# 网络程序设计实验报告

# ——手机触屏控制 PC 机鼠标

## 1. 实验目标

了解Android开发环境、Wifi技术特点，掌握Android环境下的网络程序设计基本方法。实验内容：在Android手机上读取触屏数据，将人手在触屏上的滑动轨迹通过wifi网络发送到PC机上；PC机上安装鼠标控制程序，根据手机发来的数据，鼠标的光标作相应移动。

（1） 在手机上编程，读取人手在触屏上的滑动信息；

（2） 通过wifi网络，把滑动数据发送到PC机;

（3） PC机根据wifi送达的数据控制鼠标光标移动；

网络中断时PC和手机都有错误显示；网络连接恢复时手机端要有指示。

## 2. 实验步骤

### 基本设计原理

本程序由 PC 端（服务端）、Android 端（客户端）共同组成，采用 Server-Client 设计思路，Android 端将触摸屏采集到的手势移动信息，以及按键点击信息发送到 PC 端，PC 端收到数据包后，处理手势移动信息，并调用 java.awt.Robot 库的相关方法，实现对 PC 机器鼠标的控制。

本程序 PC 端采用 Java 语言编写，采用 swing 实现前端界面，PC端首先扫描计算机的网卡信息，获得网卡绑定的ip地址，从中筛选出内网 ip，交由用户选择监听地址，建立 ServerSocket，启动服务端监听 socket，等待客户端连接。并生成包含 ip 地址和端口信息的二维码，以供客户端扫描。



客户端为 Android 应用程序，程序启动后，用户扫描PC端生成的二维码，客户端获取到 PC 端的ip地址和端口，尝试建立 TCP 连接，连接建立成功后，程序捕捉用户的手势信息，并将信息通过 TCP 报文发送到服务端。



### 实现步骤

1. 服务端初始化

* 初始化时，使用 swing 显示图形化界面，筛选内网 ip ，交给用户选择监听ip，并设置图形化界面的回调函数
* /\*\*   
   \* 服务端初始化   
   \* @param port 监听端口   
   \* @throws IOException   
   \*/   
   public MobileControlSever(Mode mode, int port) throws IOException {   
   show(); // 显示图形化界面   
   mMode = mode;   
   mPort = port;   
   mServerSocket = new ServerSocket(); // 创建服务端监听 Socket   
   try {   
   mRobot = new Robot();   
   } catch (AWTException e) {   
   e.printStackTrace();   
   }   
   mIpAddrList = getIpList(); // 扫描网卡ip，从中筛选出内网ip   
   mainWindow.setAddrList(mIpAddrList); // ip 地址显示在界面中   
   mainWindow.setComboBoxCallback(this.ipSelectedCallback()); // 设置下拉菜单的回调函数   
   mainWindow.setApplyButtonCallback(this.startButtonCallback()); // 设置开始按钮回调函数   
   mainWindow.setStopButtonCallback(this.stopButtonCallback()); // 设置停止按钮回调函数   
   mainWindow.setInfoLabel("等待设定监听 ip，完成后点击开始运行按钮");   
   this.mIpAddr = mIpAddrList.get(0);   
   }
* 当用户点击开始按钮后，执行回调函数
* /\*\*   
   \* 获取点击按钮的回调函数   
   \*   
   \* @return ActionListener 回调函数   
   \* @throws IOException   
   \*/   
   public ActionListener startButtonCallback() throws IOException {   
   ActionListener actionListener = new ActionListener() {   
   @Override   
   public void actionPerformed(ActionEvent e) {   
   System.out.println("my ip: " + mIpAddr);   
   isStop = false;   
   if (mIpAddr == null || mIpAddr.length() == 0) {   
   throw new IllegalArgumentException("IP address is null!");   
   }   
   if (mServerSocket.isClosed()) {   
   try {   
   mServerSocket = new ServerSocket();   
   } catch (IOException ioException) {   
   ioException.printStackTrace();   
   }   
   }   
   if (!mServerSocket.isBound()) {   
   try {   
   mServerSocket.bind(new InetSocketAddress(mIpAddr, mPort)); // 监听用户设定的ip地址   
   } catch (IOException exception) {   
   exception.printStackTrace(); // 端口已经被占用   
   }   
   }   
   if (!isAlive()) {   
   start(); // 启动循环线程，在循环中不断接受客户端发送的数据并处理   
   }   
   }   
   };   
   return actionListener;   
   }

