**实验五**

**基于 WinSock API 的网络通信应用程序开发**

**【实验目的】**

1. 了解相关网络协议的基本原理和工作流程；

2. 掌握使用 Socket 进行网络通信的方法；

3. 体会客户机、服务器交互模式。

**【实验任务】**

1. 通过调用 Socket 相关函数实现网络通信；

2. 实现界面系统和后台通信系统的协同配合；

3. 回顾常用控件和 GDI 对象的使用方法；

4. 感受 MFC 下多线程的基本用法。

**【实验设备】**

个人 PC，Windows11，Qt5.15.2 MSVC2019

**【计划学时】**

4-8 学时

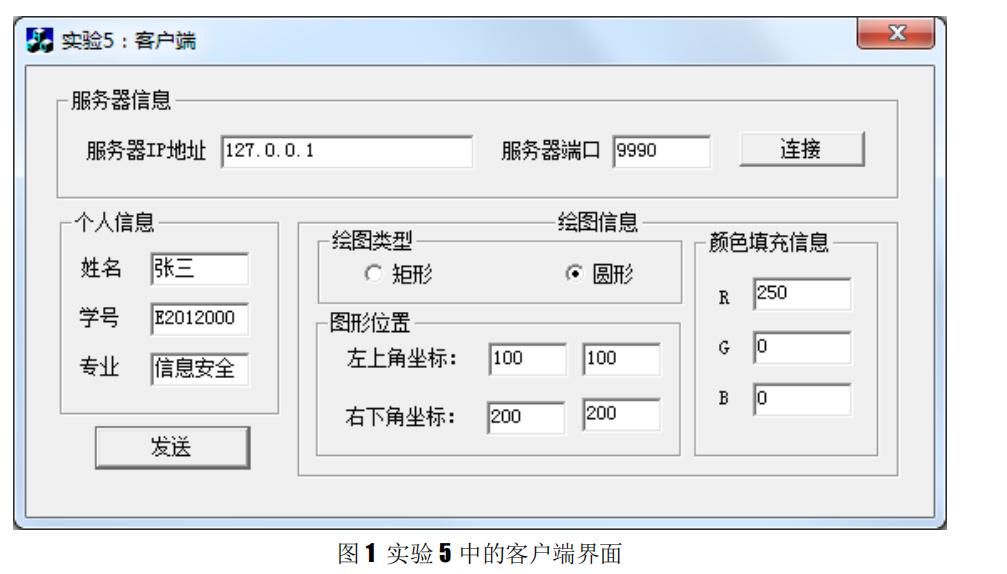
**【实验内容】**

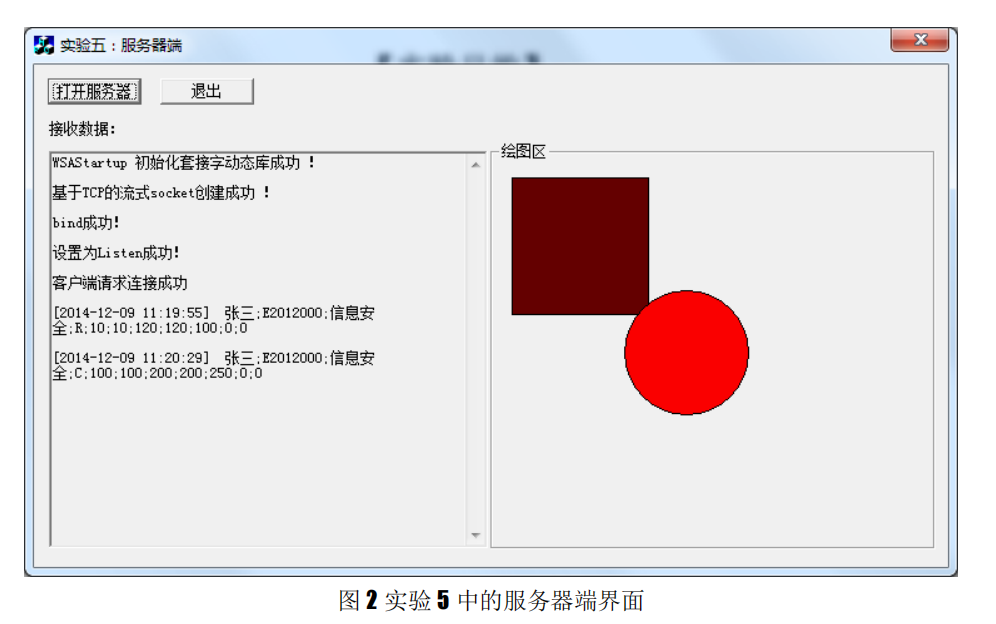
1. 分别建立服务器和客户机，以单线程/多线程方式实现客户机和服务器的连

接和通信，从客户机向服务器传输你的个人信息和绘图信息，在服务器上显示

你发送的信息并按照传输来的信息绘制图形。服务器和客户端的界面布局和功

能效果分别如图 1 和图 2 所示。





**客户端服务端通信的流程：**

采用QWebSocket实现：

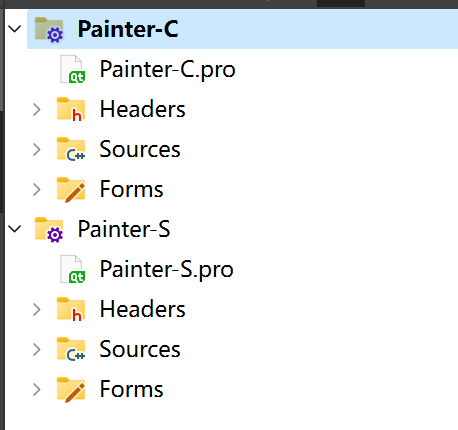
在Qt项目的工程文件中加入“QT +=websockets”

