**分布式架构的即时通讯系统wChat项目简介**

**第一部分: 项目概述**

本项目是一套基于分布式架构的即时通讯系统。支持用户注册登录、好友管理、实时消息收发等核心即时通讯场景，同时具备高并发处理、分布式协同及数据安全存储能力。

主要技术栈：c++，Qt开发，node.js邮箱服务，boost异步网络编程，http，tcp，grpc，protobuf，json，mysql，redis，单例模式，线程池，分布式

**第一部分:服务端设计**

一.重要类

1.class CServer:public std::enable\_shared\_from\_this<CServer>

主要作用，接收连接请求

(1)通过构造时传来的 boost::asio::io\_context& ioc 及端口，初始化 tcp::acceptor \_acceptor

(2)void CServer::Start()中启动异步监听

先从IOContextPool中获取io\_context, 构建会话(http或tcp)

启动监听\_acceptor.async\_accept(......)，绑定的回调函数会再次调用Start()继续监听。

2.class AsioIOContextPool :public Singleton<AsioIOContextPool>

主要作用，管理io\_contexts( std::vector<IOContext> \_IOContexts ),可以获取io\_context的引用。

(1)构造函数根据cpu核心数初始化 std::vector<WorkPtr> \_works; ,再启动对应数量线程，运行io\_context

\_works[i] = std::unique\_ptr<Work>(new Work(\_IOContexts[i].get\_executor()));

threads.emplace\_back([this, i]() {IOContexts[i].run();});

(2)boost::asio::io\_context& AsioIOContextPool::GetIOContext()，返回io\_context的引用，供外部使用

(3)void AsioIOContextPool::Stop(),先后关闭\_IOContexts，\_works，\_threads，结束服务

3.class LogicSystem:public Singleton<LogicSystem>

主要作用，接收网络线程的任务，写入任务队列，等待逻辑线程获取，并调用对应逻辑函数处理

(1)写入函数，先上锁，再写入队列，消息数量由0变为1，则条件变量发送信号

void PostMsgToQue(shared\_ptr < LogicNode> msg)

\_consume.notify\_one();

(2)构造函数中，逻辑线程注册函数，其中循环从任务队列获取任务并查找逻辑函数处理

\_worker\_thread = std::thread (&LogicSystem::DealMsg, this);

循环开始前，判断任务队列，如果为空则用条件变量阻塞等待，并释放锁

while (\_msg\_que.empty() && !\_b\_stop) {

\_consume.wait(unique\_lk);

}

4.class ChatServiceImpl final : public ChatService::Service

主要作用，作为grpc服务端，接收其他聊天服务器的请求，用于转发处理添加好友，同意申请，发送消息等信息

5.class ChatGrpcClient :public Singleton<ChatGrpcClient>

主要作用，作为grpc客户端，发送请求给其他聊天服务器，包括添加好友，同意申请，发送消息等信息

6.class MysqlMgr: public Singleton<MysqlMgr>

主要作用，提供sql服务，通过获取 MySqlPool的连接，与mysql数据库进行交互，提供各种外部可调用其函数接口

7.class RedisMgr: public Singleton<RedisMgr>, public std::enable\_shared\_from\_this<RedisMgr>

主要作用，提供redis服务，通过获取 RedisConPool的连接，与redis数据库进行交互，提供各种外部可调用其函数接口

8.class ChatConPool class MySqlPool class RedisConPool

主要作用，均是管理多个连接的线程池，在需要发送消息时，可从中取出连接，通过锁和条件变量的机制实现，

以class ChatConPool为例

(1)构造函数中，创建多个连接，放入队列std::queue<std::unique\_ptr<ChatService::Stub>>connections\_中

for (size\_t i = 0;i < poolSize\_;++i) {

std::shared\_ptr<Channel>channel = grpc::CreateChannel(.............);

connections\_.push(ChatService::NewStub(channel));// newstub 返回右值

}

(2)获取连接的智能指针

std::unique\_ptr<ChatService::Stub>getConnection();

(3)返回连接的智能指针

void returnConnection(std::unique\_ptr<ChatService::Stub> context);

9.class ConfigMgr

主要作用，读取.ini配置文件，存入std::map<std::string, SectionInfo> \_config\_map，供外部使用

10.class UserMgr : public Singleton<UserMgr>

主要作用，记录用户在该聊天服务器所在会话，用于服务端主动发出消息给对应客户端

(1)std::unordered\_map<int, std::shared\_ptr<CSession>> \_uid\_to\_session;

(2)设置用户所在会话，修改用户所在会话，删除用户所在会话

std::shared\_ptr<CSession> GetSession(int uid);

void SetUserSession(int uid, std::shared\_ptr<CSession> session);

void RmvUserSession(int uid, std::string session\_id);

二.服务器类型

1.验证码服务器VarifyServer

主要作用，生成并发送验证码给网关服务器发来的邮箱，并写入redis，用于客户端需要验证码的请求

2.网关服务器gate\_server

主要作用，接收处理http请求，处理登录注册等逻辑

主要构成类：

(1)class CServer:public std::enable\_shared\_from\_this<CServer>

(2)class AsioIOContextPool :public Singleton<AsioIOContextPool>

(3)class LogicSystem:public Singleton<LogicSystem>

(4)class HttpConnection:public std::enable\_shared\_from\_this<HttpConnection> // http会话服务管理类