1. 服务端的主运行线程

* 服务端数据包的接收和处理相关实现在一个单独的线程中执行，其主要代码如下：
* while (true) {   
   DataInputStream in = null;   
   try {   
   System.out.println("Waiting for client on port " + mServerSocket.getLocalPort() + "...");   
   // System.out.println("Server ip: "   
   // + mServerSocket.getInetAddress());   
   System.out.println("Just connected to " + server.getRemoteSocketAddress());   
   if (isStop) {   
   throw new IOException();   
   }   
   in = new DataInputStream(server.getInputStream()); // 阻塞式等待客户端数据包   
   String inStr = in.readUTF(); // 使用 UTF 格式进行解码   
   System.out.println(inStr);   
   handleTouchData(inStr); // 处理收到的数据，并模拟鼠标操作   
   } catch (SocketTimeoutException s) {   
   System.out.println("Socket timed out!");   
   break;   
   } catch (IOException e) {   
   //client disconnected   
   e.printStackTrace();   
   System.out.println("client disconnected");   
   try {   
   in.close();   
   server.close();   
   in = null;   
   server = null;   
   } catch (Exception ee) {   
   e.printStackTrace();   
   }   
   break;   
   }

1. 模拟鼠标操作

* 从客户端发送的数据为使用逗号 ”，“ 分隔的控制数据，其第一个字段表示鼠标动作命令：
  + L：鼠标左键点击
  + R：鼠标右键点击
  + M：鼠标移动
* 当第一个字段为 M 时，后续两个字段表示鼠标在 x 轴和 y 轴移动的距离
* /\*\*   
   \* 处理从客户端发送来的数据   
   \* @param t   
   \*/   
   private void handleTouchData(String t) throws IOException {   
   if (t != null && t.length() > 0) {   
   String[] segs = t.split(",");   
   int segLen = segs.length;   
   if (segLen > 1 && "mctd".equals(segs[0])) {   
   String type = segs[1];   
   if ("L".equals(type)) {   
   leftClick();   
   } else if ("R".equals(type)) {   
   rightClick();   
   } else if ("M".equals(type)) {   
   if (segLen > 3) {   
   String x = segs[2], y = segs[3];   
   System.out.println("handle move, x: " + x + ", y: " + y);   
   try {   
   int xInt = Integer.valueOf(x);   
   int yInt = Integer.valueOf(y);   
   if (xInt != 0 || yInt != 0) {   
   mouseMove(mMouseX + xInt, mMouseY + yInt);   
   }   
   } catch (NumberFormatException e) {   
   e.printStackTrace();   
   }   
   }   
   }   
   } else if (segLen == 1 && "heartbeat".equals(segs[0])) {   
   System.out.println("heart beat received");   
   mHeartBeatTime = System.currentTimeMillis();   
   mOutputStream.writeUTF("heartbeat"); // 回应心跳包   
   }   
   else {   
   System.out.println(t);   
   }   
   }   
   }
* 获取到鼠标的对应命令后，调用相关的函数，模拟鼠标的操作，以 mouseMove 为例：
* /\*\*   
   \* @param x   
   \* to x   
   \* @param y   
   \* to y   
   \*/   
   private void mouseMove(int x, int y) {   
   System.out.println("mouseMove called, x: " + x + ", y: " + y);   
   if (mRobot != null) {   
   mRobot.mouseMove(x, y);   
   }   
     
   updateCurrentMousePosition();   
   }
* 该函数对 java 的 java.awt.Robot 类进行了封装，实现了鼠标的模拟移动