引用：

#include<QtWebSockets/QWebSocket>

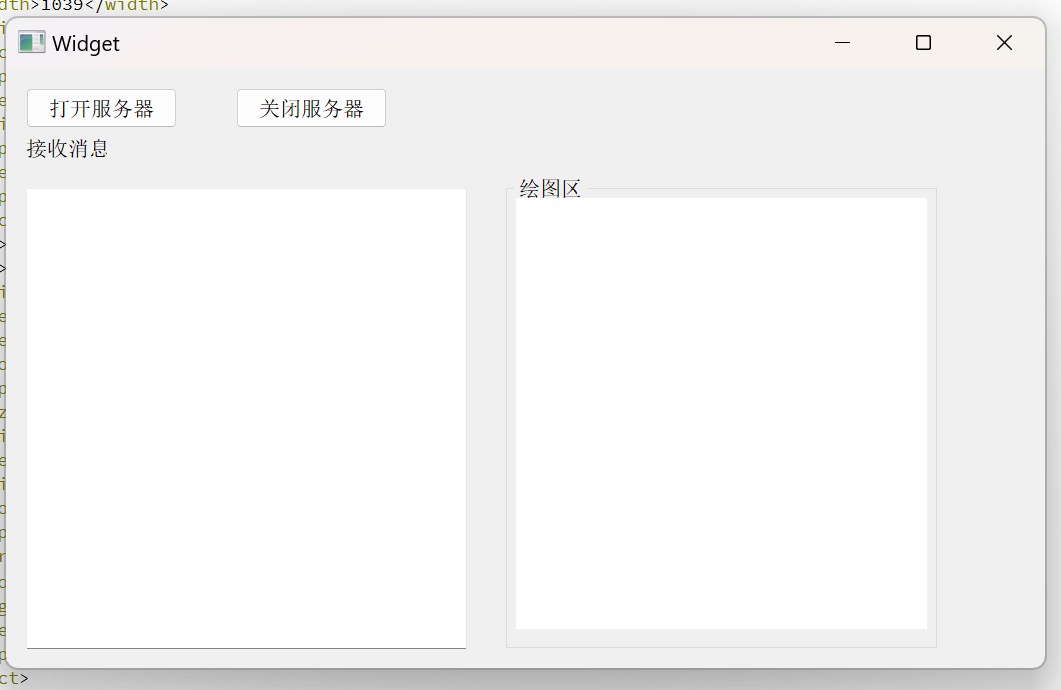
#include<QtWebSockets/QWebSocketServer>

本次实验中，一共创建了两个独立的项目，分别代表客户端与服务端

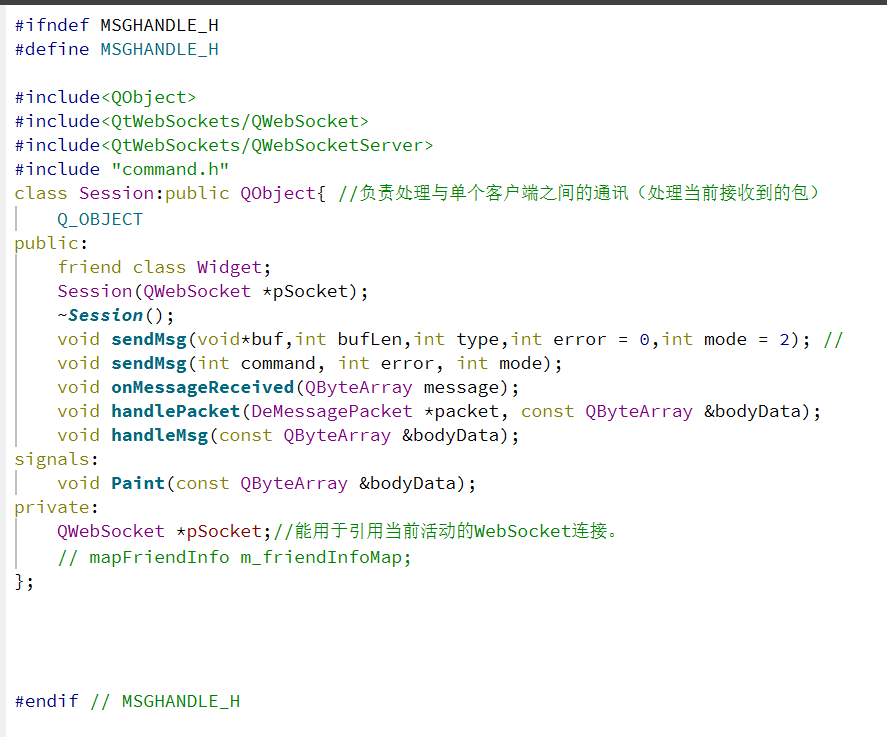


**Server端：**

Ui界面：



创建一个专门的类Session用于处理网络消息，每一个Session对象都代表一个套接字（私有成员变量pSocket实际处理网络活动），用于唯一标识某个客户端与服务端之间的连接。



创建一个QWebSocketServer对象用于开启服务端。

创建一个QWebSocket对象用于引用当前网络活动。



**服务端开启监听：**

为“打开服务器”按钮的点击信号绑定槽函数：



实例化QWebSocketServer私有成员变量，并设置监听端口为443，监听任何IPV4请求。

