|  |  |  |  |
| --- | --- | --- | --- |
|  | 2017/5/14 |  | |
|  | |  | |
| 计算机网络实验报告  *基于Java开发的IP Phone* | | | |
|  |  | |  |
|  |  | | 软件42 2141601030 欧阳鹏程 |

计算机网络实验报告

基于Java开发的IP Phone

# 实验题目

实现一个IP Phone，使用TCP进行拨号，UDP进行语音通信。

# 实验目的和要求

## 实验目的

编程实践进行网络通信，熟悉TCP编程和UDP编程。

## 实验要求

使用TCP进行拨号，UDP进行语音通信。

# 实验分析

对于实验要求中的使用TCP进行拨号，而后使用UDP进行语音通信；首先对于语音通话而言，使用UDP肯定是毋庸置疑的，而先用TCP进行拨号则是规定了必须先使用TCP进行通信才能使用UDP进行通信，不可以直接进行UDP的通信；所以我设计了必须先通过输入合法的IP和端口号进入文字发送界面，而发送文本信息是基于TCP通信的，所以另一方面，这也完成了通过TCP拨号的一个过程；而之后在进入文字聊天界面之后才能再进行语音通信（UDP通信），对于Buttons和合法输入的控制都在前端进行控制。

# 实验环境

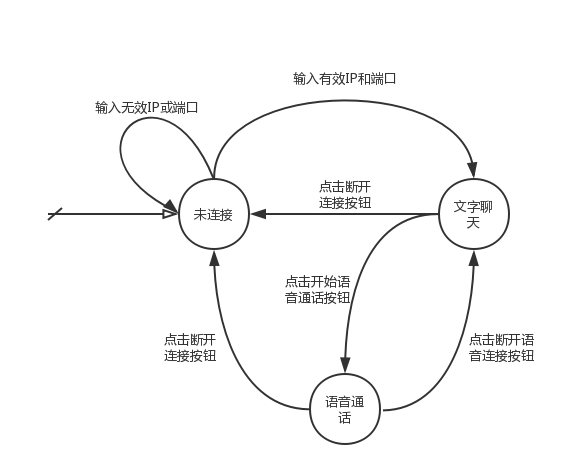
Windows 10

Intellij IDEA 2016.2.2

Eclipse

JDK1.8.0\_60

# IP Phone状态机



# 实验源码

## 前端

### 主要框架MainFrame.java：

package frontEnd;

​

import javax.swing.\*;

import java.awt.\*;

​

/\*\*

\* Created by wtupc96 on 2017/4/10.

\*/

class MainFrame extends JFrame implements Runnable {

  private MainFrame() {

      SwingUtilities.invokeLater(this);

  }

​

  public static void main(String[] args) {

      new MainFrame();

  }

​

  @Override

  public void run() {

      this.setLayout(new BorderLayout(5, 5));

      this.setDefaultCloseOperation(EXIT\_ON\_CLOSE);

      this.setVisible(true);

      this.setTitle("Computer Network: 10005");

      this.setLocationRelativeTo(null);

      this.setSize(300, 600);

      this.setResizable(false);

​

      setJMenuBar(new TopMenu());

      add(new ChattingRoom(), BorderLayout.CENTER);

      add(new MultiFuncMenu(), BorderLayout.SOUTH);

  }

}

### 顶层菜单TopMenu.java

package frontEnd;

​

import backEnd.Chat;

import backEnd.TextMessageReceiver;

import backEnd.TextMessageSender;

​

import javax.swing.\*;

import java.io.IOException;

import java.util.Date;

​

import static frontEnd.MultiFuncMenu.\*;

​

/\*\*

\* Created by wtupc96 on 2017/4/10.