1. 客户端初始化

* 客户端首先需要扫描服务端生成的二维码，扫描二维码由 zxing 库实现相关功能，并将获取到的服务端 ip 地址以字符串的形式提供给后端类
* public void onQRScanEnd(String serverIp) {   
   mHandler.removeCallbacks(heartBeatTask);   
   boolean found = !TextUtils.isEmpty(serverIp);   
   if (found) {   
   mServerIp = serverIp; // 设定服务端ip   
   } else {   
   quitAndMarkDisconnected(); // 若未获取到ip设定状态为未连接   
   }   
   connect(mServerIp, SERVER\_TCP\_PORT); // 尝试连接服务端   
   if (mClientSoc == null || !mClientSoc.isConnected()) {   
   quitAndMarkDisconnected();   
   return;   
   }   
   mOnConnectListener.onFindServerComplete(found);   
   mConnectionStatus = CONNECTED; // 设定状态为已连接   
   Thread listenThread = new listenThread();   
   listenThread.start();   
   new Thread(heartBeatTask).start(); // 启动心跳包   
   }
* 获取到服务端ip后，调用 connect 方法，尝试建立 socket 连接，并设定连接类型为长连接
* private void connect(String serverName, int port) {   
   try {   
   Log.d(TAG, "Connecting to server " + serverName + " on port " + port);   
   mClientSoc = new Socket(serverName, port);   
   mClientSoc.setKeepAlive(true);   
   Log.d(TAG, "Just connected to server " + mClientSoc.getRemoteSocketAddress());   
   } catch (UnknownHostException e) {   
   Log.e(TAG, "xxxxxxxxxxxxxxxxx", e);   
   } catch (IOException e) {   
   Log.e(TAG, "xxxxxxxxxxxxxxxxx", e);   
   }   
   }

