视频会议客户端

项目解读

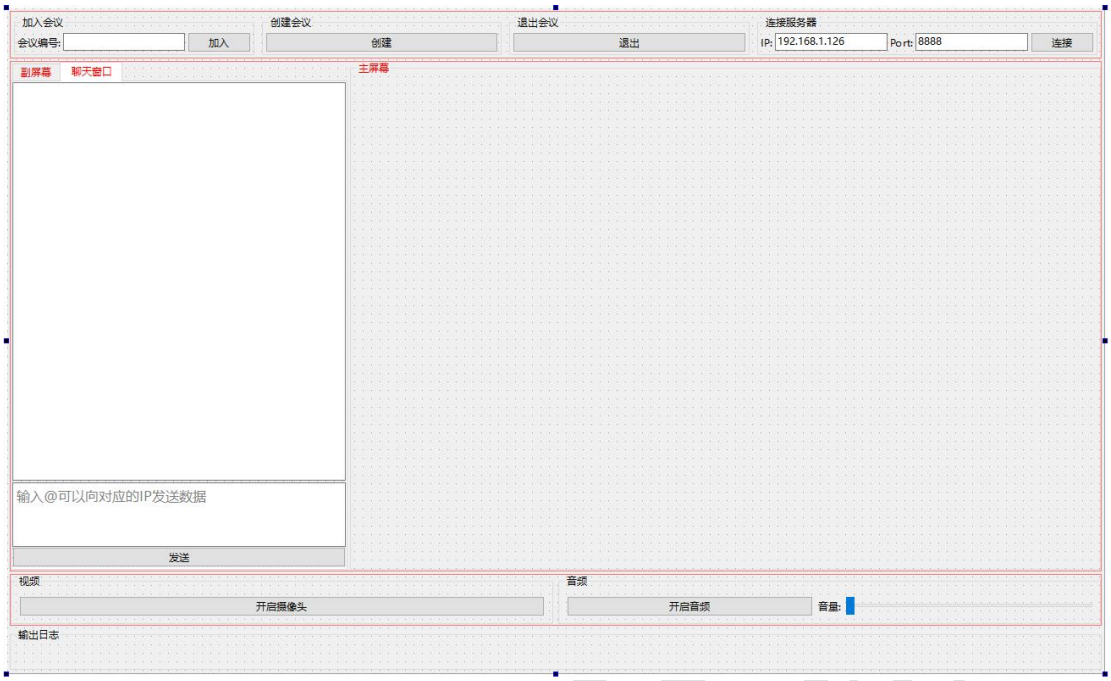
项目代码结构

源码文件

AudioInput.h AudioInput.cpp	录音
AudioOutput.h AudioOutput.cpp	播放（声音）
chatmessage.h chatmessage.cpp	聊天信息
logqueue.h logqueue.cpp	日志队列
main.cpp	入口函数
mytcpsocket.h mytcpsocket.cpp	网络通信
mytextedit.h mytextedit.cpp	文本编辑控件
myvideosurface.h myvideosurface.cpp	视频显示控件
netheader.h netheader.cpp	网络包模块
partner.h partner.cpp	房间信息
recvsolve.h recvsolve.cpp	数据接收模块
screen.h screen.cpp	屏幕工具模块
sendimg.h sendimg.cpp	图片发送
sendtext.h sendtext.cpp	文字发送
widget.h widget.cpp	主窗口

