2024-2025学年第2学期
C++课程作业

仿QQ的聊天程序
班级：24-1

成员：
2410120025 郭警豪
2410120034 陈晓豪

2025 年 6 月 12 日

目录

1 程序说明

我们小组开发了一个仿QQ的聊天程序，主要实现了以下功能：

- 用户注册与登录：用户可以通过输入用户名和密码进行注册和登录。
- 好友管理：用户可以搜索其他用户并添加为好友，查看好友列表，同意或拒绝好友申请。
- 聊天功能：用户可以与好友进行单聊，发送和接收文本消息，并查看历史聊天记录。

该程序使用Qt框架开发客户端界面，Sqlite数据库存储用户信息和聊天记录，通过Socket实现客户端与服务端的通信。

2 小组分工

郭警豪

- 注册与登录功能的实现
- 好友功能的实现
- 远程服务器的搭建

陈晓豪

- 聊天界面的开发
- 聊天记录本地化的实现

3 程序分析

3.1 功能需求

- 用户管理**：支持用户注册、登录，存储用户信息。
- 好友管理**：支持搜索用户、添加好友、查看好友列表、处理好友申请。
- 聊天功能**：支持单聊，发送和接收消息，查看历史消息。

3.2 性能需求

- 响应速度：用户操作（如登录、发送消息）应在短时间内得到响应。
- 数据存储：用户信息和聊天记录需安全存储，支持历史消息查询。
- 并发处理：支持多个用户同时在线，处理并发通信。

4 程序设计

4.1 设计思路

- 客户端：使用Qt框架开发图形用户界面，实现用户交互功能。
- 服务端：处理客户端请求，与数据库交互，存储和查询数据。
- 通信：通过Socket实现客户端与服务端的通信，传输JSON格式的数据。

4.2 数据库设计

4.2.1 用户表（users）

字段名	数据类型	描述
account	VARCHAR(20)	用户账号（主键）
password	VARCHAR(50)	密码
nickname	VARCHAR(50)	昵称

表 1: 用户表（users）结构

4.2.2 好友关系表（friendships）

4.2.3 消息表（messages）

4.3 类与函数设计

- 客户端
  - Register：注册界面类，处理用户注册逻辑。

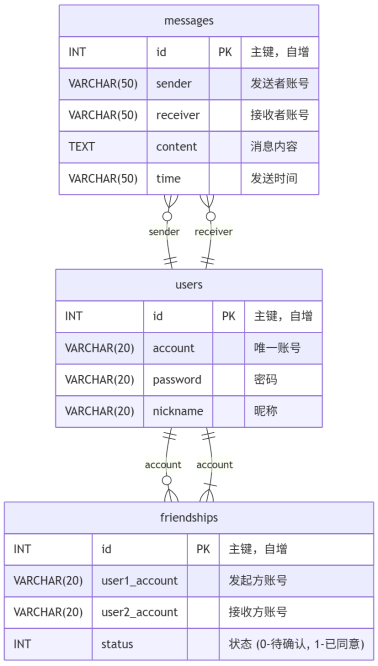

图 1: 数据库设计图(待完善)

- `MainWindow`：主窗口类，处理用户登录。
- `ChatWindow`：聊天窗口类，处理消息发送和接收。
- `Index`：主界面类，处理界面转化。
- `FriendManagement`：好友管理界面类，处理好友管理。
- `FriendItemWidget`：好友元素类，用于好友显示。
- `DragEvent`：拖动事件类，处理拖动事件。
- `Information`：信息类，处理用户信息显示与发送好友申请。
- `MessageBubbleWidget`：消息气泡类，处理聊天信息的显示。

- **服务端**
  - `Widget`：服务端主类，处理客户端请求，与数据库交互。
  - `Mythread`：线程类，用于实现多线程。

字段名	数据类型	描述
user1_account	VARCHAR(20)	用户1账号
user2_account	VARCHAR(20)	用户2账号
status	INT	好友状态（0：申请中，1：已通过）

表 2: 好友关系表（friendships）结构

字段名	数据类型	描述
sender	VARCHAR(20)	发送者账号
receiver	VARCHAR(20)	接收者账号
content	TEXT	消息内容
time	DATETIME	发送时间

表 3: 消息表（messages）结构

5 开发环境

- QT 6.9.0(windows)
- Sqlite 3.50.1(windows)
- QT 5.15.13(Ubuntu 24.04 64位)
- Sqlite 3.45.1(Ubuntu 24.04 64位)