1. 捕捉用户手势

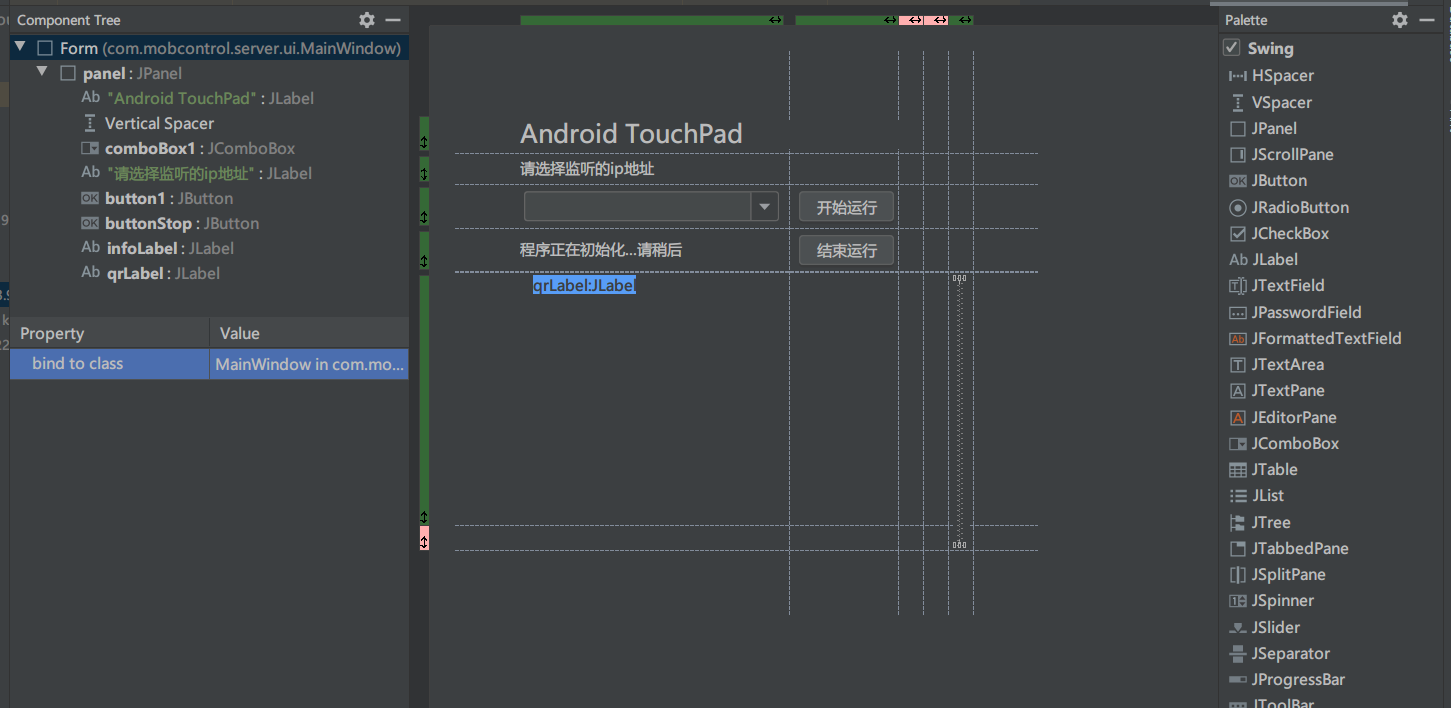
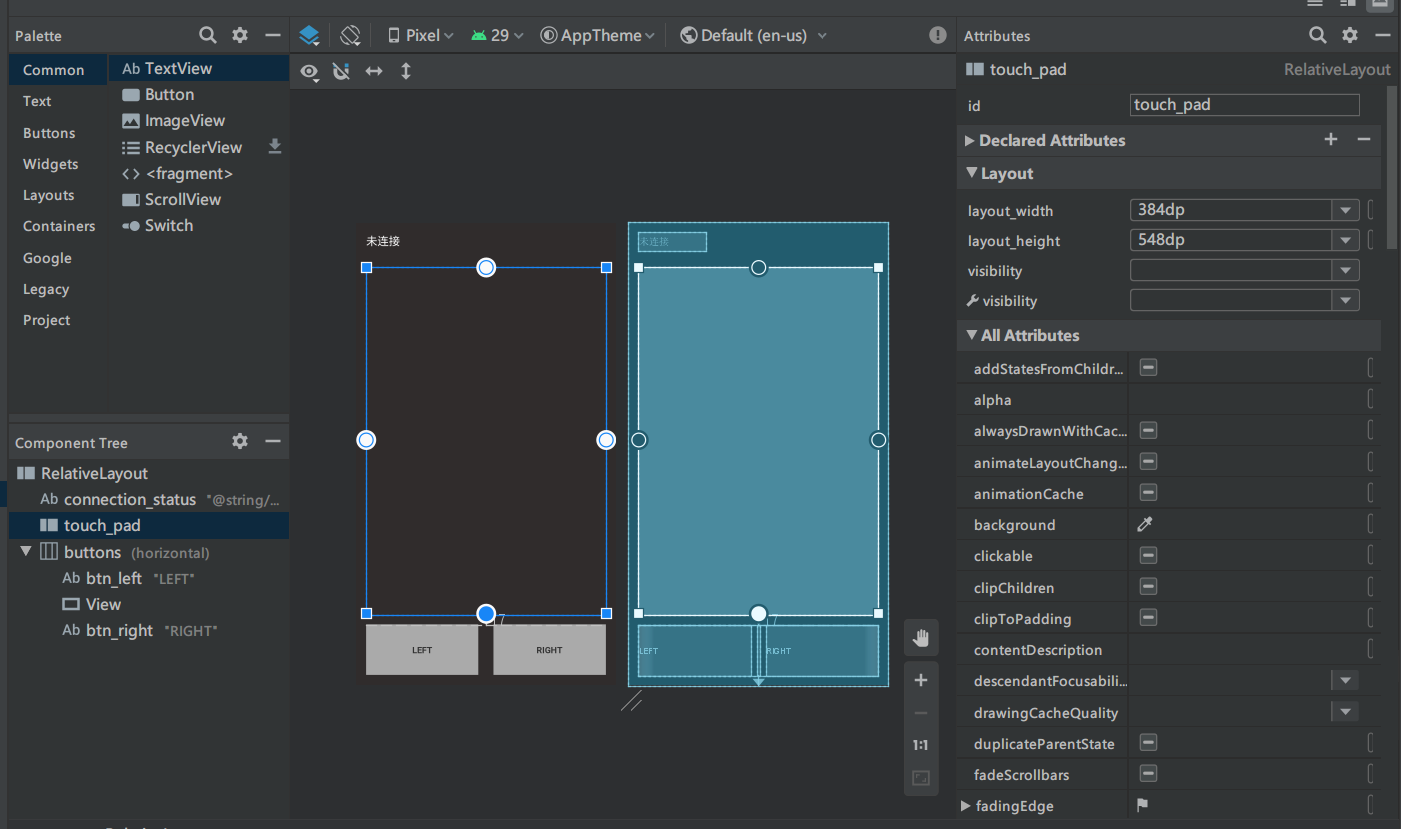
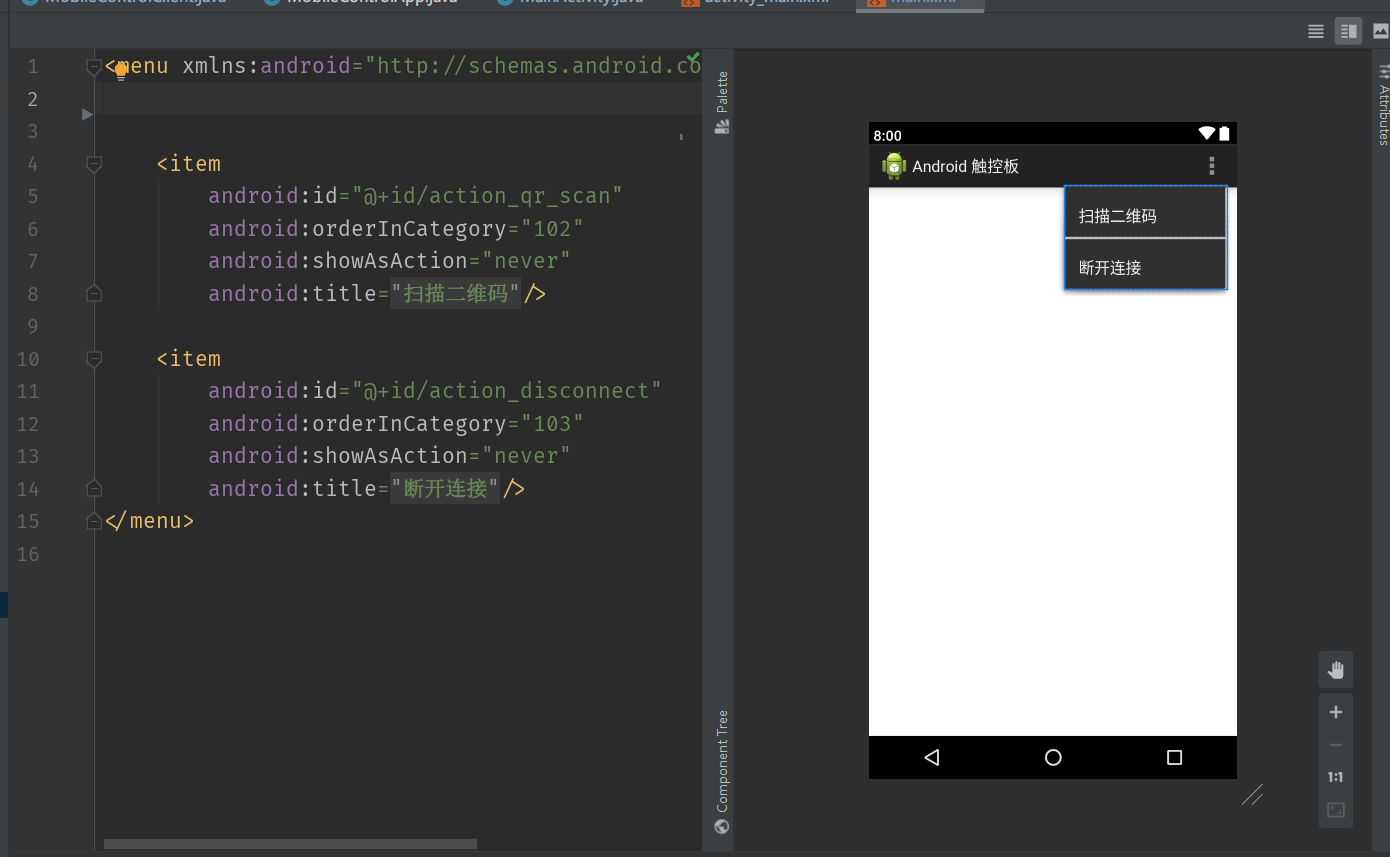
* 在 Android 界面中，建立了一个 id 为 touch\_pad 的界面元素，用户可以触摸该元素，实现手势的输入，因此，需要为该界面元素设置触控回调函数，当用户触摸该元素时，记录用户手指移动轨迹
* findViewById(R.id.touch\_pad).setOnTouchListener(new OnTouchListener() {   
     