(5)class ChatGrpcClient :public Singleton<ChatGrpcClient>

(6)class MysqlMgr: public Singleton<MysqlMgr>

(7)class RedisMgr: public Singleton<RedisMgr>, public std::enable\_shared\_from\_this<RedisMgr>

(8)class ConfigMgr

3.状态服务器status\_server

主要作用，接收网关服务器发来的登录信息，查询用户状态是否合理，生成token及为客户端分配的聊天服务器tcp\_server的端口和ip，并返回给gate\_server

主要构成类：

(1)class CServer:public std::enable\_shared\_from\_this<CServer>

(2)class AsioIOContextPool :public Singleton<AsioIOContextPool>

(3)class LogicSystem:public Singleton<LogicSystem>

(4)class StatusServiceImpl final : public StatusService::Service

(5)class ConfigMgr

(6)class MysqlMgr: public Singleton<MysqlMgr>

(7)class RedisMgr: public Singleton<RedisMgr>, public std::enable\_shared\_from\_this<RedisMgr>

4.聊天服务器tcp\_server

主要作用，与用户建立长连接，处理客户端添加好友，同意申请，发送消息等请求

主要构成类：

(1)class CServer:public std::enable\_shared\_from\_this<CServer>

(2)class AsioIOContextPool :public Singleton<AsioIOContextPool>

(3)class LogicSystem:public Singleton<LogicSystem>

(4)class HttpConnection:public std::enable\_shared\_from\_this<HttpConnection> // http服务管理类

(5)class ChatGrpcClient :public Singleton<ChatGrpcClient>

(6)class MysqlMgr: public Singleton<MysqlMgr>

(7)class RedisMgr: public Singleton<RedisMgr>, public std::enable\_shared\_from\_this<RedisMgr>

(8)class ChatServiceImpl final : public ChatService::Service

(9)class UserMgr : public Singleton<UserMgr>

(10)class ConfigMgr

**第三部分: 数据库设计**

1.user表，记录用户的主要个人数据如uid，name，email等

主要作用，通过uid查询出该账号的个人信息

2.friend表，记录好友uid及其他信息，通过self\_id建立索引

主要作用，通过self\_id，查询出其所有好友，及与其相关的信息，如昵称，标签等

3.chat\_conversations表，会话表，记录好友双方的会话id，及其他会话信息，如最后一条消息的时间，消息id

主要作用，通过user\_a\_id及user\_b\_id建立的联合索引，查询出会话id，进而快速查询聊天记录

4.chat\_messages表，聊天记录表，消息的发送者，接收者，内容，类型，时间等信息

设立外键，使该表的 conversation\_id 字段与 chat\_conversations的 id字段关联

主要作用，通过 conversation\_id 建立索引，快速查询所有该会话id下的所有聊天消息

**第四部分：主要网络操作过程**

一.登录

1.客户端在LoginDialog的Qt类输入邮箱密码校验格式后，通过httpmgr单例类以json格式发送http的post请求至gate\_server服务器

HttpMgr::GetInstance()->PostHttpReq(QUrl(gate\_url\_prefix+"/user\_login"),json\_obj, ReqId::ID\_LOGIN\_USER,Modules::LOGINMOD);

消息结构1.路由 2.消息体 3.请求id 4.消息种类

2.httpmgr注册消息的回调函数，接收gate\_server消息

QObject::connect(reply,&QNetworkReply::finished,[](){};

该回调函数在步骤10触发。

3.gate\_server的server类监听到新的http请求后，创建HttpConnection类管理连接

4.通过HttpConnection的HandleReq()方法，解析出请求类型为post请求，再移交逻辑层LogicSystem单例类处理

bool success = LogicSystem::GetInstance()->HandlePost(\_request.target(),shared\_from\_this());

5.LogicSystem通过消息路由，从构造函数即注册好的逻辑函数关联容器post\_handlers中找到对应逻辑函数，并执行

if (\_post\_handlers.find(path) == \_post\_handlers.end()) {

return false;

}

post\_handlers[path](connection);

return true;

6.登录请求逻辑函数中解析json格式，通过连接了mysql的mysqlmgr的checkpwd方法查询数据库，校验邮箱及密码，获取用户uid

bool pwd\_valid = MysqlMgr::GetInstance()->CheckPwd(email, pwd, userInfo);

7.若查询成功，会通过 StatusGrpcClient管理的grpc通道，向status\_server服务器发送包含用户uid的请求，以找到合适的tcp\_chat\_server服务器管理新连接，同时获取其ip和端口。此请求是阻塞等待的。

auto reply = StatusGrpcClient::GetInstance()->GetChatServer(userInfo.uid);

8.在status\_server中， StatusServiceImpl会异步的接收grpc请求。收到GetChatServer请求后，查询适合的tcp\_chat\_server服务器的ip及端口，再生成与uid绑定的token，返回给gate\_server。

9.完成逻辑函数后，HttpConnection将校验成功的消息及status\_server生成的token返回通过WriteResponse()方法异步的返回消息给客户端。