6 程序实现

6.1 注册与登录

6.1.1 注册

用户输入昵称和密码后使用Tcp协议传输到服务端，服务端接受到后随机生成一段8到13位的随机数，对接受的密码依次使用Md5，Sha256，Real Sha3_384加密，服务端将账号信息插入数据库中，将账号发回给服务端。

6.1.2 登录

用户输入账号和密码，传输到服务端处，服务端在数据库中查找相应账号，与接受到的密码进行比对，服务端将将校验结果发送给客户端，客户端根

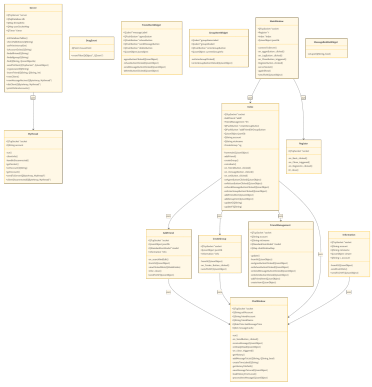

图 2: 类图(待完善)

据接受到的结果判断是否进入Index界面。

6.1.3 好友申请

客户端点击Index界面右上角的+号，点击添加好友/群按钮，进入Add Friend界面，在搜索栏处使用关键词搜索，将关键词发送给服务端，服务端将关键词作为账号和昵称在数据库中查询，并将查询结果发送给客户端，客户端将接受到的结果显示到resultlistView上。用户双击相应条目后进入Informatio n界面，客户端发出checkFriend请求，客户端检查发送者与被查看者的关系，将结果发送给客户端，客户端根据结果确定按钮的文字，若检查结果is not friend，按钮文字发送好友申请，否则为发送消息。用户点击申请好友按钮后，向服务端发送好友申请，服务端将好友关系插入数据库中，status置为0，代表好友申请未同意。

6.2 好友管理

客户端发送upadte请求，服务端查询数据库中所有与此账户有关的好友关系，将好友账号，昵称与status返回，客户端根据返回结果显示相应信息，status为0时显示同意与拒绝按钮，status为1时显示发送消息与删除好友按钮。客户端点击相应按钮后，向服务端发送相应请求，服务端对其请求进行相应处理，并向客户端发送新的好友关系结果，客户端据此更新显示。

6.3 聊天

待陈晓豪上传

6.4 聊天记录的本地存储

待陈晓豪上传

7 程序测试

测试账号：

- admin 509919938 root (此账号位于远程服务器上)

暂未完全实现

8 程序部署

服务端部署在阿里云上，服务器系统使用Ubuntu 24.04 64位，在服务器上安装Sqlite与Qt运行环境，使用Makefile编译程序。同时使用Qt的windeployqt命令打包了可在windows下运行的客户端与服务端。

9 经验总结

在本次C++课程作业中，我们小组成功开发了一个仿QQ的聊天程序。通过这个项目，我们不仅加深了对C++编程语言的理解，还学习了如何使用Qt框架进行图形用户界面的开发，以及如何通过Socket实现客户端与服务端的通信。以下是我们在开发过程中的一些经验和总结：

技术学习

- **C++编程**：通过实际项目，我们更加熟悉了C++的基本语法和高级特性，如类和对象、继承和多态等。
- **Qt框架**：我们学习了如何使用Qt框架进行界面设计和事件处理，这极大地提高了我们的界面开发效率。

- **数据库操作**：通过使用Sqlite数据库，我们掌握了基本的数据库操作，包括数据的增删改查。
- **网络编程**：我们学习了如何使用Socket进行网络通信，实现了客户端与服务端的数据交换。

项目管理

- **分工合作**：我们小组明确分工，每位成员负责不同的模块，这不仅提高了开发效率，也确保了项目的顺利进行。
- **版本控制**：使用Git进行版本控制，确保了代码的安全性和可追溯性，方便了团队协作。
- **定期会议**：我们定期举行小组会议，讨论项目进度和遇到的问题，及时调整开发计划。

问题解决

- **调试与测试**：在开发过程中，我们遇到了各种问题，通过调试和测试，我们学会了如何定位和解决问题。
- **性能优化**：针对程序的性能瓶颈，我们进行了优化，提高了程序的响应速度和数据处理能力。
- **安全问题**：我们注意到了程序的安全性，对用户密码进行了加密处理，保护了用户数据的安全。

