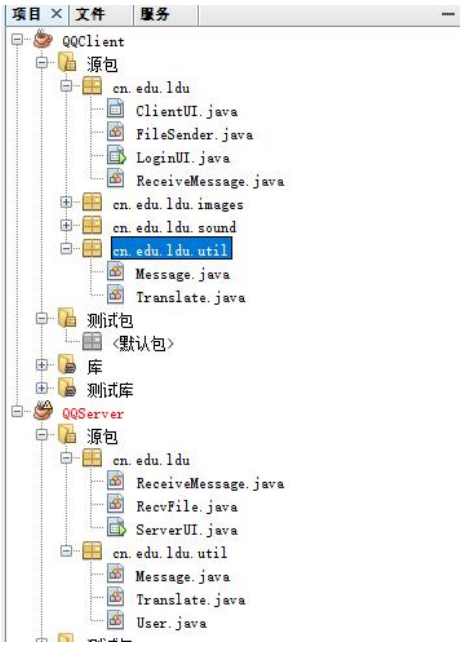
文件传输与 SwingWorker

实验介绍

TCP 协议有保证数据可靠传输的一套机制，用 TCP 协议传输文件，是个很好的选择，本次在第五章 QQUDP 案例的基础上，增加文件传输的功能设计。

SwingWoker的使用方法
实现一个SwingWorker的子类，并重写doInBackground()方法，在工作线程中实例化一个子类，调用execute()方法以启动任务。
需要重写的方法
doInBackground()：必须重写，任务的逻辑代码。
done()：任务完成以后的操作。
process()：规定相应进度如何显示。
get()：获取任务结束以后的最后结果，这个方法应该放置在done()的内部。
publish()：发布进度，向process方法传递进度参数。