通过Qt的信号-槽机制，将接收到新网络请求的信号绑定给onnewconnection槽函数。



开启监听

**处理新的连接：**

从等待队列中获取一个网络活动，赋值给pSocket私有成员变量，并以pSocket作为构造函数参数创建Session对象。将Session的私有成员pSocket收到二进制消息的信号与Session::onMessageReceived函数绑定。onMessageReceived函数负责处理接收到的二进制消息。

**处理客户端发来的消息并绘制图形：**

客户端发送的消息中包含了数据与操作类型，先有实例化的Session对象接收到消息，然后再将数据以信号的形式发送到主进程，主进程将字节数组转回Json格式，根据键提取出各个值。将个人信息显示在ui控件中；

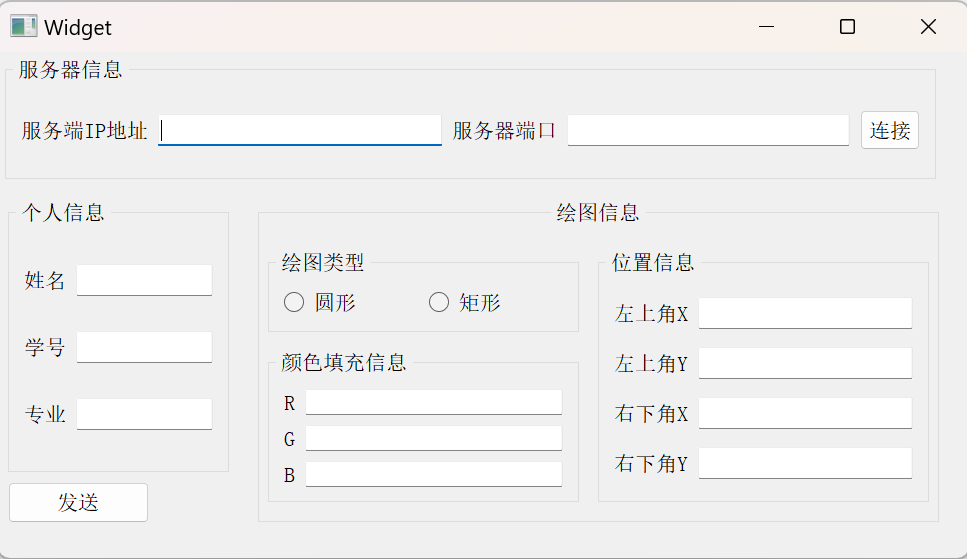
服务端的绘制逻辑是，使用一个QLabel控件，将一个QPixmap绑定到上面。QPixmap相当于一个画布，此后可以在这张画布上进行绘制。