未来展望

- **功能扩展**：我们计划在未来的版本中增加更多功能，如群聊、文件传输等，以提高程序的实用性。
- **用户体验**：我们将继续优化用户界面和交互设计，提升用户体验。
- **技术探索**：我们希望探索更多的技术，如使用更高级的数据库系统，或者尝试使用其他编程语言进行开发。

通过这次课程作业，我们不仅提升了自己的技术能力，也学会了如何进行团队合作和项目管理。我们相信这些经验将对我们未来的学习和工作产生积极的影响。

A 重要代码展示

```
void Server ::newMessageReciver(QByteArray byte,Mythread *currentThread){
  QTcpSocket *socket =currentThread->getSocket();

  QJsonObject response;

  QJsonDocument receiverdocument = QJsonDocument::fromJson(byte);

  qDebug()<<"接受的信息:"<<receiverdocument;
  if(!receiverdocument.isNull()&&receiverdocument.isObject()){
    QJsonObject receiverjsonObject =receiverdocument.object();
    QString type =receiverjsonObject.value("type").toString();
    qDebug()<<receiverjsonObject;
    if(type=="login"){
      QString account = receiverjsonObject.value("account").toString();
      QString password = receiverjsonObject.value("password").toString();
      password=crypassword(password);
      QSqlQuery query;
      query.prepare("SELECT * FROM users WHERE account = :account");
      query.bindValue(":account", account);
      if(query.exec()){
        response["type"] = "login_response";
        if(query.next()){
          int storeId = query.value(0).toInt();
          QString storeAccount = query.value(1).toString();
          QString storePassword = query.value(2).toString();
          QString snickName = query.value(3).toString();

