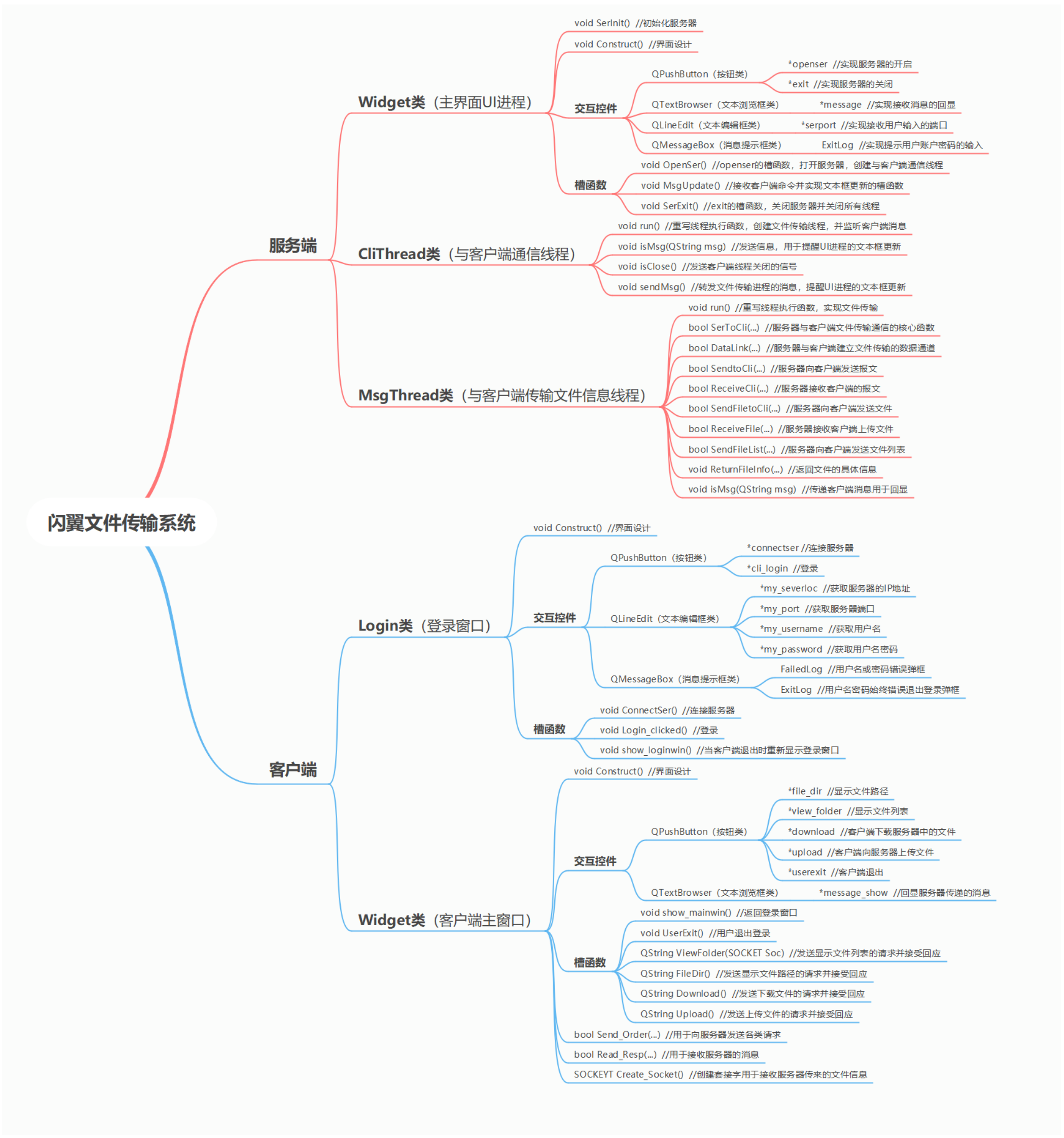
闪翼文件传输系统项目源码及其注解

项目导图



闪翼文件传输系统
服务端
Widget类（主界面UI进程）
void SerInit() //初始化服务器
void Construct() //界面设计
交互控件
QPushButton（按钮类）
*openser //实现服务器的开启
*exit //实现服务器的关闭
QTextBrowser（文本浏览框类）
*message //实现接收消息的回显
QLineEdit（文本编辑框类）
*serport //实现接收用户输入的端口
QMessageBox（消息提示框类）
ExitLog //实现提示用户账户密码的输入
槽函数
void OpenSer() //openser的槽函数，打开服务器，创建与客户端通信线程
void MsgUpdate() //接收客户端命令并实现文本框更新的槽函数
void SerExit() //exit的槽函数，关闭服务器并关闭所有线程
CliThread类（与客户端通信线程）
void run() //重写线程执行函数，创建文件传输线程，并监听客户端消息
void isMsg(QString msg) //发送信息，用于提醒UI进程的文本框更新
void isClose() //发送客户端线程关闭的信号
void sendMsg() //转发文件传输进程的消息，提醒UI进程的文本框更新
MsgThread类（与客户端传输文件信息线程）
void run() //重写线程执行函数，实现文件传输
bool SerToCli(...) //服务器与客户端文件传输通信的核心函数
bool DataLink(...) //服务器与客户端建立文件传输的数据通道
bool SendtoCli(...) //服务器向客户端发送报文
bool ReceiveCli(...) //服务器接收客户端的报文
bool SendFiletoCli(...) //服务器向客户端发送文件
bool ReceiveFile(...) //服务器接收客户端上传文件
bool SendFileList(...) //服务器向客户端发送文件列表
void ReturnFileInfo(...) //返回文件的具体信息
void isMsg(QString msg) //传递客户端消息用于回显
客户端
Login类（登录窗口）
void Construct() //界面设计
交互控件
QPushButton（按钮类）
*connectser //连接服务器
*cli_login //登录
QLineEdit（文本编辑框类）
*my_severloc //获取服务器的IP地址
*my_port //获取服务器端口
*my_username //获取用户名
*my_password //获取用户名密码
QMessageBox（消息提示框类）
FailedLog //用户名或密码错误弹框
ExitLog //用户名密码始终错误退出登录弹框
槽函数
void ConnectSer() //连接服务器
void Login_clicked() //登录
void show_loginwin() //当客户端退出时重新显示登录窗口
Widget类（客户端主窗口）
void Construct() //界面设计
交互控件
QPushButton（按钮类）
*file_dir //显示文件路径
*view_folder //显示文件列表
*download //客户端下载服务器中的文件
*upload //客户端向服务器上传文件
*userexit //客户端退出
QTextBrowser（文本浏览框类）
*message_show //回显服务器传递的消息
槽函数
void show_mainwin() //返回登录窗口
void UserExit() //用户退出登录
QString ViewFolder(SOCKET Soc) //发送显示文件列表的请求并接受回应
QString FileDir() //发送显示文件路径的请求并接受回应
QString Download() //发送下载文件的请求并接受回应
QString Upload() //发送上传文件的请求并接受回应
bool Send_Order(...) //用于向服务器发送各类请求
bool Read_Resp(...) //用于接收服务器的消息
SOCKEYT Create_Socket() //创建套接字用于接收服务器传来的文件信息

