河南理工大学

JAVA课程设计报告

2016 — 2017学年第 2 学期



课程名称 JAVA课程设计

作品题目 远程屏幕监控系统

学生姓名 刘壮飞

学 号 311509060128

专业班级 软件1501

联系电话 13937267417

指导教师 智慧来

2017年5月14日

目录

摘要…………………………………………………………………………………1

一、项目分析………………………………………………………………………2

1.1需求性分析

1.2可行性分析

1.3技术点分析

二、功能实现………………………………………………………………………3

2.1系统托盘模块……………………………………………………………3

2.1.1系统托盘是个什么东西?

2.1.2有必要实现系统托盘吗?

2.1.3怎么实现系统托盘

2.1.4实现系统托盘代码

2.2自定义协议………………………………………………………………5

2.2.1网络协议

2.2.2为什么自定义协议?

2.2.3自定义协议

2.2.4自定义协议相关类说明

2.2.5核心代码

2.3客户端模块………………………………………………………………7

2.3.1获取屏幕截图

2.3.2将图片以协议的规范发到服务器

2.3.3系统退出机制

2.3.4客户端代码

2.4服务器端模块……………………………………………………………9

2.4.1服务器逻辑

2.4.2对多客户端的处理

2.4.3服务器端对客户端消息请求的处理

三、Web服务器环境配置………………………………………………………11

3.1简述

3.2工具

3.3搭建过程

3.4使用方法

四、项目难点……………………………………………………………………12

4.1客户端循环发送图片的问题

4.2服务器端线程里操作JPanel重绘

4.3对异常的处理

五、展望…………………………………………………………………………14

六、感想…………………………………………………………………………15

附件：软件运行效果图…………………………………………………………16

摘要

远程屏幕监控系统在生活中是很常见的，学校机房的机房管理系统、PC版QQ的远程演示功能等都属于远程屏幕监控系统。监控系统的原理是通过客户端不断的截取屏幕发送到服务器端，服务器端进而将画面呈现出来的过程。本论文实现的是一个多客户端的远程屏幕监控系统。

本论文第一部分对系统进行项目分析，包括需求分析、可行性分析、相关技术分析，大致介绍了整个项目需要做的工作以及需要掌握的技术，介绍了Socket通信原理、截屏原理、Swing树、系统托盘、自定义JPanel实现显示监控图像以及多线程的知识。

第二部分分别对系统托盘模块、自定义协议模块、获取屏幕截图模块、连续发送与接收图片模块、登录、退出模块、多客户端处理模块、Swing树模块、自定义JPanel模块进行介绍。我没有直接搬上一大堆的理论知识，而是先简要介绍模块功能，然后按照正常思考的思路去实现项目需要的功能，并且去分析实现这个功能的必要性。遇到问题之后就分析出现这个问题的原因以及考虑如何去提升效率、减少存储空间等一系列优化问题。然后通过最后的分析给出一个优化后的解决方案，同时我将自己当时思考的错误点也罗列了出来，对多个处理方法都给予了尝试。针对每个模块都给出了功能的实现详细步骤以及示例代码。

第三部分是Web服务器环境配置以及程序使用说明。本项目是远程屏幕监控系统，如果要测试的话，服务器端的程序是需要部署在服务器上的，所以我将本机Web服务器环境配置的方法也讲解一下，另外还有关于本程序代码如何打包等知识都有讲解。