```

```
28          QJsonObject jsonObject;
29          jsonObject["id"] = storeId;
30          jsonObject["account"] = storeAccount;
31          jsonObject["nickname"] = snickName;
32          if(storePassword==password){
33
34            response["status"] = "success";
35            response["message"] = "Login successful";
36
37            response["account"]=storeAccount;
38            response["nickname"]=snickName;
39            response["id"]=storeId;
40            currentThread->setAccount(storeAccount);
41            threadInfo[storeAccount]=currentThread;
42            userSocketMap[storeAccount] = socket;
43          }else{
44
45            response["status"] = "failure";
46            response["message"] = "Incorrect password";
47          }
48        }else{
49
50          response["status"] = "failure";
51          response["message"] = "Account not found";
52        }
53      }
54
55    }else if(type=="register"){
56
57      QString account = QString::number(QRandomGenerator::global
    ()->bounded(static_cast<double>(1000000000LL)) + 10000000, 'f
    ', 0);
58      while(isAccountExists(account)){
59        QString account = QString::number(QRandomGenerator::
    global()->bounded(static_cast<double>(1000000000LL)) +
    10000000, 'f', 0);
60      }
61      QString nickname =receiverjsonObject.value("nickname").
    toString();
```

```
62      QString password = receiverjsonObject.value("password").
    toString();
63      password=crypassword(password);
64
65      QSqlQuery query;
66      query.prepare("INSERT INTO users (account, password,
    nickname) VALUES (?, ?, ?)");
67      query.addBindValue(account);
68      query.addBindValue(password);
69      query.addBindValue(nickname);
70
71      qDebug() << "SQL Query:" << query.lastQuery();
72      qDebug() << "Bound Parameters: account=" << account << ",
    password=" << password << ", nickname=" << nickname;
73
74      response["type"] = "register_response";
75      if (query.exec()) {
76
77          QString successMessage = QString("Registration successful
    : Account: %1, Nickname: %2").arg(account, nickname);
78          qDebug()<<successMessage;
79
80
81          response["status"] = "success";
82          response["message"] = "Registration successful";
83          response["account"] = account; // 返回生成的账号
84
85
86          insertFriend(account,account,1);
87      } else {
88          QString errorMessage = QString("Registration failed:
    Error: %1")
89          .arg(query.lastError().text());
90          qDebug()<<errorMessage;
91          response["type"] = "register_response";
92          response["status"] = "failure";
93          response["message"] = query.lastError().text();
94      }
95
```

```
96      }else if(type=="addFriend_search"){
97        QString message=receiverjsonObject.value("message").
        toString();
98
99        QSqlQuery query_n;
100       query_n.prepare("SELECT account,nickname FROM users WHERE
        nickname = :message");
101       query_n.bindValue(":message", message);
102       QSqlQuery query_a;
103       query_a.prepare("SELECT account,nickname FROM users WHERE
        account = :message");
104       query_a.bindValue(":message", message);
105       QJsonArray usersArray;
106       if(query_n.exec()){
107         while(query_n.next()){
108
109           QString account = query_n.value("account").toString();
110           QString nickname = query_n.value("nickname").toString()
        ;
111
112
113           QJsonObject userObject;
114           userObject["account"] = account;
115           userObject["nickname"] = nickname;
116
117
118           usersArray.append(userObject);
119         }
120       }
121       if (query_a.exec()) {
122         while (query_a.next()) {
123
124           QString account = query_a.value("account").toString();
125           QString nickname = query_a.value("nickname").toString()
        ;
126
127           QJsonObject userObject;
128           userObject["account"] = account;
129           userObject["nickname"] = nickname;
```

```
130
131
132             usersArray.append(userObject);
133           }
134         }
135         response["type"]="addFriend_searcher_reponse";
136         response["users"]=usersArray;
137       }else if (type == "checkFriend") {
138         qDebug() << "checkFriend";
139         response["type"] = "checkFriend_response";
140
141         QString v_account = receiverjsonObject["v_account"].
      toString();
142         QString account = receiverjsonObject["account"].toString();
143
144         QSqlQuery query;