MyServer

Headers

server.h

```
#ifndef SERVER_H
#define SERVER_H

/*
 *与server相关的头文件
```

```c
 */

//定义一些常数

//client侦听sever数据端口，应大于1024
#define DATA_PORT_PV 5850
//client与sever连接的命令端口，应大于1024
#define CMD_PROT_AC 4096

//命令报文参数缓存的大小
#define CMD_PARAM_SIZE 256
//回复报文消息缓存的大小
#define RESPONSE_CONTENT_SIZE 256
#define BACKLOG 10
#define DATA_BUFSIZE 4096

/*
 * 命令类型:
    LIST:查看文件列表
    PWD:显示当前目录
    DOWNLOAD:下载文件
    UPLOAD:上传文件
    QUIT:退出

    分别对应:int 0~4
 */
typedef enum
{
    LIST,PWD,DOWNLOAD,UPLOAD,QUIT
}CMD;

//命令报文,从客户端发往服务器
typedef struct CmdGram {
    CMD cmd;
    char content[CMD_PARAM_SIZE];
} CmdGram;

//回复报文的类型
typedef enum {
    SUCCESS, ERR
} RESPONSE;

//回复报文,从服务器发往客户端
typedef struct ResponseGram {
    RESPONSE response;
    //回复报文的具体内容
    char content[RESPONSE_CONTENT_SIZE];
} ResponseGram;
#endif // SERVER_H
```

widget.h

```cpp
#ifndef WIDGET_H
#define WIDGET_H

#include <QWidget>
#include <QTextBrowser>
#include <QPushButton>
#include <QLineEdit>
#include <QLabel>
#include <QDebug>
#include <winsock2.h>
#include <server.h>
#include <clithread.h>
#include <QMessageBox>

#pragma comment(lib, "ws2_32.lib")

class Widget : public QWidget
{
    Q_OBJECT

public:
    Widget(QWidget *parent = nullptr);
    ~Widget();

    //界面美化
    void Construct();
    //初始化服务器
    void SerInit();

private slots:
    //打开服务器
    void OpenSer();
    //消息框内容更新
    void MsgUpdate(QString msg);
    //服务器关闭
    void SerExit();

public:
    QTextBrowser *message;
    QPushButton *openser;
    QPushButton *exit;
    QLineEdit *serport;

    QMessageBox ExitLog;
    QPushButton *ExitL=new QPushButton(QObject::tr("退出"));

private:
    WSADATA wasData;
    SOCKET mSerSocket = INVALID_SOCKET;
    CliThread *mCliThread;
    int port;
};
#endif // WIDGET_H
```

clithread.h

```
#ifndef CLITHREAD_H
#define CLITHREAD_H

#include <QWidget>
#include <QThread>
#include <winsock2.h>
#include <QDebug>
#include <QString>
#include <msgthread.h>

class CliThread : public QThread
{
    Q_OBJECT
public:
    explicit CliThread(SOCKET mListen, QWidget *parent = nullptr);
    void run();
    ~CliThread();

public:
    QString msg;

signals:
    //cli发送过来的所有信息需要传递给ui界面更新，均采用此信号
    void isMsg(QString msg);
    //clithread关闭信号
    void isClose();

public slots:
    //传递文件传输线程中的信号到ui界面的槽函数
    void sendMsg(QString msg);

private:
    SOCKET mListen;
    QWidget *parent;

};

#endif // CLITHREAD_H
```

msgthread.h

```
#ifndef MSGTHREAD_H
#define MSGTHREAD_H

#include <QWidget>
#include <QThread>
#include <winsock2.h>
#include <QDebug>
#include <QString>
#include <server.h>
```

```
#include <cstring>
#include <iostream>
#include <io.h>

class MsgThread : public QThread
{
    Q_OBJECT
public:
    explicit MsgThread(SOCKET mClient, SOCKADDR_IN mCli_Addr, QWidget *parent =
nullptr);
    ~MsgThread();
    void run();

    //ser与cli之间的对接
    bool SerToCli(SOCKET Ser_Cmd_Socket, CmdGram* Command, SOCKADDR_IN
*Cli_Data_Port);
    //与cli进行数据连接
    bool DataLink(SOCKET *Ser_Data_Socket, SOCKADDR_IN *Cli_Data_Port);
    //向cli发送报文
    bool SendtoCli(SOCKET Ser_Cmd_Socket, ResponseGram* Response);
    //接收cli的报文
    bool ReceiveCli(SOCKET Ser_Cmd_Socket, char* Command);
    //向cli发送文件
    bool SendFiletoCli(SOCKET Ser_Data_Socket, char* FileName);
    //从cli接收文件
    bool ReceiveFile(SOCKET Ser_Data_Socket, char* FileName);
    //给cli发送文件列表
    bool SendFileList(SOCKET Ser_Data_Socket);
    //返回文件的具体信息
    void ReturnFileInfo(struct _finddata32_t* File_Data, char* fileInfo);

private:
    //与客户端进行文件传输通信的套接字
    SOCKET mClient;
    //客户端地址
    SOCKADDR_IN mCli_Addr;

signals:
    //客户端的相关操作通过此信号传输
    void isMsg(QString msg);

public slots:
};

#endif // MSGTHREAD_H
```