10.步骤2注册的回调函数检验成功后发送http请求成功信号

emit self->sig\_http\_finish(req\_id,res,ErrorCodes::SUCCESS,mod);

11.对应槽函数检验消息模式转发信号，消息返回给LoginDialog类，触发槽函数LoginDialog::slot\_login\_mod\_finish

if(mod == Modules::LOGINMOD){

emit sig\_login\_mod\_finish(id, res, err);

}

12.LoginDialog::slot\_login\_mod\_finish解析json格式后，根据消息id调用对应函数处理

\_handlers[id](jsonDoc.object());

通过error字段，可以知道请求是否失败，及失败类型

13.校验无误，处理函数发送信号，通知tcpmrg类启动tcp连接

emit sig\_connect\_tcp(si);

槽函数中,\_socket.connectToHost(si.Host, \_port);

14.若连接成功，触发socket回调，发送连接成功信号sig\_con\_success(true)，通知LoginDialog

QObject::connect(&\_socket, &QTcpSocket::connected, [&]() {

qDebug() << "Connected to server!";

// 连接建立后发送消息

emit sig\_con\_success(true);

});

15.槽函数LoginDialog::slot\_tcp\_con\_finish(bool bsuccess)中，将uid，token以json格式发出信号，请求tcpmgr将消息发送给tcp\_chat\_server

emit TcpMgr::GetInstance()->sig\_send\_data(ReqId::ID\_CHAT\_LOGIN, jsonString);

16.对应tcp\_chat\_server服务器收到消息，投递到逻辑层LogicSystem管理的消息队列里，等待取出处理

LogicSystem::GetInstance()->PostMsgToQue(std::make\_shared<LogicNode>(shared\_from\_this(), \_recv\_msg\_node));

17.登录消息处理函数LogicSystem::LoginHandler读出uid及token，再从Redis中读出对应uid的token\_value,比较两token是否相同

bool success = RedisMgr::GetInstance()->Get(token\_key, token\_value);

18.若相同，校验通过，再从mysql中读取用户其他消息

bool b\_base = GetBaseInfo(base\_key, uid, user\_info);// 用户基本消息

auto b\_apply = GetFriendApplyInfo(uid, apply\_list); // 用户好友被邀请列表

bool b\_friend\_list = GetFriendList(uid, friend\_list); // 用户好友列表

19.对每个好友，查询过往聊天信息，同样写入friend\_list

MysqlMgr::GetInstance()->GetMessages(uid, friend\_ele->uid, obj);

其中，先通过双方uid查询会话表

"SELECT id FROM chat\_conversations WHERE user\_a\_id = ? AND user\_b\_id = ?"

再通过会话表查询所有聊天记录

"SELECT sender\_id, receiver\_id, content, msg\_type, status, send\_time, read\_time "

"FROM chat\_messages "

"WHERE conversation\_id = ? "

"ORDER BY send\_time ASC"

20.以该账号的uid为键，tcp\_chat\_server服务器的名字为值，写入redis，方便以后查询uid所在服务器

RedisMgr::GetInstance()->Set(ipkey, server\_name);

21.再将uid与session写入服务器的usermgr，方便以后通过uid得知所在的session

UserMgr::GetInstance()->SetUserSession(uid, session);

22.逻辑函数结束，调用写函数，在其中异步发送回包。

23.客户端tcpmgr收到消息，根据消息id调用登录回包函数，将信息写入管理用户消息的单例类usermgr并发送界面跳转信号，进入主界面，登录成功

emit sig\_switch\_chatdlg();

初始写入的信息包括

用户个人信息 UserMgr::GetInstance()->SetUserInfo(user\_info);

用户好友申请列表 UserMgr::GetInstance()->AppendApplyList(jsonObj["apply\_list"].toArray());

用户好友列表 UserMgr::GetInstance()->AppendFriendList(jsonObj["friend\_list"].toArray());

二.注册过程

1.客户端在LoginDialog页面点击注册后切换到RegistDialog页面

connect(\_login\_dlg,&::LoginDialog::switchRegist,this,&MainWindow::slotSwitchRegist);

2.在注册页面输入邮箱后，点击获取验证码，先检查邮箱是否符合正则，再发出post信号给httpmgr

QRegularExpression regex(R"((\w+)(\.|\_)?(\w\*)@(\w+)(\.(\w+))+)");

bool match=regex.match(email).hasMatch();

HttpMgr::GetInstance()->PostHttpReq("/get\_varifycode",........)