145         query.prepare("SELECT COUNT(*) AS is_friend FROM
      friendships WHERE "
146           "(user1_account = :v_account AND user2_account = :account
      AND status = 1) OR "
147           "(user2_account = :v_account AND user1_account = :account
      AND status = 1);");
148         query.bindValue(":v_account", v_account);
149         query.bindValue(":account", account);
150
151         if (query.exec()) {
152           if (query.next()) {
153             int is_friend = query.value(0).toInt();
154             if (is_friend > 0) {
155               response["result"] = "is friend";
156             } else {
157               response["result"] = "is not friend";
158             }
159           }
160         } else {
161           qDebug() << "Query failed:" << query.lastError().text();
162         }
163       }else if(type=="friend_request"){
164         response["type"]="friend_request_response";
```

```
165       QString v_account=receiverjsonObject["v_account"].toString
      ();
166       QString account=receiverjsonObject["account"].toString();
167       if(insertFriend(v_account,account,0)){
168         response["result"]="insert_successed";
169       }else{
170         response["result"]="insert_not_successed";
171       }
172
173       auto it = threadInfo.find(account);
174       if (it != threadInfo.end()) {
175         qDebug() << account<<"在线";
176         Mythread* f_thread = it.value();
177         QTcpSocket* f_socket=f_thread->getSocket();
178         QJsonObject f_response;
179         find(account,f_response);
180         sendToClient(f_socket,f_response);
181       } else {
182         qDebug() << account<<"未在线";
183       }
184     }else if(type=="update"){
185       QString account =receiverjsonObject["account"].toString();
186       find(account,response);
187     }else if(type=="Agree_the_friend"){
188       QString account = receiverjsonObject["account"].toString();
189       QString friend_account = receiverjsonObject["friend_account
      "].toString();
190       qDebug() << "account:" << account << "friend_account:" <<
      friend_account;
191       QSqlQuery query;
192       query.prepare("UPDATE friendships SET status = 1 WHERE
      user1_account = ? AND user2_account = ?;");
193       query.addBindValue(friend_account);
194       query.addBindValue(account);
195       if (query.exec()) {
196         qDebug() << "Agree_the_friend" << query.lastQuery();
197         find(account,response);
198
199         auto it = threadInfo.find(friend_account);
```

```
200         if (it != threadInfo.end()) {
201           qDebug() << friend_account<<"在线";
202           Mythread* f_thread = it.value();
203           QTcpSocket* f_socket=f_thread->getSocket();
204           QJsonObject f_response;
205           find(friend_account,f_response);
206           sendToClient(f_socket,f_response);
207         } else {
208           qDebug() << friend_account<<"未在线";
209         }
210       } else {
211         qDebug() << "Update failed:" << query.lastError().text();
212       }
213       qDebug() << "Agree_the_friend" << query.lastQuery();
214     }else if (type=="Refuse_the_friend"||type=="Delete_the_friend
      "){
215       QString account = receiverjsonObject["account"].toString();
216       QString friend_account = receiverjsonObject["friend_account
      "].toString();
217       qDebug() << "account:" << account << "friend_account:" <<
      friend_account;
218
219       QSqlQuery query;
220       query.prepare("DELETE FROM friendships WHERE user1_account
      = :friend_account AND user2_account = :account;");
221       query.bindValue(":account", account);
222       query.bindValue(":friend_account", friend_account);
223
224       if (query.exec()) {
225         qDebug() << "Delete successful";
226         find(account,response);
227
228         auto it = threadInfo.find(friend_account);
229         if (it != threadInfo.end()) {
230           qDebug() << friend_account<<"在线";
231           Mythread* f_thread = it.value();
232           QTcpSocket* f_socket=f_thread->getSocket();
233           QJsonObject f_response;
234           find(friend_account,f_response);
```

```
235           sendToClient(f_socket,f_response);
236         } else {
237           qDebug() << friend_account<<"未在线";
238         }
239       } else {
240         qDebug() << "Delete failed:" << query.lastError().text();
241       }
242     }
243     else if(type == "chat_message"){
244       QString from = receiverjsonObject["from"].toString();
245       QString to = receiverjsonObject["to"].toString();
246       qDebug() << "to_user:::" << from << to;