Sources

widget.cpp

```
#include "widget.h"

Widget::Widget(QWidget *parent)
    : QWidget(parent)
```

```
{
    Construct();
    SerInit();
}

Widget::~Widget()
{
    ::closesocket(mSerSocket);
    WSACleanup();
}

//服务器端初始化
void Widget::SerInit()
{
    if(WSAStartup(MAKEWORD(2, 2), &wasData) != 0)
    {
        message->append("初始化失败");
        return;
    }
    return;
}

//启动服务器
void Widget::OpenSer()
{
    //先获取输入端口
    QString thePort;
    thePort = serport->text();
    this->port = thePort.toInt();
    message->append("服务器端口号为: " + thePort);

    //创建套接字
    SOCKET mListen;
    mListen = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
    if(mListen ==INVALID_SOCKET)
    {
        message->append("创建socket失败");
        WSACleanup();
        return;
    }

    //将ser套接字与本地端口地址绑定
    SOCKADDR_IN ser;
    ser.sin_family = AF_INET;
    ser.sin_port = htons(port);
    ser.sin_addr.S_un.S_addr = htonl(INADDR_ANY);
    memset(&(ser.sin_zero), 0, sizeof(ser.sin_zero));
    if(bind(mListen, (SOCKADDR*)&ser, sizeof(ser)) != 0)
    {
        WSACleanup();
        qDebug()<<"绑定socket与本地端口失败";
        message->append("绑定socket与本地端口失败");
        return;
    }

    //进入监听状态
    if(listen(mListen, 5) != 0)
    {
        WSACleanup();
        qDebug()<<"监听错误";
        message->append("监听错误");
        return;
    }

    message->append("服务器打开成功");
    message->append("-------------------------服务器开始工作-----------------------
--");
    openser->setStyleSheet("QPushButton{border-
image:url(\":/logo/openser3.png\");}");
    openser->setEnabled(false);
    mSerSocket = mListen;

    //启动后台线程与客户端进行连接
    mCliThread = new CliThread(mSerSocket);
    mCliThread->start();

    //信号槽连接
    connect(mCliThread, &CliThread::isMsg, this, &Widget::MsgUpdate);
}

//消息框内容更新
void Widget::MsgUpdate(QString msg)
{
    message->append(msg);
    return;
}

//Ser退出
void Widget::SerExit()
{
    mCliThread->terminate();
    mCliThread->quit();
    delete mCliThread;
    closesocket(mSerSocket);
    WSACleanup();
    close();
}

//服务器界面美化
void Widget::Construct()
{
    this->setWindowTitle("闪翼文件传输系统-服务器");
    this->resize(1200, 800);
    this->setWindowIcon(QIcon(":/logo/logo.png"));
    this->setFixedSize(this->width(), this->height());

    QImage  image;
    image.load(":/logo/ser_background.png");
    QPalette palet;
    palet.setBrush(this->backgroundRole(),QBrush(image));
    this->setPalette(palet);

    openser = new QPushButton(this); openser->resize(200,70); openser-
>move(920,400);
    QString styleSheetString1("QPushButton{border-
image:url(\":/logo/openser1.png\");}");
```

```
    styleSheetString1+="QPushButton:hover{border-
image:url(\":/logo/openser2.png\");}";
    styleSheetString1+="QPushButton:pressed{border-
image:url(\":/logo/openser3.png\");}";
    openser->setStyleSheet(styleSheetString1);
    connect(openser, &QPushButton::clicked, this, &Widget::OpenSer);

    exit = new QPushButton(this); exit->resize(80,80); exit->move(980,665);
    QString styleSheetString2("QPushButton{border-
image:url(\":/logo/exit1.png\");}");
    styleSheetString2+="QPushButton:hover{border-
image:url(\":/logo/exit2.png\");}";
    styleSheetString2+="QPushButton:pressed{border-
image:url(\":/logo/exit3.png\");}";
    exit->setStyleSheet(styleSheetString2);
    connect(exit, &QPushButton::clicked,
            [=]()
    {
        ExitLog.show();
    });

    serport = new QLineEdit(this);
    serport->setStyleSheet("QLineEdit{border-width:0;border-
style:outset;background-color:rgba(242,242,242,0)}");
    serport->resize(245, 50);
    serport->move(900, 322);
    serport->setFont(QFont("Consolas", 15, QFont::Bold));

    message = new QTextBrowser(this);
    message->resize(820, 700);
    message->move(45, 60);
    message->setFont(QFont("Consolas", 12, QFont::Bold));

    //设置退出对话框
    ExitLog.setWindowTitle("闪翼");
    ExitLog.setWindowIcon(QIcon(":/logo/logo.png"));
    ExitLog.setText("感谢您选择闪翼文件传输，欢迎您下次使用，再见~");
    ExitLog.addButton(ExitL, QMessageBox::RejectRole);
    connect(ExitL, &QPushButton::released, this, &Widget::SerExit);
}
```

clithread.cpp

```
#include "clithread.h"

CliThread::CliThread(SOCKET mListen, QWidget *parent) : QThread(parent)
{
    this->mListen = mListen;
    this->parent = parent;
}

CliThread::~CliThread()
{
    closesocket(mListen);
    emit isClose();
}

void CliThread::run()
{
    //客户端地址
    sockaddr_in cli_address;
    int size = sizeof(cli_address);

    while(!isInterruptionRequested())
    {
        //每次接收新客户端时，将之前的地址信息清0
        memset(&cli_address, 0, sizeof(cli_address));

        //等待新客户端连接
        SOCKET cli_socket = accept(mListen, (SOCKADDR*)&cli_address, &size);
        if(cli_socket == INVALID_SOCKET)
        {
            msg = "创建cli_socket失败";
            emit isMsg(msg);
            return;
        }

        msg = "客户端连接成功";
        emit isMsg(msg);

        //开启新线程与客户端进行通信
        MsgThread* FileThread = new MsgThread(cli_socket, cli_address, parent);
        FileThread->start();
        //绑定信号和槽，转发消息给ui进程进行界面更新
        connect(FileThread, &MsgThread::isMsg, this, &CliThread::sendMsg);

        connect(this, &CliThread::isClose,
                [=]()
        {
            msg = "客户端断开连接";
            emit isMsg(msg);
            FileThread->terminate();
            FileThread->quit();
            delete FileThread;
        });
    }
}

void CliThread::sendMsg(QString msg)
{
    emit isMsg(msg);
    return;
}
```

