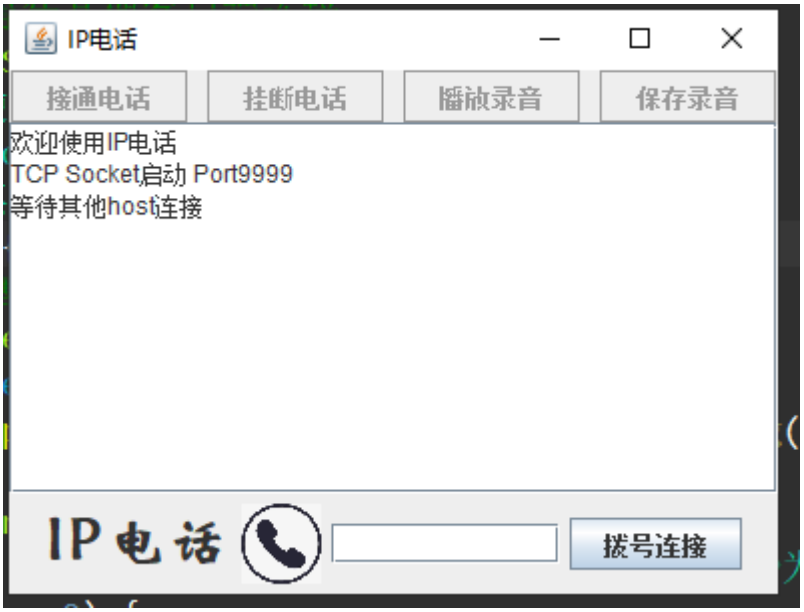
计算机网络课内实验报告

姓名	陈进前
学号	2176112788
应用	IP 电话

一、 应用简述

本应用是一款 IP 电话，在同一局域网下的两个主机可以进行相互通话。其具体功能有：通过 IP 地址拨号通话，通话录音保存。通话时，双方先建立 TCP 连接，以发送控制信息，如接受通话请求，挂断电话等，然后再通过 UDP 发送数据报，以实现实时通话。在通话时，先通过麦克风，将音频信号捕捉存入数据缓冲中，然后使用 UDP 协议，发送至对方的 socket。经过局域网测试，可以完成通话。



二、 主类、内部类、关键方法描述。

该应用的主类为 PhoneCall，一个通话只实例化一个 PhoneCall 对象。在主类中定义了 Swing 组件，TCP 连接的输出流（用于控制启动和结束通话），UDPSocket（端口号为 9000，用于发送数据），接收缓冲 buf，双方通信的录音格式 AudioFormat，读取音频数据的 TargetDataLine，（ TargetDataLine 接口提供从目标数据行的缓冲区读取所捕获数据的方法），源音频数据 SourceDataLine（可以充当混频器的源，将字节写入数据行），以及字节数组输入输出流 ByteArrayInputStream bais，ByteArrayOutputStream baos，音频输入流 AudioInputStream ais。

内部类 Record，其实现了 Runnable 接口，可以作为线程加入线程池执行。其主要功能为定义存放录音的字节数组 bts，循环执行，先将 bts 中的数据写入 baos（用于将来的录音播放及保存），然后再将 bts 中的数