247       QString content = receiverjsonObject["content"].toString();
248       QString time = receiverjsonObject["time"].toString();
249
250       QJsonObject chatData;
251       chatData["type"] = "chat_message";
252       chatData["from"] = from;
253       chatData["to"] = to;
254       chatData["content"] = content;
255       chatData["time"] = time;
256
257       QJsonDocument doc(chatData);
258       QByteArray data = doc.toJson();
259
260
261       QSqlQuery saveChat;
262       saveChat.prepare("INSERT INTO messages (sender, receiver,
    content, time) VALUES (:from, :to, :content, :time)");
263       saveChat.bindValue(":from", from);
264       saveChat.bindValue(":to", to);
265       saveChat.bindValue(":content", content);
266       saveChat.bindValue(":time", time);
267
268       if (!saveChat.exec()) {
269         qDebug() << "Failed to save message:" << saveChat.
    lastError().text();
270       } else {
271         qDebug() << "Message saved successfully";
```

```
272       }
273
274
275       if (userSocketMap.contains(to)) {
276         userSocketMap[to]->write(data);
277         response["status"] = "sent";
278       } else {
279         response["status"] = "offline";
280       }
281     }
282     else if(type == "get_history") {
283       QString a1 = receiverjsonObject["from"].toString();
284       QString a2 = receiverjsonObject["to"].toString();
285
286       QSqlQuery query;
287       query.prepare("SELECT sender, receiver, content, time FROM
    messages WHERE "
288       "(sender = :a1 AND receiver = :a2) OR (sender = :a2 AND
    receiver = :a1) "
289       "ORDER BY time ASC");
290       query.bindValue(":a1", a1);
291       query.bindValue(":a2", a2);
292
293
294       QMap<QString, QString> nicknameMap;
295       QSqlQuery nicknameQuery;
296       nicknameQuery.prepare("SELECT account, nickname FROM users
    WHERE account = :a1 OR account = :a2");
297       nicknameQuery.bindValue(":a1", a1);
298       nicknameQuery.bindValue(":a2", a2);
299       if (nicknameQuery.exec()) {
300         while (nicknameQuery.next()) {
301           QString account = nicknameQuery.value("account").
    toString();
302           QString nickname = nicknameQuery.value("nickname").
    toString();
303           nicknameMap[account] = nickname;
304         }
305       }
```

```
306
307       QJsonArray msgArray;
308       if (query.exec()) {
309         while (query.next()) {
310           QString sender = query.value("sender").toString();
311           QJsonObject msg;
312           msg["name"] = nicknameMap.value(sender, sender);//发送
      者的名称
313           msg["from"] = sender;
314           msg["to"] = query.value("receiver").toString();
315           msg["content"] = query.value("content").toString();
316           msg["time"] = query.value("time").toString();
317           msgArray.append(msg);
318         }
319       }
320
321       response["type"] = "get_history_response";
322       response["messages"] = msgArray;
323
324
325     } else if(type == "create_group") {
326       QString groupName = receiverjsonObject["group_name"].
      toString();
327       QString ownerAccount = receiverjsonObject["owner_account"].
      toString();
328
329
330       QSqlQuery query;
331       query.prepare("INSERT INTO `groups` (group_name,
      owner_account) VALUES (:group_name, :owner_account)");
332       query.bindValue(":group_name", groupName);
333       query.bindValue(":owner_account", ownerAccount);
334
335       response["type"] = "create_group_response";
336
337       if (query.exec()) {
338         qint64 groupId = query.lastInsertId().toLongLong();
339
340
```

```
341         QSqlQuery addMember;
342         addMember.prepare("INSERT INTO group_members (group_id,
        account, role) VALUES (:group_id, :account, 'owner')");
343         addMember.bindValue(":group_id", groupId);
344         addMember.bindValue(":account", ownerAccount);
345         addMember.exec();
346 
347         response["status"] = "success";
348         response["group_id"] = QString::number(groupId);
349         response["message"] = "Group created successfully";
350         if (!addMember.exec()) {
351           qDebug() << "Failed to add group member:" << addMember.
        lastError().text();
352         } else {
353           qDebug() << "Added group member successfully.";
354         }
355       } else {
356         response["status"] = "failure";
357         response["message"] = query.lastError().text();
358       }
359 
360     }
361 
362   }
363   sendToClient(socket,response);
364 }
```

