浙江农林大学

信息工程学院

# 课 程 设 计 报 告

课程名称：Socket 编程

课程编号：R3550021

设计内容：网络编程

专　　业：计算机科学与技术

班　　级：计算机 153 班

学　　号：201505010315

姓　　名：倪畅

指导教师：罗显贵

设计地点：机房、图书馆和寝室

设计时间：2017 年 11 月 12 日

**2017 年 11 月 19 日**

# 目录

## 写在前面：

因本人对 Java 语言比较熟悉，所以此次课程设计主要用 Java 语言实现，编译器为 Eclipse 和 Android Studio。另外，本课程设计源码以及实验截图在随同打包的压缩文件夹中，老师可以查阅。

## 课程设计主要内容：

**题目一** 开发教师上课系统，要求有：

a) 软件分为两个部分：教师机（服务端）和学生机（客户端）

b) 完成上课系统的基本要求如：教师机桌面共享、教师上课过程的录像、教师向学生群发文件等。

c) 能同时支持 40 个学生上课

**题目三** 开发一个基于 P2P 技术局域网聊天程序，要求：

a) 服务器仅负责联系所有已登陆的客户端，但并不负责转发客户端之间通信信息。

b) 已登陆客户端可以选择与任何其它已登陆客户端进行聊天，传输文件，视频和语音聊天等。

c) 具有一定的实用性

**题目七** 与手机通信编程，要求如下：

a) 软件以 Windows 服务的形式在计算机后台运行

b) 软件以邮件或短信端口的形式与手机终端通信

c) 通过与软件的通信，手机终端可以远程操控计算机，如关机、重启、启动某个应用程序等。

## 题目一 开发教师上课系统

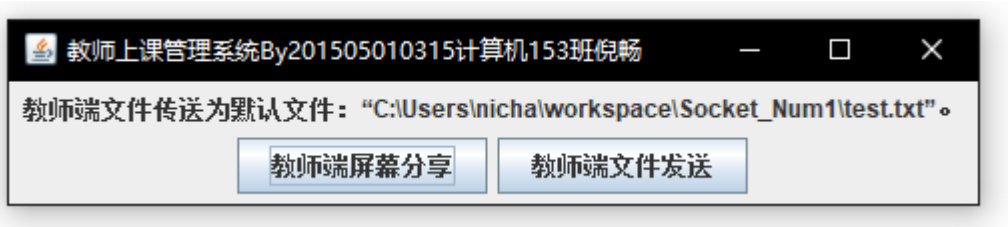
### 解题思路：

教师上课系统有教师机和学生机两个部分，教师机桌面共享，在功能实现上面，说白了，就是教师端的机器不断截取屏幕信息，以图片的形式发送到每一个学生端的电脑上面，由此学生能够看见老师在电脑上的操作，这就是所谓的教师机桌面共享。再来说说教师向学生群发文件：客户端与服务端建立连接，选择客户端本地文件，先将文件名及大小等属性发送给服务端，再将文件通过流的方式传输给服务端。传输的进度打印到控制台中，直到传输完成。服务端接收客户端的请求（阻塞式），每接收到一个客户端请求连接后，就新开一个处理文件的线

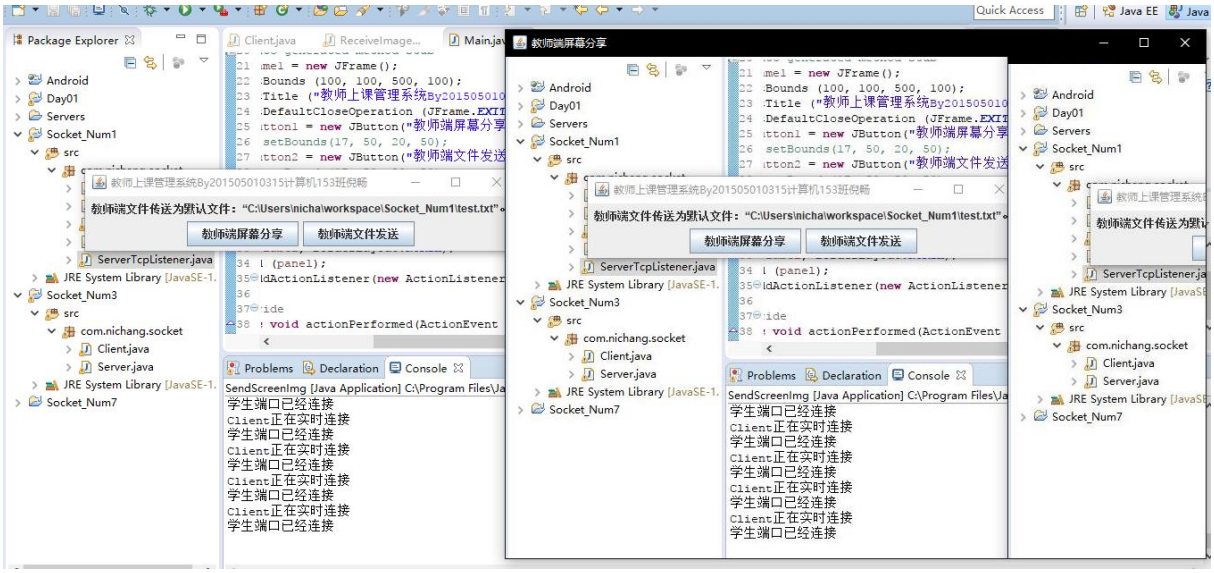
程，开始写入流，将文件到服务器的指定目录下，并与传输过来的文件同名。为了方便调试，我这里将文件路径固定，所以教师机传输文件是不可修改的。