\*/

class TopMenu extends JMenuBar {

  private static final JMenuItem jBreakMenuItem = new JMenuItem("连接");

  private static final JMenuItem jQuitMenuItem = new JMenuItem("退出");

  private static Chat chat;

  private static TextMessageSender textMessageSender;

  private static TextMessageReceiver textMessageReceiver;

  private static String input = "";

  private static boolean flag = false;

  private static Thread sendThread;

  private static Thread receiveThread;

​

  static {

      jQuitMenuItem.addActionListener(e -> {

          if (JOptionPane.

                  showConfirmDialog(null, "你确定退出吗?", "退出", JOptionPane.YES\_NO\_OPTION) == 0) {

              System.exit(0);

          }

      });

​

      jBreakMenuItem.addActionListener(e -> {

          if (jBreakMenuItem.getText().equals("连接")) {

              input = JOptionPane.showInputDialog("请输入你想要连接的IP地址和端口号：");

              if (input != null &&

                      input.matches("^(\\d{1,3}\\.){3}\\d{1,3}:\\d{1,10}$")) {

                  chat = new Chat(input);

​

                  try {

                      textMessageSender = new TextMessageSender(input);

                      flag = true;

                      sendThread = new Thread(new Runnable() {

                          String message;

​

                          @Override

                          public void run() {

                              while (true) {

                                  if ((message = textMessageSender.receiveMessage()) == null) {

                                      break;

                                  }

                                  ChattingRoom.getjChattingRoomTextField().append(new Date().toString() + "   " + input + "说：\n" + message + "\n\n");

                              }

                              sendThread.interrupt();

                              jBreakMenuItem.doClick();

                          }

                      });

​

                      sendThread.start();

                  } catch (IOException e1) {

                      receiveThread = new Thread(new Runnable() {

                          String message;

​

                          @Override

                          public void run() {

                              textMessageReceiver = new TextMessageReceiver();

                              flag = false;

                              while (true) {

                                  if ((message = textMessageReceiver.receiveMessage()) == null) {

                                      break;

                                  }

                                  ChattingRoom.getjChattingRoomTextField().append(new Date().toString() + "   " + input + "说：\n" + message + "\n\n");

                              }

                              receiveThread.interrupt();

                              jBreakMenuItem.doClick();

                          }

                      });

​

                      receiveThread.start();

                  }

​

                  getjTalkButton().setEnabled(true);

                  getjVideoButton().setEnabled(true);

                  getjMessageTextArea().setEnabled(true);

​

                  jBreakMenuItem.setText("断开连接");

              } else {

                  JOptionPane.showMessageDialog(null, "请输入合法的IP地址和端口号！");

              }

          } else {

              chat.soundStop();

​

              JButton jTalkButton = getjTalkButton();

              JButton jVideoButton = getjVideoButton();

              jTalkButton.setEnabled(false);

              jTalkButton.setText("开始语音");

              jVideoButton.setEnabled(false);

              jVideoButton.setText("开始视频");

              getjMessageTextArea().setEnabled(false);

​

              jBreakMenuItem.setText("连接");

          }

      });

  }

​

  TopMenu() {

      this.add(jBreakMenuItem);

      this.add(jQuitMenuItem);

  }

​

  public static boolean getFlag() {

      return flag;

  }

​

  public static Chat getChat() {

      return chat;

  }

​

  public static TextMessageSender getTextMessageSender() {

      return textMessageSender;

  }

​

  public static TextMessageReceiver getTextMessageReceiver() {

      return textMessageReceiver;

  }

}

​

### 聊天框ChattingRoom.java

package frontEnd;

​

import javax.swing.\*;

​

/\*\*

\* Created by wtupc96 on 2017/4/10.

\*/

class ChattingRoom extends JPanel {

  private static final JTextArea jChattingRoomTextField = new JTextArea(23, 0);

  private static final JScrollPane jScrollPane = new JScrollPane(jChattingRoomTextField);

​

  static {

      jScrollPane.setHorizontalScrollBarPolicy(JScrollPane.HORIZONTAL\_SCROLLBAR\_AS\_NEEDED);

      jScrollPane.setVerticalScrollBarPolicy(JScrollPane.VERTICAL\_SCROLLBAR\_AS\_NEEDED);

      jChattingRoomTextField.setLineWrap(true);

      jChattingRoomTextField.setSize(290, 400);

      jChattingRoomTextField.setEditable(false);

  }

​

  ChattingRoom() {

      add(jScrollPane);

  }

​

  public static JTextArea getjChattingRoomTextField() {

      return jChattingRoomTextField;

  }

}

### 底部菜单MultiFuncMenu.java

package frontEnd;

​

import backEnd.Chat;

​

import javax.swing.\*;

import javax.swing.event.DocumentEvent;

import javax.swing.event.DocumentListener;

import java.awt.\*;

import java.util.Date;

​

import static frontEnd.ChattingRoom.getjChattingRoomTextField;

import static frontEnd.TopMenu.\*;

​

/\*\*

\* Created by wtupc96 on 2017/4/10.