据不断发送到目标 ip 的端口号 3000。

内部类 play，同样实现了 Runnable 接口，可以作为线程加入线程池，主要功能是将 bts 中的数据写入源数据行 sd 中，然后混频播放。

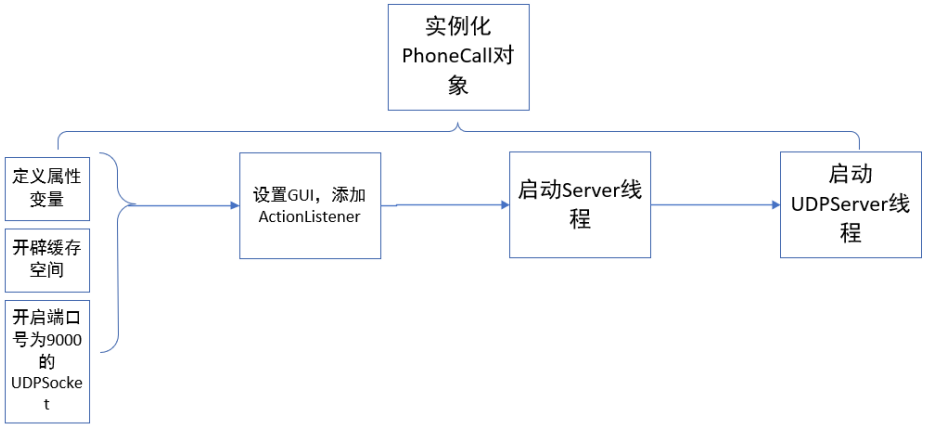
内部类 Server，实现了 Runnable 接口，可以作为线程加入线程池并执行。其主要功能是等待其他主机拨号。该内部类会创建一个端口号为 9999（默认，也可自己指定）的 TCP socket。随后一直阻塞，直到有人连接，连接后获取该主机的 IP，并选择是否与其进行通话。若通话接通，则调用 Capture 方法，开始捕捉麦克风的音频信号，并将执行 Record 线程不断发送数据到目的主机的 3000 端口。若拒绝通话，则断开连接，等待其他人再次连接。

内部类 Call，实现了 Runnable 接口，可以作为线程加入线程池并执行，其主要功能是 TCP 拨号（用于连接到其他主机的 9999 端口请求与其通话）。

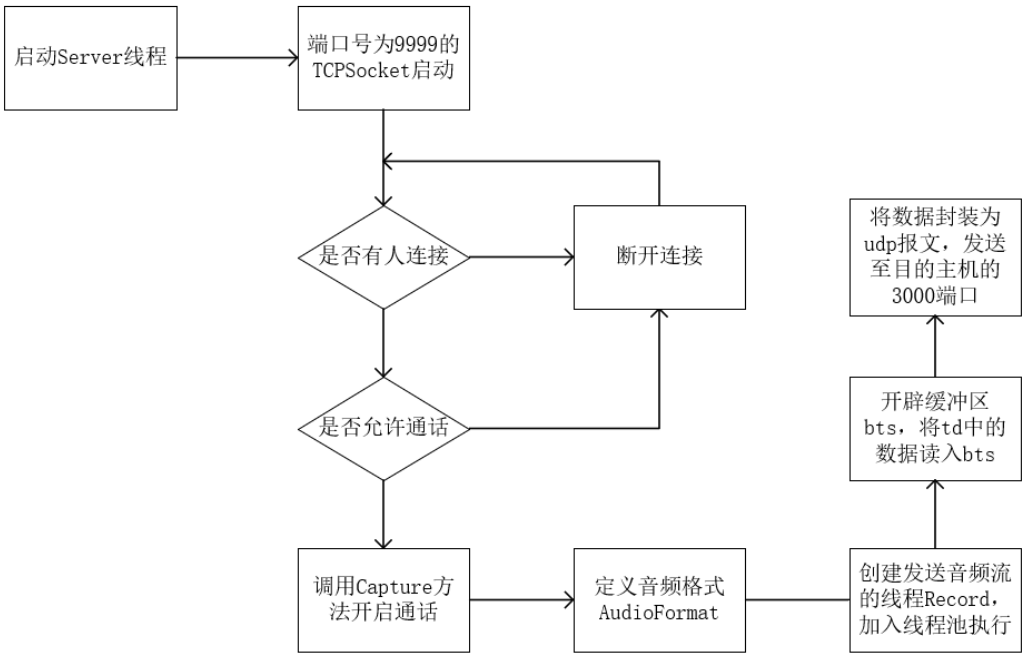
内部类 UDPServer，实现了 Runnable 接口，可以作为线程加入线程池并执行，当通话允许后，开启此线程，创建 DatagramSocket，端口号为 3000，接收另一主机 UDP 发送的报文，并调用 play()方法将字节数组写入 SourceDataline，进行混频播放。