资源文件

widget.ui	界面布局



resource.qrc：

1.jpg 默认会议头像

3.gif 等待画面

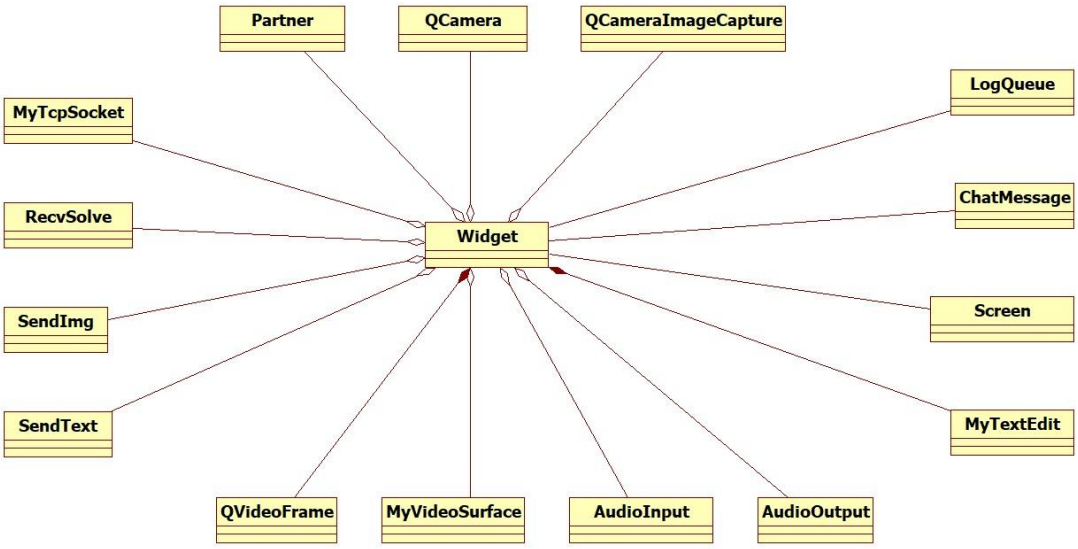
2.wav 用户上线音效

项目架构

本项目整体属于典型的 CS 架构（客户端——服务器架构）中的客户端。

客户端采用 qt 的信号和槽机制来实现前端展示和后台数据交互的对接。

项目类关系



项目类功能

AudioInput	录音
AudioOutput	播放声音
QCamera	摄像头
QCameraImageCapture	截屏
ChatMessage	聊天消息
LogQueue	日志队列
MyTcpSocket	网络通信
MyTextEdit	文字输入窗框
MyVideoSurface	视频播放组件
Partner	房间信息
RecvSolve	数据接收模块
Screen	屏幕信息
SendImg	图片发送模块
SendText	文本发送模块
Widget	主窗口

项目流程

应用初始化

入口函数 main→
屏幕初始化 Screen::init()→

窗口初始化 Widget::Widget(QWidget *parent)→

开启日志 LogQueue
控件初始化
视频传输初始化
网络传输初始化
文字信息传输初始化
摄像头初始化
接收处理初始化
音频初始化

消息发送

用户点击发送按钮→
on_sendmsg_clicked 响应→
发送 PushText 信号→
SendText::push_Text 响应→
消息入 textqueue 队列→
SendText::run(线程函数)响应→
加入 queue_send 队列→
MyTcpSocket::run(线程函数)响应→
MyTcpSocket::sendData 响应→
QTcpSocket::write 最终执行发送

视频发送

Widget::cameraImageCapture 捕获摄像头帧画面→
发射信号 pushImg→
SendImg::ImageCapture→
pushToQueue 入队列→
SendImg::run 出队列→
入队列 queue_send→
MyTcpSocket::run(线程函数)响应→
MyTcpSocket::sendData 响应→
QTcpSocket::write 最终执行发送

接收数据

信号 readyRead→
MyTcpSocket::recvFromSocket→
queue_recv 入队列→
线程函数 RecvSolve::run 响应→

queue_recv 出队列→
发射信号 datarecv→
Widget::datasolve 响应→
消息处理

项目技术栈

多线程技术

利用多线程来处理网络收发、视频帧传输、消息发送、消息接收等等过程。
其中视频帧传输属于高频触发内容，使用线程避免其对 UI 线程的过度占用。
而消息发送和消息接收从网络收发线程中剥离出来，这样让消息处理过程不至于干扰网络收发过程。网络收发只高效处理收发队列，而对消息进行编码封包等操作可以完全不参与。这些编码和封包操作放在消息收发线程里面，按照需要自行处理。
考虑到除了视频传输消息之外，其他消息触发频率大多数都比较低，这样的形式可以提高网络线程的利用效率。即多个消息并行处理，最后生成发送数据队列，交给网络线程全力发送。避免了个别消息处理的高耗时导致整个收发过程的阻塞！