**实验截图：**

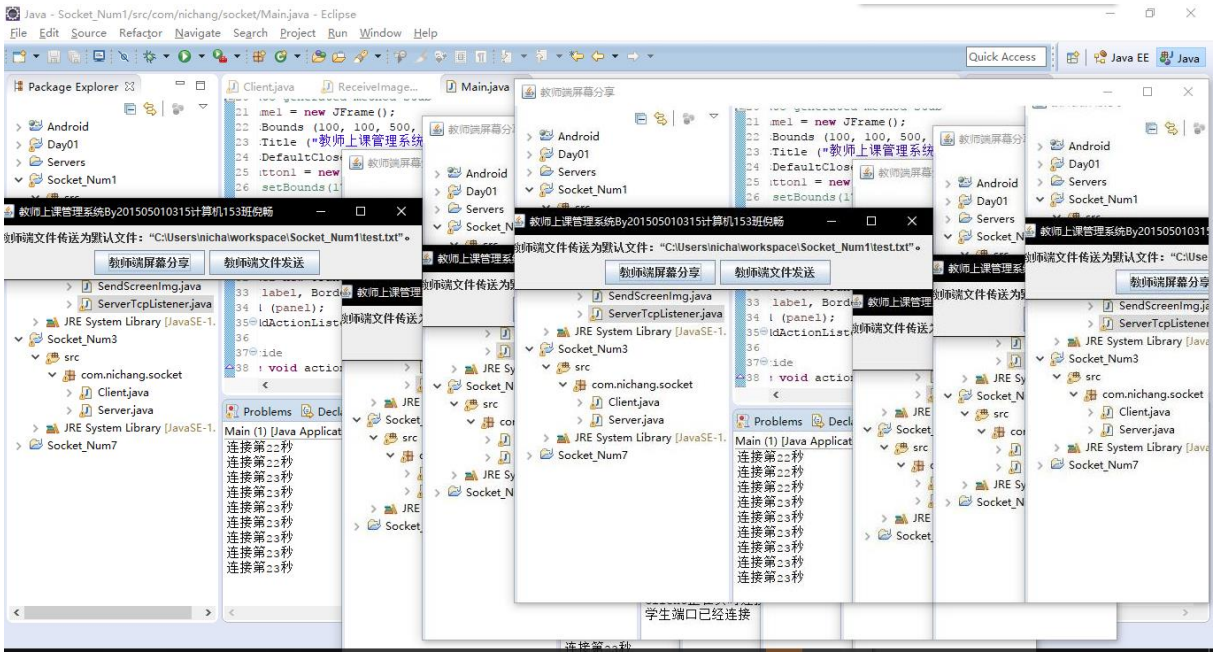
**主界面：**



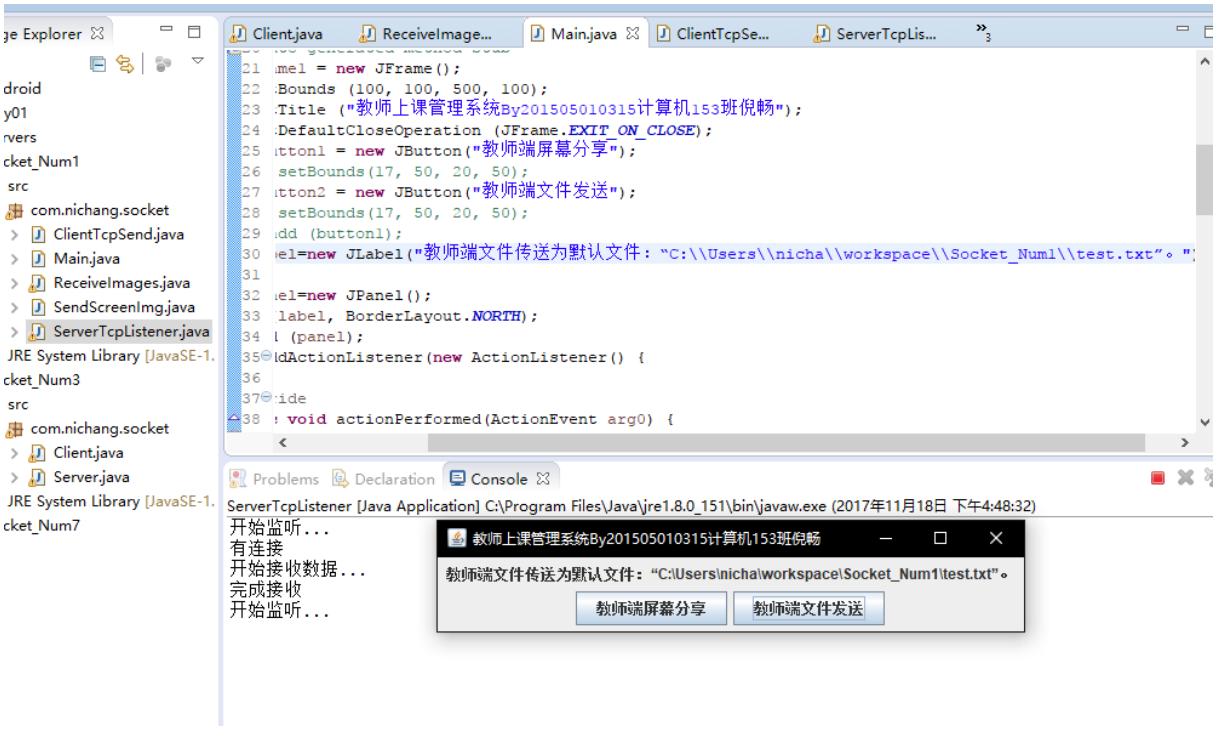

**教师屏幕分享（单人）：**



**教师屏幕分享（多人）：**



**教师群发文件：**




**解题部分关键代码：**

教师机桌面共享部分：

服务端：

```
public  class SendScreenImg extends Thread
{
    public static int SERVERPORT=8000;
    private ServerSocket serverSocket;
    private Robot robot;
    public  Dimension screen;
    public Rectangle rect ;
    private Socket socket;

    public static void main(String args[])
    {
        new SendScreenImg(SERVERPORT).start();
    }
    public SendScreenImg(int SERVERPORT)
    {
        try {
            serverSocket = new ServerSocket(SERVERPORT);
            serverSocket.setSoTimeout(864000000);
            robot = new Robot();
        } catch (Exception e) {
```

```
            e.printStackTrace();
        }
        screen = Toolkit.getDefaultToolkit().getScreenSize();
        rect = new Rectangle(screen);
    }

    @Override
    public void run()
    {
        while(true)
        {
            try{
                socket = serverSocket.accept();
                System.out.println("学生端口已经连接");
                ZipOutputStream zip = new ZipOutputStream(new
DataOutputStream(socket.getOutputStream()));
                zip.setLevel(9);

                BufferedImage img = robot.createScreenCapture(rect);
                zip.putNextEntry(new ZipEntry("test.jpg"));
                ImageIO.write(img, "jpg", zip);
                if(zip!=null)zip.close();
                System.out.println("Client正在实时连接");

            } catch (IOException ioe) {
                System.out.println("连接断开");
            } finally {
                if (socket != null) {
                    try {
                        socket.close();
                    } catch (IOException e) {e.printStackTrace();}
                }
            }
        }
    }
}
```

客户端：

```
public  class ReceiveImages extends  Thread{
    public BorderInit frame ;
    public Socket socket;
    public String IP;
    public ReceiveImages(BorderInit frame,String IP)
    {
        this.frame = frame;
        this.IP=IP;
    }
```

```
    public void run() {
        while(frame.getFlag()){
            try {
                socket = new Socket(IP,8000);
                DataInputStream ImgInput = new
DataInputStream(socket.getInputStream());
                ZipInputStream imgZip = new ZipInputStream(ImgInput);

                imgZip.getNextEntry();            //到Zip文件流的开始处
                Image img = ImageIO.read(imgZip);  //按照字节读取Zip图片流里面的图片
                frame.jlbImg.setIcon(new ImageIcon(img));
                System.out.println("连接第
"+(System.currentTimeMillis()/1000)%24%60+"秒");
                frame.validate();
                TimeUnit.MILLISECONDS.sleep(10);// 接收图片间隔时间
                imgZip.close();

            } catch (IOException | InterruptedException e) {
                System.out.println("连接断开");
            }finally{
                try {
                    socket.close();
                } catch (IOException e) {}
            }
        }
    }
}

//Client端窗口辅助类，专门用来显示从教师端收到的屏幕信息
class BorderInit extends JFrame
{
    private static final long serialVersionUID = 1L;
    public JLabel jlbImg;
    private boolean flag;
    public boolean getFlag(){
        return this.flag;
    }
    public BorderInit()
    {
        this.flag=true;
        this.jlbImg = new JLabel();
        this.setTitle("教师端屏幕分享" );
        this.setSize(800, 600);
        this.add(jlbImg);
        this.setLocationRelativeTo(null);
        this.setExtendedState(Frame.MAXIMIZED_BOTH);
```

```
        this.setDefaultCloseOperation(DISPOSE_ON_CLOSE);
        this.setVisible(true);
        this.validate();
        this.addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
                flag=false;
                BorderInit.this.dispose();
                System.out.println("窗体关闭");
                System.gc();
            }
        });
    }
}
```

