目录

[AcountManage 2](#_Toc38215599)

[DataBase 2](#_Toc38215600)

[Ihandler 2](#_Toc38215601)

[MainWindow 2](#_Toc38215602)

[NewUserVector 2](#_Toc38215603)

[Protocol 2](#_Toc38215604)

[Room 2](#_Toc38215605)

[RoomManage 2](#_Toc38215606)

[RoomVector 2](#_Toc38215607)

[SocketHandle 2](#_Toc38215608)

[StrategyHandle 2](#_Toc38215609)

[User 2](#_Toc38215610)

[UserHandle 2](#_Toc38215611)

# AcountManage

# DataBase

# Ihandler

virtual void *Handle*(QTcpSocket \*socket ,const Protocol &p) = 0;//纯虚函数

被AcountManage和RoomManage两个类继承

# MainWindow

负责连接Tcp

将所有的登录账户压入一个容器里面QVector<SocketHandle\*> sockets;（//保存所有客户端套接字信息的动态数组）

# NewUserVector

这是个单例，只有Tcp程序不退出运行，就源源不断地将所有账户压入该类的私有变量QVector<SocketHandle\*> sockets;中

这个类存在的意义：当主播创建直播间或者主播退出直播间时候，所有登录进来的账户，在客厅左边都要刷新已存在的主播房间名

# Protocol

# Room

在创建主播房间时候，总是：

Room host\_room(hostname,name,socket,ip,camera\_port,audio\_port);

New 一个主播房间实例

然后压入QVector<Room>动态数组里面

# RoomManage

# RoomVector

将QVector<Room> chat\_room;定义成私有成员

在主播创建房间时，通过

bool RoomVector::CheckRoom(const Room &room) {

QVector<Room>::iterator it;

for(it = chat\_room.begin(); it!=chat\_room.end(); it++) {

//确保一个客户端只能创建一个直播间,并且不能和已存在的房间重名

if(room.GetHostRoomName() == it->GetHostRoomName() || room.GetSocket() == it->GetSocket()) {//判断条件就两条

return true;

}

}

return false;//此处与其他不同，当为假的时候才能创建房间

}

确保一个主播只能创建一个直播间，并且还不能与已存在的直播间重名

在关闭直播间时候，判断是否是主播关闭的：

当判断是主播退出房间之后

要让在这个直播间所有的观众返回到大厅

当主播退出该直播间后，将该直播间从Qvec<Room>容器里面删除

所有已登录账户的大厅刷新现存在的直播间

//====关闭直播账户====//

void RoomManage::CloseHostRoom(QTcpSocket \*socket, const Protocol &p) {

QString host\_name = p["host\_name"].toString();//房间名

QString user\_name = p["user\_name"].toString();//账户名

Protocol pRet(Protocol::CloseRoom);//------------------------------

Protocol pUse(Protocol::UserLset);

qDebug() <<"服务器接收到房间名/账户名:" <<host\_name<<user\_name;

RoomVector \*rv = RoomVector::GetInstance();

QVector<Room>& chatrooms = rv->GetAllChat();

for(QVector<Room>::iterator it = chatrooms.begin(); it!=chatrooms.end(); it++) {

if(it->GetHostRoomName() == host\_name) {//找到主播房间名

QVector<Account\_t>& per = it->GetAudience();//找到这个主播房间下对应的所有观众（放在容器里面）

//判断是否为主播关闭窗口：QVector<Room>存了所有关于主播房间的类，找到这个含有房间名/账户名的类，与传过来的user\_name 对比

if(it->GetHostPreson() == user\_name) {

for(QVector<Account\_t>::iterator ip = per.begin(); ip!= per.end(); ip++) {

//下面两行的意思是(要关闭的主播房间)里面所的观众窗口全部退出观看窗口，返回到大厅

pRet["result"] = "CloseHostRoomTrue";

ip->socket->write(pRet.pack());//遍历，所有游客关闭窗口

this->AccountRefresh(ip->socket,ip->name);

}

//防止直播间只有主播自己的情况，无法进入上述for循环，则直接读到如下语句

pRet["result"] = "CloseHostRoomTrue";

it->GetSocket()->write(pRet.pack());

this->AccountRefresh(it->GetSocket(),it->GetHostPreson());