第四部分是我在写项目的过程中的犯的一些错误以及项目的难点，第五部分是对该系统后续的一些功能的设想，第六部分是我的一些感想，第七部分是项目运行效果的展示。

关键字：屏幕监控；Socket；Swing；自定义协议；Web服务器环境配置

**一、项目分析**

**1.1需求性分析**

项目的初始阶段就是对整个系统进行预估，这有利于我们对整个系统的理解，屏幕监控系统需要实现的功能有：

a.客户端登录、退出

b.客户端截屏以及连续发送图像

c.客户端系统托盘功能

d.服务器端连续接收图像以及客户端其他请求

e.服务器端显示连接用户的用户树

f.客户端退出后用户树刷新

g.客户端发送图像后显示在服务器端

**1.2可行性分析**

需求性分析里提到的功能能否实现呢?我们在这里进行讨论：

a.通过构造自定义协议实现,都是通过将这些请求构造成协议从而发送到服务器

b.截屏功能通过Robot类实现，然后将BufferedImage转化为字节数组输出流，再转化为字节数组，并以协议的方式发送到服务器实现图像的连续发送。

c.使用系统托盘对象SystemTray来实现。

d.可以通过自定义协议工具类提供的解析数据的方法解析出数据，并根据消息类型进行相应的处理。

e.用户树使用JTree实现，DefaultTreeCellRenderer可以设置树的外观，为JTree设置节点选中监听器可以监听节点选中事件

f. 用DefaultTreeModel的reload()方法实现

g.可以自定义JPanel，通过paint(g)方法绘制图片

**1.3技术点分析**

**Socket**

网络上的两个程序通过一个双向的通信连接实现数据的交换，这个连接的一端称为一个socket，java的API提供了对Socket的支持。

**自定义网络协议**

网络协议为[计算机网络](http://baike.baidu.com/item/%E8%AE%A1%E7%AE%97%E6%9C%BA%E7%BD%91%E7%BB%9C)中进行数据交换而建立的规则、标准或约定的集合。为了满足我们的需求，我们需要自定义一个协议，并为其提供发送消息、解析消息的功能。

**系统托盘**

系统托盘是个特殊区域，通常在桌面的底部，项目中涉及到了对系统托盘的一些操作，我们为客户端提供系统托盘功能，可以方便用户关闭监控。

**IO流**

流是一种抽象概念，它代表了数据的无结构化传递。按照流的方式进行输入输出，数据被当成无结构的字节序或字符序列。从流中取得数据的操作称为提取操作，而向流中添加数据的操作称为插入操作，用来进行输入输出操作的流就称为IO流。换句话说，IO流就是以流的方式进行输入输出。我们主要使用的有DataOutputStream、DataInputSream、ByteArrayoOutputStream等。

**屏幕截图**

使用Robot类实现屏幕截取以及事件回放操作。

**AWT与SWING**

抽象窗口工具包，该包提供了一套与本地图形界面进行交互的[接口](http://baike.baidu.com/item/%E6%8E%A5%E5%8F%A3/15422203)，是Java提供的用来建立和设置Java的[图形用户界面](http://baike.baidu.com/item/%E5%9B%BE%E5%BD%A2%E7%94%A8%E6%88%B7%E7%95%8C%E9%9D%A2)的基本工具；以抽象窗口工具包为基础使跨平台应用程序可以使用任何可插拔的外观风格。该项目主要是用到了窗口以及树控件、树的刷新、树的节点外观、节点选择事件处理等技术。

**自定义JPanel**

JPanel代表一个面板，通过实现一个继承自JPanel的DrawPanel，重写其paint(g)方法实现将图像画到视图上，如果不断修改绘制的图片，在速度达到的情况下可以实现屏幕监控画面显示的功能。

**多线程**

多线程是指从软件或者硬件上实现多个线程并发执行的技术。具有多线程能力的计算机因有硬件支持而能够在同一时间执行多于一个线程，进而提升整体处理性能。多线程是为了同步完成多项[任务](http://baike.baidu.com/item/%E4%BB%BB%E5%8A%A1)，不是为了提高运行效率，而是为了提高[资源](http://baike.baidu.com/item/%E8%B5%84%E6%BA%90)使用效率来提高系统的效率。

**二、功能实现**

**2.1系统托盘模块**

**2.1.1系统托盘是个什么东西？**

系统托盘是个特殊区域，通常在桌面的底部，在那里，用户可以随时访问正在运行中的那些程序。在微软的Windows里，系统托盘常指任务栏的状态区域；在每个系统里，托盘是所有正运行在[桌面环境](http://baike.baidu.com/item/%E6%A1%8C%E9%9D%A2%E7%8E%AF%E5%A2%83)里的应用程序共享的区域。

**2.1.2有必要实现系统托盘吗？**

回答是肯定的，当前的大部分软件都会提供一个系统托盘让用户更加方便的操作，QQ的系统托盘左键可以打开QQ窗口，右键可以选择退出账号、注销账号、更改状态等一系列操作客户端是负责将屏幕截图发到服务器以及执行一些收到的指令，也需要与服务器端做一些交互，比如：登录、发消息、退出等操作，如果把这些处理操作放到系统托盘里可以增大用户粘性，使用户可以更方便使用系统。