教师向学生群发文件部分：

服务端：

```
public class ServerTcpListener implements Runnable {

    @Override
    public void run() {
    }
    public static void main(String[] args) {
        try {
            final ServerSocket server = new ServerSocket(33456);
            Thread th = new Thread(new Runnable() {

                @Override
                public void run() {
                    while (true) {
                        try {
                            System.out.println("开始监听...");
                            Socket socket = server.accept();
                            System.out.println("有连接");
                            receiveFile(socket);
                        } catch (Exception e) {
                            e.printStackTrace();
                        }
                    }
                }
            });
            th.run();
        } catch (Exception ex) {
            ex.printStackTrace();
        }
    }

    public static void receiveFile(Socket socket) throws IOException {
```

```
        byte[] inputByte = null;
        int length = 0;
        DataInputStream din = null;
        FileOutputStream fout = null;
        try {
            din = new DataInputStream(socket.getInputStream());

            fout = new FileOutputStream(new File("./"+din.readUTF()));
            inputByte = new byte[1024];
            System.out.println("开始接收数据...");
            while (true) {
                if (din != null) {
                    length = din.read(inputByte, 0, inputByte.length);
                }
                if (length == -1) {
                    break;
                }
                System.out.println(length);
                fout.write(inputByte, 0, length);
                fout.flush();
            }
            System.out.println("完成接收");
        } catch (Exception ex) {
            ex.printStackTrace();
        } finally {
            if (fout != null)
                fout.close();
            if (din != null)
                din.close();
            if (socket != null)
                socket.close();
        }
    }
}
```

客户端：

```
public class ClientTcpSend {

    public static void main(String[] args) {
        int length = 0;
        byte[] sendByte = null;
        Socket socket = null;
        DataOutputStream dout = null;
        FileInputStream fin = null;
        try {
          try {
            socket = new Socket();
```

```
            socket.connect(new InetSocketAddress("127.0.0.1", 33456),10 *
1000);
            dout = new DataOutputStream(socket.getOutputStream());
            File file = new
File("C:\\Users\\nicha\\workspace\\Socket_Num1\\test.txt");
            fin = new FileInputStream(file);
            sendByte = new byte[1024];
            dout.writeUTF(file.getName());
            while((length = fin.read(sendByte, 0, sendByte.length))>0){
                dout.write(sendByte,0,length);
                dout.flush();
            }
            } catch (Exception e) {
            } finally{
                if (dout != null)
                    dout.close();
                if (fin != null)
                    fin.close();
                if (socket != null)
                    socket.close();
        }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

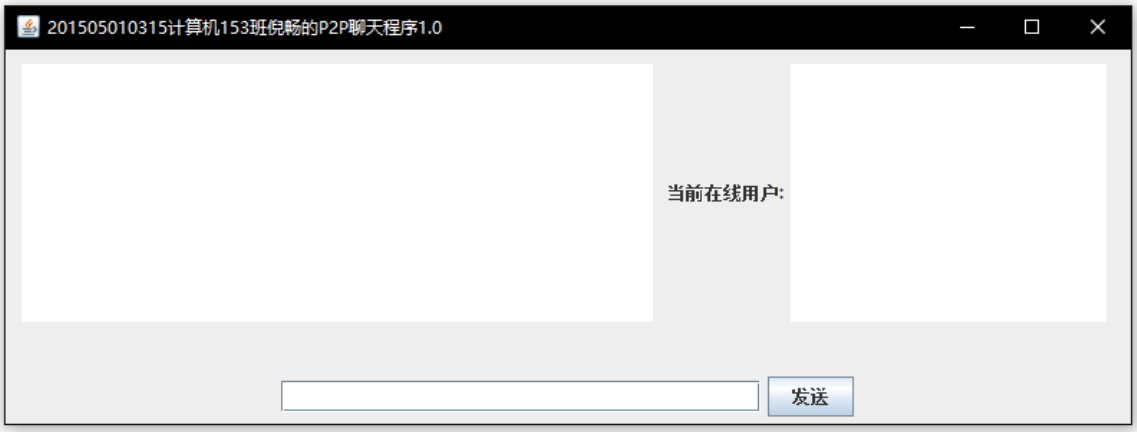
## 题目三 开发一个基于 P2P 技术局域网聊天程序

**解题思路：**

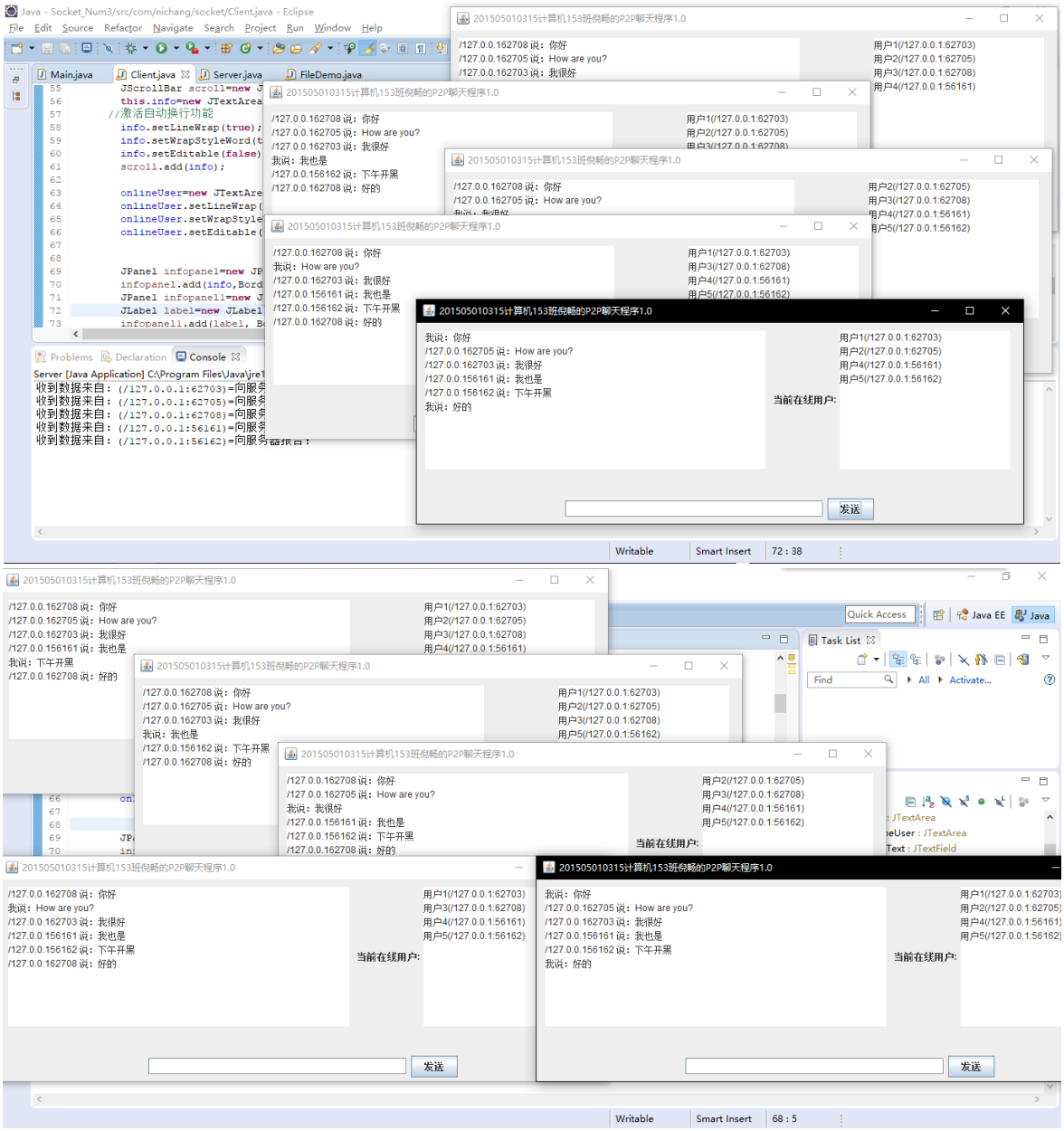
对于这题目，需要一个服务端和一个客户端。首先我先讲讲服务端是怎么实现的：首先，我先用一个专门的线程将用户的所有 IP 和端口先记录下来，然后设置了一个 UDPsocket 的接口（当然，这里需要一个能接收数据的 UDP 包）。然后再给在线的用户一个线程，这个线程负责给在线用户发送当前所有在线用户的信息，最后还要在服务端设置一个监听端口的线程，时刻监听这个端口有没有信息从客户端传进来。总之，服务端主要有两个操作，一是阻塞接收客户端的 socket 并做响应处理，二是检测客户端的心跳，如果客户端一段时间内没有发送心跳则移除该客户端，由 Server 创建 ServerSocket，然后启动两个线程池去处理这两件事，其中 SocketHolder 存储当前跟服务端交互的 socket 集合。再来说说客户端的实现：利用已经学的 Java 知识，简单的给这个聊天室一个界面，设置了按钮的监听事件，和服务端保持在同一个端口以便进行消息的发送。最关键的一点是要写一个方法，来给其他在线用户发送心跳，使得能保持 session 有效。这样，一个简单的基于 P2P 技术局域网聊天程序就能得以实现了。

实验截图：

客户端主界面：




多人聊天调试：



解题部分关键代码：

服务端：

