        case TYPE_RECEIVE_MESSAGE: {
                chatMessage
msg(data.value("from_email").toString(),data.value("time").toString(),data.value("message").toStri
ng());

db->append_message(data.value("from_email").toString(),data.value("message").toString(),data.v
```

```
alue("time").toString(),data.value("from_email").toString());
                emit sig_receive_message(msg);
                break;
            }
            case TYPE_GET_USER_INFO: {
                info["username"]=data.value("username").toString();
                info["avatar"]=data.value("avatar").toString();
                emit info_refreshed();
                break;
            }
            case TYPE_FETCH_FRIEND_LIST: {
                qDebug()<<"Fetched "<<data.value("count").toInt()<<" friends";
                friend_email_list.clear();
                friends.clear();
                for(auto i:data.value("list").toArray()) {
                    friend_email_list.append(i.toObject().value("email").toString());

friends.append(userStruct(i.toObject().value("username").toString(),i.toObject().value("email").toS
tring(),i.toObject().value("avatarmd5").toString()));

get_avatar(i.toObject().value("email").toString(),i.toObject().value("avatarmd5").toString());
                    db->add_friend(i.toObject().value("email").toString());
                }
                emit friend_list_refreshed();
                break;
            }
            case TYPE_SEARCH_USER: {
                if(data.value("count").toInt()>0) {
                    userStruct
```

```
user(data.value("username").toString(),data.value("email").toString(),data.value("avatarmd5").toStr
ing());
                    emit user_found(user);
                }
                else emit user_not_found();
                break;
            }
            case TYPE_GET_AVATAR: {
                if(data.value("email").toString()==info.value("email")) fetch_friend_list();
                if(data.value("result").toInt()==GET_AVATAR_OK) {
                    if(XMY_Utilities::checkDir(".cache")){
                        QString
filename=".cache\\"+XMY_Utilities::emailtomd5(data.value("email").toString())+".png";

XMY_Utilities::save_pic_from_base64(data.value("avatar").toString(),filename);
                        emit avatar_got(data.value("email").toString());
                    }
                }

                break;
            }
            default: emit general_return(data.value("result").toInt());
            }
        }

        void loginsession::slot_connection_error()
        {
            establish_connect();
        }
```

```
void loginsession::slot_connected()
{
    emit general_return(CONNECTION_CONNECTED);
    qDebug()<<"connected";
}


void loginsession::slot_disconnected()
{
    qDebug("disconnected");
    emit general_return(CONNECTION_ERROR);
emit sig_logout();
```

该部分实现了服务端和客户端判断接受的消息种类以及进行的对应操作

```
void RegisterWindow::on_okButton_clicked()
{
    if(is_empty()) {
        ui->showLabel->setStyleSheet("color:red;");
        ui->showLabel->setText("Please enter all information.");
        ui->usernameEdit->setFocus();
        return;
    }
    if(passwordTooLong()) {
        ui->showLabel->setStyleSheet("color:red;");
        ui->showLabel->setText("Password too long!");
        return;
    }
```

```
        if(passwordTooShort()) {
            ui->showLabel->setStyleSheet("color:red;");
            ui->showLabel->setText("Password too short!");
            return;
        }
        if(!confirm_pwd()) {
            ui->showLabel->setStyleSheet("color:red;");
            ui->showLabel->setText("Confirm error!");
            return;
        }
        QString username = ui->usernameEdit->text();
        QString password = ui->passwordEdit->text();
        emit register_info(email, username, password);
        ui->showLabel->setStyleSheet("color:green;");
        ui->showLabel->setText("Registration request sent.");
        close();
    }
    void VerificationDialog::slot_register(QString email, QString username,QString password) {
        QJsonObject user_info;
        user_info.insert("type",TYPE_REGISTER);
        user_info.insert("email",email);
        user_info.insert("username", username);
        QString hash_password=QCryptographicHash::hash(password.toUtf8(),
QCryptographicHash::Sha384).toHex();
        user_info.insert("password",hash_password);
        QJsonDocument data(user_info);
        socket->write(data.toJson(QJsonDocument::Compact));
        close();
    }
```

该部分实现了账户的注册功能