//当主播退出该直播房间后，将该直播房间从QVector<Room>容器中删除

rv->EraseRoom(\*it);

//主播房间关闭成功，直播间大厅自动刷新已存在的主播间

NewUserVector\* instance = NewUserVector::GetInstance();

QVector<SocketHandle\*>& sh = instance->GetAllSockets();

for(QVector<SocketHandle\*>::iterator it = sh.begin(); it != sh.end(); it++) {

QTcpSocket\* socketTmp = (\*it)->GetSocket();

this->RoomListHandle(socketTmp,p);

}

break;

} else {

}

}

}

return ;

}

# SocketHandle

# StrategyHandle

# User

# UserHandle

总结：RoomListUi.cpp和HostRoomUi.cpp要用ClientHandle 作为打包中转站

因为：

1. 统一解压包，便于管理
2. 例如游客退出主播房间时，必须在ClientHandle打包时带入当前账户的账户名

不然在服务器怎么判定是主播退出账户，还是游客退出账户？

1. 传递的包p[“ result ”]或者设置pRet[“ result ”]的类型不同，想要去做的事则不同

由QVector<Account\_t>返回的包是游客的

由QVector<Room>返回的包是主播的

总结一下 游客加入主播房间到离开主播房间的整个过程

一 游客：

connect(ui\_->listWidget,SIGNAL(itemDoubleClicked(QListWidgetItem\*)),this,SLOT(onDoubleClickedHostRoom(QListWidgetItem\*)));//双击加入主播房间(注意有参数传递)

二 ：

void RoomListUi::onDoubleClickedHostRoom(QListWidgetItem \*item) {

QString str = item->text();

emit sigJoinHostRoom(str);

}

三 void ClientHandle::onJoinHostRoom(QString host\_room\_name) {

QString name = ui\_->lineName->text();

Protocol p(Protocol::JoinRoom);

p["host\_room\_name"] = host\_room\_name;

p["user\_name"] = name;

qDebug()<<"客户端双击传递的房间名："<<host\_room\_name;

this->tcp\_socket->write(p.pack());

}

四 服务器

void RoomManage::JoinHostRoom(QTcpSocket \*socket, const Protocol &p) {

DataBase \*instance = DataBase::GetInstance();

instance->CreatConnection();

double money = 0.0f;

int level = 0;

QString host\_room\_name = p["host\_room\_name"].toString();

QString name = p["user\_name"].toString();

QSqlQuery query;

query.prepare("select \* from tb\_user where user\_name = :user\_name");

query.bindValue(":user\_name", name);

bool ret = query.exec();

if(!ret) {

qDebug()<<query.lastError().text();

} else {

while(query.next()) {

money = query.value(4).toDouble();

level = query.value(5).toInt();

}

}

qDebug()<<"服务器查询到金额/积分"<<money<<level;

instance->RemoveConnection();

RoomVector \*room\_vector = RoomVector::GetInstance();

QVector<Room> chatrooms = room\_vector->GetAllChat();

Protocol pRet(Protocol::JoinRoom);

QVector<Room>::iterator it;

for(it = chatrooms.begin(); it != chatrooms.end(); it++) {

if(host\_room\_name == it->GetHostRoomName()) {

//此循环来防止重复加入

QVector<Account\_t> audience = it->GetAudience();

for(QVector<Account\_t>::iterator ip = audience.begin(); ip != audience.end(); ip++) {

//遍历该房间的观众容器，如果容器里有名字与要加入的游客名字相同，则退出加入

if(ip->name == name) {

qDebug()<<"成员已经存在";

return;

}

}

Account\_t visitor = {name,socket};//创建结构体，用来存放游客

room\_vector->InstertVisitor(host\_room\_name, visitor);//把游客加到对应的主播房间的游客容器里面去

pRet["result"] = "JoinHostRoomTrue";

pRet["host\_room\_name"] = host\_room\_name;

pRet["user\_name"] = it->GetHostPreson();

pRet["audience"] = name;

pRet["money"] = money; //游客账户余额

pRet["level"] = level;

pRet["address"] = it->GetAddress();

pRet["camera\_port"] = it->GetCameraPort();

pRet["audio\_port"] = it->GetAudioPort();

socket->write(pRet.pack());