\*/

class MultiFuncMenu extends JPanel {

  private static final JTextArea jMessageTextArea = new JTextArea(10, 0);

  private static final JButton jTalkButton = new JButton("开始语音");

  private static final JButton jVideoButton = new JButton("开始视频");

  private static final JButton jSendButton = new JButton("发送");

​

  private static final GridLayout btnGridLayout = new GridLayout(1, 3, 3, 3);

  private static final JPanel jBtnPanel = new JPanel();

  private static final JScrollPane jScrollPane = new JScrollPane(jMessageTextArea);

​

  private static Chat chat;

​

  static {

      jScrollPane.setHorizontalScrollBarPolicy(

              JScrollPane.HORIZONTAL\_SCROLLBAR\_AS\_NEEDED);

      jScrollPane.setVerticalScrollBarPolicy(

              JScrollPane.VERTICAL\_SCROLLBAR\_AS\_NEEDED);

      jMessageTextArea.setLineWrap(true);

      jSendButton.setEnabled(false);

      jTalkButton.setEnabled(false);

      jVideoButton.setEnabled(false);

      jMessageTextArea.setEnabled(false);

      jBtnPanel.setLayout(btnGridLayout);

      jBtnPanel.add(jTalkButton);

      jBtnPanel.add(jVideoButton);

      jBtnPanel.add(jSendButton);

​

      jMessageTextArea.getDocument().addDocumentListener(new DocumentListener() {

          @Override

          public void insertUpdate(DocumentEvent e) {

              setJSendButtonState();

          }

​

          @Override

          public void removeUpdate(DocumentEvent e) {

              setJSendButtonState();

          }

​

          @Override

          public void changedUpdate(DocumentEvent e) {

              setJSendButtonState();

          }

      });

​

      jTalkButton.addActionListener(e -> {

          chat = getChat();

​

          if (jTalkButton.getText().equals("开始语音")) {

              chat.soundStart();

              jTalkButton.setText("结束语音");

          } else {

              chat.soundStop();

              jTalkButton.setText("开始语音");

          }

      });

​

      jSendButton.addActionListener(e -> {

          String message = jMessageTextArea.getText();

          System.out.println(message);

          getjChattingRoomTextField().append(new Date().toString() + " 我：\n" + message + "\n\n");

​

          System.out.println(getFlag());

​

          if (TopMenu.getFlag()) {

              getTextMessageSender().sendMessage(message);

          } else {

              getTextMessageReceiver().sendMessage(message);

          }

          jMessageTextArea.setText("");

      });

​

      jVideoButton.addActionListener(e -> {

          // TODO: 2017/4/10

      });

  }

​

  MultiFuncMenu() {

      setLayout(new BorderLayout(2, 2));

      add(jScrollPane, BorderLayout.CENTER);

      add(jBtnPanel, BorderLayout.SOUTH);

  }

​

  public static JTextArea getjMessageTextArea() {

      return jMessageTextArea;

  }

​

  public static JButton getjTalkButton() {

      return jTalkButton;

  }

​

  public static JButton getjVideoButton() {

      return jVideoButton;

  }

​

  private static void setJSendButtonState() {

      if (jMessageTextArea.getText().equals("")) {

          jSendButton.setEnabled(false);

      } else {

          jSendButton.setEnabled(true);

      }

  }

}

## 后端

### 文本发送类TextMessageSender.java

package backEnd;

​

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

import java.io.PrintWriter;

import java.net.Socket;

​

/\*\*

\* Created by wtupc96 on 2017/4/16.