三、 执行过程分析

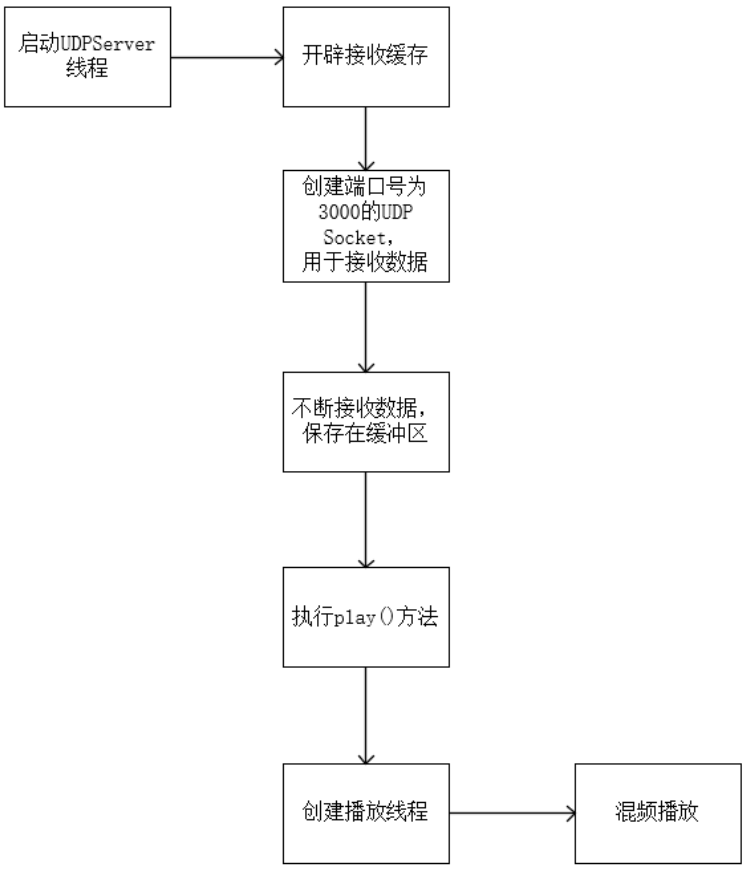
应用启动时，实例化 PhoneCall 对象。



启动 Server 线程的一系列操作：



启动 UDPServer 线程的一系列操作：



四、 缺点与展望

该应用主体功能基本完善，但由于时间和技术能力的限制，仍然存在着许多缺点有待日后进一步改善。如：

① play(byte[] data)方法用于播放收到的音频数据，执行时会创建一个新的 play 线程。而 play 方法是在 udpserver 线程中调用执行的，位于 while 循环之内，在不断循环的过程中，不断地再删除原有线程，创建新的线程，是一种系统资源的浪费，也可能造成实时性下降的问题。

② 通过 TCP 连接来建立通话，然后通过 UDP 发送数据，思路方向没错。但是如果存在恶意阻塞的情况，即某主机不断要求通话，就会造成本机的 9999 端口一直被反复连接，别人打不进来电话。

