第 35 卷 第 5 期 计 算 机 工 程 2009 年 3 月
Vol.35 No.5 Computer Engineering March 2009
·网络与通信· 文章编号：1000—3428(2009)05—0074—02 文献标识码：A 中图分类号：TP393.02

基于 Java 的网络实时远程监控系统设计

阎秀英，周亚建，胡正名
(北京邮电大学信息处理与智能技术重点实验室数字内容研究中心，北京 100876)

摘 要：随着网络技术的发展，远程屏幕监控越来越多地应用在各种场合。虽然现在有很多远程屏幕监控软件，但使用 Java 开发的在 Linux 下的屏幕监控系统还很少。该文提出了在 Linux 下基于 Java 设计网络实时监控系统的方法，分析了其中的关键技术并予以实现。在网络教学中，该系统能使教师随时监控到网络中任何一台学生机的操作状态，经测试具有良好的实时性。

关键词：屏幕监控；SWT_AWT 桥；Socket 技术

Design of Real-time Remote Network Monitor System Based on Java

YAN Xiu-ying, ZHOU Ya-jian, HU Zheng-ming
(Digital Content Research Center, Key Laboratory of Information Processing and Intelligent Technology,
Beijing University of Posts and Telecommunications, Beijing 100876)

【Abstract】 With the development of network technology, remote screen monitor is used in more and more sceneries. Although there are many remote monitor systems, rarely exists ones based on Java on the Linux platform. In this paper, a method to design a real-time remote system is proposed and implemented. The frame of the system is presented, and also the key techniques are expatiated. In the network teaching system, the tutor can monitor any student's screen at any time using this system. According to the test, the system has the expectable real-time advantage.

【Key words】 screen monitor; SWT_AWT bridge; Socket technique

随着计算机网络技术的发展与应用，屏幕监控越来越受人们的关注。如为确保网络信息的安全，越来越多的网管人员需要实时监控联网计算机的运行状态和各种操作；在远程教育中，学生要实时地看到教师的屏幕变化、教师需要随时监控学生的屏幕。

1 系统架构介绍及关键技术分析

1.1 系统架构

本文提出了一种用 Java 对局域网内联网计算机进行实时监控的系统，系统采用客户机/服务器模式。当要对装有服务器端程序的主机进行屏幕监控时，就在客户端使用服务器端主机的 IP 发送连接请求。服务器端检测到连接请求后建立连接，同时开始对本机进行抓屏，转变为数据流向客户端发送。客户端连续接收，每隔一定时间(文中采用 1 000 ms)刷新界面，实现实时监控。设计总框架与基本过程如图 1 所示。

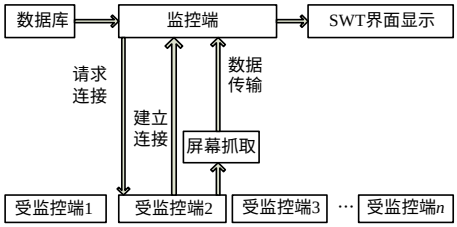

图 1 系统结构

在图 1 中，当监控端监控完受监控端 1 时，中断与受监控端 1 的连接。假设接下来要监控受监控端 2。首先从数据库获取网络中受监控端 2 的 IP 地址(也可利用一些工具直接获取，而不使用数据库)，获取目标 IP 地址之后，发送连接请求，受监控端接收连接请求，建立连接。此时，受监控端 2 开始自动抓取本机屏幕信息，打包成图像数据流，向监控端发送。监控端接收数据流，利用 SWT 进行连续显示。

1.2 系统关键技术分析

1.2.1 连接的建立

网络中的 2 台机器可以通过 Socket 通信。Socket 是 2 个进程间通信链的网络端点，一般由一个地址加一个端口来标识。Socket 通信机制是一种底层的通信机制，通过 Socket 的数据是原始字节流信息，通信双方必须根据约定的协议对数据进行处理和解释。

Socket 通信机制提供了 2 种通信方式：面向连接的方式(TCP)和无连接方式(UDP 数据报)。在面向连接的方式中，通信双方在通信开始之前必须进行一次连接过程，建立提供可靠的、全双工字节流服务的通信链路。在无连接方式中，不需要连接过程，一次网络 I/O 以一个数据报形式进行。

Java 同时支持这 2 种通信方式，为了保证信息传输的可靠性，本文选用面向连接的通信方式。在 Java 开发平台中，这种方式表现为流式 I/O 模式。每个 Socket 有 2 个流：一个输入流和一个输出流。只要向 Socket 的输出流写，一个进程就可以通过网络连接向其他进程发送数据；同样，通过读 Socket 的输入流，就可以读取传输来的数据。

1.2.2 屏幕数据的抓取与传送

屏幕截取接近操作系统的底层操作，在 Windows 操作系统下，该操作几乎全部用 VC, VB 实现。事实上，使用 Java JDK1.4 的 Robot 类来实现屏幕截取更加简单。本文系统在 Linux 下采用 Robot[1-2]类实现。

作者简介：阎秀英(1984－)，女，硕士，主研方向：下一代网络安全；周亚建，讲师；胡正名，教授、博士生导师

收稿日期：2008-06-21 **E-mail：**xyyan1984@yahoo.com.cn

Robot 类用于产生与本地操作系统有关的底层输入，测试应用程序运行或自动控制应用程序运行。该类的对象可以完成对屏幕(屏幕上正在运行的应用程序界面)像素的拷贝，完成屏幕图像截取操作。