**2.1.3怎么实现系统托盘？**

JAVA的API提供了一系列关于系统托盘的类与方法，为软件添加系统托盘功能的步骤如下：

a.我们先把图片放到src同级目录下

b.首先获取图片的Image

c.根据Image创建托盘图标TrayIcon

d.创建系统托盘对象SystemTray

e.创建弹出菜单PopupMenu，并为其添加MenuItem以及为MenuItem添加点击事件

f.为托盘图标TrayIcon添加弹出菜单PopupMenu

g.为系统托盘SystemTray添加托盘图标

**2.1.4实现系统托盘代码**

public void showSystemTray() {

Image image = Toolkit.getDefaultToolkit().getImage("img/icon.png");

final TrayIcon trayIcon = new TrayIcon(image);// 创建托盘图标

trayIcon.setToolTip("屏幕监控系统\n客户端");// 设置提示文字

final SystemTray systemTray = SystemTray.getSystemTray();//托盘

final PopupMenu popupMenu = new PopupMenu(); // 创建弹出菜单

MenuItem item = new MenuItem("退出");

item.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

//菜单项点击处理逻辑

}

});

popupMenu.add(item);

trayIcon.setPopupMenu(popupMenu); // 为托盘图标加弹出菜单

try {

systemTray.add(trayIcon); // 为系统托盘加托盘图标

} catch (AWTException e) {

e.printStackTrace();

}

}

**2.2自定义协议**

**2.2.1网络协议**

网络协议为[计算机网络](http://baike.baidu.com/item/%E8%AE%A1%E7%AE%97%E6%9C%BA%E7%BD%91%E7%BB%9C)中进行数据交换而建立的规则、标准或约定的集合。协议就是一个交换数据的规则，我们也可以自定义一个交换数据的规则，并按照规则进行数据的传输，这样我们就自定义了一个协议。

**2.2.2为什么自定义协议？**

在客户端与服务器端进行Socket通信时，两者进行通信主要通过连接取得socket对象，在使用socket取得输入、输出流实现读取、写入数据功能。以客户端向服务器端发送一条消息为例：

a.两者通过Socket建立连接，客户端有一个Socket对象，客户端连接成功后，服务器端也会得到一个Socket对象。

b.客户端得到Socket的输出流，往输出流中写字符串消息，客户端Socket关闭

c.服务器端得到Socket的输入流，从输入流中读取出字符串消息，服务器端Socket关闭

这样就实现了从客户端将消息发送到服务器端，但是如果发送多条消息就会出问题了，你会说在客户端写个死循环让它一直运行，并且发送消息之后不关闭Socket连接不就可以了吗？

我最开始的想法也是这样的，在服务器端获取到连接的Socket后，循环接受客户端发送的消息，循环结束的条件是输入流中没有数据。

这个方法对于文本类消息是可行的，但是客户端发送的是图片字节数组，如果还是这个方法，那么就要考虑读字节数组时读到什么位置算是读到了一张图片，想来想去，自定义协议是可以解决这个问题的。

**2.2.3自定义协议**

|  |  |  |
| --- | --- | --- |
| type  1字节 | totalLen  4字节 | bytes[]  长度不定 |

type：消息类型

totalLen：该数据的长度=消息长度+5字节

bytes[]：实际的消息

**2.2.4自定义协议相关类说明**

我定义两个工具类：Protocol、Result

Protocol类的静态常量如下表:

|  |  |
| --- | --- |
| 常量 | 含义 |
| TYPE\_IMAGE | 发送的是图片 |
| TYPE\_LOAD | 客户端登录 |
| TYPE\_LOGOUT | 客户端退出 |

Protocol类封装了协议的两个使用方法，方法签名如下：

static void send(int,byte[],DataOutputStream)

该方法将字节数组写到指定输出流中，第一个参数是消息类型（静态常量），第二个参数是字节数组型的数据，第三个参数是输出流。

static Result getResult(DataInputStream)

该方法从指定输入流中读取数据，返回一个包含了这些数据的Result对象

Result类封装了一个消息的数据，包括type、totalLen、data[]