   float lastX, lastY, downX, downY;   
   long downTime;   
     
   @Override   
   public boolean onTouch(View v, MotionEvent event) {   
   switch (event.getAction()) {   
   case MotionEvent.ACTION\_DOWN: //手指按下   
   downX = lastX = event.getX();   
   downY = lastY = event.getY();   
   downTime = System.currentTimeMillis();   
   break;   
   case MotionEvent.ACTION\_MOVE: // 手指移动   
   float x = event.getX(); // 记录移动向量   
   float y = event.getY();   
   float dx = x - lastX;   
   float dy = y - lastY;   
   lastX = x;   
   lastY = y;   
     
   TouchData td = new TouchData();   
   td.setType(TouchData.TOUCH\_TYPE\_MOVE);   
   td.setX(mSpeed \* (int) dx);   
   td.setY(mSpeed \* (int) dy);   
   send(td);   
   break;   
   case MotionEvent.ACTION\_UP: // 手指抬起   
   x = event.getX();   
   y = event.getY();   
   dx = x - downX;   
   dy = y - downY;   
   if (dx < 2 && dy < 2) {   
   // this is a click event   
   TouchData td\_c = new TouchData();   
   long tx = System.currentTimeMillis() - downTime;   
   Log.d(TAG, "tx " + tx);   
   td\_c.setType(tx > mLongClickTime ? TouchData.TOUCH\_TYPE\_LONG\_CLICK   
   : TouchData.TOUCH\_TYPE\_CLICK);   
   send(td\_c);   
   }   
   break;   
   }   
   return true;   
   }   
   });

1. 发送用户手势到服务端

* /\*\*   
   \* 向服务端发送手势数据   
   \* @param td   
   \*/   
   private void send(TouchData td) {   
   if (td == null || !mControlClient.isConnected()) {   
   return;   
   }   
   StringBuilder sb = new StringBuilder();   
   sb.append(td.getHead()).append(",")   
   .append(td.getType()).append(",")   
   .append(td.getX()).append(",")   
   .append(td.getY());   
   String jsonStr = sb.toString();   
   Log.d(TAG, "send: " + jsonStr);   
   mControlClient.send(jsonStr);   
   }

至此，实现了 Android 手机控制 PC 端鼠标操作。

### 前端界面设计

* 服务端界面设计
* 服务端采用 swing 进行界面设计：
* 
* 界面中设计了下拉菜单 comboBox 用于选择监听ip，两个按钮用于控制服务的启动停止，qrLabel 用于显示供客户端连接的二维码，使用 JLabel 标签显示程序的重要日志信息
* 客户端界面设计
* 
* 客户端顶部显示了当前的连接状态，中间为一片空白的 Layout 区域，用于模拟触控板，底部两个按钮模拟鼠标的左键和右键
* 
* 点击程序的右上角菜单可以选择扫描二维码或断开连接