③ UDP 数据报文直接明文传输，未被加密。极其容易造成通话信息的泄漏。

④ 该应用只适合于局域网使用，在网络上由于 NAT 协议的存在，不清楚目标主机的 IP 地址，所以无法直接点对点进行通话。

针对以上缺点与不足，提出如下展望：

① 更改 udpserver 线程的调用逻辑，避免重创建同一播放线程。

② 增加黑名单功能，将恶意连接阻塞在防火墙之外。

③ 报文在传输前应该进行加密，接收后进行解密。由于本应用是实时性较强的应用，数据流式传输，可以考虑使用 CTR 等方法进行高速流加密。

④ 为穿越 NAT 层，可以考虑设置第三方服务器进行转发。

五、 程序源码

```
import javax.sound.sampled.*;
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.*;
import java.net.*;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
```

```
public class PhoneCall extends JFrame implements ActionListener {

    //定义所需要的组件
    JPanel jp1,jp3;
    JScrollPane jp2;
    JLabel jl1=null;
    JLabel pic;
    JButton captureBtn,stopBtn,playBtn,saveBtn, tcp_btn;
    JTextField jtextIp;
    static JTextArea jt_aArea=new JTextArea();

    // 新建线程池来处理 拨号、连接、播放音频等线程
    ExecutorService cachedThreadPool = Executors.newCachedThreadPool();

    // 电话类定义
    PhoneCall mr;

    // TCP 连接的输出流
    static PrintWriter pw=null;

    // 是否接听电话      volatile 关键字保证多线程下变量能修改的值能被及时地更新以便其他线程读取
    volatile int onPhone = -1;

    // 是否成功建立 TCP 连接的标志
    volatile static boolean is_Connected = false;

    //客户端在 9000 端口监听接收到的数据，这是 UDP 发送的端口    datagram Java 规定的数据报格式
    DatagramSocket ds = new DatagramSocket(9000);
```

```
//获取本机 IP 地址+
InetAddress loc = InetAddress.getLocalHost();

// 建立接收缓冲
byte[] buf = new byte[1024];

//定义录音格式
AudioFormat af = null;

//定义目标数据行,可以从中读取音频数据,该 TargetDataLine 接口提供从目标数据行的缓冲区读取所捕获数据的方法。
TargetDataLine td = null;

//定义源数据行,源数据行是可以写入数据的数据行。它充当其混频器的源。应用程序将音频字节写入源数据行，这样可处理字节缓冲并将它们传递给混频器。
SourceDataLine sd = null;

//定义字节数组输入输出流
ByteArrayInputStream bais = null;
ByteArrayOutputStream baos = null;

//定义音频输入流
AudioInputStream ais = null;

//定义停止录音的标志，来控制录音线程的运行
Boolean stopflag = false;

//构造函数
public PhoneCall() throws SocketException, UnknownHostException {
```

```
        //组件初始化
        jp1 = new JPanel();
        jp2 = new JScrollPane(jt_aArea);
        jp3 = new JPanel();

        //定义字体
        Font myFont = new Font("华文新魏",Font.BOLD,30);
        jl1 = new JLabel("IP 电话");
        jl1.setFont(myFont);
        jp1.add(jl1);

        pic = new JLabel();
        ImageIcon img = new ImageIcon("D:\\Phone.jpg");

img.setImage(img.getImage().getScaledInstance(40,40,Image.SCALE_DEFAULT));
        pic.setIcon(img);
        // 定义输入 IP 地址，尝试用 TCP 连接
        jtextIp = new JTextField(10);
        // 拨号按钮
        tcp_btn = new JButton("拨号连接");
        tcp_btn.addActionListener(this);
        tcp_btn.setActionCommand("tcp_btn");

        jp1.add(pic);
        jp1.add(jtextIp);
        jp1.add(tcp_btn);


        jt_aArea.setEditable(false);
```

```
// 下方按钮
captureBtn = new JButton("接通电话");
//对开始录音按钮进行注册监听
captureBtn.addActionListener(this);
captureBtn.setActionCommand("captureBtn");
//对停止录音进行注册监听
stopBtn = new JButton("挂断电话");
stopBtn.addActionListener(this);
stopBtn.setActionCommand("stopBtn");
//对播放录音进行注册监听
playBtn = new JButton("播放录音");
playBtn.addActionListener(this);
playBtn.setActionCommand("playBtn");
//对保存录音进行注册监听
saveBtn = new JButton("保存录音");
saveBtn.addActionListener(this);
saveBtn.setActionCommand("saveBtn");



this.add(jp1,BorderLayout.SOUTH);
this.add(jp2,BorderLayout.CENTER);
this.add(jp3,BorderLayout.NORTH);
jp3.setLayout(null);
jp3.setLayout(new GridLayout(1, 4,10,10));
jp3.add(captureBtn);
jp3.add(stopBtn);
jp3.add(playBtn);
jp3.add(saveBtn);
//设置按钮的属性
captureBtn.setEnabled(false);
```

```
        stopBtn.setEnabled(false);
        playBtn.setEnabled(false);
        saveBtn.setEnabled(false);
        //设置窗口的属性
        this.setSize(400,300);
        this.setTitle("IP 电话");
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setLocationRelativeTo(null);
        this.setVisible(true);

        // 开始监听自己的 9999 端口，等待其他用户连接
        Server server = new Server();