**2.2.5核心代码**

//客户端登录功能

Protocol.send(Protocol.TYPE\_LOAD, "client".getBytes(), dos);

//服务器端接收消息功能

Result result =Protocol.getResult(dis);

//以协议的规范向输出流中写数据

dos.writeByte(type);

dos.writeInt(totalLen);

dos.write(bytes);

dos.flush();

//以协议的规范从输入流读取数据

Protocol类的getResult方法核心:

byte type = dis.readByte();

int totalLen=dis.readInt();

byte[] bytes=new byte[totalLen-4-1];

dis.readFully(bytes);

**2.3客户端模块**

**2.3.1获取屏幕截图**

BufferedImage bfImage = robot.createScreenCapture(new Rectangle(0, 0, width, height));

其中width，height是屏幕的宽高，截取屏幕用的是JAVA提供的Robot类，Robot类可以模拟用户的行为，如控制鼠标、打字等一系列操作。所以可以使用Robot实现屏幕截图以及事件回放的功能。

**2.3.2将图片以协议的规范发到服务器**

我们获取到了屏幕截图的BufferedImage，那么应该怎么将它传到服务器呢？

先看我的初期想法：

通过ImageIO将BufferedImage写到一个File里，然后从这个文件的输入流中读出数据至字节数组中，然后使用协议工具类把这条消息发出去就可以了。

这里会产生问题：应该把BufferedImage写到哪个File，我们有三个选择：

a.每次获得一个BufferedImage就新建一个File

b.程序启动后初始化创建一个File

c.写到临时文件中，并设置程序退出删除临时文件

针对情况a，可以顺利的发送图片。但是由于获得屏幕截图后需要创建文件，向文件里写数据，然后再读出来，给我一种感觉就是好像做了重复的工作，效率不高；并且每得到BufferedImage后就创建一个File，因为系统每隔50ms就会截取一张图片，每张图片都大于1M。我测试的时候仅仅运行了十多分钟我的一个磁盘就被写满了，这样势必会产生两个问题：效率问题、空间占用问题。

针对情况b，系统在获取BufferedImage后往一个固定的File里写数据，从这个File里读出数据，这个方法只生成一个文件，然后不断的在这个文件里进行读写操作，这样处理的话内存占用确实会少很多，但效率不会提升，并且在实际中使用的时候发现系统会报异常：文件损坏错误。异常的原因其实也很简单，因为屏幕截取的速率是很快的，截图之后就会往File里写数据，如果此时上张图片还没有读完，那么就会导致文件损坏。

针对情况c，经我试验，任何用处都没有，临时文件在程序结束后也没有删除，虽然这个方法失败了，但也是一种尝试。

经查阅资料发现，通过直接将BufferedImage转化为字节数组从而达到我们对空间、速率上的要求，以下是具体过程：

经查阅资料发现ImageIO类可以直接将BufferedImage对象写到字节数组输出流中，然后我们在将字节数组输出流的数据转化为字节数组就可以了。

ByteArrayOutputStream baos = new ByteArrayOutputStream();

ImageIO.write(buffedImage, "png", baos);

Protocol.send(Protocol.TYPE\_IMAGE, baos.toByteArray(), dos);

**2.3.3系统退出机制**

我为客户端的托盘提供了[右键菜单>退出]选项，客户端有一个布尔类型的变量标志是否生存，系统会检测这个值然后执行循环：截图，发送，休眠。当用户选

中退出时，系统先想服务器发送一个退出请求，然后更改变量标志为false，并关闭掉socket连接以及输入、输出流，正常退出。为了让系统更加的健壮，我们要对系统异常捕获以及处理异常。

**2.3.4客户端代码**

final Client client = new Client();

client.conn("192.168.1.101", 33000);//连接服务器

client.load();//登录

client.showSystemTray();//显示托盘

while (client.isLive) {

client.sendImage(client.getScreenShot());

try {

Thread.sleep(50);

} catch (InterruptedException e) {

System.exit(0);

}

}

**2.4服务器端模块**

**2.4.1服务器逻辑**

服务器端程序运行后，创建ServerSocket，然后不断的接受连接上的Socket，每当客户端连接上，就将Socket交到一个线程手里，由该线程负责该客户端的所有交互行为，服务器应该有一个Map保存客户端IP与Socket的对应关系