### 连接状态监测

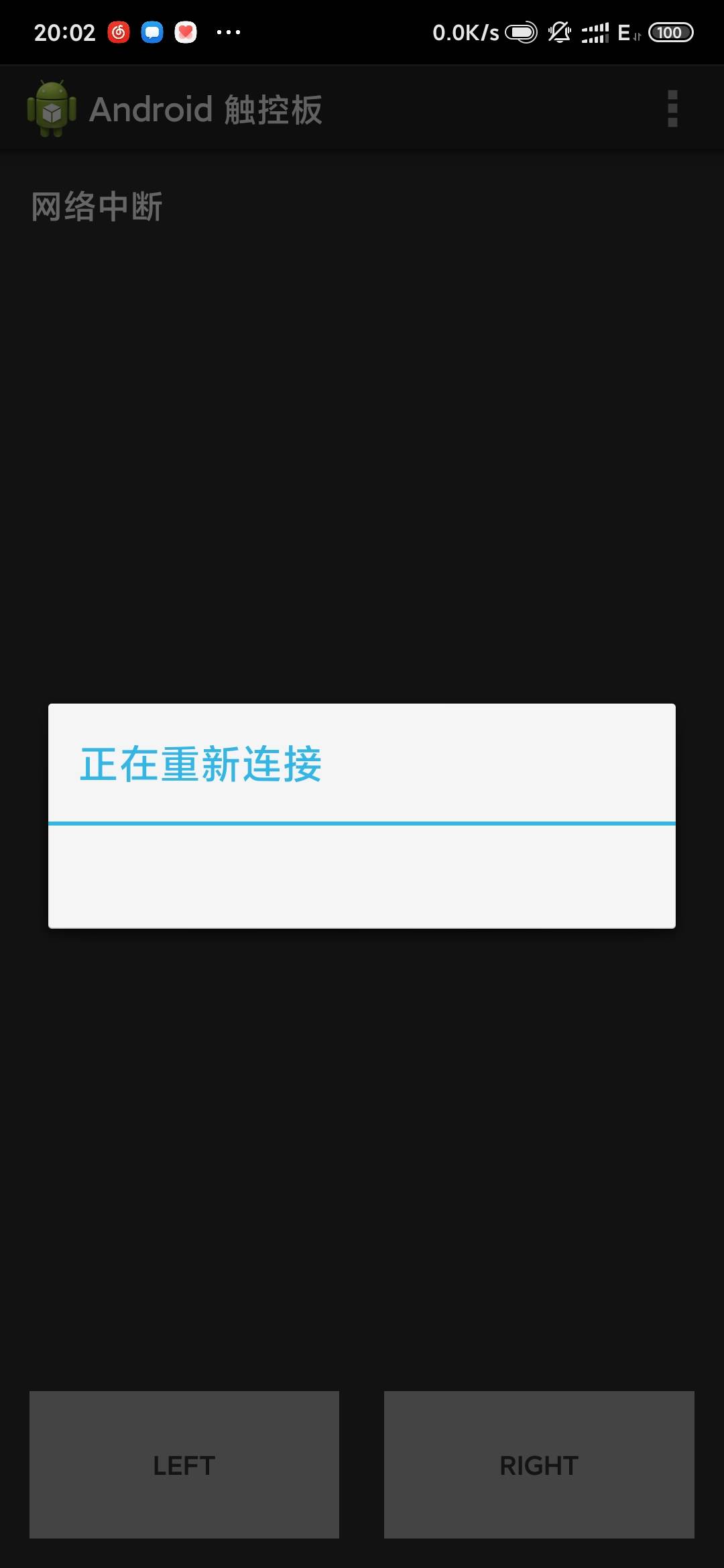
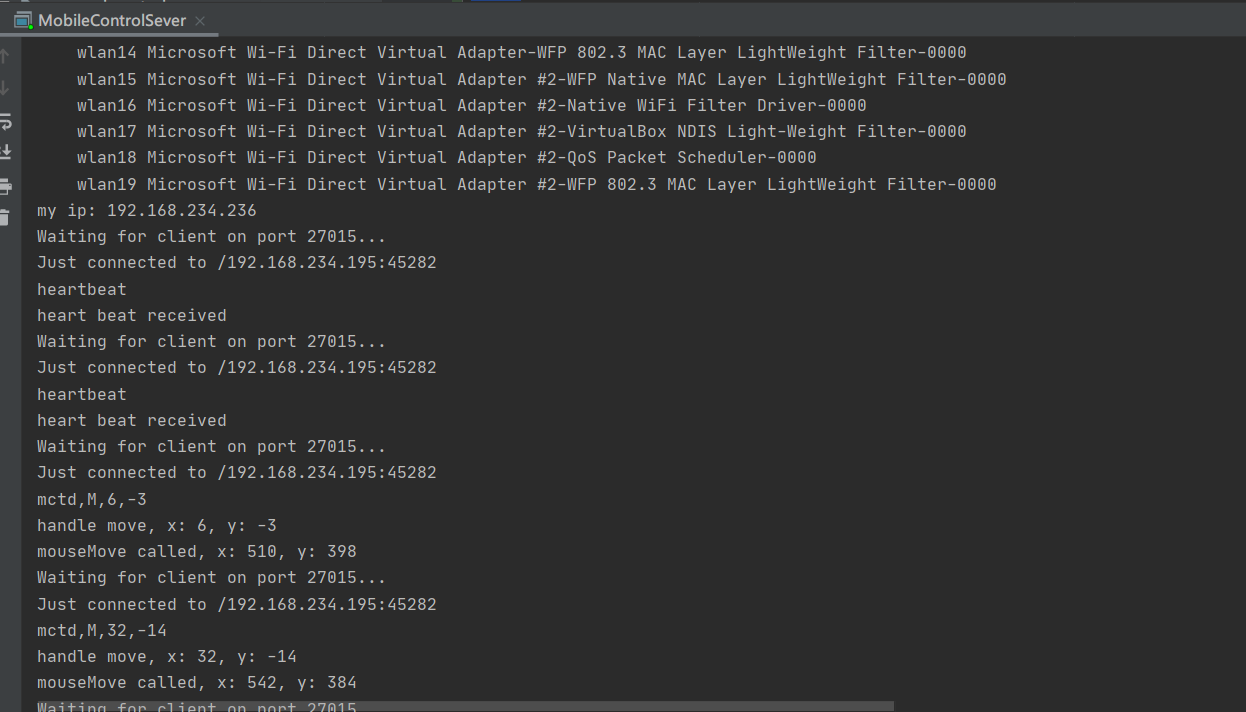
由于网络连接可能存在不稳定性，因此设计了心跳机制，监测并维持客户端和服务端的连接，客户端每隔 1s 会向服务端发送一个心跳包：