        // 将自己本机连接的线程放到线程池中运行
        cachedThreadPool.execute(server);

        // UDP 接收端口
        UDPServer udpServer = new UDPServer();
        cachedThreadPool.execute(udpServer);
    }

    public static void main(String[] args) throws SocketException,
UnknownHostException {

        //创造一个客户端实例
        PhoneCall mr = new PhoneCall();

    }

    public void actionPerformed(ActionEvent e) {
```

```
if(e.getActionCommand().equals("captureBtn"))
{
    //点击开始录音按钮后的动作
    //停止按钮可以启动
    captureBtn.setEnabled(false);
    stopBtn.setEnabled(true);
    playBtn.setEnabled(false);
    saveBtn.setEnabled(false);

    // 获取对方的对方的 IP 并且向这个 IP 发送音频流数据
    String ip = jtextIp.getText();
    try {
        InetAddress to_ip = InetAddress.getByName(ip);
        //调用录音的方法
        capture(to_ip);
    } catch (UnknownHostException e1) {
        e1.printStackTrace();
    }

}else if (e.getActionCommand().equals("stopBtn")) {
    //点击停止录音按钮的动作
    captureBtn.setEnabled(true);
    stopBtn.setEnabled(false);
    playBtn.setEnabled(true);
    saveBtn.setEnabled(true);
    //调用停止录音的方法，停止向对方发送数据
    stop();

}else if(e.getActionCommand().equals("playBtn"))
{
```

```
                //调用播放录音的方法
                play();
            }else if(e.getActionCommand().equals("saveBtn"))
            {
                //调用保存录音的方法
                save();
            }else if(e.getActionCommand().equals("tcp_btn"))
            {
                //  对输入的 IP 拨号
                String ip = jtextIp.getText();
                jt_aArea.append("尝试连接"+ip+"\r\n");
                //  在线程池中开启线程来拨号
                Call call = new Call(ip);
                cachedThreadPool.execute(call);
            }

        }
        //开始通话，捕捉 麦克风 获取音频流的数据
        public void capture()
        {
            try {
                //af 为 AudioFormat 也就是音频格式
                af = getAudioFormat();
                DataLine.Info info = new DataLine.Info(TargetDataLine.class,af);
                td = (TargetDataLine)(AudioSystem.getLine(info));
                //打开具有指定格式的行，这样可使行获得所有所需的系统资源并变
得可操作。
                td.open(af);
                //允许某一数据行执行数据 I/O
                td.start();
```

```
                //创建播放录音的线程
                Record record = new Record();
                cachedThreadPool.execute(record);

            } catch (LineUnavailableException ex) {
                ex.printStackTrace();
                return;
            }
        }

        //开始通话，捕捉 麦克风 获取音频流的数据
        public void capture(InetAddress ip)
        {
            try {
                //af 为 AudioFormat 也就是音频格式
                af = getAudioFormat();
                DataLine.Info info = new DataLine.Info(TargetDataLine.class,af);
                td = (TargetDataLine)(AudioSystem.getLine(info));
                //打开具有指定格式的行，这样可使行获得所有所需的系统资源并变
得可操作。
                td.open(af);
                //允许某一数据行执行数据 I/O
                td.start();

                //创建发送音频流的线程
                Record record = new Record(ip);
                cachedThreadPool.execute(record);

            } catch (LineUnavailableException ex) {
```

```
                ex.printStackTrace();
                return;
            }
        }

        //停止录音
        public void stop()
        {
            stopflag = true;
        }

        //播放录音,这是本机的录音，保存到缓存中
        public void play()
        {
            //将 baos 中的数据转换为字节数据
            byte audioData[] = baos.toByteArray();
            //转换为输入流
            bais = new ByteArrayInputStream(audioData);
            af = getAudioFormat();
            ais = new AudioInputStream(bais, af, audioData.length/af.getFrameSize());

            try {
                DataLine.Info dataLineInfo = new DataLine.Info(SourceDataLine.class,
af);