3.httpmgr发出post请求，连接回调函数

QNetworkReply \*reply =\_manager.post(request,data);

Object::connect(reply,&QNetworkReply::finished,.........);

4.gate\_server的httpmgr接收到后转交给对应LogicSystem的post模块的逻辑函数处理

bool success = LogicSystem::GetInstance()->HandlePost(\_request.target(), shared\_from\_this());

5.逻辑函数解析数据后，通过grpc调用邮件发送服务器VarifyServer( node.js 编写 )的发送验证码功能

GetVarifyRsp rsp = VerifyGrpcClient::GetInstance()->GetVarifyCode(email);

6.VarifyServer中grpc.Server绑定的对应异步接收函数，接收到信号

server.addService(message\_proto.VarifyService.service, { GetVarifyCode: GetVarifyCode })

async function GetVarifyCode(call, callback)

7.在redis中查找是否已有该邮箱的验证码

let query\_res = await redis\_module.GetRedis(const\_module.code\_prefix+call.request.email);

8.没有则随机生成4位验证码

uniqueId = uuidv4();

uniqueId = uniqueId.substring(0, 4);

9.调用redis模块的写函数，设置键值对及过期日期

async function SetRedisExpire(key,value, exptime){...

await RedisCli.set(key,value)

await RedisCli.expire(key, exptime);.....}

10.编辑好邮件后发送

let send\_res = await emailModule.SendMail(mailOptions);

11.用户收到邮箱验证码后，在注册页面输入其他注册信息，点击确认按钮，触发槽函数

void RegistDialog::on\_sure\_btn\_clicked()

12.槽函数读取用户输入后，再发出post信号给httpmgr, httpmgr发出post请求，连接回调函数

HttpMgr::GetInstance()->PostHttpReq("/user\_register",........);

QNetworkReply \*reply =\_manager.post(request,data);

13.gate\_server的httpmgr接收到后转交给对应LogicSystem的post模块的逻辑函数处理

bool success = LogicSystem::GetInstance()->HandlePost(\_request.target(), shared\_from\_this());

14.逻辑函数先查找redis中email对应验证码是否存在

bool b\_get\_varify = RedisMgr::GetInstance()->Get(CODEPREFIX + email, varify\_code);

15.再调用数据库函数，写入用户数据

int uid = MysqlMgr::GetInstance()->RegUser(name, email, pwd);

std::unique\_ptr < sql::PreparedStatement > stmt(

con->\_con->prepareStatement("CALL reg\_user(?,?,?,?,@result)"));

16.数据库的reg\_user函数，主要接收用户数据，检查用户命及email是否存在，再生成并返回新的uid，写入新用户数据

IF EXISTS (SELECT 1 FROM `user` WHERE `name` = new\_name) THEN

IF EXISTS (SELECT 1 FROM `user` WHERE `email` = new\_email) THEN

SELECT `id` INTO @new\_id FROM `user\_id`;

INSERT INTO `user` (`uid`, `name`, `email`, `pwd`, `icon`)

VALUES (@new\_id, new\_name, new\_email, new\_pwd, new\_icon);

17.服务端返回注册消息成功，客户端跳转到登录界面，注册完成

三.修改密码

1.客户端在LoginDialog页面修改密码后切换到ResetDialog页面

2.同注册过程一样，先获取邮箱验证码，再将新的用户信息，发送给gate\_server

3.httpmgr接收并转交给对应逻辑函数处理

bool success = LogicSystem::GetInstance()->HandlePost(\_request.target(), shared\_from\_this());

4.逻辑函数先查找redis中email对应的验证码是否合理

bool b\_get\_varify = RedisMgr::GetInstance()->

Get(CODEPREFIX + src\_root["email"].asString(), varify\_code);

5.再查询数据库，检查邮箱是否存在

bool email\_valid = MysqlMgr::GetInstance()->CheckEmail(email);

6.更新密码

bool b\_up = MysqlMgr::GetInstance()->UpdatePwd(name, pwd);

7.服务端返回注册消息成功，客户端跳转到登录界面，修改密码完成

四.搜索好友

1.用户在客户端主界面的搜索框中输入，触发槽函数，显示搜索列表

connect(ui->search\_edit,&QLineEdit::textChanged,this,&ChatDialog::slot\_text\_changed);

2.在弹出的搜索列表中点击选择搜索类型为添加好友，触发槽函数，槽函数中向tcpmgr发出携带搜索内容的信号

connect(this, &QListWidget::itemClicked, this, &SearchList::slot\_item\_clicked);

emit TcpMgr::GetInstance()->sig\_send\_data(ReqId::ID\_SEARCH\_USER\_REQ, jsonData);

3.触发tcpmgr的写槽函数，向tcp\_chat\_server发送消息

void TcpMgr::slot\_send\_data(ReqId reqId, QByteArray dataBytes)

4.tcp\_chat\_server转到逻辑层LogicSystem的对应函数处理

void LogicSystem::SearchInfo(std::shared\_ptr<CSession> session, const short& msg\_id, const string& msg\_data)

5.逻辑函数判断是uid搜索还是名字搜索，调用对应搜索函数

6.搜索函数中均采取先在redis找，若找不到，再在mysql查找，成功查找则写回redis的查询方式，查询信息的类型为基础信息

bool b\_base = RedisMgr::GetInstance()->Get(base\_key, info\_str);

user\_info = MysqlMgr::GetInstance()->GetUser(uid);

RedisMgr::GetInstance()->Set(base\_key, redis\_root.toStyledString());

7.逻辑函数结束，触发send函数，返回结果给客户端

8.客户端tcpmgr处理函数检查结果是否正确，发出不同信号，SearchList接收并弹出相应窗口

handlers.insert(ID\_SEARCH\_USER\_RSP, [this](ReqId id, int len, QByteArray data) {....}）；

若失败，弹出提示窗口 find\_dlg = std::make\_shared<FindFailDlg>(this);