当解析客户端发来的数据后，服务端根据颜色RGB值；图形种类；坐标，通过pixmap实例化一个QPainter对象进行绘制。



**Client：**

UI界面：



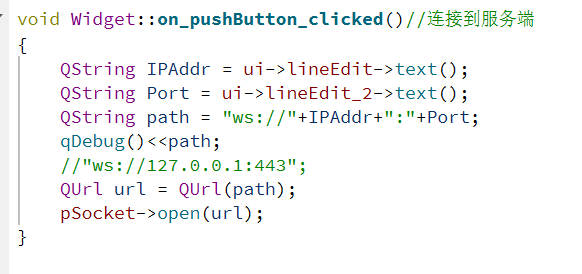
创建一个私有成员变量QWebSocket \*pSocket，用于引用当前网络活动。



**与服务端建立连接**

在“连接”按钮绑定的槽函数中，实例化pSocket成员变量，并为其绑定事件。指定服务端地址后，打开连接。（此处使用的是127.0.0.1）

当接收到连接建立成功的信号后，调用sendMsg函数向服务端发送一个参数为CommandEnum\_Prover的请求。



在“发送”按钮的点击信号绑定的槽函数中，从ui控件获取信息，封装为json格式，再转为字节数字，调用sendMsg方法发送到服务端

**网络通信方法实现与格式：**

**发送消息方法：sendMsg**

以CA为例：

void sendMsg(void\*buf,int bufLen,int type,int error = 0,int mode = 2);

接受五个参数，其中起作用的为前三个参数，分别代表数据内容，数据长度，指令类型。

通过强制类型转换，将数据最终转为QByteArray格式，与创建的包头包体凭借后，调用pSocket发送数据。

注：调用该方法时，传入的数据一般为json格式的QByteArray

注：发送数据时，需要对数据进行封装。本次实验中采用的方法是，现创建一个QJsonobj对象，对其键值对进行赋值后，转为一个QJsonDocument对象。调用其toJson方法（参数为QJsonDocument::Compact）转为QByteArray后进行发送



**接收消息：onMessageReceived：**

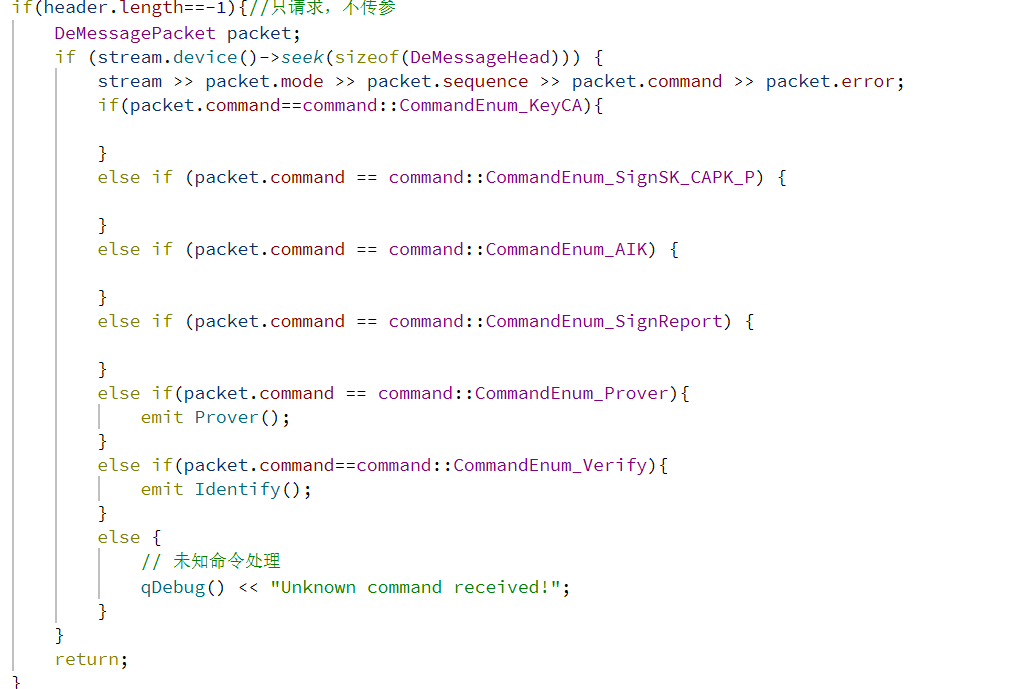
void onMessageReceived(QByteArray message);