msgthread.cpp

```cpp
#include "msgthread.h"

MsgThread::MsgThread(SOCKET mClient, SOCKADDR_IN mCli_Addr, QWidget *parent) :
QThread(parent)
{
    this->mClient = mClient;
    this->mCli_Addr = mCli_Addr;
}

MsgThread::~MsgThread()
{
    int ret = closesocket(mClient);
    if( ret == 0 )
    {
        qDebug()<< "关闭成功";
    }
    else
    {
        qDebug()<< "关闭失败";
    }
}

void MsgThread::run()
{
    //给客户端发送报文
    ResponseGram responseCli;
    responseCli.response = SUCCESS;
    strcpy(responseCli.content, "SUCCESS");
    if(SendtoCli(mClient, &responseCli));
    else
    {
        emit isMsg("为客户端创建线程成功，但与客户端发送信息失败");
    }

    //开始获取报文命令
    CmdGram commandCli;
    while(true)
    {
        if(ReceiveCli(mClient, (char*)&commandCli));
        else
        {
            emit isMsg("获取客户端命令失败");
            break;
        }
        if(SerToCli(mClient, &commandCli, &mCli_Addr));
        else
        {
            emit isMsg("获取客户端命令成功，但服务器未响应");
            break;
        }
    }

    closesocket(mClient);
}
```

```
bool MsgThread::SerToCli(SOCKET Ser_Cmd_Socket, CmdGram *Command, SOCKADDR_IN
*Cli_Data_Port)
{
    SOCKET Ser_Data_Socket;
    FILE* fp = NULL;
    ResponseGram* response;
    //回复报文，从ser发往cli
    ResponseGram Response;
    Response.response = SUCCESS;
    strcpy(Response.content, "SUCCESS");

    response = &Response;

    //cli从ser下载文件
    if(Command->cmd == DOWNLOAD)
    {
        emit isMsg("收到下载文件命令");
        fp = fopen(Command->content, "rb");//直接打开命令所在的的文件目录
        if(fp == NULL)
        {
            response->response = ERR;
            strcpy(response->content, "打开文件错误\n\n");

            //发送错误报文
            if(SendtoCli(Ser_Cmd_Socket, response))
            {
                emit isMsg("与客户端报告错误失败");
                return false;
            }
            return false;
        }
        else
        {
            //发送成功报文
            response->response = SUCCESS;
            strcpy(response->content, "打开成功，正在传送......");
            if(SendtoCli(Ser_Cmd_Socket, response))//发送报文过后开始传输文件
            {
                //建立连接
                if(DataLink(&Ser_Data_Socket, Cli_Data_Port))
                {
                    if(SendFiletoCli(Ser_Data_Socket, Command->content))//发送文件
                    {
                        fclose(fp);
                        return true;
                    }
                    else
                        return false;
                }
                else
                    return false;
            }
            else
                return false;
        }
    }
    else if(Command->cmd == UPLOAD)//上传文件
    {
        emit isMsg("收到上传文件命令");
        //先检查服务器中是否存在同名文件
        char fileName[128];
        strcpy(fileName, Command->content);

        FILE* fp = fopen(fileName, "r+");

        if(fp != NULL)//文件能够打开，说明服务器中存在同名文件，提出警告
        {
            response->response = SUCCESS;
            strcpy(response->content, "警告：服务器中已存在该文件，将会覆盖文件\n");
            remove(fileName);
        }
        else
        {
            response->response = SUCCESS;
            strcpy(response->content, "SUCCESS");
        }

        //给cli发送报文
        if(SendtoCli(Ser_Cmd_Socket, response));
        else
        {
            emit isMsg("文件符合上传条件，但与客户端通信失败");
            return false;
        }

        //建立数据传输通道
        if(DataLink(&Ser_Data_Socket, Cli_Data_Port));
        else
        {
            emit isMsg("建立数据传输通道失败");
            return false;
        }

        //接收cli的文件
        if(ReceiveFile(Ser_Data_Socket, fileName));
        else
        {
            emit isMsg("接收客户端文件失败");
            return false;
        }

        return true;
    }

    else if(Command->cmd == QUIT)//退出
    {
        emit isMsg("接收到客户端退出命令");
        response->response = SUCCESS;
        strcpy(response->content, "服务器已接收到断开请求");

        if(SendtoCli(Ser_Cmd_Socket, response));
        else
        {
            emit isMsg("服务器已成功接收客户端断开信息，但与客户端通信失败");
            return false;
        }
        closesocket(Ser_Data_Socket);
        return true;
    }

    else if(Command->cmd == LIST)//列出文件目录
    {
        emit isMsg("收到列出文件目录命令");
        //建立连接
        if(DataLink(&Ser_Data_Socket, Cli_Data_Port));
        else
        {
            emit isMsg("建立数据传输连接失败");
            return false;
        }

        //发送文件目录
        if(SendFileList(Ser_Data_Socket))
        {
            emit isMsg("发送文件目录成功");
            return true;
        }
        else
        {
            emit isMsg("发送文件目录失败");
            return false;
        }
    }

    else if(Command->cmd == PWD)//显示文件路径
    {
        emit isMsg("收到显示当前文件路径命令");
        response->response = SUCCESS;
        if(GetCurrentDirectoryA(RESPONSE_CONTENT_SIZE, response->content));
        else
        {
            strcpy(response->content, "获取当前目录失败");
            return false;
        }
        if(SendtoCli(Ser_Cmd_Socket, response))
            return true;
        else
        {
            emit isMsg("获取当前目录成功，但与客户端发送信息失败");
            return false;
        }
    }

    else
    {
        return true;
    }
}

//向cli发送报文
bool MsgThread::SendtoCli(SOCKET Ser_Cmd_Socket, ResponseGram* Response)
{
    if(send(Ser_Cmd_Socket, (char*)Response, sizeof (ResponseGram), 0) ==
SOCKET_ERROR)
    {
        int error = WSAGetLastError();
        emit isMsg("向客户端发送报文失败");
        qDebug()<<error;
        return false;
    }

    emit isMsg("向客户端发送报文成功");
    return true;
}

//接收来自客户端的命令
bool MsgThread::ReceiveCli(SOCKET Ser_Cmd_Socket, char* Command)
{
    int flag;
    int not_recv = sizeof(CmdGram);

    //开始读取数据
    for(; not_recv > 0;)
    {
        flag = recv(Ser_Cmd_Socket, Command, not_recv, 0);
        if(flag == SOCKET_ERROR)
        {
            emit isMsg("接收命令错误或客户端退出");
            return false;
        }
        else
        {
            emit isMsg("接收命令成功");
        }


        //字符串指针加int的意义：偏移
        not_recv -= flag;
        Command += flag;
    }

    return true;
}

//建立数据通道
bool MsgThread::DataLink(SOCKET *Ser_Data_Socket, SOCKADDR_IN *Cli_Data_Port)
{
    //创建ser端数据的socket
    SOCKET ser_data_socket;
    ser_data_socket = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
    if(ser_data_socket == INVALID_SOCKET)
    {
        emit isMsg("创建数据传输套接字失败");
        return false;
    }

    //获取客户端数据传输端口号
    SOCKADDR_IN cli_data_port;
    std::memcpy(&cli_data_port, Cli_Data_Port, sizeof(SOCKADDR_IN));
    cli_data_port.sin_port = htons(DATA_PORT_PV);
    //连接
    if(::connect(ser_data_socket, (SOCKADDR*)&cli_data_port, sizeof(SOCKADDR)))
    {
        emit isMsg("与客户端的连接出现问题");
        closesocket(ser_data_socket);
        return false;
    }

    *Ser_Data_Socket = ser_data_socket;
    emit isMsg("与客户端数据传输连接成功");
    return true;
}

//给cli发送文件
bool MsgThread::SendFiletoCli(SOCKET Ser_Data_Socket, char *FileName)
{
    //开始读文件
    char buf[2048];

    FILE *fp = fopen(FileName, "rb");
    if(fp == NULL)
    {
        emit isMsg("文件不存在");
        fclose(fp);
        closesocket(Ser_Data_Socket);
        return false;
    }

    while(1)
    {
        int size = fread(buf, 1, 2048, fp);

        if(send(Ser_Data_Socket, buf, size, 0) == SOCKET_ERROR)
        {
            emit isMsg("发送文件过程中发生错误");
            closesocket(Ser_Data_Socket);
            return false;
        }
        if(size < 2048)
            break;
    }

    fclose(fp);
    closesocket(Ser_Data_Socket);
    emit isMsg("文件发送完成");
    return true;
}

//从cli接收文件
bool MsgThread::ReceiveFile(SOCKET Ser_Data_Socket, char *FileName)
{
    //先创建文件，再读取
    char buf[2048];
    int recv_size;

    FILE* fp = fopen(FileName, "wb");
    if(fp == NULL)
    {
        emit isMsg("创建文件过程中发生错误");
        fclose(fp);
        closesocket(Ser_Data_Socket);
        return false;
    }

    while(1)
    {
        recv_size = recv(Ser_Data_Socket, buf, 2048, 0);
        if(recv_size == SOCKET_ERROR)
        {
            emit isMsg("上传文件时发生错误");
            fclose(fp);
            closesocket(Ser_Data_Socket);
            return false;
        }

        if(recv_size == 0)
            break;

        fwrite(buf, 1, recv_size, fp);
    }

    fclose(fp);
    closesocket(Ser_Data_Socket);
    emit isMsg("完成上传");
    return true;
}

//向cli发送文件列表
bool MsgThread::SendFileList(SOCKET Ser_Data_Socket)
{
    long handle;
    struct _finddata_t File_Data;
    char fileInfo[500];
    const char column[100] = "           Name           |       Updata Time
|     Size(Bytes)     \n";

    //使用通配符找到当前目录，并将其信息存入指针
    handle = _findfirst32("*", &File_Data);

    if(handle == -1)
    {
        emit isMsg("服务器读取文件列表失败");
        if(send(Ser_Data_Socket, "获取文件列表失败", sizeof("获取文件列表失败"), 0));
        else
        {
            emit isMsg("服务器发送信息失败");
            return false;
        }
    }
    else
    {
        send(Ser_Data_Socket, column, sizeof(column), 0);
        while(true)
        {
            //查找下一个文件
            int FindNext = _findnext32(handle, &File_Data);

            if(FindNext == -1)
            {
                emit isMsg("服务器已查找完所有文件");
                break;
            }
            else
            {
                ReturnFileInfo(&File_Data, fileInfo);
                if(send(Ser_Data_Socket, fileInfo, sizeof(fileInfo), 0));
                else
                {
                    emit isMsg("服务器发送信息失败");
                    return false;
                }
            }
        }
    }

    closesocket(Ser_Data_Socket);
    _findclose(handle);
    return true;
}

//返回文件具体信息
void MsgThread::ReturnFileInfo(struct _finddata32_t *File_Data, char *fileInfo)
{
    char fileinfo[500];
    int size;

    time_t fileUpdata_time = File_Data->time_write;
    tm* time_wanted;
    time_wanted = NULL;
    time_wanted = localtime(&fileUpdata_time);

    size = File_Data->size;

    sprintf(fileinfo, "| %24s|   %d/%d/%d %d:%d:%d    | %d \n",
            File_Data->name, 1900 + time_wanted->tm_year, 1 + time_wanted-
>tm_mon, time_wanted->tm_mday,
            time_wanted->tm_hour, time_wanted->tm_min, time_wanted->tm_sec,
size);

    strcpy(fileInfo, fileinfo);
}
```