若成功但是是已添加的好友，则跳转到聊天界面 emit sig\_jump\_chat\_item(si);

若为未添加的，则弹出申请界面 find\_dlg = std::make\_shared<FindSuccessDlg>(this);

五.发出添加好友申请

1.在搜索好友成功完成的界面中，点击添加按钮，进入申请信息填写界面ApplyFriend。

2.编辑申请认证信息，填写备注名，标签等后，点击确认后，发出携带申请信息的信号给tcpmgr

3.tcp\_chat\_server转到逻辑层LogicSystem的对应函数处理

void AddFriendApply(std::shared\_ptr<CSession> session, const short& msg\_id, const std::string& msg\_data);

4.逻辑函数先将申请写入数据库,获取申请方的基本信息

MysqlMgr::GetInstance()->AddFriendApply(uid, touid);

bool b\_info = GetBaseInfo(base\_key, uid, apply\_info);

5.查询对方账号是否在同一个tcp\_chat\_server

bool b\_ip = RedisMgr::GetInstance()->Get(to\_ip\_key, to\_ip\_value);

6.1.若在，则通过对方uid查询对方所在的session，将申请信息补全后再通过该session发送申请信息

auto session = UserMgr::GetInstance()->GetSession(touid);

session->Send(return\_str, ID\_NOTIFY\_ADD\_FRIEND\_REQ);

6.2.若不在，则将申请信息补全后通过grpc发送到对方所在tcp\_chat\_server

ChatGrpcClient::GetInstance()->NotifyAddFriend(to\_ip\_value,add\_req);

6.3若对方未登录，则直接返回

7.逻辑函数结束后，服务器返回是否发送成功消息，客户端tcpmgr接收，简单处理

session->Send(return\_str, ID\_AUTH\_FRIEND\_RSP);

handlers.insert(ID\_ADD\_FRIEND\_RSP，.....);

8.2.对方服务器的grpc服务端ChatServiceImpl收到信息，对应函数先通过uid查询所在session，再发送消息

auto session = UserMgr::GetInstance()->GetSession(touid);

session->Send(return\_str, ID\_NOTIFY\_ADD\_FRIEND\_REQ);

9.对方接收到消息tcpmgr发出信号给chatdialog，后者先判断是否已有这一申请，若没有，将其写入UserMgr的\_apply\_list

UserMgr::GetInstance()->AddApplyList(std::make\_shared<ApplyInfo>(apply));

并且客户端对应部件（side\_contact\_lb，con\_user\_list）右上角显示红点，提示有新消息

10.调用void ApplyFriendPage::AddNewApply(std::shared\_ptr<AddFriendApply> apply)，将新申请写入ApplyFriendList apply\_friend\_list，以在申请列表显示，以及写入未回复的申请unordered\_map<int, ApplyFriendItem\*>unauth\_items

六.接受好友申请

1.受邀方点击side\_contact\_lb，切换至联系人列表栏

connect(ui->side\_chat\_lb, &StateWidget::clicked, this, &ChatDialog::slot\_side\_chat);

2.受邀方点击ApplyFriendItem里的addBtn后，弹出好友信息编辑画面AuthenFriend

emit this->sig\_auth\_friend(\_apply\_info);

3.受邀方在AuthenFriend中编辑好好友信息后，点击确认，向tcpmgr发送信号，tcpmgr发送消息给服务器

emit TcpMgr::GetInstance()->sig\_send\_data(ReqId::ID\_AUTH\_FRIEND\_REQ, jsonData);

4.服务端对应逻辑函数void LogicSystem::AuthFriendApply

解析json后先获取邀请方基本信息，并写入返回消息体 rtvalue

bool b\_info = GetBaseInfo(base\_key, touid, user\_info);

5.修改记录在数据库的好友申请条目，将条目状态修改为1(同意)

MysqlMgr::GetInstance()->AuthFriendApply(uid, touid);其中主要执行

(con->\_con->prepareStatement("UPDATE friend\_apply SET status = 1 ""WHERE from\_uid = ? AND to\_uid = ?")

6.更新数据库好友列表，添加好友

MysqlMgr::GetInstance()->AddFriend(uid, touid,back\_name);其中主要执行

添加邀请方为受邀方的好友

"INSERT IGNORE INTO friend(self\_id, friend\_id, back) ""VALUES (?, ?, ?)"

从好友申请表查询邀请方给受邀方的昵称

"SELECT back FROM friend\_apply ""WHERE from\_uid = ? AND to\_uid = ?"

添加受邀方为邀请方的好友

"INSERT IGNORE INTO friend(self\_id, friend\_id, back) ""VALUES (?, ?, ?)"