Listing 1: 服务端接受信息并处理

```
1   #include "chatwindow.h"
2   #include "ui_chatwindow.h"
3   #include "dragevent.h"
4   #include <QFile>
5   #include <QDir>
6   #include <QDesktopServices>
7   ChatWindow::ChatWindow(QTcpSocket *socket, const QString &
      selfAccount, const QString &friendAccount, const QString &
      friendName, QWidget *parent)
8   : QWidget(parent)
```

```
9   , ui(new Ui::ChatWindow)
10  {
11    ui->setupUi(this);
12    setWindowFlag(Qt::FramelessWindowHint);
13
14    ui->MessageListWidget->setSpacing(5);
15    ui->MessageListWidget->setSelectionMode(QAbstractItemView::
      NoSelection);
16    ui->MessageListWidget->setFocusPolicy(Qt::NoFocus);
17    ui->MessageListWidget->setVerticalScrollMode(
      QAbstractItemView::ScrollPerPixel);
18    ui->MessageListWidget->setStyleSheet(R"(
19    QListWidget {
20      background: transparent;
21      border: none;
22    }
23    QListWidget::item {
24      background: transparent;
25      border: none;
26      margin: 0px;
27      padding: 0px;
28    }
29    QListWidget::item:hover {
30      background: transparent;
31    }
32    QListWidget::item:selected {
33      background: transparent;
34    }
35    )");
36    ui->EditArea->installEventFilter(this);
37    this->installEventFilter(new DragEvent());
38    this->socket = socket;
39    this->selfAccount = selfAccount;
40    this->friendAccount = friendAccount;
41    this->friendName = friendName;
42    if(selfAccount!=friendName){
43      ui->FriendName->setText(friendName);
44    }
45    qDebug() << "to_user:::" << selfAccount << friendAccount;
```

```
46    loadHistoryFromLocal();
47    getHistory();
48
49    connect(ui->close,&QToolButton::clicked,this,&ChatWindow::
      on_close_triggered);
50    connect(ui->SendButton, &QPushButton::clicked, this, &
      ChatWindow::on_SendButton_clicked);
51  }
52  bool ChatWindow::eventFilter(QObject *obj, QEvent *event) {
53    if (obj == ui->EditArea && event->type() == QEvent::KeyPress)
       {
54      QKeyEvent *keyEvent = static_cast<QKeyEvent*>(event);
55      if ((keyEvent->key() == Qt::Key_Enter || keyEvent->key() ==
      Qt::Key_Return) &&
56      keyEvent->modifiers() == Qt::NoModifier) {
57        emit on_SendButton_clicked();
58        return true;
59      }
60    }
61    return QWidget::eventFilter(obj, event);
62    ui->EditArea->setFocus();
63  }
64
65  ChatWindow::~ChatWindow()
66  {
67    delete ui;
68  }
69
70
71
72  void ChatWindow::on_SendButton_clicked()
73  {
74    QString text = ui->EditArea->toPlainText();
75    while (!text.isEmpty() && text.endsWith('\n')) {
76      text = text.chopped(1);
77    }
78    if (text.isEmpty()) return;
79
80    QJsonObject object;
```

```
81      object["type"] = "chat_message";
82      object["from"] = this->selfAccount;
83      object["to"] = this->friendAccount;
84
85      object["content"] = text;
86      object["time"] = QDateTime::currentDateTime().toString("yyyy-
        MM-dd HH:mm:ss");
87      QByteArray data = QJsonDocument(object).toJson();
88      socket->write(data);
89
90      QString messageLine =  text;
91      addMessageToList(messageLine, "皇帝",true);
92      ui->EditArea->clear();
93      saveMessageToLocal(object);
94    }
95
96    void ChatWindow::receiveMessage(const QJsonObject &js) {
97      qDebug()<<__func__<<js;
98      if (js["from"].toString() == friendAccount) {
99        QString time = js["time"].toString();
100       QString content = js["content"].toString();
101       QString messageLine = content;
102       addMessageToList(messageLine, js["name"].toString(),false);
103       saveMessageToLocal(js);
104     }
105   }
106
107   void ChatWindow::onReadyRead(QJsonObject jsonobject)
108   {
109     qDebug() << __func__ << jsonobject;
110     qDebug() << "Data arrived";
111
112     QString type = jsonobject["type"].toString();
113     if (type == "chat_message") {
114       receiveMessage(jsonobject);
115     }
116     else if (type == "get_history_response") {
117       QJsonArray history = jsonobject["messages"].toArray();
118       for (const QJsonValue &val : history) {
```

```
119         QJsonObject msg = val.toObject();
120         processNewMessage(msg);
121       }
122     }
123   }
124
125
126   void ChatWindow::on_close_triggered()
127   {
128     this->hide();
129   }
130
131   void ChatWindow::getHistory()
132   {
133     QJsonObject json;
134     json["type"] = "get_history";
135     json["from"] = this->selfAccount;
136     json["to"] = this->friendAccount;
137     QByteArray data = QJsonDocument(json).toJson();
138     socket->write(data);
139   }
140
141   void ChatWindow::addMessageToList(const QString &text, const
      QString &name, bool isOwnMessage)
142   {
143
144     QWidget *container = new QWidget;
145     QVBoxLayout *layout = new QVBoxLayout(container);
146     layout->setSpacing(2);
147     layout->setContentsMargins(10, 0, 10, 0);
148
149
150     QLabel *nameLabel = new QLabel(name);
151     nameLabel->setStyleSheet("color: gray; font-size: 10px;");
152     if (isOwnMessage) {
153       nameLabel->setAlignment(Qt::AlignRight);
154     } else {
155       nameLabel->setAlignment(Qt::AlignLeft);
156     }
```

```
157
158
159     MessageBubbleWidget *bubble = new MessageBubbleWidget(text,
        isOwnMessage);
160
161
162     layout->addWidget(nameLabel);
163     layout->addWidget(bubble);
164
165
166     QListWidgetItem *item = new QListWidgetItem(ui->
        MessageListWidget);
167     item->setSizeHint(container->sizeHint());