                sd = (SourceDataLine) AudioSystem.getLine(dataLineInfo);
                sd.open(af);
                sd.start();
                //创建播放进程
                Play py = new Play();
                cachedThreadPool.execute(py);
```

```
        } catch (Exception e) {
            e.printStackTrace();
        }finally{
            try {
                //关闭流
                if(ais != null)
                {
                    ais.close();
                }
                if(bais != null)
                {
                    bais.close();
                }
                if(baos != null)
                {
                    baos.close();
                }

            } catch (Exception e) {
                e.printStackTrace();
            }
        }
}

// 重载 play 方法，这是播放从 UDP 报文中获取到的报文
public void play(byte[] audioData)
{
        //转换为输入流
        bais = new ByteArrayInputStream(audioData);
        af = getAudioFormat();
```

```
            ais = new AudioInputStream(bais, af, audioData.length/af.getFrameSize());

            try {
                DataLine.Info dataLineInfo = new DataLine.Info(SourceDataLine.class,
af);
                sd = (SourceDataLine) AudioSystem.getLine(dataLineInfo);
                sd.open(af);
                sd.start();
                //创建播放进程
                Play py = new Play();
                cachedThreadPool.execute(py);
            } catch (Exception e) {
                e.printStackTrace();
            }finally{
                try {
                    //关闭流
                    if(ais != null)
                    {
                        ais.close();
                    }
                    if(bais != null)
                    {
                        bais.close();
                    }

                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
```

```
}

// 保存自己的录音
public void save()
{
    //取得录音输入流
    af = getAudioFormat();

    byte audioData[] = baos.toByteArray();
    bais = new ByteArrayInputStream(audioData);
    ais = new AudioInputStream(bais,af, audioData.length / af.getFrameSize());
    //定义最终保存的文件名
    File file = null;
    //写入文件
    try {
        //以当前的时间命名录音的名字
        //将录音的文件存放到 F 盘下语音文件夹下
        File filePath = new File("./audio");
        if(!filePath.exists())
        {//如果文件不存在，则创建该目录
            filePath.mkdir();
        }
        file = new File(filePath.getPath()+"/"+System.currentTimeMillis()+".mp3");
        AudioSystem.write(ais, AudioFileFormat.Type.WAVE, file);
    } catch (Exception e) {
        e.printStackTrace();
    }finally{
        //关闭流
        try {
```

```
                if(bais != null)
                {
                    bais.close();
                }
                if(ais != null)
                {
                    ais.close();
                }
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
}
//设置 AudioFormat 的参数
public AudioFormat getAudioFormat()
{
    //下面注释部分是另外一种音频格式，两者都可以
    AudioFormat.Encoding encoding = AudioFormat.Encoding.
            PCM_SIGNED ;
    float rate = 8000f;
    int sampleSize = 16;
    String signedString = "signed";
    boolean bigEndian = true;
    int channels = 1;
    return new AudioFormat(encoding, rate, sampleSize, channels,
            (sampleSize / 8) * channels, rate, bigEndian);
}
//录音类，因为要用到 PhoneCall 类中的变量，所以将其做成内部类
// 此时，我们将捕捉到的音频字符数组通过 UDP 发送
```

```
class Record implements Runnable
{
    InetAddress ip;
    Record() {}

    Record(InetAddress ip) {
        this.ip = ip;
    }

    //定义存放录音的字节数组,作为缓冲区
    byte bts[] = new byte[10000];
    //将字节数组包装到流里，最终存入到 baos 中
    //重写 run 函数
    public void run() {
        System.out.println("开始录音");
//          System.out.println(loc);
        // 定义发送的数据报
        DatagramPacket dp_send = new DatagramPacket(bts,1000, ip, 3000);