添加新的好友会话，用于快速查询聊天记录，记录会话属性

"INSERT IGNORE INTO chat\_conversations ("

"user\_a\_id, user\_b\_id, last\_msg\_id, update\_time"

") VALUES (?, ?, 0, NOW())"

7.查询邀请方所在chat\_tcp\_server

bool b\_ip = RedisMgr::GetInstance()->Get(to\_ip\_key, to\_ip\_value);

8.不在则通过grpc向对应服务器发送，再完成操作，

ChatGrpcClient::GetInstance()->NotifyAuthFriend(to\_ip\_value, auth\_req);

9.在则先查询所在session，再发送对方同意的消息

auto session = UserMgr::GetInstance()->GetSession(touid);

session->Send(return\_str, ID\_NOTIFY\_AUTH\_FRIEND\_REQ);

10.邀请方客户端tcpmgr收到消息，发出携带基本信息的信号

emit sig\_add\_auth\_friend(auth\_info);

11.ChatDialog接收信号，先检验是否已经是好友，再将其存入聊天列表

connect(TcpMgr::GetInstance().get(), &TcpMgr::sig\_add\_auth\_friend, this, &ChatDialog::slot\_add\_auth\_friend);

auto bfriend = UserMgr::GetInstance()->CheckFriendById(auth\_info->\_uid);

UserMgr::GetInstance()->AddFriend(auth\_info);

12.创建新ChatUserWid，将其放入chat\_user\_list显示在好友列表栏，及存入\_chat\_items\_added

auto\* chat\_user\_wid = new ChatUserWid();

ui->chat\_user\_list->setItemWidget(item, chat\_user\_wid);

\_chat\_items\_added.insert(auth\_info->\_uid, item);

13.同时ContactUserList收到消息，同样先判断是否为好友再，将新好友添加人联系人列表栏

connect(TcpMgr::GetInstance().get(), &TcpMgr::sig\_add\_auth\_friend,this,&ContactUserList::slot\_add\_auth\_firend);

bool isFriend = UserMgr::GetInstance()->CheckFriendById(auth\_info->\_uid);

auto \*con\_user\_wid = new ConUserItem();

this->setItemWidget(item, con\_user\_wid);

邀请方客户端响应完成

14.服务端逻辑函数结束后，服务器向受邀方返回是否成功同意对方发出的申请

session->Send(return\_str, ID\_AUTH\_FRIEND\_RSP);

受邀方客户端tcpmgr接收，并发出信号

handlers.insert(ID\_AUTH\_FRIEND\_RSP，.....);

emit sig\_auth\_rsp(rsp);

15.ApplyFriendPage收到信号，将对应申请条目上的红点删除

connect(TcpMgr::GetInstance().get(), &TcpMgr::sig\_auth\_rsp, this, &ApplyFriendPage::slot\_auth\_rsp);

find\_iter->second->ShowAddBtn(false);

16.ChatDialog收到信号，同样先判断是否为好友，再添加到聊天列表

connect(TcpMgr::GetInstance().get(), &TcpMgr::sig\_auth\_rsp, this,&ChatDialog::slot\_auth\_rsp);

auto bfriend = UserMgr::GetInstance()->CheckFriendById(auth\_rsp->\_uid);

ui->chat\_user\_list->setItemWidget(item, chat\_user\_wid);

17.同时ContactUserList收到消息，同样先判断是否为好友再，将新好友添加人联系人列表栏

connect(TcpMgr::GetInstance().get(), &TcpMgr::sig\_auth\_rsp,this,&ContactUserList::slot\_auth\_rsp);

bool isFriend = UserMgr::GetInstance()->CheckFriendById(auth\_info->\_uid);

auto \*con\_user\_wid = new ConUserItem();

this->setItemWidget(item, con\_user\_wid);

受邀方客户端响应完成

七.消息发送

1.发送方在位于ChatDialog里的Chatpage点击发送后，读取messagetextedit里的消息，

再根据消息大小分段或直接通过tcpmgr发送，消息包括fromuid，touid，消息体，消息体又包括消息id，消息内容

connect(ui->chatEdit,&MessageTextEdit::send,this,&ChatPage::on\_send\_btn\_clicked);

emit TcpMgr::GetInstance()->sig\_send\_data(ReqId::ID\_TEXT\_CHAT\_MSG\_REQ, jsonData);

2.服务端对应逻辑函数将消息存入mysql

MysqlMgr::GetInstance()->AddMessage(uid, touid, arrays.toStyledString());

3查询redis 查找touid对应的server ip

bool b\_ip = RedisMgr::GetInstance()->Get(to\_ip\_key, to\_ip\_value);

4.若在同一tcp\_chat\_server下，直接查询其所在session发送，否则通过grpc转交给对应tcp\_chat\_server发送

auto session = UserMgr::GetInstance()->GetSession(touid);

ChatGrpcClient::GetInstance()->NotifyTextChatMsg(to\_ip\_value, text\_msg\_req, rtvalue);

5.逻辑函数结束，发送方收到是否成功的回包 //ui设置送达等标记 todo...