private final Runnable heartBeatTask = new Runnable() {   
 @Override   
 public void run() {   
 if (mConnectionStatus != CONNECTED) {   
 return;   
 }   
 try {   
 send("heartbeat");   
 } catch (Exception e) {   
 e.printStackTrace();   
 return;   
 }   
 mHandler.postDelayed(heartBeatTask, 1000);   
 mUIHandler.postDelayed(heartBeatTimeoutTask, 3000); // 三秒未收到回应，判定连接中断，尝试重新连接   
 }   
 };

服务端接收到客户端的心跳包之后，同样回应一个心跳包作为响应，服务端还会运行一个看门狗线程，当超过3秒未收到心跳包，则判定链接中断，等待客户端重新连接

/\*\*   
 \* 看门狗线程   
 \*/   
 class WatchDogThread extends Thread {   
 public void run() {   
 while(true) {   
 try {   
 Thread.sleep(500);   
 long delta = System.currentTimeMillis() - mHeartBeatTime;   
 if (isStop) {   
 return;   
 }   
 if (delta > 5000) {   
 System.out.println("Heartbeat lost, close socket");   
 mIsAlive = false;   
 mSocket.close();   
 mainWindow.setInfoLabel("网络中断，等待客户端重新连接");   
 break;   
 }   
 } catch (InterruptedException | IOException e) {   
 e.printStackTrace();   
 }   
 }   
 }   
 }

## 3. 实验结果

* 服务端初始化：
* 
* 客户端初始化：
* 
* 服务端丢失连接：
* 
* 客户端丢失连接
* 
* 服务端控制台日志：
* 
* 客户端 Logcat 日志：
* 