main.cpp

```
#include "widget.h"

#include <QApplication>

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    Widget w;
    w.show();
    return a.exec();
}
```

MyClient

Headers

client.h

```
#ifndef SEVER_H
#define SEVER_H

/*
 *与client相关的头文件
 */

//下面定义一些常数

//client侦听sever数据端口 > 1024
#define DATA_PORT_PV 5850
//client与sever连接的命令端口, > 1024
#define CMD_PROT_AC 4096

//命令报文参数缓存的大小
#define CMD_PARAM_SIZE 256
//回复报文消息缓存的大小
#define RESPONSE_CONTENT_SIZE 256
#define BACKLOG 10
#define DATA_BUFSIZE 4096

/*
 * 命令类型:
    LIST:查看文件列表
    PWD:显示当前目录
    DOWNLOAD:下载文件
    UPLOAD:上传文件
    QUIT:退出

    分别对应:int 0~4
 */
#include <windows.h>

//全局变量
extern SOCKET My_Socket;
```

```
typedef enum
{
    LIST,PWD,DOWNLOAD,UPLOAD,QUIT
} CMD;

//命令报文,从客户端发往服务器
typedef struct CmdGram {
    CMD cmd;
    char content[CMD_PARAM_SIZE];
} CmdGram;

//回复报文的类型
typedef enum {
    SUCCESS, ERR
} RESPONSE;

//回复报文,从服务器发往客户端
typedef struct ResponseGram {
    RESPONSE response;
    char content[RESPONSE_CONTENT_SIZE];
} ResponseGram;
#endif // SEVER_H
```

login.h

```
#ifndef LOGIN_H
#define LOGIN_H

#include <QWidget>
#include <QDialog>
#include <QIcon>
#include <QFont>
#include <QLabel>
#include <QLineEdit>
#include <QPushButton>
#include <QMessageBox>
#include <QDebug>
#include <windows.h>
#include <client.h>
#include <tchar.h>
#include <iostream>
#include <QFileDialog>
#include <QInputDialog>
#include <imagehlp.h>

class Login : public QDialog
{
    Q_OBJECT
public:
    explicit Login(QWidget *parent = nullptr);

    void Construct();

public:
    QLineEdit *my_severloc;
    QString serip;
    QPushButton *connectser;
    QLineEdit *my_port;
    QString serport;

    QLineEdit *my_username;
    QString user;
    QLineEdit *my_password;
    QString psd;
    QPushButton *cli_login;
    const QString Name = "yyy";
    const QString Password = "20210521";
    int chances;

    QMessageBox FailedLog, ExitLog;
    QPushButton *Retry=new QPushButton(QObject::tr("重试"));
    QPushButton *ExitL=new QPushButton(QObject::tr("退出"));


public:
    QString IP_NUM;
    QString Port_NUM;

    WSADATA wasData;

signals:
    void mainshow();

public slots:
    void Login_clicked();
    void show_loginwin();

    void ConnectSer();
};

#endif // LOGIN_H
```