程序结构和功能



ClientUI：客户机界面构建 UDP 套接字和发送消息面板

FileSender：继承 SwingWorker 类，负责文件传输

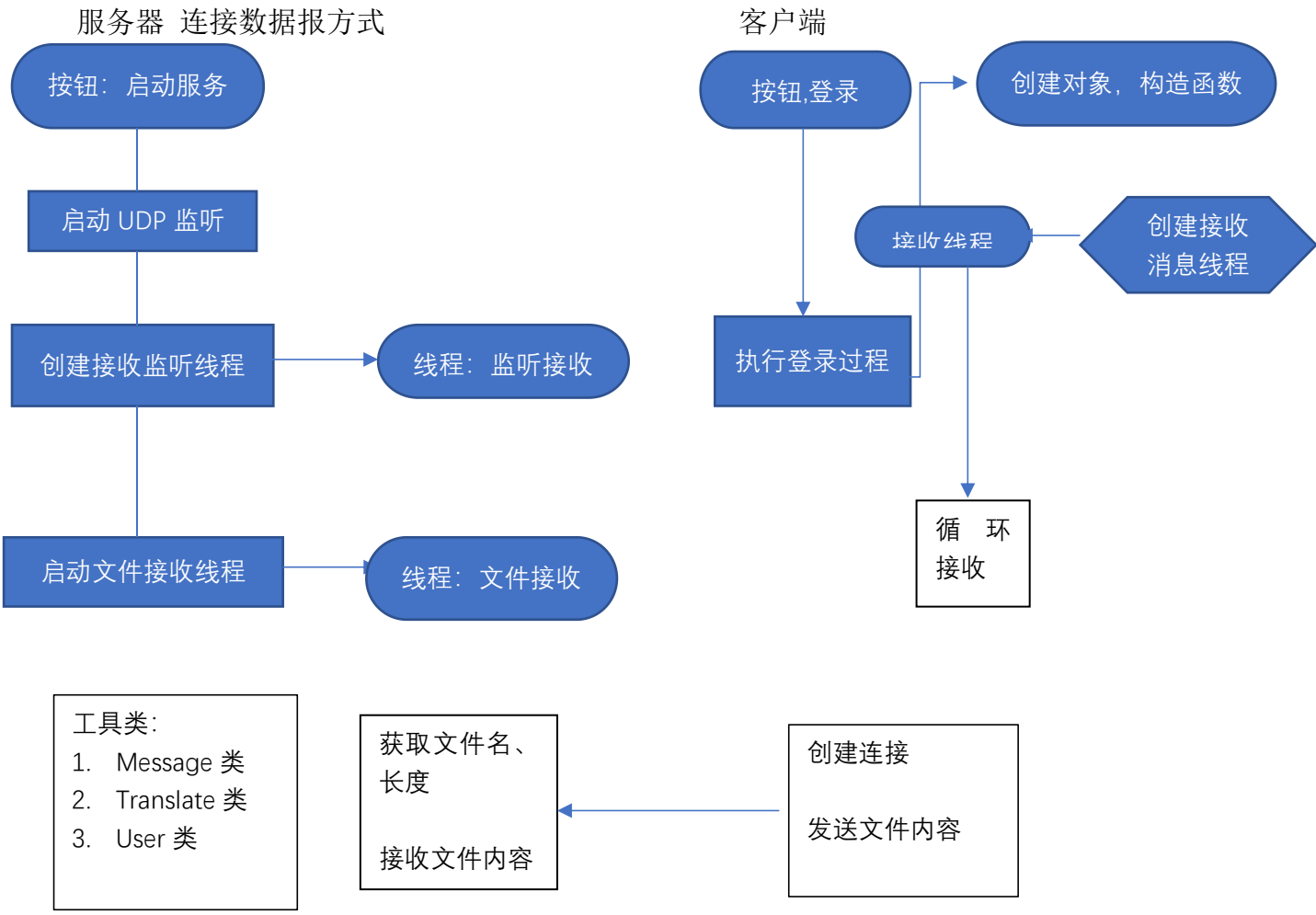
LoginUI：登录面板和验证登录

ReceiveMessage：接收所有消息线程

ServerUI：UDP 数据套接字侦听指定端口，消息接收线程和文件接收线程

RecvFile：继承 SwingWorker，接收文件到指定的文件夹

程序流程图



程序关键代码

选择文件

```
//发送文件
private void uploadFileActionPerformed(java.awt.event.ActionEvent evt) {
    //打开文件选择对话框
    JFileChooser fileChooser=new JFileChooser();
    fileChooser.setDialogTitle("选择上传文件");
    fileChooser.setApproveButtonText("选择");
    int choice=fileChooser.showOpenDialog(this); //显示对话框
    if (choice==JFileChooser.APPROVE_OPTION) { //点击选择按钮
        File file=fileChooser.getSelectedFile();//获取文件对象
        //启动发送文件线程
        SwingWorker<List<String>,String> sender=new FileSender(file,msg,this);
        sender.addPropertyChangeListener(new PropertyChangeListener() {
     public  void propertyChange(PropertyChangeEvent evt) {
         if ("progress".equals(evt.getPropertyName())) {
             progressBar.setValue((Integer)evt.getNewValue());
         }
     }
});
```

发送文件

```
public class FileSender extends SwingWorker<List<String>,String>{
    private File file; //文件
    private Message msg;//消息类
    private ClientUI parentUI; //父类
    private Socket fileSocket; //传送文件的套接字
    private static final int BUFSIZE=8096; //缓冲区大小
    private int progress=0; //文件传送进度
    private String lastResults=null; //传送结果
    //构造函数
```

```
@Override
protected List<String> doInBackground() throws Exception {
    fileSocket=new Socket();
    //连接服务器
    SocketAddress remoteAddr=new InetSocketAddress(msg.getToAddr(),msg.getToPort());
    fileSocket.connect(remoteAddr);
    //构建套接字输出流
    DataOutputStream out=new DataOutputStream(
                         new BufferedOutputStream(
                         fileSocket.getOutputStream()));
    //构建文件输入流
    DataInputStream in=new DataInputStream(
                       new BufferedInputStream(
                       new FileInputStream(file)));
    long fileLen=file.length();  //计算文件长度
    //发送文件名称、文件长度
    out.writeUTF(file.getName());
    out.writeLong(fileLen);
    out.flush();
    //传送文件内容
    int numRead=0; //单次读取的字节数
    int numFinished=0; //总完成字节数
    byte[] buffer=new byte[BUFSIZE];
    while (numFinished<fileLen && (numRead=in.read(buffer))!=-1) { //文件可读
        out.write(buffer,0,numRead);  //发送
        out.flush();
        numFinished+=numRead; //已完成字节数
        Thread.sleep(200); //演示文件传输进度用
        publish(numFinished+"/"+fileLen+"bytes");
        setProgress(numFinished*100/(int)fileLen);
    }//end while
```

```
@Override
protected void process(List<String> middleResults) {
    for (String str:middleResults) {
        parentUI.progressLabel.setText(str);
    }
}
// doInBackground函数执行完毕
```