        DatagramPacket dp_receive = new DatagramPacket(buf, 1024);

        baos = new ByteArrayOutputStream();
        try {
            System.out.println("ok3");
            stopflag = false;
            int i=0;
            while(stopflag != true)
            {
                i++;
                //  当停止通话没按下时，该线程一直执行
                //  从数据行的输入缓冲区读取音频数据到
```

```
                    //  要读取 bts.length 长度的字节,cnt 是实际读取的字节数
                    //  将 td 输入音频流中缓冲区的内容读取到 bts 缓存中
                    int cnt = td.read(bts, 0, bts.length);
                    if(cnt > 0)
                    {
                        // 从数据线的输入缓冲区读取音频数据到 输出流 baos 中
                        // 下面这一句是为了重放录音的时候，将从 td 中捕捉到的字符流保存到 baos 中
                        baos.write(bts, 0, cnt);
                        // 将 bts 数组，也就是从 td 中读取到的音频流缓冲通过 UDP 发送
                        // 这个时候收到的内容应该是一个 byte[] 数组
                        ds.send(new DatagramPacket( bts, cnt, ip, 3000));
                    }

                }
            } catch (Exception e) {
                e.printStackTrace();
            }finally{
                try {
                    //关闭打开的字节数组流
                    if(baos != null)
                    {
                        baos.close();
                    }
                } catch (IOException e) {
                    e.printStackTrace();
                }finally{
                    td.drain();
```

```
                    td.close();
                }
            }
        }

    }
    //播放类,同样也做成内部类
    class Play implements Runnable
    {
        //播放baos中的数据即可,我们这个时候将所有的数据都保存到了 baos 中
了
        public void run() {
            byte bts[] = new byte[10000];
            try {
                int cnt;
                //读取数据到缓存数据
                while ((cnt = ais.read(bts, 0, bts.length)) != -1)
                {
                    if (cnt > 0)
                    {
                        //写入缓存数据
                        //将音频数据写入到混频器
                        sd.write(bts, 0, cnt);
                    }
                }
            } catch (Exception e) {
                e.printStackTrace();
            }finally{
                sd.drain();
                sd.close();
```

```
            }
        }
    }

    //  将本机作为服务端，开启 9999 端口等待连接
    //  接听其他人拨打的电话
    class Server implements Runnable
    {
        // 默认开启 9999 端口
        private int port = 9999;
        Server() {}
        // 可以指定端口
        Server(int port) {
            this.port = port;
        }
        public void run() {
            try {
                // 开启本机 9999 端口等待其他客户端连接
                ServerSocket ss = new ServerSocket(port);
                jt_aArea.append("欢迎使用 IP 电话"+"\r\n");
                jt_aArea.append("TCP Socket 启动"+" Port"+port+"\r\n");
                jt_aArea.append("等待其他 host 连接"+"\r\n");
                while (!is_Connected) {
                    // 这个方法将会阻塞，一直等待其他的用户连接，然后才会
继续进行
                    Socket s =  ss.accept();
                    synchronized (Server.class) {
                        String other_ip =
s.getRemoteSocketAddress().toString().split(":")[0].substring(1);
                        InetAddress ip = InetAddress.getByName(other_ip);
```

```
                    System.out.println("对方的 IP 是"+other_ip+"\r\n");
                    int is_connect = JOptionPane.showConfirmDialog(mr,"IP
为"+ip+"的用户呼叫您","是否接听电话",0);
                    if(is_connect == 0) {
                        // 确定接听
                        // 修改接听变量
                        is_Connected = true;

                        System.out.println("这个时候选择接听对方电话");
                        jt_aArea.append("已接听"+ip+"的用户电话\r\n");
                        // 这个时候获取本地的音频,向对方发送消息
                        capture(ip);