```
public class Server extends Thread{
    //存储所有的用户IP与端口
    public static List<Map> userList=new ArrayList<Map>();
    //在线用户IP
    public static Map users=new HashMap();
    public static int index=1;
    private DatagramSocket server;
    public Server(DatagramSocket server)
    {
        this.server=server;
    }
    //线程负责给在线用户发送当前所有在线用户的信息
    public void run()
    {
        try
        {
            DatagramPacket sendPacket;
            StringBuffer msg;
            while(true)
            {
                for(Map user:Server.userList)
                {
                    //服务器数据，标记server:
                    msg=new StringBuffer("server:");
                    for(Map map:Server.userList)
                    {

if(!map.get("id").toString().equals(user.get("id").toString()))
                        {

msg.append(map.get("id")+"#"+map.get("ip")+":"+map.get("port"));
                            msg.append(",");
                        }
                    }
                    if(!msg.toString().equals("server:"))
                    {
                        byte[] data=msg.toString().getBytes();
                        //构造发送报文
                        sendPacket = new DatagramPacket(data, data.length,
(InetAddress)user.get("ip"), (Integer)user.get("port"));
                        server.send(sendPacket);
                    }
                }
```

```
                Thread.sleep(2000);
            }
        }catch(Exception e){}
    }
    public static void main(String args[])throws Exception
    {
        int port=20000;
        //创建一个UDPsocket
        DatagramSocket server = new DatagramSocket(port);
        byte[] buf = new byte[1024];
        //接收数据的udp包
        DatagramPacket packet = new DatagramPacket(buf, buf.length);
        //开启服务
        new Server(server).start();
        String msg;
        //循环接收数据
        while(true)
        {
            server.receive(packet);
            msg=new String(packet.getData(),0,packet.getLength());
            if(msg!=null&&msg.equals("bye"))
                break;
            if(msg.length()>0)
            {
                System.out.println("收到数据来自:
("+packet.getAddress()+":"+packet.getPort()+")="+msg);

if(!users.containsKey(packet.getAddress()+":"+packet.getPort()))
                {
                    Map map=new HashMap();
                    map.put("id", index);
                    map.put("ip", packet.getAddress());
                    map.put("port", packet.getPort());
                    userList.add(map);
                    users.put(packet.getAddress()+":"+packet.getPort(), index);
                    index++;
                }
            }
        }
        server.close();
    }
}
```

客户端:

```
public class Client extends Thread implements ActionListener{
    //是否停止
    public static int STOP=0;
```

```
//在线用户列表
public static Map<String,SocketAddress> userMap=new HashMap();
private DatagramSocket client;
private JFrame frame;
//聊天信息
private JTextArea info;
//在线用户
private JTextArea onlineUser;
private JTextField msgText;
private JButton sendButton;
public Client(DatagramSocket client)throws Exception
{
    this.client=client;
    this.frame=new JFrame("201505010315计算机153班倪畅的P2P聊天程序1.0");
    frame.setSize(800, 300);
    sendButton=new JButton("发送");
    JScrollBar scroll=new JScrollBar();
    this.info=new JTextArea(10,40);
  //激活自动换行功能
    info.setLineWrap(true);
    info.setWrapStyleWord(true);
    info.setEditable(false);
    scroll.add(info);
    onlineUser=new JTextArea(10,20);
    onlineUser.setLineWrap(true);
    onlineUser.setWrapStyleWord(true);
    onlineUser.setEditable(false);
    JPanel infopanel=new JPanel();
    infopanel.add(info,BorderLayout.WEST);
    JPanel infopanel1=new JPanel();
    JLabel label=new JLabel("当前在线用户:");
    infopanel1.add(label, BorderLayout.NORTH);
    infopanel1.add(onlineUser, BorderLayout.SOUTH);
    infopanel.add(infopanel1,BorderLayout.EAST);
    JPanel panel=new JPanel();
    msgText=new JTextField(30);
    panel.add(msgText);
    panel.add(sendButton);
    frame.add(infopanel,BorderLayout.NORTH);
    frame.add(panel,BorderLayout.SOUTH);
    frame.setVisible(true);
    sendButton.addActionListener(this);
    frame.addWindowListener(new  WindowAdapter(){
       public  void  windowClosing(WindowEvent  e){
          System.exit(0);
       }
```

```
        });
    }
    /**
     * 给其他在线用户发送心跳 保持session有效
     */
    private void sendSkip()
    {
        new Thread(){
            public void run()
            {
                try
                {
                    String msg="skip";
                    while(true)
                    {
                        if(STOP==1)
                            break;
                        if(userMap.size()>0)
                        {
                             for (Entry<String, SocketAddress> entry :
userMap.entrySet()) {
                                DatagramPacket data=new
DatagramPacket(msg.getBytes(),msg.getBytes().length,entry.getValue());
                                client.send(data);
                            }
                        }
                        //每10s发送一次心跳
                        Thread.sleep(10*1000);
                    }
                }catch(Exception e){}
            }
        }.start();
    }
    //主要任务是接收数据
    //可以是其他用户发来的信息，也可以是服务器发来的在线用户数据
    public void run()
    {
        try
        {
            String msg;
            DatagramPacket data;
            //执行心跳
            sendSkip();
            while(true)
            {
                if(STOP==1)
```