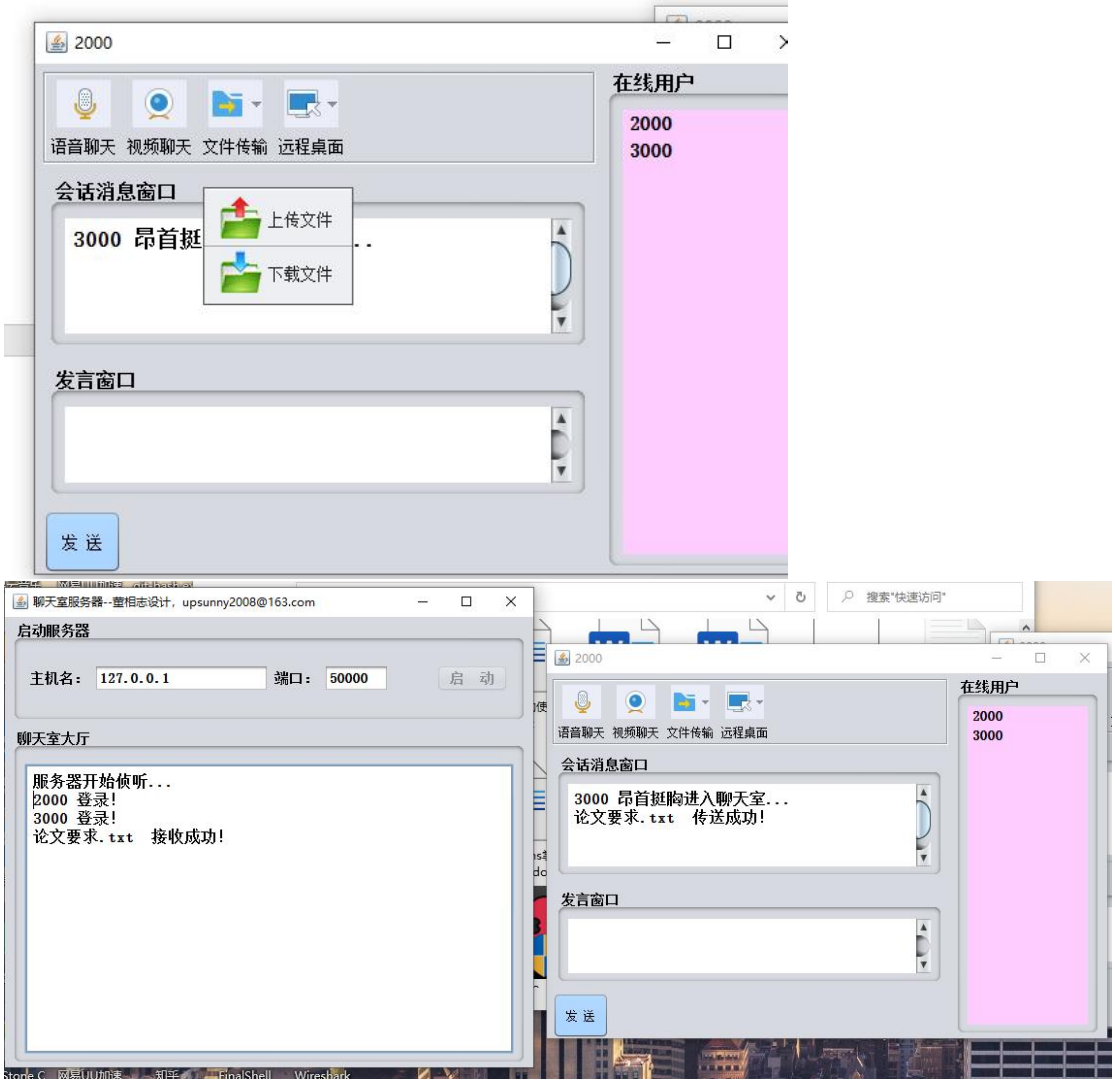
FileSender 类继承 SwingWorker 类，重写 doInBackground()方法，当线程创建的时候，作为工作线程会开始执行这个方法，在传输文件的同时，计算当前文件传输完成的百分比，同步显示传输进度。Publish()方法发送数据到 process()方法中的 list 列表中。便于 process 处理并显示中间结果，该方法自动执行。

服务器文件接收线程和消息接收线程

```
private void btnStartActionPerformed(java.awt.event.ActionEvent evt) {
    try {
        //获取服务器工作地址端口
        String hostName=txtHostName.getText();
        int hostPort=Integer.parseInt(txtHostPort.getText());
        //创建UDP数据报套接字,在指定端口侦听
        DatagramSocket serverSocket=new DatagramSocket(hostPort);
        txtArea.append("服务器开始侦听...\n");
        //创建并启动UDP消息接收线程
        Thread recvThread=new ReceiveMessage(serverSocket,this);
        recvThread.start();
        //创建并启动文件接收线程
        new Thread(new Runnable() {
            public void run() {
                try {
                SocketAddress serverAddr=new InetSocketAddress(InetAddress.getByName(hostName),hostPort);
                ServerSocket listenSocket=new ServerSocket(); //创建侦听套接字
                listenSocket.bind(serverAddr); //绑定到工作地址
                int processors=Runtime.getRuntime().availableProcessors();//CPU数
                ExecutorService fixedPool=Executors.newFixedThreadPool(processors*2);//创建固定大小线程池
                while (true) { //处理所有客户机连接
                    Socket fileSocket=listenSocket.accept();//如果无连接，则阻塞，否则接受连接并创建新的会话套接字
                    //文件接收线程为SwingWorker类型的后台工作线程
                    SwingWorker<Integer,Object> recver=new RecvFile(fileSocket,ServerUI.this); //创建客户线程
                    fixedPool.execute(recver); //用线程池调度客户线程运行
                }//end while
                } catch (IOException ex) {
                    JOptionPane.showMessageDialog(null, ex.getMessage(), "错误提示", JOptionPane.ERROR_MESSAGE);
                }//end try catch
            }//end run()
```

点击按钮连接服务器之后，创建并启动 UDP 消息接收线程，再另外开辟一个文件接收线程，在文件接收线程中一直循环处理所有客户机连接，连接成功后创建 SwingWorker 类型的后台工作线程，用线程池调度的方式调度客户机运行。

实验运行结果



upload 目录下显示了接收的文件。