网络编程

使用了 Qt 的网络编程技术。
利用信号和槽来处理网络收发。
这种形式极大的简化了网络处理过程。
使用者可以专注于数据的处理而不是网络流程的控制。

协议封装

网络包结构：

消息类型 四字节
消息数据 参考消息长度
消息长度 四字节
房间地址 四字节

消息类型：

```
IMG_SEND = 0,
IMG_RECV,
AUDIO_SEND,
AUDIO_RECV,
TEXT_SEND,
TEXT_RECV,
```

CREATE_MEETING,
EXIT_MEETING,
JOIN_MEETING,
CLOSE_CAMERA,

CREATE_MEETING_RESPONSE = 20,
PARTNER_EXIT = 21,
PARTNER_JOIN = 22,
JOIN_MEETING_RESPONSE = 23,
PARTNER_JOIN2 = 24,
RemoteHostClosedError = 40,
OtherNetError = 41

消息长度 消息数据的长度，单位字节
房间地址 标记房间的地址信息

视频会议技术

视频采用 QCamera 类来获取摄像头内容，通过 QAudioInput 来获取音频数据
将画面和声音数据同步传输出去，然后经由服务器转发，从而实现远程视频会议需求

项目问题

请简单介绍一下项目

本项目是一个基于 Qt 开发的视频会议系统的客户端项目。
该项目将多线程编程和 Qt 的信号和槽机制有机结合，不仅仅提高了网络传输效率，也有效的简化了网络功能的实现。

请问在项目开发过程中遇到了哪些问题，你是如何解决的

首先是视频传输对 UI 线程的影响问题。
因为视频数据产生频率较高（每秒 16~25 帧不等）传输数据时长也不小（依据摄像头清晰度从几十 k 到数百 k）。如果每次都使用 widget 响应摄像头消息，那么会大量占用 UI 线程。导致繁忙的时候，用户界面操作出现卡顿感。
所以我做了下面两件事情：
1 统一了消息数据格式。
2 开启了一个处理线程专门负责响应摄像头帧数据。
统一消息数据格式，使得所有需要使用网络的界面操作，都可以变成一个消息数据队列。

开启专门的线程处理这些消息请求，可以避免这些消息对用户界面线程的挤占。
这样处理之后，整个界面的流畅程度大为提升。

其次是界面操作繁多，其中不少都涉及网络操作。如果每个操作都要等待网络应答才返回，会导致程序运行卡死。
为了解决这个问题，我将网络收发分离，制定了两个队列。
同时添加了自定义信号：
当收到数据解析出数据包之后，会将解析出来的消息数据放入消息应答队列中。然后开线程专门处理这个队列，向 widget 发送处理信号
当要发出网络消息的时候，会将消息封装，然后放入消息发送队列中。
然后开线程专门处理这个队列，向网络有序发送数据包。
这样处理之后，整个收发过程开始变得有序起来。
而且发送操作无需一直阻塞。应答包到来后，会自动触达 widget 层。

请问你是如何设计该项目的（该项目的架构是如何设计的）？

要实现一个视频会议系统，最关键的是视频画面和声音数据的送达。
这样涉及音视频的内容，最好使用 CS 架构而不是 BS 架构。
因为视频或者音频数据，使用自定义的协议传输，效率可能会更高一些。
所以我在这里选择了 CS 架构。
客户端开发我选择了相对较为成熟的 Qt 技术。
现在的 Qt 技术不仅仅在跨平台方面更成熟而且低成本，在设备调用封装上也比较容易上手。
所以这块我选择了 Qt，这样研发成本低，研发效率高。
考虑到音视频服务器对内存的消耗比较大，所以服务器选择 Linux 系统。
Windows 服务器对内存的利用效率相对 Linux 而言还是要低一点的。
整体的架构我就是这样思考的。