```
                    break;
                byte[] buf=new byte[1024];
                DatagramPacket packet = new DatagramPacket(buf, buf.length);
                client.receive(packet);
                msg=new String(packet.getData(),0,packet.getLength());
                if(msg.length()>0)
                {
                    if(msg.indexOf("server:")>-1)
                    {
                        //服务器数据 格式server:ID#IP: PORT，。。
                        String
userdata=msg.substring(msg.indexOf(":")+1,msg.length());
                        String[] user=userdata.split(",");
                        for(String u:user)
                        {
                            if(u!=null&&u.length()>0)
                            {
                                String[] udata=u.split("#");
                                String ip=udata[1].split(":")[0];
                                int
port=Integer.parseInt(udata[1].split(":")[1]);
                                ip=ip.substring(1,ip.length());
                                SocketAddress adds=new InetSocketAddress(ip,port);
                                userMap.put(udata[0], adds);
                                //给对方打洞 发送空白报文
                                data=new DatagramPacket(new byte[0],0,adds);
                                client.send(data);
                            }
                        }
                        //更新在线用户列表
                        this.onlineUser.setText("");
                        for (Map.Entry<String, SocketAddress> entry :
userMap.entrySet()) {
                            this.onlineUser.append("用户
"+entry.getKey()+"("+entry.getValue()+")\n");
                        }
                    }
                    else if(msg.indexOf("skip")>-1);
                    else
                    {
                        //普通消息

this.info.append(packet.getAddress().toString()+packet.getPort()+" 说:
"+msg);
                        this.info.append("\n");
                    }
```

```
                }
            }
        }
        catch(Exception e){}
    }
    public static void main(String args[])throws Exception
    {
        String serverIP="127.0.0.1";
        int port=20000;
        //构造一个目标地址
        SocketAddress target = new InetSocketAddress(serverIP, port);
        DatagramSocket client = new DatagramSocket();
        String msg="向服务器报告！";
        byte[] buf=msg.getBytes();
        //向服务器发送上线数据
        DatagramPacket packet=new DatagramPacket(buf,buf.length,target);
        client.send(packet);
        new Client(client).start();
    }
    //按钮事件
    @Override
    public void actionPerformed(ActionEvent e) {
        if(e.getSource()==this.sendButton)
        {
            try{
                String msg=this.msgText.getText();
                if(msg.length()>0)
                {
                    this.info.append("我说: "+msg);
                    this.info.append("\n");
                    for (Map.Entry<String, SocketAddress> entry :
userMap.entrySet()) {
                        DatagramPacket data=new
DatagramPacket(msg.getBytes(),msg.getBytes().length,entry.getValue());
                        client.send(data);
                    }