6.接收方收到消息后，解析json打包成TextChatMsg类，发出信号携带其的信号

auto msg\_ptr = std::make\_shared<TextChatMsg>(jsonObj["fromuid"].toInt(),

sonObj["touid"].toInt(),jsonObj["text\_array"].toArray());

emit sig\_text\_chat\_msg(msg\_ptr);

7.ChatDialog接收

connect(TcpMgr::GetInstance().get(), &TcpMgr::sig\_text\_chat\_msg, this, &ChatDialog::slot\_text\_chat\_msg);

8.其中，先从chat\_user\_list中查找uid所对应的chat\_item

chat\_wid->updateLastMsg(msg->\_chat\_msgs);

9.再更新聊天页面chat\_page显示的聊天记录，若当前打开的不是对方，则不变

UpdateChatMsg(msg->\_chat\_msgs);

10.若找不到对应的chat\_item，则从联系人列表里查找，并创建新的chat\_item添加进chat\_user\_list

auto fi\_ptr = UserMgr::GetInstance()->GetFriendById(msg->\_from\_uid);

\_chat\_items\_added.insert(msg->\_from\_uid, item);

**第七部分：客户端设计**

自定义的一些控件类

1.class ClickedBtn : public QPushButton

(1)重定义这些函数，使鼠标点击和触碰按钮时，按钮可以根据qss的设置显示不同的图案

virtual void enterEvent(QEnterEvent \*event)override;

virtual void leaveEvent(QEvent \*event)override;

virtual void mousePressEvent(QMouseEvent \*event)override;

virtual void mouseReleaseEvent(QMouseEvent \*event)override;

// 中间部分

2.class CustomizeEdit : public QLineEdit

主要作用，作为搜索框

(1)重写void focusOutEvent(QFocusEvent \*event) override，发出失去焦点的信号给外部

(2)添加输入长度限制的方法，便于搜索结果

3.class SearchList : public QListWidget

主要作用，在主界面以列表显示由搜索框search\_edit的输入搜索到的用户

(1)重写bool eventFilter(QObject \*watched, QEvent \*event) override；根据鼠标位置显示滚动条，跳转滚动幅度

(2)是主界面窗口ChatDialog的一个成员

(3)成员class LoadingDlg : public QDialog 显示等待搜索结果的.gif 画面

4.class AddUserItem : public ListItemBase

主要作用，作为搜索结果的分类栏，添加进SearchList

5.class FindFailDlg : public QDialog

主要作用，搜索失败时，弹出的窗口，提醒用户

6.class FindSuccessDlg : public QDialog

主要作用，搜索成功时，弹出的窗口，处理接下来的添加好友操作

7.class ApplyFriend : public QDialog

主要作用，编辑申请信息的窗口，可填写备注名，标签等

// 左边部分

8.class StateWidget : public QWidget

主要作用，作为侧边导航栏按钮，用于切换会话列表栏，联系人列表栏等不同功能区，

(1)添加右上角红点显示功能，用于提示用户有新消息出现

(2)重写了鼠标焦点出入，松按函数，使之在不同状态显示不同样式

(3)重写了鼠标松开函数，使之能像按钮类，发出按下信号

// 右边部分

9.class ApplyFriendPage : public QWidget

主要作用，保存好友申请信息，并通过ApplyFriendList显示好友申请列表

(1)unordered\_map<int, ApplyFriendItem\*>\_unauth\_items保存好友申请信息

10.class ApplyFriendList: public QListWidget

主要作用，存储显示好友申请信息的ApplyFriendItem

(1)重写bool eventFilter(QObject \*watched, QEvent \*event) override

11.class ApplyFriendItem : public ListItemBase

主要功能，作为ListItemBase记录在ApplyFriendList，显示好友申请信息

(1)void ApplyFriendPage::loadApplyList()；在登录时读取usermgr里保存的好友申请列表

(2)void AddNewApply(std::shared\_ptr<AddFriendApply> apply);添加新来的好友申请

12.class AuthenFriend : public QDialog

主要功能，好友信息编辑画面，点击对应ApplyFriendItem的按钮后弹出

(1)void SlotApplySure();确认回调，将同意信息打包发给tcpmgr

13.class ChatPage : public QWidget

主要作用，作为聊天框口，显示好友间发送的消息，及输入和发送用户消息

(1)void ChatPage::on\_send\_btn\_clicked()，点击发送后触发，

读取保存在MessageTextEdit \* chatEdit里的信息，发送生成消息唯一消息id，发出发送信号给tcpmgr

调用ui->chat\_data\_list->appendChatItem(pChatItem);刷新本地消息列表

(2)发出emit sig\_append\_send\_chat\_msg(txt\_msg);触发Chatdialog的slot\_append\_send\_chat\_msg其中，

将发送的消息存储到Chatdialog下的chat\_user\_list对应的ChatUserWid

(通过QMap<int,QListWidgetItem\*>\_chat\_items\_added 存储的item与uid的关系查找)更新聊天窗口的消息，

再将消息放入UserMgr的用户的好友信息里

(3)void ChatPage::SetUserInfo(std::shared\_ptr<UserInfo> user\_info)，读取Chatdialog发来的好友信息，修改界面，

调用chat\_data\_list->removeAllItem清除与上一个聊天对象的聊天记录，

调用AppendChatMsg，读取与该好友的聊天记录

(4)void ChatPage::AppendChatMsg(std::shared\_ptr<TextChatData> msg)读取与该好友的聊天记录，

设置消息条目ChatItemBase\* pChatItem = new ChatItemBase(role);

设置消息气泡pBubble = new TextBubble(role, msg->\_msg\_content);

将气泡放入pChatItem->setWidget(pBubble);