\*/

public class TextMessageSender {

  private static Socket sendSocket;

  private static BufferedReader bufferedReader;

​

  private static PrintWriter write;

​

  public TextMessageSender(String host) throws IOException {

      int index = host.indexOf(':');

      sendSocket = new Socket(host.substring(0, index), Integer.valueOf(host.substring(index + 1)));

      bufferedReader = new BufferedReader(new InputStreamReader(sendSocket.getInputStream()));

      write = new PrintWriter(sendSocket.getOutputStream());

  }

​

  public void sendMessage(String message) {

      write.println(message);

      write.flush();

  }

​

  public void releaseResources() {

      try {

          bufferedReader.close();

          write.close();

          sendSocket.close();

      } catch (IOException e) {

          e.printStackTrace();

      }

  }

​

  public String receiveMessage() {

      try {

          return bufferedReader.readLine();

      } catch (IOException e) {

          e.printStackTrace();

      }

      return null;

  }

}

​

### 文本接收类TextMessageReceiver.java

package backEnd;

​

import java.io.\*;

import java.net.ServerSocket;

import java.net.Socket;

​

/\*\*

\* Created by wtupc96 on 2017/4/16.

\*/

public class TextMessageReceiver {

  private static ServerSocket serverSocket;

  private static Socket getSocket;

  private static BufferedReader bufferedReader;

  private static BufferedWriter bufferedWriter;

​

  public TextMessageReceiver() {

      try {

          serverSocket = new ServerSocket(9527);

          System.out.println("TCP 9527");

          getSocket = serverSocket.accept();

          bufferedReader = new BufferedReader(new InputStreamReader(getSocket.getInputStream()));

          bufferedWriter = new BufferedWriter(new OutputStreamWriter(getSocket.getOutputStream()));

      } catch (IOException e) {

          e.printStackTrace();

      }

  }

​

  public String receiveMessage() {

      try {

          return bufferedReader.readLine();

      } catch (IOException e) {

          e.printStackTrace();

      }

      return null;

  }

​

  public void releaseResources() {

      try {

          bufferedReader.close();

          bufferedWriter.close();

          serverSocket.close();

          getSocket.close();

      } catch (IOException e) {

          e.printStackTrace();

      }

  }

​

  public void sendMessage(String message) {

      try {

          bufferedWriter.write(message);

          bufferedWriter.flush();

      } catch (IOException e) {

          e.printStackTrace();

      }

  }

}

​

### 语音聊天核心类Chat.java

package backEnd;

​

public class Chat {

  private final SoundReceiver soundReceiver;

  private final SoundSender soundSender;

​

  public Chat(String targetPeer) {

      int indexOfPort = targetPeer.indexOf(':');

​

      soundSender = new SoundSender(targetPeer.substring(0, indexOfPort));

​

      soundReceiver = new SoundReceiver();

  }

​

  public void soundStart() {

      soundReceiver.start();

      soundSender.start();

  }

​

  public void soundStop() {

      soundReceiver.stop();

      soundSender.stop();

  }

}

​

### 语音发送类SoundSender.java

package backEnd;

​

import javax.sound.sampled.\*;

​

class SoundSender extends UDPSender implements Runnable {

​

  private final int bufferLength;

  private TargetDataLine line;

  private Thread thread;

  private boolean isStart;

​

  public SoundSender(String groupAddress) {

      super(groupAddress, 9999);

      AudioFormat format = new AudioFormat(8000, 16, 2, true, true);

      DataLine.Info info = new DataLine.Info(TargetDataLine.class, format);

      try {

          line = (TargetDataLine) AudioSystem.getLine(info);

          line.open(format, line.getBufferSize());

      } catch (LineUnavailableException e) {

          e.printStackTrace();

      }

      this.bufferLength = 1024;

      isStart = false;

  }

​

​

  public void run() {

      byte[] buffer = new byte[bufferLength];

      while (isStart && !thread.isInterrupted()) {

          line.read(buffer, 0, buffer.length);

          send(buffer);

      }

  }

​

  public void start() {

      if (thread == null || !thread.isAlive()) {

          thread = new Thread(this);

          line.start();

          thread.start();

          isStart = true;

      }

  }

​

  public void stop() {

      thread.interrupt();

      line.stop();

      isStart = false;

  }

​

}

​

### 语音接受类SoundReceiver.java

package backEnd;