}

}

//当加房间后，自动刷新直播间群成员列表

this->VisitorBarRefresh(socket, p);

return;

}

五 服务器

当游客加入主播房间后 主播和加入的游客房间列表都刷新

void RoomManage::VisitorBarRefresh(QTcpSocket \*socket, const Protocol &p) {

QString host\_room\_name = p["host\_room\_name"].toString();

Protocol pRet(Protocol::UserLset);

RoomVector \*room\_vector = RoomVector::GetInstance();

QVector<Room> chatrooms = room\_vector->GetAllChat();

int count = 1;

//在所有封装主播房间号的动态数组里找到该主播号

for(QVector<Room>::iterator it = chatrooms.begin(); it != chatrooms.end(); it++) {

if(host\_room\_name == it->GetHostRoomName() || socket == it->GetSocket()) {

QVector<Account\_t> users = it->GetAudience();

qDebug() << "刷新群成员列表";

//遍历该主播间放有全部观众的容器，每遍历出一个观众，就存到一个返回包里面

for(QVector<Account\_t>::iterator ia = users.begin(); ia != users.end(); ia++) {

pRet[QString::number(count)] = ia->name;

qDebug()<< ia->name<<";";

count++;

}

pRet["result"] = "UserListTrue";

//游客群成员(每个人)页面的列表刷新

for(QVector<Account\_t>::iterator ia = users.begin(); ia != users.end(); ia++) {

ia->socket->write(pRet.pack());

}

// //主播群成员页面的列表刷新

it->GetSocket()->write(pRet.pack());

}

}

return;

}

六 游客

case Protocol::JoinRoom:

if(p["result"].toString() == "JoinHostRoomTrue") {

room\_ui\_->hide();

host\_ui\_->show();

host\_ui\_->UpdateVisitorRoomInformation(p);

} else if(p["result"].toString() == "JoinRoomFalse"){

QMessageBox::critical(this, "直播房间", "加入失败");

} else {

QMessageBox::critical(this, "直播房间", "未知错误");

}

break;

七

//更新游客的房间信息

void HostRoomUi::UpdateVisitorRoomInformation(Protocol p) {

QString host\_name = p["user\_name"].toString();

QString host\_room\_name = p["host\_room\_name"].toString();

QString audience = p["audience"].toString();

double money = p["money"].toDouble();

int level = p["level"].toInt();

QString address = p["address"].toString();

int camera\_port = p["camera\_port"].toInt();

int audio\_port = p["audio\_port"].toInt();

/\*设置开启视频和音频的IP,视频的端口号，音频的端口号\*/

this->SetAddress(address, camera\_port, audio\_port);

/\*把money和lever转化成QString类型为显示在HostRoomUi.ui页面\*/

QString qmoney = QString::number(money,'f',2);

QString qlevel = QString::number(level,10);

qDebug()<<"客户端接收到金额/积分"<<qmoney<<qlevel;

/\*实现money和lever显示在游客房间HostRoomUi.ui页面\*/

VisitorRoomShow(host\_room\_name, qmoney, qlevel, host\_name,audience);

/\*设置窗口标题\*/

this->setWindowTitle("\*\*\*\*\*\*" + host\_room\_name + "直播间(观众窗口)\*\*\*\*\*\*");

}

游客关闭直播间

八 游客

//===关闭直播间===//

void HostRoomUi::onCloseHostRoom() {

QString host\_name = ui\_->le\_RoomName->text();

emit sigCloseHostRoom(host\_name);

}

九

//======主播或者游客关闭房间======//

void ClientHandle::onCloseHostRoom(QString str) {

QString name = ui\_->lineName->text();

Protocol p(Protocol::CloseRoom);

p["host\_name"] = str;

p["user\_name"] = name;

qDebug() <<"客户端发出房间名/账户名:" << str << name;

this->tcp\_socket->write(p.pack());