该函数即为上文中提到的，与接收到二进制消息信号绑定的槽函数。

首先，会对接收到的数据进行格式校验



根据消息长度判断是否为请求：



若不是请求，则检查消息长度，并处理数据，调用handlePacket函数进行处理



**处理数据：handlePacket：**

根据解析出的指令，执行相应的函数，每个端的函数不同，具体功能放到后续的远程证明部分介绍

注：接收到的数据类型为QByteArray，应先将QByteArray转为QJsonDocumnet，再转为QJsonobj，根据key提取value。



**客户端代码：**

#include "widget.h"

#include "ui\_widget.h"

#include <QMessageBox>

Widget::Widget(QWidget \*parent)

: QWidget(parent)

, ui(new Ui::Widget)

{

ui->setupUi(this);

pSocket = new QWebSocket;

connect(pSocket, &QWebSocket::binaryMessageReceived, this, &Widget::onMessageReceived);

connect(pSocket, &QWebSocket::connected, this, [=]{ //!!!确保与CA端成功建立连接后再发送请求

qDebug()<<QString::fromLocal8Bit("连接到服务器");

});

// 处理失败情况

connect(pSocket, &QWebSocket::disconnected, this, [=]{

qDebug() << "WebSocket disconnected";

qDebug()<<QString::fromLocal8Bit("连接丢失");

QApplication::quit();

});

}

Widget::~Widget()

{

delete ui;

}

void Widget::onMessageReceived(QByteArray message) {//处理数据包

}

void Widget::handlePacket(DeMessagePacket \*packet, const QByteArray &bodyData){

}

void Widget::sendMsg(void\* buf, int bufLen, int type, int error, int mode) {

// 创建包头并赋值

char\* charPath = static\_cast<char\*>(buf); //强制类型转换

//QString qstrFilePath(charPath);

DeMessageHead header;

header.encoded = '0';

header.version = '0';

header.length = bufLen;

//qDebug()<<sizeof(header);

// 创建消息体

DeMessagePacket packet;

packet.mode = mode;

packet.sequence = 0;

packet.command = type;

packet.error = error;

// 将消息头和消息体序列化为 QByteArray

QByteArray messageData;

QDataStream stream(&messageData, QIODevice::WriteOnly);

stream<<header.mark[0];

stream<<header.mark[1];

stream << header.encoded;

stream << header.version;

stream << header.length;

// 序列化消息体

stream << packet.mode;

stream << packet.sequence;

stream << packet.command;

stream << packet.error;

if (buf) {

// 将用户数据复制到消息体部分

QByteArray userData(charPath);

stream << userData; // 序列化用户数据

//qDebug()<<sizeof(userData);

}

// 发送消息

qDebug()<<sizeof(messageData);

this->pSocket->sendBinaryMessage(messageData); // 假设 pSocket 是有效的 QWebSocket 指针

}

void Widget::sendMsg(int command, int error, int mode) {

// 设定消息头

DeMessageHead header;

header.encoded = '0';

header.version = '0';

header.length = -1; // length 设为 -1 表示无用户数据

qDebug() << sizeof(header);

// 创建消息体

DeMessagePacket packet;

packet.mode = mode;

packet.sequence = 0;

packet.command = command;

packet.error = error;

// 将消息头和消息体序列化为 QByteArray

QByteArray messageData;

QDataStream stream(&messageData, QIODevice::WriteOnly);

stream << header.mark[0];

stream << header.mark[1];

stream << header.encoded;

stream << header.version;

stream << header.length;

// 序列化消息体

stream << packet.mode;

stream << packet.sequence;

stream << packet.command;

stream << packet.error;

// 发送消息，无用户数据

pSocket->sendBinaryMessage(messageData); // 假设 pSocket 是有效的 QWebSocket 指针

}

void Widget::on\_pushButton\_clicked()//连接到服务端

{

QString IPAddr = ui->lineEdit->text();

QString Port = ui->lineEdit\_2->text();

QString path = "ws://"+IPAddr+":"+Port;

qDebug()<<path;

//"ws://127.0.0.1:443";

QUrl url = QUrl(path);

pSocket->open(url);

}

void Widget::on\_pushButton\_2\_clicked()//发送消息到服务器

{

//获取输入，打包为Json

QJsonObject json;