168
169     ui->MessageListWidget->addItem(item);
170     ui->MessageListWidget->setItemWidget(item, container);
171     ui->MessageListWidget->scrollToBottom();
172   }
173
174   QWidget* ChatWindow::createTimeLabel(const QString &time)
175   {
176     QLabel *label = new QLabel(time);
177     label->setAlignment(Qt::AlignCenter);
178     label->setStyleSheet("color: gray; font-size: 12px; padding:
        5px;");
179
180     QWidget *wrapper = new QWidget;
181     QHBoxLayout *layout = new QHBoxLayout(wrapper);
182     layout->addWidget(label);
183     layout->setAlignment(Qt::AlignCenter);
184     layout->setContentsMargins(0, 0, 0, 0);
185     return wrapper;
186   }
187
188   QString ChatWindow::getHistoryFilePath() const {
189     QString filename = QString("%1_%2.json").arg(selfAccount).arg
        (friendAccount);
190
191
```

```
192     QString currentPath = QDir::currentPath();
193     QString dirPath = currentPath + "/chat_history";
194     QDir dir;
195
196
197     if (!dir.exists(dirPath)) {
198       qDebug() << "正在创建目录: " << dirPath;
199       if (!dir.mkpath(dirPath)) {
200         qDebug() << "创建目录失败! ";
201       }
202     }
203
204     QString fullPath = dirPath + "/" + filename;
205     qDebug() << "create" << fullPath;
206     return fullPath;
207   }
208
209   void ChatWindow::saveMessageToLocal(const QJsonObject &msg) {
210     QString filePath = getHistoryFilePath();
211     qDebug() << "save" << filePath;
212
213     QFile file(filePath);
214     QJsonArray historyArray;
215
216
217     if (file.exists()) {
218       qDebug() << "文件已存在，尝试读取历史记录...";
219       if (file.open(QIODevice::ReadOnly)) {
220         QByteArray data = file.readAll();
221         QJsonDocument doc = QJsonDocument::fromJson(data);
222         if (doc.isArray()) {
223           historyArray = doc.array();
224         }
225         file.close();
226       }
227     }
228
229
230     QString messageId = msg["id"].toString();
```

```
231     if (messageId.isEmpty()) {
232       QString content = msg["content"].toString();
233       QString time = msg["time"].toString();
234       messageId = QString("%1_%2").arg(time).arg(content);
235     }
236 
237     bool exists = false;
238     for (const QJsonValue &val : historyArray) {
239       QJsonObject existingMsg = val.toObject();
240       QString existingId = existingMsg["id"].toString();
241       if (existingId.isEmpty()) {
242         QString existingContent = existingMsg["content"].toString
    ();
243         QString existingTime = existingMsg["time"].toString();
244         existingId = QString("%1_%2").arg(existingTime).arg(
    existingContent);
245       }
246       if (existingId == messageId) {
247         exists = true;
248         break;
249       }
250     }
251 
252     if (!exists) {
253       historyArray.append(msg);
254 
255       if (file.open(QIODevice::WriteOnly | QIODevice::Truncate))
    {
256         QJsonDocument doc(historyArray);
257         qint64 bytesWritten = file.write(doc.toJson());
258         qDebug() << " 写入文件成功，共写入" << bytesWritten << "字
    节";
259         file.close();
260       } else {
261         qDebug() << " 无法打开文件进行写入:
    " << file.errorString();
262       }
263     } else {
264       qDebug() << " 消息已存在，未写入文件: " << messageId;
265     }
```

```
266
267 }
268 void ChatWindow::loadHistoryFromLocal() {
269   QString filePath = getHistoryFilePath();
270   qDebug() << "get" << filePath;
271
272   QFile file(filePath);
273   if (!file.exists()) return;
274
275   if (file.open(QIODevice::ReadOnly)) {
276     QByteArray data = file.readAll();
277     QJsonDocument doc = QJsonDocument::fromJson(data);
278     if (doc.isArray()) {
279       QJsonArray history = doc.array();
280       for (const QJsonValue &val : history) {
281         QJsonObject msg = val.toObject();
282         processNewMessage(msg);
283       }
284     }
285     file.close();
286   }
287 }
288
289 void ChatWindow::processNewMessage(const QJsonObject &msg) {
290   QString messageId = msg["id"].toString();
291   if (messageId.isEmpty()) {
292
293     QString content = msg["content"].toString();
294     QString time = msg["time"].toString();
295     messageId = QString("%1_%2").arg(time).arg(content);
296   }
297
298   if (!messageCache.contains(messageId)) {
299     QString sender = msg["from"].toString();
300     QString content = msg["content"].toString();
301     QString name = msg["name"].toString();
302     bool isOwn = (sender == selfAccount);
303
304     QDateTime timestamp = QDateTime::fromString(msg["time"].
```

```
    toString(), "yyyy-MM-dd HH:mm:ss");
305
306
307       if (lastMessageTime.isNull() ||
308       lastMessageTime.date() != timestamp.date() ||
309       lastMessageTime.secsTo(timestamp) > TIME_THRESHOLD_SECONDS)
          {
310           QListWidgetItem* item = new QListWidgetItem(ui->
    MessageListWidget);
311           QWidget *timeWidget = createTimeLabel(timestamp.toString(
    "yyyy-MM-dd HH:mm:ss"));
312           item->setSizeHint(timeWidget->sizeHint());
313           ui->MessageListWidget->addItem(item);
314           ui->MessageListWidget->setItemWidget(item, timeWidget);
315       }
316
317
318       addMessageToList(content, name, isOwn);
319
320
321       saveMessageToLocal(msg);
322
323
324       messageCache.insert(messageId);
325
326
327       lastMessageTime = timestamp;
328     } else {
329       qDebug() << " 消息已存在，跳过： " << messageId;
330     }
331   }
```

Listing 2: chatwindow的实现