widget.h

```
#ifndef WIDGET_H
#define WIDGET_H

#include <QWidget>
#include <QPushButton>
#include <QTextBrowser>
#include <windows.h>
#include "client.h"
#include <QDebug>
#include <windows.h>
#include <tchar.h>
#include <iostream>
#include <QFileDialog>
```

```
#include <QLineEdit>
#include <QInputDialog>
#include <imagehlp.h>
#include <QTextStream>

//QTextStream out(stdout);

class Widget : public QWidget
{
    Q_OBJECT

public:
    Widget(QWidget *parent = nullptr);
    ~Widget();

    void Construct();

public:
    QPushButton *view_folder;
    QPushButton *file_dir;
    QPushButton *userexit;
    QPushButton *download;
    QPushButton *upload;
    QTextBrowser *message_show;

public:
    //客户端向服务器发送请求
    bool Send_Order(SOCKET CliSocket, CmdGram *Command);
    //接收服务器的回复消息
    bool Read_Resp(SOCKET CliSocket, ResponseGram *Response);

signals:
    void Exit();

private slots:
    void show_mainwin();
    void UserExit();

    QString ViewFolder(SOCKET Soc);
    QString FileDir();
    QString Download();
    QString Upload();

};
#endif // WIDGET_H
```

Sources

login.cpp

```
#include "login.h"

Login::Login(QWidget *parent) : QDialog(parent)
{
    Construct();
    chances = 3;
}

//连接到serip:serport
SOCKET My_Socket;

void Login::ConnectSer()
{
    //从输入框中获取服务器地址和端口
    serip = my_severloc->text();
    serport = my_port->text();

    IP_NUM = serip;
    Port_NUM = serport;
    SOCKADDR_IN addr;
    if(WSAStartup(MAKEWORD(2, 2), &wasData) != 0)
    {
        qDebug()<<"WinSock初始化失败";
    }

    //创建套接口
    My_Socket = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
    if(My_Socket == INVALID_SOCKET)
    {
        qDebug()<<"创建套接口失败";
    }

    addr.sin_family = AF_INET;
    addr.sin_port = htons(Port_NUM.toInt());

    char* temp;
    QByteArray Tran = IP_NUM.toLatin1();
    temp = Tran.data();
    addr.sin_addr.S_un.S_addr = inet_addr(temp);
    memset(&(addr.sin_zero), 0, sizeof(addr.sin_zero));

    if (::connect(My_Socket, (SOCKADDR*)&addr, sizeof(SOCKADDR)) ==
SOCKET_ERROR)  //避免与QTconnect冲突
    {
        qDebug()<<"未能连接上服务器";
        closesocket(My_Socket);
        WSACleanup();
        return;
    }

    qDebug()<<"连接成功，可进行登录";
    cli_login->setEnabled(true);
    return;
}

void Login::Login_clicked()
{
    user = my_username->text();
    psd = my_password->text();

    if((user == Name) && (psd == Password))
    {
        this->close();
        emit mainshow();
    }
    else
    {
        //拥有三次机会输入
        chances--;
        if(chances >= 1)
        {
            switch (chances)
            {
            case 1:
                FailedLog.setText("用户名或密码错误，您还有1次输入机会！");
                break;
            case 2:
                FailedLog.setText("用户名或密码错误，您还有2次输入机会！");
                break;
            default:
                break;
            }
            FailedLog.show();
        }
        else
        {
            ExitLog.show();
        }
    }
}

void Login::show_loginwin()
{
    this->show();
    //清空文本框，重新连接服务器
    my_severloc->clear();
    my_port->clear();
    my_username->clear();
    my_password->clear();
    //设置光标
    my_severloc->setFocus();
    cli_login->setEnabled(false);
}

void Login::Construct()
{
    this->setWindowTitle("闪翼文件传输系统-客户端登录");
    this->resize(1200, 800);
    this->setWindowIcon(QIcon(":/logo/logo.png"));
    setFixedSize(this->width(), this->height());

    QImage  image;
    image.load(":/logo/login_background.png");
    QPalette palet;
    palet.setBrush(this->backgroundRole(),QBrush(image));
    this->setPalette(palet);

    //服务器地址输入布局
    my_severloc = new QLineEdit(this);
    my_severloc->setStyleSheet("QLineEdit{border-width:0;border-
style:outset;background-color:rgba(242,242,242,0)}");
    my_severloc->resize(300, 45);
    my_severloc->move(250, 340);
    my_severloc->setFont(QFont("Consolas", 15, QFont::Bold));

    my_port = new QLineEdit(this);
    my_port->setStyleSheet("QLineEdit{border-width:0;border-
style:outset;background-color:rgba(242,242,242,0)}");
    my_port->resize(300, 45);
    my_port->move(250, 435);
    my_port->setFont(QFont("Consolas", 15, QFont::Bold));

    //登录界面布局
    my_username = new QLineEdit(this);
    my_username->setStyleSheet("QLineEdit{border-width:0;border-
style:outset;background-color:rgba(242,242,242,0)}");
    my_username->resize(300, 55);
    my_username->move(780, 316);
    my_username->setFont(QFont("Consolas", 15, QFont::Bold));

    my_password = new QLineEdit(this);
    my_password->setStyleSheet("QLineEdit{border-width:0;border-
style:outset;background-color:rgba(242,242,242,0)}");
    my_password->resize(300, 55);
    my_password->move(780, 424);
    my_password->setFont(QFont("Consolas", 15, QFont::Bold));
    my_password->setEchoMode(QLineEdit::Password);

    //按钮设置
    connectser = new QPushButton(this); connectser->resize(220,60); connectser-
>move(295,515);
    QString styleSheetString1("QPushButton{border-
image:url(\":/logo/connect1.png\");}");
    styleSheetString1+="QPushButton:hover{border-
image:url(\":/logo/connect2.png\");}";
    styleSheetString1+="QPushButton:pressed{border-
image:url(\":/logo/connect3.png\");}";
    connectser->setStyleSheet(styleSheetString1);
    connect(connectser, &QPushButton::clicked, this, &Login::ConnectSer);

    cli_login = new QPushButton(this); cli_login->resize(220,60); cli_login-
>move(800,515);
    QString styleSheetString2("QPushButton{border-
image:url(\":/logo/login1.png\");}");
    styleSheetString2+="QPushButton:hover{border-
image:url(\":/logo/login2.png\");}";
    styleSheetString2+="QPushButton:pressed{border-
image:url(\":/logo/login3.png\");}";
    cli_login->setStyleSheet(styleSheetString2);
    connect(cli_login, &QPushButton::clicked, this, &Login::Login_clicked);
    cli_login->setEnabled(false);

    //设置重试对话框
    FailedLog.setWindowTitle("用户登录");
    FailedLog.setWindowIcon(QIcon(":/logo/logo.png"));
    FailedLog.addButton(Retry, QMessageBox::AcceptRole);
    connect(Retry, &QPushButton::released,
            [=]()
    {
        my_username->clear();
        my_password->clear();
    });

    //设置失败对话框
    ExitLog.setWindowTitle("用户登录");
    ExitLog.setWindowIcon(QIcon(":/logo/logo.png"));
    ExitLog.setText("您的机会已用完，请您退出登录。");
    ExitLog.addButton(ExitL, QMessageBox::RejectRole);
    connect(ExitL, &QPushButton::released,
            [=]()
    {
        closesocket(My_Socket);
        WSACleanup();
        close();
    });
}
```