}

十 服务器 要判断是游客退出直播间（else后面执行的内容）

//====关闭直播账户====//

void RoomManage::CloseHostRoom(QTcpSocket \*socket, const Protocol &p) {

QString host\_name = p["host\_name"].toString();//房间名

QString user\_name = p["user\_name"].toString();//账户名

Protocol pRet(Protocol::CloseRoom);//------------------------------

Protocol pUse(Protocol::UserLset);

qDebug() <<"服务器接收到房间名/账户名:" <<host\_name<<user\_name;

RoomVector \*rv = RoomVector::GetInstance();

QVector<Room>& chatrooms = rv->GetAllChat();

for(QVector<Room>::iterator it = chatrooms.begin(); it!=chatrooms.end(); it++) {

if(it->GetHostRoomName() == host\_name) {//找到主播房间名

QVector<Account\_t>& per = it->GetAudience();//找到这个主播房间下对应的所有观众（放在容器里面）

//判断是否为主播关闭窗口：QVector<Room>存了所有关于主播房间的类，找到这个含有房间名/账户名的类，与传过来的user\_name 对比

if(it->GetHostPreson() == user\_name) {

for(QVector<Account\_t>::iterator ip = per.begin(); ip!= per.end(); ip++) {

//下面两行的意思是(要关闭的主播房间)里面所的观众窗口全部退出观看窗口，返回到大厅

pRet["result"] = "CloseHostRoomTrue";

ip->socket->write(pRet.pack());//遍历，所有游客关闭窗口

this->AccountRefresh(ip->socket,ip->name);

}

//防止直播间只有主播自己的情况，无法进入上述for循环，则直接读到如下语句

pRet["result"] = "CloseHostRoomTrue";

it->GetSocket()->write(pRet.pack());

this->AccountRefresh(it->GetSocket(),it->GetHostPreson());

//当主播退出该直播房间后，将该直播房间从QVector<Room>容器中删除

rv->EraseRoom(\*it);

//主播房间关闭成功，直播间大厅自动刷新已存在的主播间

NewUserVector\* instance = NewUserVector::GetInstance();

QVector<SocketHandle\*>& sh = instance->GetAllSockets();

for(QVector<SocketHandle\*>::iterator it = sh.begin(); it != sh.end(); it++) {

QTcpSocket\* socketTmp = (\*it)->GetSocket();

this->RoomListHandle(socketTmp,p);

}

break;

} else {

//确定是游客退出直播间，遍历该主播房间里面所有的游客

for(QVector<Account\_t>::iterator ip = per.begin(); ip != per.end(); ip++) {

if(ip->socket == socket) {

pRet["result"] = "CloseHostRoomTrue";

socket->write(pRet.pack()); //返回信号，关闭游客窗口

this->AccountRefresh(socket,user\_name); //这是说通过遍历数据库，把最新的账户剩余多少钱传到大厅

per.erase(ip); //从该直播间的"游客动态数组"中删除成员

this->RoomListHandle(socket,p); //退出直播间的游客返回大厅时候自动刷新左边栏的主播房间

//下面几行都是将刷新该直播间还有多有游客，存到返回包里面

int count = 1;

for(QVector<Account\_t>::iterator ia = per.begin(); ia != per.end(); ia++) {

pUse[QString::number(count)] = ia->name;

count++;

}

pUse["result"] = "UserListTrue";

for(QVector<Account\_t>::iterator ia = per.begin(); ia != per.end(); ia++) {

qDebug()<<"关闭游客窗口，群成员列表："<<ia->name;

//主播房间的所有游客，他们的ui界面上"群成员列表"刷新

ia->socket->write(pUse.pack());

}

//主播房间的"群成员列表"刷新

it->GetSocket()->write(pUse.pack());

break;

}

}

}

}

}

return ;

}

11 游客

case Protocol::CloseRoom:

if(p["result"].toString() == "CloseHostRoomTrue") {

host\_ui\_->close();

room\_ui\_->show();

}else if(p["result"].toString() == "CloseHostRoomFalse"){

QMessageBox::critical(this, "关闭直播间", "关闭失败");

}else{

QMessageBox::critical(this, "关闭直播间", "未知错误");

}