服务器端接受到客户端发来的登录请求，将它的IP作为Key，Socket作为Value保存到Map中，将连接上的用户显示在控制界面View的用户树上

服务器端接收到客户端发送的发送图片请求，将字节数组转化为BufferedImage，将这张图片重绘到控制界面的屏幕监控视图上

ByteArrayInputStream bai=new ByteArrayInputStream(data);

BufferedImage buff=ImageIO.read(bai);

//为屏幕监控视图设置BufferedImage

Server.view.centerPanel.setBufferedImage(buff); Server.view.centerPanel.repaint();

服务器接受到客户端发送的退出请求，从控制界面View的用户树上删除该客户端IP，从Map中删除该客户端IP，关闭当前客户端的Socket，释放掉资源。

**2.4.2对多客户端的处理**

界面分为两部分：左侧的用户树，右侧的监控区域

用户树显示当前连接到的所有客户端IP地址，右侧的监控区域显示当前监控的客户端的屏幕图象。

在一个客户端的情况下，服务器端接收到客户端的图像就显示在监控区域上。当有多个客户端连接的时候，如果服务器端不对消息进行过滤的话，那么在监控区域上会轮流显示各个客户端的屏幕监控，所以需要在服务器端标记当前监控的IP地址，当有客户端发过来图像的话，将客户端IP与标记的IP进行比对，如果相同才把图像显示出来，否则就将消息丢弃掉。这里要涉及到几个知识点：用户树的刷新、用户树的选中事件处理、用户树的节点样式

我们用以下的方式创建一棵树

model=new DefaultTreeModel(root);

JTree tree=new JTree(model);

//刷新用户树

DefaultMutableTreeNode node1=new DefaultMutableTreeNode(nodeString);

root.add(node1);

model.reload();

//用户树节点选中事件的处理

tree.addTreeSelectionListener(new TreeSelectionListener() {

@Override

public void valueChanged(TreeSelectionEvent e) {

JTree tree=(JTree) e.getSource();

DefaultMutableTreeNode selectionNode = (DefaultMutableTreeNode) tree.getLastSelectedPathComponent();

String nodeName=selectionNode.toString();

Server.curKey=nodeName;

}

});

用户树的节点样式是通过DefaultTreeCellRenderer来实现的

DefaultTreeCellRenderer cr=new DefaultTreeCellRenderer();

cr.setBackgroundNonSelectionColor(Color.darkGray);

cr.setTextNonSelectionColor(Color.white);

tree.setCellRenderer(cr);

**2.4.3服务器端对客户端消息请求的处理**

客户端与服务器的一系列交互在客户端连接上之后就被交予一个线程来全权负责了，服务器端通过协议工具类的解析数据的方法获取到消息类型、消息长度以及消息内容，将消息类型以及消息内容交予一个函数来处理

Result result =Protocol.getResult(dis);

handleType(result.getType(),result.getData());

private void handleType(int type,byte[] data) {

switch (type) {

case 1://处理图片请求

break;

case 2://处理登录请求

break;

case 3://处理退出请求

break;

}

}

**三、Web服务器环境配置**

**3.1简述**

本系统分为两个端：server端、client端，使用时，应该将Server端部署在服务器上，服务器可以使用远程服务器，也可以使用本机PC作为服务器进行测试，以下以本机PC搭建局域网服务器为例

**3.2工具**

win7系统的笔记本

**3.3搭建过程**

3.1开始>控制面板>程序与功能>打开或关闭window功能>展开Internet信息服务，选择Web管理工具、万维网服务以及其子选项，点击确定。

3.2开始>控制面板>管理工具>Internet信息服务管理器>点击默认的网站，配置网站路径、端口之后即可访问了。

**3.4使用方法**

**a.查询服务器IP**

在服务器端打开命令提示符，输入ipconfig，找到其IP地址，即IPV4地址

**b.修改代码**

本代码客户端默认连接ip为127.0.0.1的服务器，即本机，所以我们修改这个IP。在Client类中有一个main方法，它是程序的入口，当客户端连接时是调用Client对象的conn方法，只需要将这个方法中的参数修改为查询到的服务器IP地址即可。