widget.cpp

```
#include "widget.h"

Widget::Widget(QWidget *parent)
    : QWidget(parent)
{
    Construct();
}

Widget::~Widget()
{
    closesocket(My_Socket);
    WSACleanup();
}

//客户端向服务器发送请求
bool Widget::Send_Order(SOCKET CliSocket, CmdGram *Command)
{
    int size = sizeof(CmdGram);
    if(send(CliSocket, (char*)Command, size, 0) == SOCKET_ERROR)
    {
        message_show->append("向服务器发送请求失败");
        closesocket(CliSocket);
        WSACleanup();
        return false;
    }

    return true;
}

//接收服务器的回复消息
bool Widget::Read_Resp(SOCKET CliSocket, ResponseGram *Response)
{
    int num;
    int size = sizeof(ResponseGram);
    int mark;
    for(num = 0, mark = 1 ; num < size; )
    {
        mark = recv(CliSocket, (char*)Response + num , size - num, 0);
        if(mark <= 0)
        {
            message_show->append("读取服务器回复消息错误");
            closesocket(CliSocket);
            return false;
        }
        num += mark;
    }
    return true;
}

//创建套接字用于接收从服务器端传来的文件信息
SOCKET Create_Socket()
{
    SOCKET Soc;
    SOCKADDR_IN addr;

    unsigned int mark_socket=(Soc=socket(AF_INET,SOCK_STREAM,IPPROTO_TCP));
    if(mark_socket==INVALID_SOCKET)
    {
        qDebug()<<"Create Socket Error";
        WSACleanup();
        exit(-1);
    }

    addr.sin_addr.s_addr=htonl(INADDR_ANY);
    addr.sin_family=AF_INET;
    addr.sin_port=htons(DATA_PORT_PV);
    memset(&(addr.sin_zero), 0, sizeof(addr.sin_zero));

    int mark_bind=bind(Soc,(SOCKADDR *) &addr , sizeof(SOCKADDR));
    if(mark_bind==SOCKET_ERROR)
    {
        qDebug()<<"Bind Addr Error";
        closesocket(Soc);
        WSACleanup();
        exit(-1);
    }
    //
    int mark_listen=listen(Soc,1);
    if(mark_listen==SOCKET_ERROR)
    {
        qDebug()<<"listen error";
        closesocket(Soc);
        WSACleanup();
        exit(-1);
    }
    return Soc;
}


QString Widget::ViewFolder(SOCKET Soc)
```

```
{
     QString Sum_Data;
     int size;
     int num;
     CmdGram C_Packet;
     SOCKET new_Soc;
     SOCKET temp_Soc;
     SOCKADDR_IN addr;
     char buffer[DATA_BUFSIZE];

     new_Soc = Create_Socket();
     C_Packet.cmd = LIST;
     Soc = My_Socket;
     Send_Order(Soc,& C_Packet);
     size = sizeof(SOCKADDR_IN);
     unsigned int mark_list = (temp_Soc=accept(new_Soc , (SOCKADDR*) &addr ,
&size));

     Sum_Data = "";
     while(true)
     {
         num = recv(temp_Soc, buffer, DATA_BUFSIZE-1, 0);
         if(num == SOCKET_ERROR)
         {
             qDebug()<<"Server Error";
             closesocket(temp_Soc);
             closesocket(new_Soc);
             closesocket(Soc);
             WSACleanup();
             exit(-1);
         }
         if(num == 0)
             break;
         Sum_Data += QString(buffer);
     }

     closesocket(new_Soc);
     closesocket(temp_Soc);

     message_show->append("********************");
     message_show->append(Sum_Data);
     message_show->moveCursor(message_show->textCursor().End);
     return Sum_Data;
}

QString Widget::FileDir()
{
    CmdGram View_Packet;
    ResponseGram Re_View_Packet;
    View_Packet.cmd = PWD; //显示当前目录
    //message_show->setText("********************");
    Send_Order(My_Socket, &View_Packet);
    Read_Resp(My_Socket, &Re_View_Packet);
    message_show->append("********************");
    message_show->append(Re_View_Packet.content);
    message_show->moveCursor(message_show->textCursor().End);
    return QString(Re_View_Packet.content);
```

```
}


QString Widget::Download()
{
        FILE *file;   //server端的文件
        char buffer[DATA_BUFSIZE];
        CmdGram C_Packet;
        ResponseGram R_Packet;
        SOCKET new_Soc;
        SOCKET temp_Soc;
        SOCKADDR_IN addr;
        int size;
        int num;
        QString Server_pos =  this->FileDir();
        Server_pos = this->FileDir();         //Server 根目录
        //qDebug<<Server_pos;
        QString Server_File = QFileDialog::getOpenFileName(0 , "文件标题" ,
Server_pos);
        //Server_pos.append(Server_File);
        message_show->append("download ....");
        message_show->append(Server_File);  //文件目录

        message_show->append(Server_pos);
        message_show->moveCursor(message_show->textCursor().End);

        QString file_get = QFileInfo(Server_File).fileName(); //从路径中获取文件名
        message_show->append(file_get);
        message_show->moveCursor(message_show->textCursor().End);

        QByteArray Target = file_get.toLatin1();
        strcpy(C_Packet.content, Target.data());

        QString DownPath = "D:\\yyy\\courses\\Network\\yyy Network\\MyClient\\" +
file_get; //此处可改
        QByteArray Location = DownPath.toLatin1(); //Location为目标路径
        C_Packet.cmd = DOWNLOAD;

        file = fopen(Location.data(), "wb");
        if(file == NULL)
        {
            qDebug()<<"Open File Error";
            return QString("Open File Error");
        }

        new_Soc = Create_Socket();
        Send_Order(My_Socket, &C_Packet);
        Read_Resp(My_Socket, &R_Packet);
        if(R_Packet.response == ERR)
        {
            qDebug()<<"No Right File";
            fclose(file);
            closesocket(new_Soc);
            return QString("No Right File");
        }
        size = sizeof(SOCKADDR_IN);

        temp_Soc = accept(new_Soc,(SOCKADDR *)&addr, &size);
        if(temp_Soc == INVALID_SOCKET)
        {
            qDebug()<<"Accept File Error";
            closesocket(new_Soc);
            fclose(file);
            return QString("Accept File Error");
        }

        for(num = 2; num > 0; )
        {
            num = recv(temp_Soc, buffer, DATA_BUFSIZE, 0);