                    this.msgText.setText("");
                }
            }
            catch(Exception ee){}
        }
    }
}
```

## 题目七 与手机通信编程

### 解题思路：

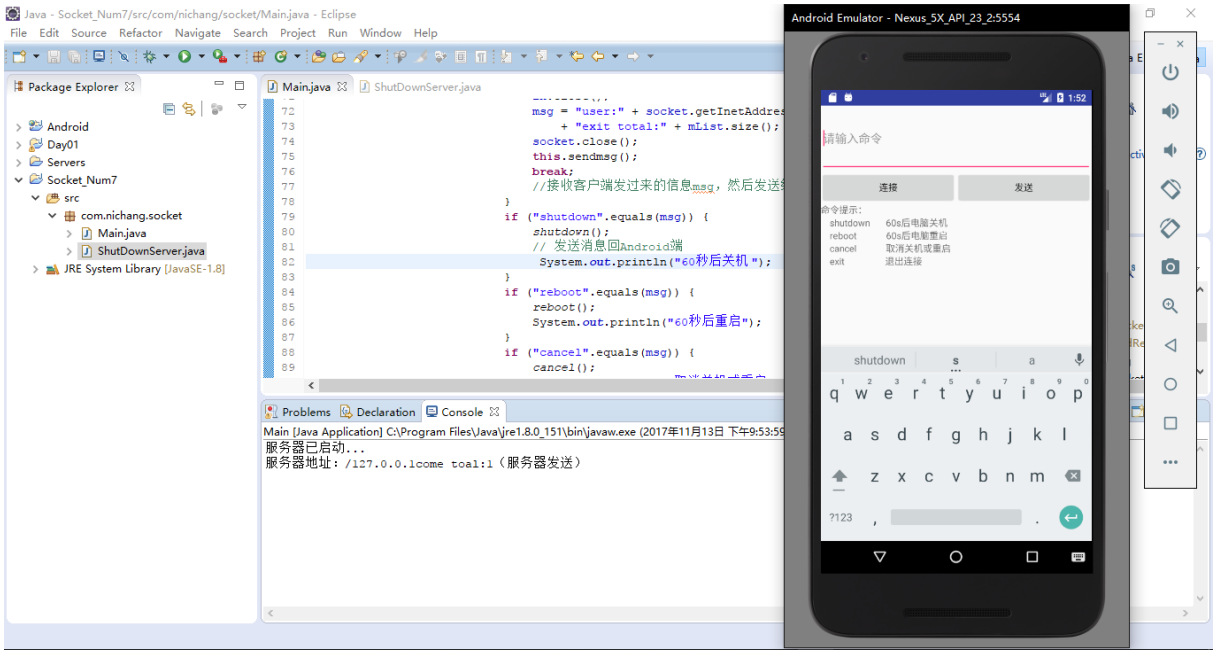
在 Android 中可以直接利用 java 中的 Socket 与 ServerSocket 构建 Socket 通信。客户端发送的文字信息将在服务器端接收并显示，服务器每接收到客户端的文字信息，就会对该文字信息进行判定，如果符合一些规定的指令，就会进入相应的函数模块，进行调用系统指令来操控电脑。这道题的关键是在于 Socket 通信首先要定义好服务端的 IP 地址和端口号，只有 IP 地址和端口号匹配，才能建立连接，才能实现后续的操作。还有一点，因为安卓手机远程操控电脑需要上网权限，所以，还要在安卓的代码里加上权限：

“<uses-permission android:name="android.permission.INTERNET" />”。

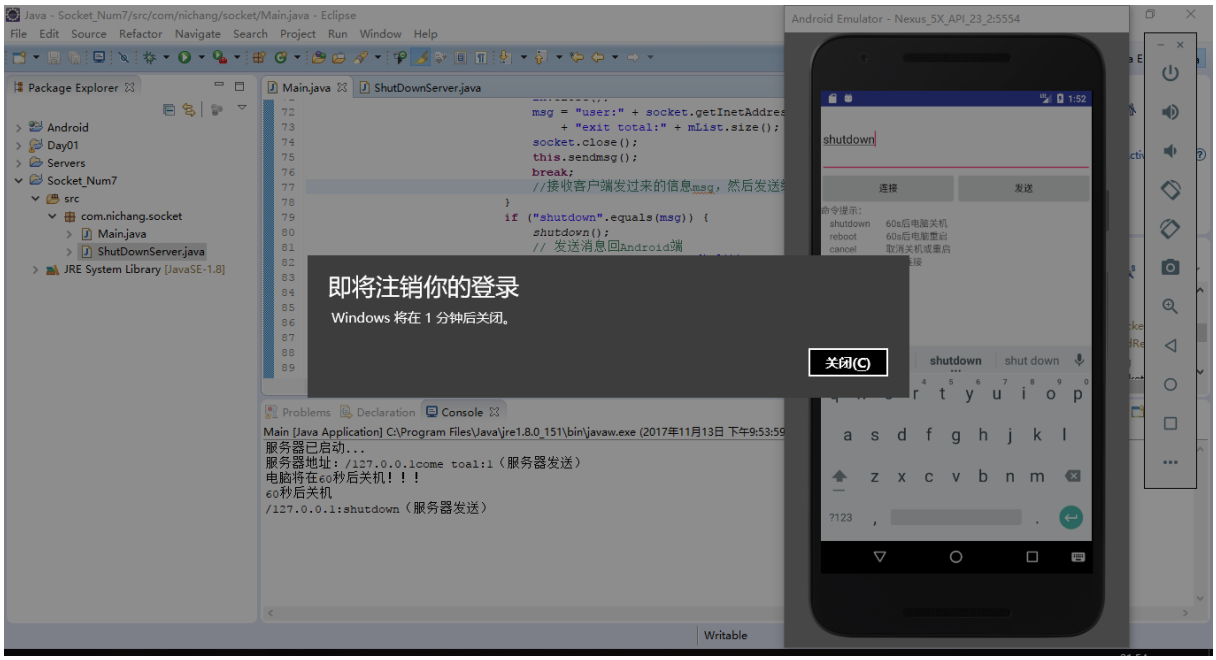
### 实验截图：

**手机客户端主界面：**

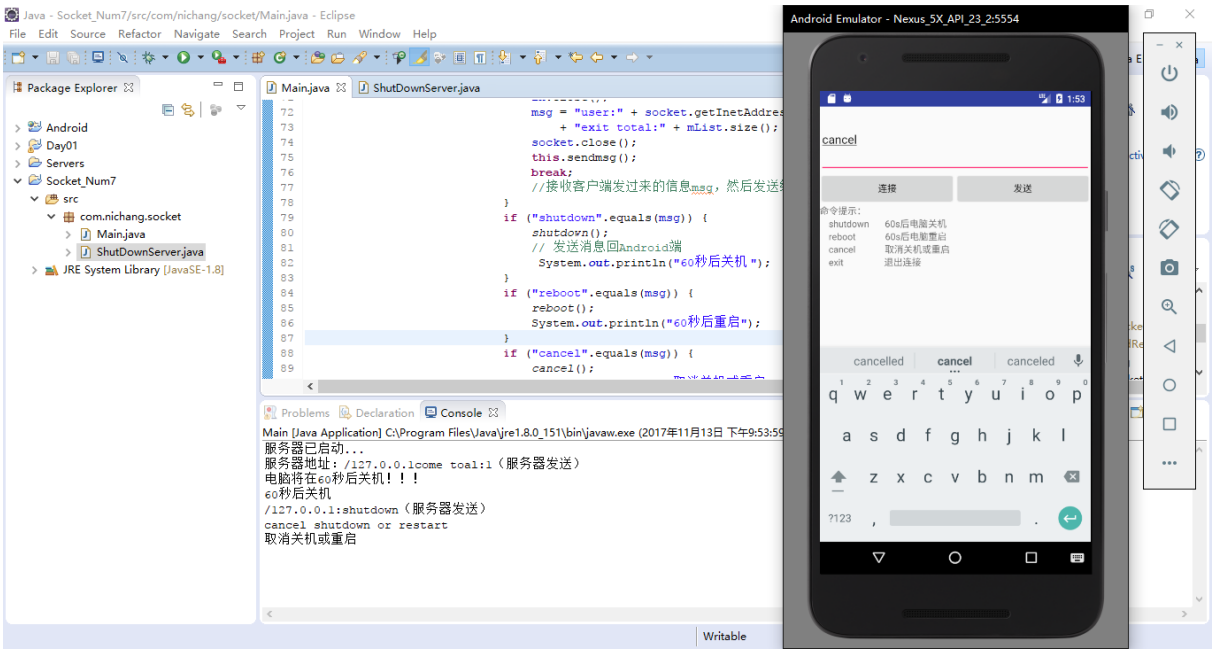
**手机远程连接电脑：**



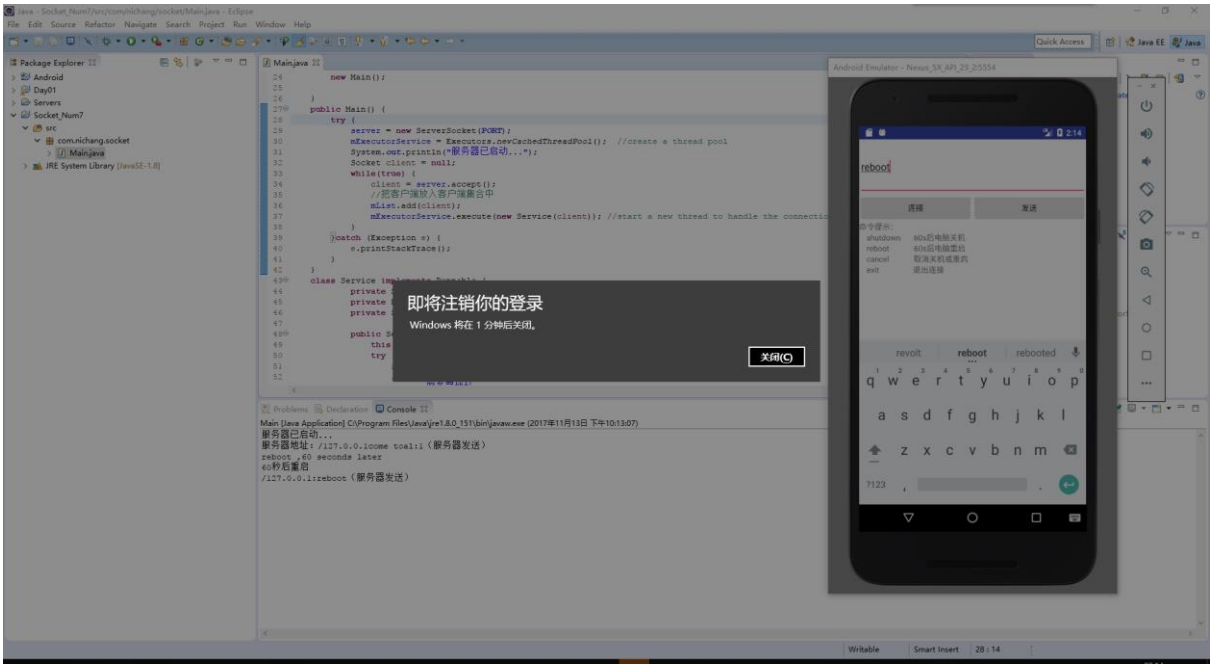

**手机远程电脑关机：**



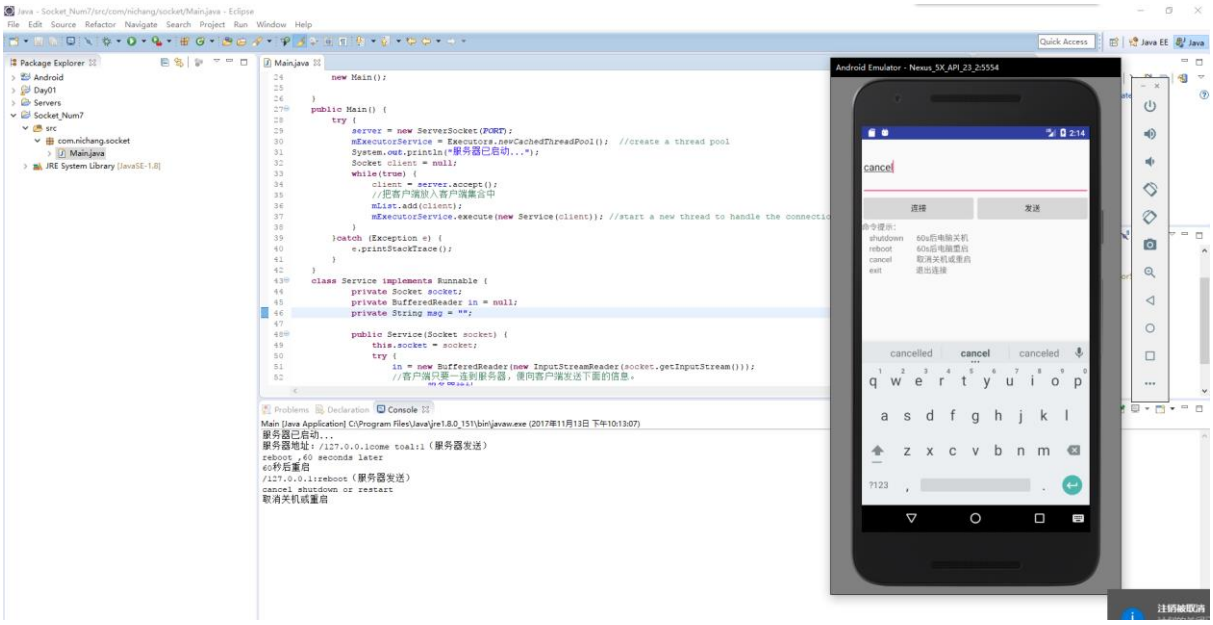
**手机取消远程电脑关机：**



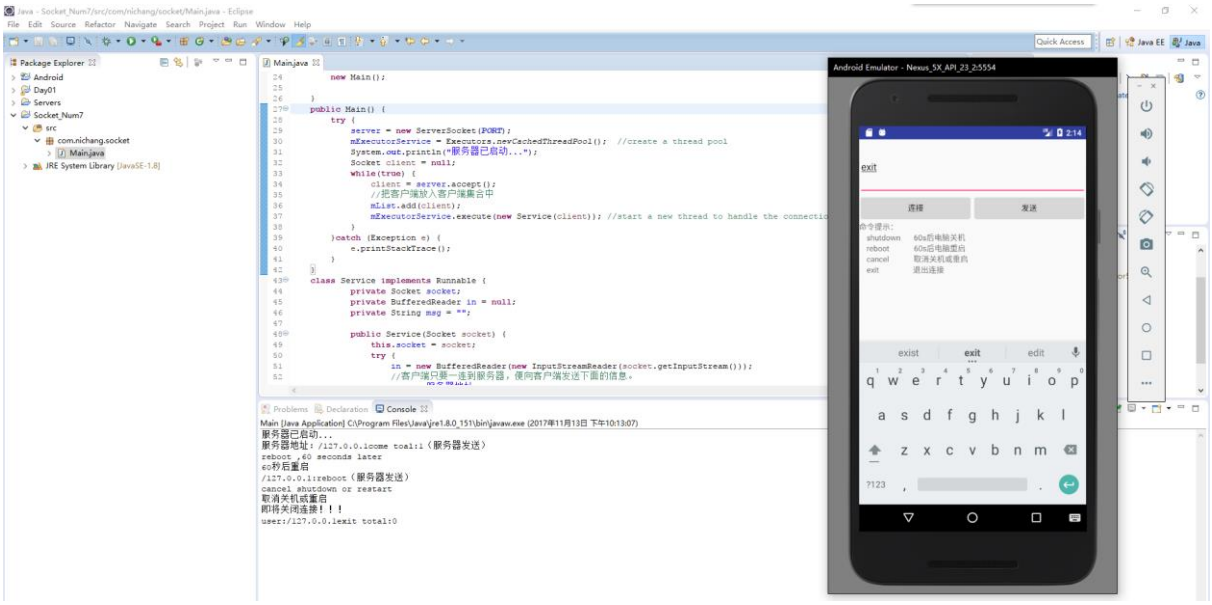

**手机远程重启电脑：**



**手机取消远程重启电脑：**



**手机断开远程连接：**



**解题部分关键代码：**

服务端：