​

import com.sun.org.apache.xerces.internal.impl.dv.util.HexBin;

​

import javax.sound.sampled.\*;

​

class SoundReceiver extends UDPReceiver implements Runnable {

​

  private SourceDataLine line;

  private Thread thread;

  private boolean isStart;

​

  public SoundReceiver() {

      super("localhost", 9527, 1024);

      AudioFormat format = new AudioFormat(8000, 16, 2, true, true);

      DataLine.Info info = new DataLine.Info(SourceDataLine.class, format);

      try {

          line = (SourceDataLine) AudioSystem.getLine(info);

          line.open(format, 10240);

      } catch (LineUnavailableException e) {

          e.printStackTrace();

      }

  }

​

  public void run() {

      while (isStart && !thread.isInterrupted()) {

          byte[] data = super.receive();

          line.write(data, 0, data.length);

​

          // 进一步将音频保存到文件中

          saveToFile(data);

      }

  }

​

  private void saveToFile(byte[] data) {

      System.out.print(HexBin.encode(data));

  }

​

  public void start() {

      if (thread == null || !thread.isAlive()) {

          thread = new Thread(this);

          line.start();

          isStart = true;

          contact();

          thread.start();

      }

  }

​

  public void stop() {

      thread.interrupt();

      line.stop();

      isStart = false;

      close();

  }

}

​

### 语音接收底部类UDPReceiver.java

package backEnd;

​

import com.sun.org.apache.xerces.internal.impl.dv.util.HexBin;

​

import javax.sound.sampled.\*;

​

class SoundReceiver extends UDPReceiver implements Runnable {

​

  private SourceDataLine line;

  private Thread thread;

  private boolean isStart;

​

  public SoundReceiver() {

      super("localhost", 9527, 1024);

      AudioFormat format = new AudioFormat(8000, 16, 2, true, true);

      DataLine.Info info = new DataLine.Info(SourceDataLine.class, format);

      try {

          line = (SourceDataLine) AudioSystem.getLine(info);

          line.open(format, 10240);

      } catch (LineUnavailableException e) {

          e.printStackTrace();

      }

  }

​

  public void run() {

      while (isStart && !thread.isInterrupted()) {

          byte[] data = super.receive();

          line.write(data, 0, data.length);

​

          // 进一步将音频保存到文件中

          saveToFile(data);

      }

  }

​

  private void saveToFile(byte[] data) {

      System.out.print(HexBin.encode(data));

  }

​

  public void start() {

      if (thread == null || !thread.isAlive()) {

          thread = new Thread(this);

          line.start();

          isStart = true;

          contact();

          thread.start();

      }

  }

​

  public void stop() {

      thread.interrupt();

      line.stop();

      isStart = false;

      close();

  }

}

### ​语音发送底部类UDPSender.java

package backEnd;

​

import java.io.IOException;

import java.net.DatagramPacket;

import java.net.DatagramSocket;

import java.net.InetAddress;

​

class UDPSender {

​

  private final DatagramPacket dgp;

  private DatagramSocket s;

  private InetAddress group;

​

  UDPSender(String groupAddress, int port) {

      try {

          s = new DatagramSocket();

          group = InetAddress.getByName(groupAddress);

      } catch (IOException e) {

          e.printStackTrace();

      }

      dgp = new DatagramPacket(new byte[0], 0, group, port);

  }

​

  public void close() {

      if (s != null) {

          s.close();

      }

  }

​

  void send(byte[] buffer) {

      dgp.setData(buffer);

      dgp.setLength(buffer.length);

      try {

          s.send(dgp);

      } catch (IOException e) {

          e.printStackTrace();