**c.程序打包**

项目下有三个包：Server、Client、Util，需要注意的是Util包下存放的是客户端和服务器都需要用到的工具类，所以不管是服务器还是客户端在打包时都需要包含Util包，另外客户端用到了系统托盘图标，所以打包时需要用到图片，需要包含img文件夹。

**d.程序运行**

在搭建好web环境的PC机上运行服务器端，在其他电脑上运行客户端。

要注意先运行服务器端程序，再运行客户端程序，客户端与服务器端建立连接成功后会显示系统托盘，服务器端运行后会有一个窗口，当有用户连接上时在服务端右侧界面会显示客户端PC机屏幕，当有多个客户端连接到服务端时，左侧用户树会显示所有客户端IP，点击树节点就会切换监控用户。

**四、项目难点**

**4.1客户端循环发送图片的问题**

图片是字节数组，循环发送的话会导致服务器端找不到图片截止的标志，无法获

取图片。就是因为这个需求我才会想到自定义协议，通过构造一个协议发到服务器，服务器就可以知道读取到什么位置是一张图片。为了性能与空间上的要求，我们需要找一种可以直接把图片转化为字节数组的方法

ByteArrayOutputStream baos = new ByteArrayOutputStream();

ImageIO.write(buffedImage, "png", baos);

转化之后baos.toByteArray()就可以将其转化为字节数组。

以下方法可以将字节数组转化为图片

ByteArrayInputStream bai=new ByteArrayInputStream(data);

BufferedImage buff=ImageIO.read(bai);

**4.2服务器端线程里操作JPanel重绘**

在线程里操作，那么要把这个Panel设为静态的，程序里有好几个地方都需要设置成静态的，如保存客户IP与Socket的Map、View的实例对象

这是自定义的一个继承自JPanel的类DrawPanel，根据我们的需求，我们要在这个JPanel上不断绘制从客户端传回来的图像，所以该类需要提供一个方法来设置绘制的图像。方法签名如下：

void setBufferedImage(BufferedImage bufferedImage)

将bufferedImage设置为当前要绘制的BufferedImage

void paint(Graphics g)

重写该方法，内部是利用g绘制当前的BufferedImage到视图上

当操作JPanel重绘时只需要两步就可以了：

a.调用该类实例对象的setBufferedImage(BufferedImage bufferedImage)方法设置要绘制的图像

b.调用实例对象的repaint()方法重绘JPanel

**4.3对异常的处理**

客户端异常退出时会造成空指针异常、CG异常、连接关闭异常、IO异常、文件读写异常等，这些异常都是在项目中出现的。在出现异常后，不要惊慌，首先根据系统提示一步一步排查，当排查到某行代码上时，仔细分析出现错误的原因，比如空指针异常，那么要考虑变量值为什么是空，要考虑变量是否是静态的，要发散思维看待问题。为了使系统更加健壮，我们要捕获异常以及对异常进行处理。

**五、展望**

任何一个优秀的作品都需要不断的改进，那么我来说一下因为时间原因来不及完成的几个功能以及大概实现思路，也算是对这个系统的展望吧。

**5.1事件回放功能**

事件回放功能是当客户端与服务器端建立连接后，客户端向服务器发送屏幕截图的同时，将客户端的鼠标、键盘等事件一起发送出去。当服务器接收到之后，利用Robot对象对整个事件进行事件回放处理，那么服务器端的监控区域不仅仅只显示客户端的屏幕，而且还有鼠标、键盘等事件。

**5.2远程控制功能**

目前只实现了远程监控功能，在服务器端只能看到客户端发生了什么，但并不能对客户端进行任何操作。由于在上边定义了一个协议，所以实现这个功能也很简单。只需要在拓展几种消息类型即可，服务器端确定要控制的客户端IP，并将服务器端的事件封装之后用协议工具类将消息发送到客户端，客户端根据消息类型进行判断，如果是控制类型的指令，那么就将事件对象在客户端进行回放。这样就实现了一个简单的控制功能。