                    }else {
                        // 拒绝对方电话
                        System.out.println("挂断对方电话");
                        jt_aArea.append("已拒绝"+ip+"的用户电话\r\n");
                        // 向对方发送 goodbye 让对方直到被拒绝
                        OutputStream out = s.getOutputStream();
                        out.write("goodbye".getBytes());
                        out.flush();
                        s.shutdownInput();
                        s.shutdownOutput();
                    }
                }
                // 当本次通话结束之后，又重新监听
                System.out.println("结束一次");
            }


        } catch (IOException e) {
```

```
                e.printStackTrace();
            }

        }
    }

    // TCP 拨号连接线程,这是作为拨号端,新建线程进行拨号
    // TCP  拨号，单独用一个线程来拨号
    class Call implements Runnable
    {
        private String ip;
        Call(String ip) {
            this.ip = ip;
        }
        //  拨号
        public void run() {
            try {
                jt_aArea.append("尝试连接"+ip+"\r\n");
                //  默认连接对方的  9999  端口,这是规定的每个人的 TCP 端口
                //  只有连接成功，才能进行下一步的  UDP  传送数据
                Socket s=new Socket(ip,9999);
                InputStreamReader isr=new InputStreamReader(s.getInputStream());
                BufferedReader br=new BufferedReader(isr);
                pw=new PrintWriter(s.getOutputStream(),true);
                jt_aArea.append("连接成功"+ip+"\r\n");
                InetAddress myip = InetAddress.getLocalHost();
                pw.println(myip+"向您发起连接\r\n");

                // TCP  连接成功后，才能开始通话,这里是
                is_Connected = true;
```

```
                captureBtn.setEnabled(true);
            } catch (UnknownHostException e) {
                jt_aArea.append("IP 输入有误！请检查后重新输入！"+"\r\n");
                System.out.println("IP 输入有误！请检查后重新输入！");
            }
            catch (NoRouteToHostException e) {
                jt_aArea.append("IP 输入不可达！请检查后重新输入！"+"\r\n");
                System.out.println("IP 输入不可达！请检查后重新输入！");
            }
            catch (Exception e) {
                e.printStackTrace();
                // 出现异常，则断开连接标志
                is_Connected = false;
                captureBtn.setEnabled(false);
            }
        }
    }


    // UDP 接收服务
    // 现在已经能向某个用户连接并且发送音频了
    class UDPServer implements Runnable {

        // 当收到连接之后，开启此线程，然后从这里获取 UDP 发送的报文
        // 同时，从自己本机捕捉音频，将音频输出流转换为字符数组然后通过
UDP 发送
        public void run() {
            String str_send = "Hello UDPclient";
            byte[] buf = new byte[10000];
            //服务端在 3000 端口监听接收到的数据
```

```
// 客户端接收 UDP 的端口
DatagramSocket ds_receive = null;
try {
    ds_receive = new DatagramSocket(3000);
} catch (SocketException e) {
    e.printStackTrace();
}

//接收从客户端发送过来的数据
DatagramPacket dp_receive = new DatagramPacket(buf, 10000);
System.out.println("server is on，waiting for client to send data......");
boolean f = true;
int i=0;
while(f){
    try{
        //服务器端接收来自客户端的数据
        ds_receive.receive(dp_receive);
        // 这里是接听之后，才选择接听消息，不然就一直循环在这里，用 volatile 变量来同步
        while (!is_Connected) {}
        System.out.println("第："+i++ + "次收到 UDP 数据报，说明确实是按照缓冲来发送的，以下是接收到的字符流");
        byte[] byte_receive = dp_receive.getData();
        play(byte_receive);
        //数据发动到连接端的 9000 端口
        DatagramPacket dp_send= new DatagramPacket(str_send.getBytes(),str_send.length(),dp_receive.getAddress(),9000);
        try {
            ds_receive.send(dp_send);
        } catch (IOException e) {
```

```
                    e.printStackTrace();
                }
                dp_receive.setLength(10000);
            }catch (IOException e) {
                e.printStackTrace();
            }

        }
        ds.close();
    }
  }
}
```