// 获取姓名

QString name = ui->lineEdit\_3->text().trimmed();

if(name.isEmpty()) {

QMessageBox::warning(this, QString::fromLocal8Bit("警告"), QString::fromLocal8Bit("姓名不能为空"));

return;

}

json["name"] = name;

// 获取学号

QString studentId = ui->lineEdit\_4->text().trimmed();

if(studentId.isEmpty()) {

QMessageBox::warning(this, QString::fromLocal8Bit("警告"), QString::fromLocal8Bit("学号不能为空"));

return;

}

json["studentId"] = studentId;

// 获取专业

QString major = ui->lineEdit\_5->text().trimmed();

if(major.isEmpty()) {

QMessageBox::warning(this, QString::fromLocal8Bit("警告"), QString::fromLocal8Bit("专业不能为空"));

return;

}

json["major"] = major;

// 获取红色RGB值

QString red = ui->lineEdit\_6->text().trimmed();

if(red.isEmpty()) {

QMessageBox::warning(this, QString::fromLocal8Bit("警告"), QString::fromLocal8Bit("红色RGB值不能为空"));

return;

}

json["red"] = red;

// 获取绿色RGB值

QString green = ui->lineEdit\_7->text().trimmed();

if(green.isEmpty()) {

QMessageBox::warning(this, QString::fromLocal8Bit("警告"), QString::fromLocal8Bit("绿色RGB值不能为空"));

return;

}

json["green"] = green;

// 获取蓝色RGB值

QString blue = ui->lineEdit\_8->text().trimmed();

if(blue.isEmpty()) {

QMessageBox::warning(this, QString::fromLocal8Bit("警告"), QString::fromLocal8Bit("蓝色RGB值不能为空"));

return;

}

json["blue"] = blue;

// 获取左上角X坐标

QString leftTopX = ui->lineEdit\_9->text().trimmed();

if(leftTopX.isEmpty()) {

QMessageBox::warning(this, QString::fromLocal8Bit("警告"), QString::fromLocal8Bit("左上角X坐标不能为空"));

return;

}

json["leftTopX"] = leftTopX;

// 获取左上角Y坐标

QString leftTopY = ui->lineEdit\_10->text().trimmed();

if(leftTopY.isEmpty()) {

QMessageBox::warning(this, QString::fromLocal8Bit("警告"), QString::fromLocal8Bit("左上角Y坐标不能为空"));

return;

}

json["leftTopY"] = leftTopY;

// 获取右下角X坐标

QString rightBottomX = ui->lineEdit\_11->text().trimmed();

if(rightBottomX.isEmpty()) {

QMessageBox::warning(this, QString::fromLocal8Bit("警告"), QString::fromLocal8Bit("右下角X坐标不能为空"));

return;

}

json["rightBottomX"] = rightBottomX;

// 获取右下角Y坐标

QString rightBottomY = ui->lineEdit\_12->text().trimmed();

if(rightBottomY.isEmpty()) {

QMessageBox::warning(this, QString::fromLocal8Bit("警告"), QString::fromLocal8Bit("右下角Y坐标不能为空"));

return;

}

json["rightBottomY"] = rightBottomY;

//获取图形种类

if(ui->radioButton->isChecked())

json["shape"]= QString::fromLocal8Bit("圆形");

else

json["shape"]=QString::fromLocal8Bit("矩形");

QJsonDocument jsonDoc(json);

QByteArray byteArray = jsonDoc.toJson(QJsonDocument::Compact);

sendMsg(byteArray.data(), byteArray.size(), CommandEnum\_Paint, 0, 0);

}

**【实验总结】**

**Qt网络通信：**

本次实验中使用的是QWebSocket库以及QWebSocketServer库。分别用户与客户端与服务端。服务端指定监听地址和端口后，通过listen方法开始监听。客户端在指定服务端IP,端口后，可调用open方法连接到服务端。双发之间的通信可借助套接字的binaryMessageReceived信号，表示接收到二进制数据。具体的数据格式可由通信双方自定义。

**QPixmap的使用：**

在上一次的实验（实验三）中，我采用的绘制方法是创建一个类并重写paintEvent虚函数。这样的好处是封装了绘制部分的代码，提高了程序的可维护性。本次实验采用QPixmap，将其作为一个控件的画布，在此基础上进行绘制。好处是实现简单，可以直接创建QPainter对象进行绘制。缺点是当绘图区域改变时，需要调节QPixmap与控件的属性，不利于长期维护。