```
public class Main {
    private static final int PORT = 9999;
    private List<Socket> mList = new ArrayList<Socket>();
```

```
    private ServerSocket server = null;
    private ExecutorService mExecutorService = null; //thread pool
    public static void main(String[] args) {
        new Main();
    }
    public Main() {
        try {
            server = new ServerSocket(PORT);
            mExecutorService = Executors.newCachedThreadPool();  //create a
thread pool
            System.out.println("服务器已启动...");
            Socket client = null;
            while(true) {
                client = server.accept();
                //把客户端放入客户端集合中
                mList.add(client);
                mExecutorService.execute(new Service(client)); //start a new
thread to handle the connection
            }
        }catch (Exception e) {
            e.printStackTrace();
        }
    }
    class Service implements Runnable {
            private Socket socket;
            private BufferedReader in = null;
            private String msg = "";
            public Service(Socket socket) {
                this.socket = socket;
                try {
                    in = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
                    //客户端只要一连到服务器，便向客户端发送下面的信息。
                    msg = "服务器地址:" +this.socket.getInetAddress() + "come toal:"
                        +mList.size()+"（服务器发送）";
                    this.sendmsg();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
   @Override
            public void run() {
                try {
                    while(true) {
                        if((msg = in.readLine())!= null) {
                            //当客户端发送的信息为：exit时，关闭连接
```

```
                    if(msg.equals("exit")) {
                        System.out.println("即将关闭连接！！！");
                        mList.remove(socket);
                        in.close();
                        msg = "user:" + socket.getInetAddress()
                            + "exit total:" + mList.size();
                        socket.close();
                        this.sendmsg();
                        break;
                        //接收客户端发过来的信息msg，然后发送给客户端。
                    }
                    if ("shutdown".equals(msg)) {
                        shutdown();
                        // 发送消息回Android端
                         System.out.println("60秒后关机 ");
                    }
                    if ("reboot".equals(msg)) {
                        reboot();
                        System.out.println("60秒后重启");
                    }
                    if ("cancel".equals(msg)) {
                        cancel();
                        System.out.println("取消关机或重启");
                    } else {
                        msg = socket.getInetAddress() + ":" + msg+"（服务器
发送）";

                        this.sendmsg();
                    }
                }
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    /**
     * 循环遍历客户端集合，给每个客户端都发送信息。
     */
    public void sendmsg() {
        System.out.println(msg);
        int num =mList.size();
        for (int index = 0; index < num; index ++) {
            Socket mSocket = mList.get(index);
            PrintWriter pout = null;
            try {
                pout = new PrintWriter(new BufferedWriter(
                        new
```

```
OutputStreamWriter(mSocket.getOutputStream())),true);
                    pout.println(msg);
                }catch (IOException e) {
                    e.printStackTrace();
                }
            }
        }
    }
 // 关机
    private static void shutdown() throws IOException {
        Runtime.getRuntime().exec("shutdown -s -t 60");
        System.out.println("电脑将在60秒后关机！！！");
    }
    // 重启
    private static void reboot() throws IOException {
        Runtime.getRuntime().exec("shutdown -r -t 60");
        System.out.println("reboot ,60 seconds later ");
    }
    // 取消关机或重启
    private static void cancel() throws IOException {
        Runtime.getRuntime().exec("shutdown -a");
        System.out.println("cancel shutdown or restart");
    }
}
```

客户端：