      }

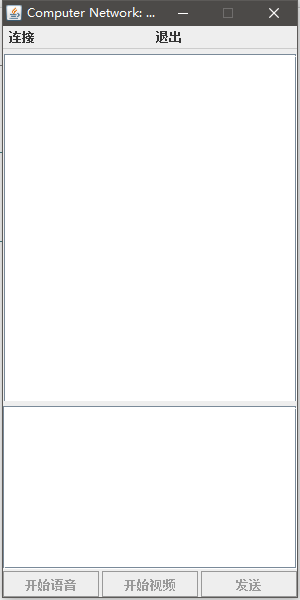
  }

}

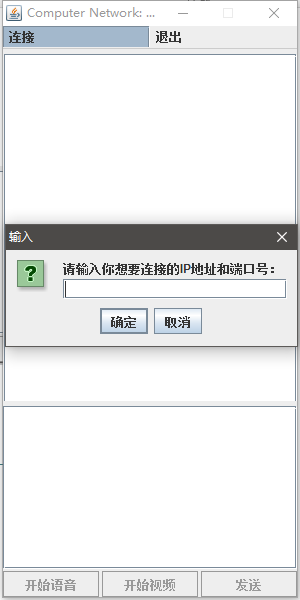
​

# 实验结果

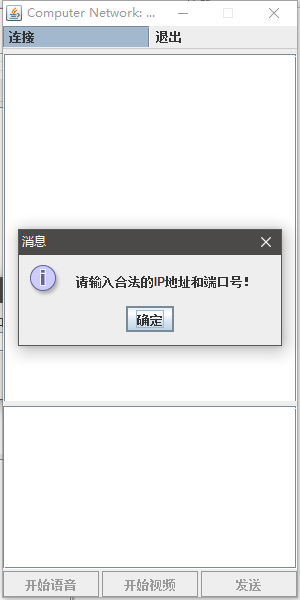
## 系统初始界面，没有连接的情况时下面的按钮都是不能按的



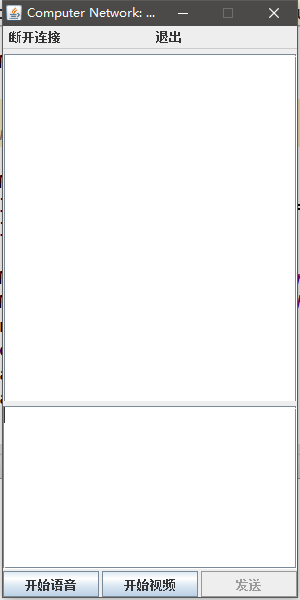
## 点击“连接”，输入IP地址和端口：



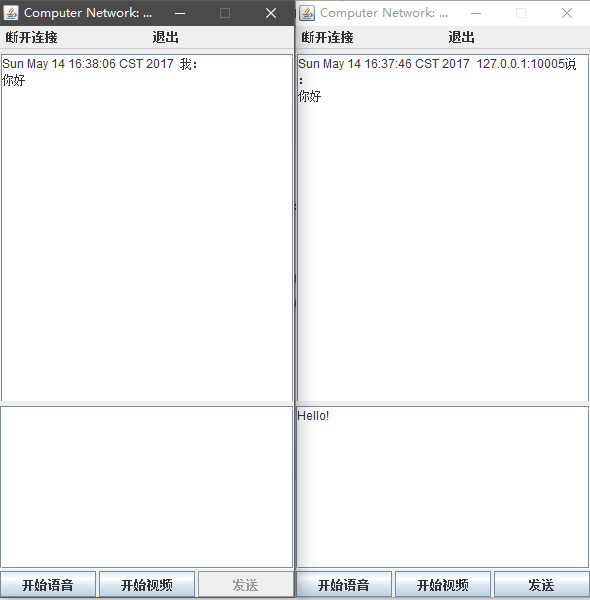
## 输入不合法的IP地址或者端口



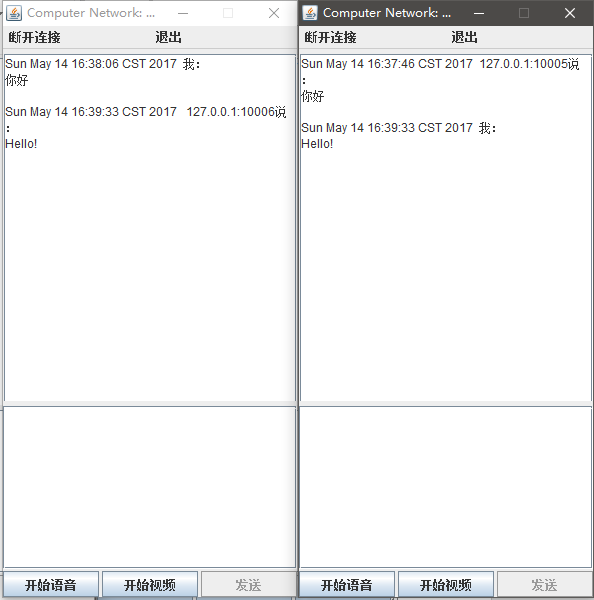
## 连接成功界面，连接成功之后才可以进行语音通信



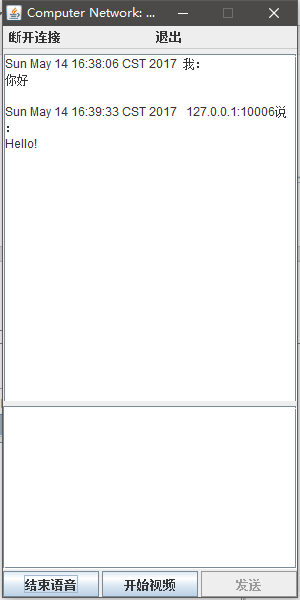
## 连接之后客户端之间就可以通信了，只有输入区有文字才能使用发送按键



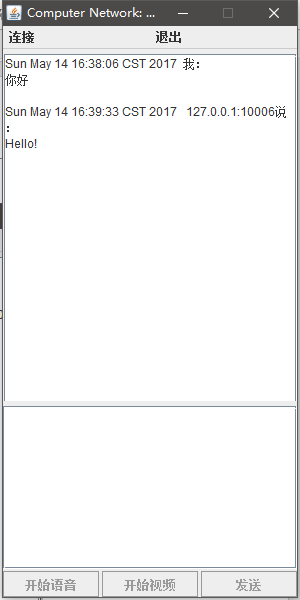
## 另一客户端返回信息



## 发起语音通话



## 断开连接，按键都变成不可用



# 讨论和分析

为了避免抛出异常，必须保持输入的IP和端口合法，所以使用input.matches("^(\\d{1,3}\\.) {3}\\d{1,3}:\\d{1,10}$")来检查是否是合法的输入格式。

每个客户端都包含一个Socket和一个SocketServer，即每个客户端都包含了客户端和服务端的功能，连接首先尝试目标IP有没有开启服务端，自己首先作为客户端进行尝试连接，若触发了Timeout事件，即目标IP没有服务端，则自身开启服务端变成服务端模式，等待其他的客户端来连接。做到了P2P的TCP通信。

public TextMessageSender(String host) throws IOException {

      int index = host.indexOf(':');

      sendSocket = new Socket(host.substring(0, index),

Integer.valueOf(host.substring(index + 1)));

      bufferedReader = new BufferedReader(

new InputStreamReader(sendSocket.getInputStream()));

      write = new PrintWriter(sendSocket.getOutputStream());

}

这一段建立了文字发送的Socket客户端，而

public TextMessageReceiver() {

      try {

          serverSocket = new ServerSocket(9527);

          System.out.println("TCP 9527");

          getSocket = serverSocket.accept();

          bufferedReader = new BufferedReader(

new InputStreamReader(getSocket.getInputStream()));

          bufferedWriter = new BufferedWriter(

new OutputStreamWriter(getSocket.getOutputStream()));

      } catch (IOException e) {

          e.printStackTrace();

      }

}

则建立了文字接收的Socket服务端。

同理，

UDPReceiver(String groupAddress, int port, int bufferSize) {

      try {

          s = new MulticastSocket(port);

          group = InetAddress.getByName(groupAddress);

      } catch (IOException e) {

          e.printStackTrace();

      }

      buffer = new byte[bufferSize];

      dgp = new DatagramPacket(buffer, bufferSize);

      isClose = false;

      this.port = port;

      this.bufferSize = bufferSize;

  }

和

UDPSender(String groupAddress, int port) {

      try {

          s = new DatagramSocket();

          group = InetAddress.getByName(groupAddress);

      } catch (IOException e) {

          e.printStackTrace();

      }

      dgp = new DatagramPacket(new byte[0], 0, group, port);

}

分别建立了UDP报文的接收端和发送端，用于语音通信。