            fwrite(buffer, sizeof(char), num, file);
        }


        fclose(file);

        closesocket(temp_Soc);
        closesocket(new_Soc);

        qDebug()<<"Get File Successfully";
        return QString("Get File Successfully");
}

QString Widget::Upload()
{
    QString My_File=QFileDialog::getOpenFileName(0 , "文件标题" , ".");
    FILE *file;
    char buffer[DATA_BUFSIZE];
    CmdGram C_Packet;
    ResponseGram R_Packet;
    SOCKET new_Soc;
    SOCKET temp_Soc;
    SOCKADDR_IN addr;
    int size;
    int num;

    QString file_get=QFileInfo(My_File).fileName();
    QByteArray Target=file_get.toLatin1();
    strcpy(C_Packet.content , Target.data());
    QByteArray Location=My_File.toLatin1();
    C_Packet.cmd=UPLOAD;

    message_show->append("Upload......");
    message_show->moveCursor(message_show->textCursor().End);

    qDebug()<<Location.data();

    file=fopen(Location.data() , "rb");
    if(file==NULL)
    {
        qDebug()<<"Open File Error";
        return QString("Open File Error");
    }

    new_Soc=Create_Socket();
    Send_Order(My_Socket , &C_Packet);
    Read_Resp(My_Socket , &R_Packet);
    if(R_Packet.response==ERR)
    {
        qDebug()<<"Same File Has Existed";
        fclose(file);
        return QString("Same File Has Existed");
    }

    size =sizeof(SOCKADDR_IN);
    unsigned int mark_accept=(temp_Soc = accept(new_Soc , (SOCKADDR*) &addr ,
&size));
    if(mark_accept== INVALID_SOCKET)
    {
        qDebug()<<"upload error";
        closesocket(new_Soc);
        fclose(file);
        return QString("upload error");
    }

    num=0;
    while (true)
    {
        num=fread(buffer , sizeof(char) , DATA_BUFSIZE , file);
        send(temp_Soc , buffer , num , 0);
        if(num<DATA_BUFSIZE)
        {
            qDebug()<<"File Upload Successfully";
            break;
        }
    }
    closesocket(temp_Soc);
    closesocket(new_Soc);
    fclose(file);
    message_show->append("Upload Successfully");
    message_show->moveCursor(message_show->textCursor().End);
    return QString("File Upload Successfully");
}

void Widget::show_mainwin()
{
    this->show();
}

void Widget::UserExit()
{
    //清除全部内容
    message_show->clearHistory();
    closesocket(My_Socket);
    this->close();
    emit Exit();
}

void Widget::Construct()
{
    this->setWindowTitle("闪翼文件传输系统-客户端");
    this->resize(1200, 800);
    this->setWindowIcon(QIcon(":/logo/logo.png"));
    this->setFixedSize(this->width(), this->height());
    QImage  image;
    image.load(":/logo/widget_background.png");
    QPalette palet;
    palet.setBrush(this->backgroundRole(),QBrush(image));
    this->setPalette(palet);

    file_dir = new QPushButton(this); file_dir->resize(200,70); file_dir-
>move(960,180);
    QString styleSheetString2("QPushButton{border-
image:url(\":/logo/filedir1.png\");}");
    styleSheetString2+="QPushButton:hover{border-
image:url(\":/logo/filedir2.png\");}";
    styleSheetString2+="QPushButton:pressed{border-
image:url(\":/logo/filedir3.png\");}";
    file_dir->setStyleSheet(styleSheetString2);
    connect(file_dir, &QPushButton::clicked, this, &Widget::FileDir);

    view_folder = new QPushButton(this); view_folder->resize(200,70);
view_folder->move(960,280);
    QString styleSheetString1("QPushButton{border-
image:url(\":/logo/viewfolder1.png\");}");
    styleSheetString1+="QPushButton:hover{border-
image:url(\":/logo/viewfolder2.png\");}";
    styleSheetString1+="QPushButton:pressed{border-
image:url(\":/logo/viewfolder3.png\");}";
    view_folder->setStyleSheet(styleSheetString1);
    connect(view_folder, &QPushButton::clicked, this, &Widget::ViewFolder);

    upload = new QPushButton(this); upload->resize(200,70); upload-
>move(960,430);
    QString styleSheetString3("QPushButton{border-
image:url(\":/logo/upload1.png\");}");
    styleSheetString3+="QPushButton:hover{border-
image:url(\":/logo/upload2.png\");}";
    styleSheetString3+="QPushButton:pressed{border-
image:url(\":/logo/upload3.png\");}";
    upload->setStyleSheet(styleSheetString3);
    connect(upload, &QPushButton::clicked, this, &Widget::Upload);

    download = new QPushButton(this); download->resize(200,70); download-
>move(960,530);
    QString styleSheetString4("QPushButton{border-
image:url(\":/logo/download1.png\");}");
    styleSheetString4+="QPushButton:hover{border-
image:url(\":/logo/download2.png\");}";
    styleSheetString4+="QPushButton:pressed{border-
image:url(\":/logo/download3.png\");}";
    download->setStyleSheet(styleSheetString4);
    connect(download, &QPushButton::clicked, this, &Widget::Download);

    userexit = new QPushButton(this); userexit->resize(80,80); userexit-
>move(1020,665);
    QString styleSheetString5("QPushButton{border-
image:url(\":/logo/exit1.png\");}");
    styleSheetString5+="QPushButton:hover{border-
image:url(\":/logo/exit2.png\");}";
    styleSheetString5+="QPushButton:pressed{border-
image:url(\":/logo/exit3.png\");}";
```

```
    userexit->setStyleSheet(styleSheetString5);
    connect(userexit, &QPushButton::clicked, this, &Widget::UserExit);

    message_show = new QTextBrowser(this);
    message_show->resize(900, 700);
    message_show->move(50, 60);
    message_show->setFont(QFont("Consolas", 12, QFont::Bold));
}
```

main.cpp

```
#include "widget.h"

#include <QApplication>
#include "login.h"

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    Widget w;
    Login m;
    m.show();
    //点击登录对话框的登录按钮，进入主界面
    QObject::connect(&m,SIGNAL(mainshow()),&w,SLOT(show_mainwin()));
    //点击主界面的注销，返回登录对话框
    QObject::connect(&w,SIGNAL(Exit()),&m,SLOT(show_loginwin()));

    return a.exec();
}
```