请问你是如何实现视频会议功能的？

我这里是基于笔记本或者台式电脑的环境进行开发的。
这些设备上摄像头一般没用录音功能。
录音一般都是声卡的功能。
所以我使用摄像头来获取视频内容，使用音频设备来获取声音内容。
然后将两者同时采集并发向服务器，再向房间内的其他客户端转发这些内容。
其他客户端收到这些内容，将视频内容显示到界面，同时播放声音内容。
通过上面这些操作，最终达成视频会议功能。

本机的视频是如何同步到其他客户端的？

我们是借助一个服务器来实现视频内容同步的。
具体来说是这样的：
1 用户需要向服务器申请开启一个房间，然后获得一个房间号。
2 这个时候服务器会开启一个进程专门来服务这个房间。
3 其他用户填入房间号加入这个房间。
4 视频用户向服务器发送视频内容。
5 服务器收到视频数据后，将内容转发给房间内的其他用户。
6 其他用户收到视频内容，然后将内容呈现在界面之上。
经过这些过程后，视频将同步到其他客户端。
综上所述，服务器的 CPU 性能、带宽和内存大小将会影响整个系统的表现。

客户端如何解决卡顿的问题？

首先是分析卡顿的原因是什么。
主要是三个方面：
1 视频数据传输是否流畅
2 视频数据展示是否流畅
3 物理环境是否支持
针对上面三个方面，我依次做了下面的应对手段：
传输方面，我采用了多线程和收发分离的方式来提高流畅度
视频展示方面，我专门使用了 MyVideoSurface 类进行展示。这样主窗口的其他控件就不会干扰到画面的显示了。
物理环境方面要保证物理网络带宽、客户端运行机器的配置满足基本要求。

通信协议是如何设计的？

通信协议设计的比较简单，这样方便维护，也能提高传输的有效荷载。
由于视频会议都是有房间属性的，所以协议中每个消息都需要包含一个房间消息。
然后消息长度信息也需要配备，方便解包。
最后，消息类型要独立出来，这样可以简化处理代码。
基于上面的需要，我设计了现在的协议格式。

请结合本项目，说明一下 Qt 的信号和槽机制

Qt 的信号和槽机制非常的灵活。不仅仅可以用在界面上，也可以用在诸如网络通信和设备数据处理之中。
具体到我的项目之中，我印象最深的一点就是网络模块这里。

传统的网络功能处理，往往需要我们自己开线程来维护 recv 或者 send。
复杂一点的，比如 IOCP，比如 EPoll，需要我们自己来等待事情的发生。
然后自己写处理函数。
但是在 Qt 中，我们只需要设置好网络参数，设置好信号响应槽就行了。
比如网络有数据到达了，可以读了，我们只需要设计好相关的槽函数即可。
网络数据会以参数的形式给到我们的槽函数。
我们直接处理就行了。
这比以前的模式要方便太多了。
而且 Qt 有完善的网络错误处理异常信息。
我们不再需要自己去弄这块儿，只需要定义好如何去处理就行。
之前的模式需要我们自己全面考虑，逐个应对。
总体来讲，信号和槽机制还是有不少便利的。

请问你是如何将网络通信数据同步到客户端的（客户端是如何响应网络数据的）

首先收到网络数据后，会触发信号 readyRead
在该信号的响应槽里面，我们会调用 MyTcpSocket::recvFromSocket 函数
接着解析消息，放入 queue_recv 入队列
当队列有数据的时候，线程函数 RecvSolve::run 函数会触发。
该线程会将消息从 queue_recv 队列中取出。
接着会发射信号 datarecv 到 widget
然后在 Widget::datasolve 函数处理该消息
此时会依据消息类型进行各种处理，其中一部分处理会影响到用户界面。
上面就是整个网络通信数据同步到客户端的整个流程

请问你是如何处理用户离线问题的

用户离线我们是通过服务器为中心来处理的。
一旦用户出现网络问题或者异常离线。
首先收到通知的是服务器的房间进程。
然后服务器会通知房间内的其他客户端。
这样该用户的状态会同步到其他客户端。
如果用户这个时候恢复了，会有一个登录消息发送到服务器。
然后服务器会转发给其他客户端。
这样其他客户端又会显示该用户的新状态。