```
public class MainActivity extends Activity implements Runnable {
    private TextView tv_msg = null;
    private EditText ed_msg = null;
    private Button btn_send = null;
    private Button btn_start = null;
    // private Button btn_login = null;
    private static final String HOST = "10.0.2.2";
    private static final int PORT = 9999;
    private Socket socket = null;
    private BufferedReader in = null;
    private PrintWriter out = null;
    private String content = "";
    //接收线程发送过来信息，并用 TextView 显示
    public Handler mHandler = new Handler() {
        public void handleMessage(Message msg) {
            super.handleMessage(msg);
            tv_msg.setText(content);
        }
    };
```

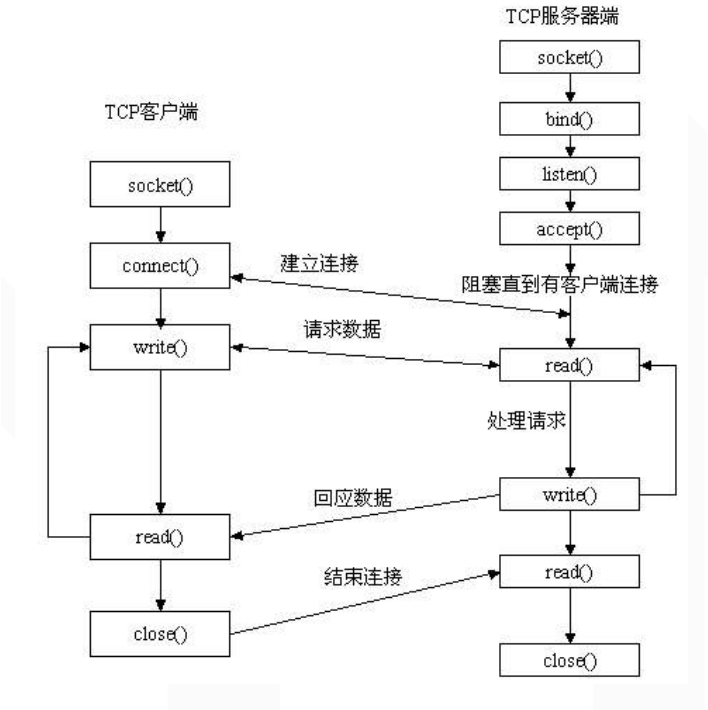
```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    tv_msg = (TextView) findViewById(R.id.textview);
    ed_msg = (EditText) findViewById(R.id.edittext);
    btn_send = (Button) findViewById(R.id.button4);
    btn_start = (Button) findViewById(R.id.button1);
    btn_start.setOnClickListener(new Button.OnClickListener() {
        @Override
        public void onClick(View v) {
           new Thread(){
               @Override
               public void run() {
                   try {
                       socket = new Socket(HOST, PORT);
                       in = new BufferedReader(new InputStreamReader(socket
                               .getInputStream()));
                       out = new PrintWriter(new BufferedWriter(new OutputStreamWriter(
                               socket.getOutputStream())), true);
                   } catch (IOException ex) {
                       ex.printStackTrace();
                       ShowDialog("login exception" + ex.getMessage());
                   }
               }
           }.start();
        }
    });
    btn_send.setOnClickListener(new Button.OnClickListener() {
        @Override
        public void onClick(View v) {
            // TODO Auto-generated method stub
            String msg = ed_msg.getText().toString();
            if (socket.isConnected()) {
                if (!socket.isOutputShutdown()) {
                    out.println(msg);
                }
            }
        }
    });
    //启动线程，接收服务器发送过来的数据
    new Thread(MainActivity.this).start();
```

```
    }
    /**
     * 如果连接出现异常，弹出 AlertDialog!
     */
    public void ShowDialog(String msg) {
        new AlertDialog.Builder(this).setTitle("notification").setMessage(msg)
                .setPositiveButton("ok", new DialogInterface.OnClickListener() {
                    @Override
                    public void onClick(DialogInterface dialog, int which) {
                    }
                }).show();
    }
    public void run() {
        try {
            while (true) {
                if (!socket.isClosed()) {
                    if (socket.isConnected()) {
                        if (!socket.isInputShutdown()) {
                            if ((content = in.readLine()) != null) {
                                content += "\n";
                                mHandler.sendMessage(mHandler.obtainMessage());
                            } else {
                            }
                        }
                    }
                }
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

## Socket 编程学习心得与总结

在计算机通信领域，Socket 被翻译为“套接字”，它是计算机之间进行通信的一种约定或一种方式。通过 Socket 这种约定，一台计算机可以接收其他计算机的数据，也可以向其他计算机发送数据。学习 Socket，也就是学习计算机之间如何通信，并编写出实用的程序。在这门课中，最重要的莫过于是 IP 地址和端口号这两样东西。IP 地址和端口能够在广袤的互联网中定位到要通信的程序，协议和数据传输方式规定了如何传输数据，有了这些，两台计算机就可以通信了。

服务器端先初始化 Socket，然后与端口绑定(bind)，对端口进行监听(listen)，调用 accept 阻塞，等待客户端连接。在这时如果有个客户端初始化一个 Socket，然后连接服务器(connect)，如果连接成功，这时客户端与服务器端的连接就建立了。客户端发送数据请求，服务器端接收请求并处理请求，然后把回应数据发送给客户端，客户端读取数据，最后关闭连接，一次交互结束。下面的这张图具体形象的描述出了 Socket 的工作原理。




当然了，在本文开始我就说过我是通过 Java 来实现的，Java 在包 java.net 中提供了两个类 Socket 和 ServerSocket，分别用来表示双向连接的客户端和服务端。这是两个封装得非常好的类，使用很方便。

网络是和 Java 紧密结合的。Java API 提供用于创建套接字 Socket 的类来便于程序的网络通信。一般而言，一台计算机只有单一的连到网络的物理连接，所有的数据都通过此链接对内、对外送达特定的计算机，这就是端口。网络程序设计中的端口，并非真实的物理存在，而是一种假相的连接装置。

网络中的套接字（Socket）用于将应用程序与端口连接起来，套接字是一个假想的连接装置，就像插座用于连接电器（应用程序）与电线（端口）一样。Java 将套接字抽象化为类，如果要使用套接字，只需创建 Socket 对象即可。

创建一个服务器，需要创建一个服务器套接字 ServerSocket，并把它附加到一个端口上，服务器从这个端口监听连接，端口标志套接字上的 TCP 服务，端口号的范围从0~65536,0~1024是为特权服务保留的端口号。java.net 包中的ServerSocket 类用于表示服务器套接字，其主要功能是等待来自网络上的请求，他可以通过该指

定的端口来等待连接的套接字。服务器套接字一次可以与一个套接字连接。如果多台客户机同时提出连接请求，服务器套接字会将请求连接的客户机存入队列中，然后从中取出一个套接字，与服务器新建的套接字连接起来。若请求的连接数大于最大容纳的数，则多出的连接请求被拒绝。队列的默认大小是 50。

调用 ServerSocket 类的 accept()方法会返回一个和客户端 Socket 对象相连接的 Socket 对象，服务器端的 Socket 对象使用 getOutputStream()方法获得的输出流将指向客户端 Socket 对象使用 getInputStream()方法获得的那个输入流；同样，服务器端的 Socket 对象使用 getInputStream()方法获得的输入流将指向客户端 Socket 对象使用 getOutputStream()方法获得的那个输出流。也就是说，当服务器想输出流写入信息时，客户端通过相对应的输入流就能读取服务器发送过来的消息，反之亦然。得到一个 socket 之后，就可以利用 socket.getInputStream()或者 socket.getOutputStream()方法得到输入输出流，这样通过流的方式就可以进行网络通信了。

本次课程设计是我第一次学习网络编程。以前的课程从来没有接触到网络方面的知识。这次的新知识对我的挑战还算不是太大，通过努力，在这次课程设计中我还是学到了不少网络编程新知识。其实，实践应该是我们的重点，在以后的工作中要的就是我们的实际的动手能力，如果我们在学习期间就是只学了书本上的知识，那样对理论的了解是不够深刻的，只有通过实践才能激发学习兴趣。总之，我觉得实践才是检验理论的唯一标准。在今后，我将进一步学习了解网络编程，深度理解运用 Socket 编程，并将所学的知识运用到我的其他项目中，比如我的个人网站和微信公众号，在实际的项目中运用，知识的掌握速度最快。