14.class MessageTextEdit : public QTextEdit

主要作用，位于chatpage中，作为输入框，供用户输入消息

(1)重写了void MessageTextEdit::dragEnterEvent(QDragEnterEvent \*event) ，筛选可拖入的文件类型

(2)重写了void MessageTextEdit::dropEvent(QDropEvent \*event)，处理拖入的mimeData

(3)QStringList MessageTextEdit::getUrl(QString text)，从source->text()获取纯净的文件路径

(4)void MessageTextEdit::insertMsgList(QVector<MsgInfo> &list, QString flag, QString text, QPixmap pix)

将文件类型，文件路径，预览图组成的MsgInfo写入对应消息表

(5)void MessageTextEdit::insertImages(const QString &url)，处理图片文件，主要将图片缩放成预览图，写入mMsgList

(6)void MessageTextEdit::insertTextFile(const QString &url)，处理其他文件，主要将其他文件写入mMsgList

(7)QPixmap MessageTextEdit::getFileIconPixmap(const QString &url)，将其他文件的主要信息提取，生成预览图

15.class ChatView:public QWidget

主要作用，位于chatpage中，作为显示聊天记录的列表框

(1)使用QScrollArea \*m\_pScrollArea来保存每一条消息，并实现滚轮功能

(2)void ChatView::appendChatItem(QWidget \*item)消息从尾部插入

(3)void ChatView::prependChatItem(QWidget \*item)消息从头部插入，暂未实现

(4)void ChatView::insertChatItem(QWidget \*before, QWidget \*item)消息从中部插入，暂未实现

(5)void ChatView::removeAllItem()清空所有消息

(6)实现细节，使用一个权重极大的widget将添加到m\_pScrollArea中，使得其他聊天消息被顶到上部

16.class ChatItemBase : public QWidget

作用作用，作为一条消息的布局，最终在chatpage中显示

(1)采用网格布局，组合显示头像，名字，消息气泡，通过m\_role记录是否为自己的消息，构造具体布局

(2)void ChatItemBase::setUserName(const QString &name)设置名字

(3)void ChatItemBase::setUserIcon(const QPixmap &icon)设置头像

(4)void ChatItemBase::setWidget(QWidget \*w)设置聊天气泡，通过替换在构造函数提前设置在布局中的widget实现

pGLayout->replaceWidget(m\_pBubble, w);

17.class BubbleFrame : public QFrame

主要作用，美化消息气泡，重写void paintEvent(QPaintEvent \*e)override;画出圆角方形气泡及连接头像和消息的小三角形

18.class TextBubble : public BubbleFrame

主要作用，将文本信息加入消息气泡，添加设定宽高及行数的函数，显示在消息气泡上，使之更美观

19.class ChatUserList: public QListWidget

主要作用，在主界面以列表显示会话列表栏

(1)重写bool eventFilter(QObject \*watched, QEvent \*event) override；根据鼠标位置显示滚动条，跳转滚动幅度

并且可以在滚动到底部时，发送信号通知聊天界面加载更多聊天内容

(2)是主界面窗口ChatDialog的一个成员

20.class ChatUserWid : public ListItemBase

主要作用，作为会话列表栏，添加进ChatUserList

(1)添加根据好友信息会话栏界面的方法

(2)提供实时改变界面方法

(3)std::shared\_ptr<UserInfo>\_user\_info，保存了聊天对象的基本信息，及存储聊天记录

std::vector<std::shared\_ptr<TextChatData>> \_chat\_msgs;

根窗口mainwindow维护四个窗口

LoginDialog\*\_login\_dlg; // 1.登录窗口

RegistDialog\* \_regist\_dlg;// 2.注册窗口

ResetDialog\* \_reset\_dlg; // 3.重置密码窗口

ChatDialog\* \_chat\_dlg; // 4.主界面窗口

初始为1窗口，可按钮切换2 3窗口，登录成功后，切换至4窗口

class ChatDialog : public QDialog

(1)在void ChatDialog::ShowSearch(bool bsearch)中，通过bsearch和ChatUIMode改变列表栏显示的内容

(2)重写bool eventFilter(QObject\*watched,QEvent\* event)override;添加鼠标点击时位置检测，检测是否点击

(3)右边部分QStackedWidget stackedWidget 下有三个同时仅显示一个窗口

(4)\_cur\_chat\_uid 记录当前聊天对向的uid

(5)connect(ui->chat\_user\_list, &QListWidget::itemClicked, this, &ChatDialog::slot\_item\_clicked);

通过点击聊天列表切换会话对象

(6)void ChatDialog::slot\_append\_send\_chat\_msg(std::shared\_ptr<TextChatData> msgdata)

将发送的消息存储到Chatdialog下的chat\_user\_list对应的ChatUserWid

(通过QMap<int,QListWidgetItem\*>\_chat\_items\_added 存储的item与uid的关系查找)更新聊天窗口的消息，

再将消息放入UserMgr的用户的好友信息里

**第八部分：提升与改进**

1.服务端添加心跳检查，避免僵尸连接。

2.添加日志系统

3.线程池获取资源可改为无锁队列，提高并发

4.优化好友搜索功能，使之可以使用多种信息搜索，及模糊搜索

5.改进mysql表结构及查询方式

6.添加分布式锁，避免同一账号在多处登录