**5.3聊天模块**

其实论文写到这里聊天模块的实现已经没有难度了，其实在客户端登录的时候就用到了这个功能。客户端登录是向服务器发送了一个字符串，只不过服务器端接收到消息后并没有回应。如果客户端让用户输入字符串，系统将消息发送到服务器端，服务器端对消息进行提醒、显示，并给服务器端提供一个输入区域，将回应的消息发送到指定IP的客户端。那么一个简单的1对1的即时聊天模块就算实现了。

**5.4 UI优化**

现在的系统UI太粗糙了，因为Swing本身做UI不是很有优势，我对这方面了解的不是很深。对UI优化可以使用已有的LookAndFeel或者使用其他的开源库。后续可以尝试为服务器端界面加上工具栏以及图标、点击效果。

**5.5 Log日志记录**

Log日志记录功能是当系统运行时将Log日志输出到log文件中，Log日志保存的是一个软件的运行状态。

**六、感想**

通过做这个项目发现对零碎的小知识点掌握的不牢固，例如图片与字节数组的转换、系统托盘实现等一些功能都是现学现用的。其中在线程中操作View中的对象是其中最让我头疼的地方，因为那个对象没有用static导致一直获取不到数据，一下子就困惑了一天。目前系统已经完成了，回头想想，整个系统的开发并不是很难，流程很清晰，整个的难点就是那个循环发送字节数组的问题，使用自定义协议这个方法后问题也就解决了。项目开发中可能会遇到一些奇怪的事情，但是不要着急，一步一步缩小查找范围还是可以找到错误的原因的。项目主要用到了Socket通信原理，socket通信有很多的应用，我们可以用socket做即时通信软件、联网游戏、文件上传下载工具等等。突然发现学的东西还是挺有用的，在这里要谢谢任课老师在课堂上对知识点进行认真的讲解，我从中学到了很多。

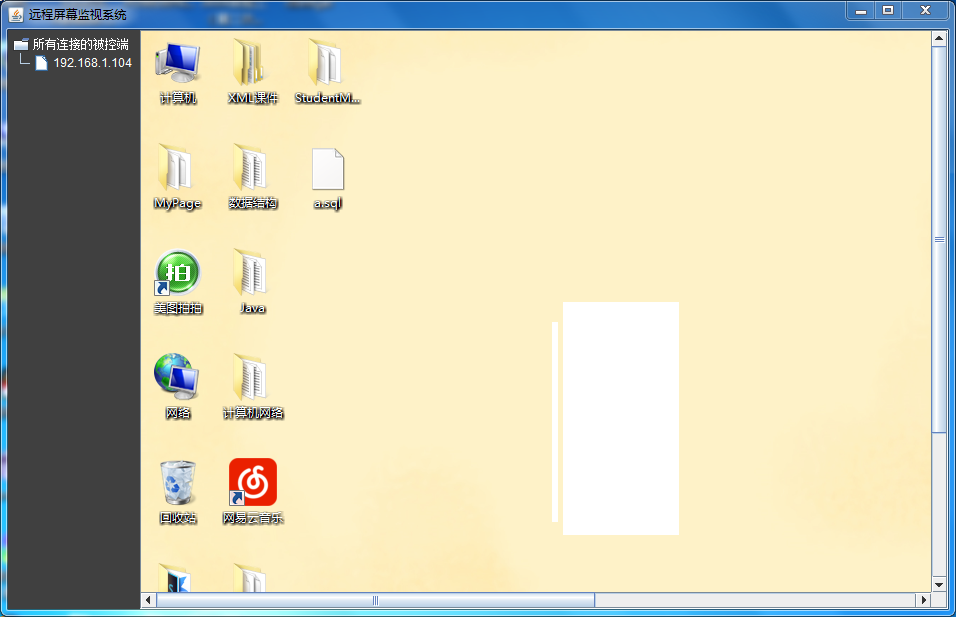
总之呢，写软件是个很磨人的过程，我们可能会一直犯错，有时候感觉自己看出错在哪了，改正过之后竟然还是有错，并且有时候的错误会使我们感觉很无奈。如果能够坚持下来写几个小项目，那么对自己的能力绝对是一个大的提升。

**附件1：软件运行效果图**

图1：服务器端运行，此时无客户端连接

****

图2：服务器端运行，此时有客户端连接，左侧为用户树，树节点为客户端IP，右侧为监控区域，显示的是被控端的屏幕画面

****