Robot 类提供的方法 createScreenCapture()，可以直接将全屏幕或某个屏幕区域的像素拷贝到一个 BufferedImage 对象中，将该对象写入到一个图像文件之中，就完成了屏幕到图像的拷贝过程。Java 应用程序可以调用此对象，完成对特定应用程序的屏幕截取，如果将此功能配合网络，就可以实现对远端计算机屏幕的监视。

1.2.3 基于 SWT_AWT[3]的监控图像实时显示

AWT(Abstract Window Toolkit)[4]是 Java1.0 版本提供的第 1 个支持图形化用户界面 GUI 设计的工具集。AWT 类库中的各种操作被定义成在一个抽象窗口中进行。抽象窗口使得界面的设计能够独立于界面的实现，使利用 AWT 开发的 GUI 能够适用于所有的平台系统，且具有良好的布局管理，使组件有合适的位置。但由于 Sun 仅仅实现了 Java 支持的平台中共有的那些部件，使得 AWT 无法满足开发者的需求，不适宜构建丰富的桌面图形界面。

SWT(Standard Widget Tookit)[4-6]是 IBM 开发的图形用户界面工具。它在一定程度上是将 AWT 与 SWING 混合的一种解决方法。它向开发人员提供了一组窗口小部件。SWT 与 AWT 的主要区别在于 SWT 使用操作系统的底层 GUI 窗口小部件，直接调用操作系统的图形库，从而使得 Java 应用程序界面与操作系统的习惯完全一致。更为重要的是，对本地方法的直接调用大幅度提高了 Java 应用程序的运行速度。

虽然 SWT 优势明显，但实际工程中并不能舍弃 AWT，有时基于 SWT 的界面必须借助 AWT 才能满足需求。而要想将 SWT 与 AWT 结合起来使用，必须要用 SWT_AWT 桥才可以完成任务。本文的监控显示就是在基于 SWT 的容器上利用 SWT_AWT 桥建立起 AWT 底层框架，而后将接收来的监控数据显示在 AWT 的组件中。

2 系统实现

2.1 建立连接的实现

Java 的有连接通信一般经历下列 4 个基本步骤：(1)创建 Socket，建立连接；(2)打开连接到 Socket 的输入/输出流；(3)按照协议对 Socket 进行读/写操作；(4)关闭 Socket。这里只简述建立连接的过程。

Java.net 包中的 2 个类：Socket 和 ServerSocket，它们分别表示连接的 Client 端和 Server 端。系统中在服务器端用 ServerSocket(12000)，将本地 Socket 绑定到端口 12000，等待连接。在客户机端用 Socket(serverIP,12000)创建 Socket 连接到指定主机的指定端口。当 Server 端监听到连接请求时，使用 accept()成功建立连接。该过程如图 2 所示。

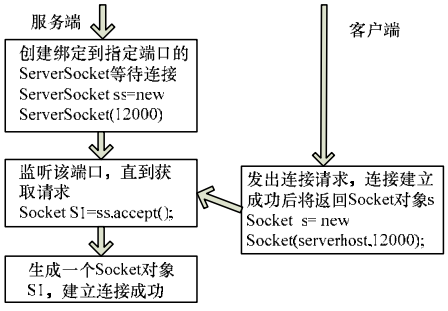

图 2 利用 Java 实现客户端与服务器端连接的过程

2.2 屏幕数据抓取与传送的实现

Robot 类提供的方法.createScreenCapture()，可以将全屏幕或屏幕某个区域的像素拷贝到一个 BufferedImage 对象中，将该对象写入到一个图像文件之中，就完成了屏幕图像抓取。其实现代码如下：

```
Robot robot=new Robot();
toolkit=Toolkit.getDefaultToolkit(); BufferedImage newImage=
robot.createScreenCapture(new Rectangle(toolkit.getScreenSize()));
```

在 TCP/IP 协议组中，TCP 是一种面向连接的协议，为用户提供可靠的数据传输服务。但是它提供的传输效率不够好。因此，在远程监控系统中将数据压缩后再进行传输。

2.3 基于 SWT 监控图像显示的实现

在客户端获得与服务器的连接后用 javax.imageio.ImageIO 类的 read()方法从端口读取监控数据，形成图片，主要代码如下：

```
clientsocket=new java.net.Socket(stuip, 12000);
cimage=javax.imageio.ImageIO.read
(clientsocket.getInputStream());
label_2.setIcon(new ImageIcon(cimage));
```

其中，ClientSocket 是本地 Socket；stuip 是被监控机器的 IP 地址；label_2 是利用 AWT 中 Label 组件实例化的可更新显示图片的对象。监控端收到图像数据流后在 SWT_AWT 桥搭建的界面中显示。其具体组件组织结构如图 3 所示。

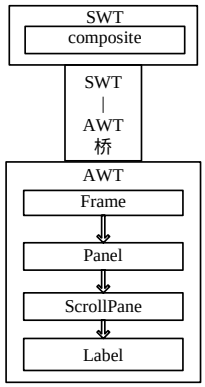

图 3 组件结构

最终监控的图像画在 AWT 的 Label 中。它的上层容器用了 ScrollPane，用以实现滚动，这样可以在应用程序界面特定区域内看到被监控机器完整的屏幕图片。因为是实时监控，在系统中用了定时器，每隔 1 000 ms 重绘一次界面。实验结果如图 4、图 5 所示，在监控端(图 5)应用程序界面的右半区域实时显示被监控端(图 4)的屏幕变化。

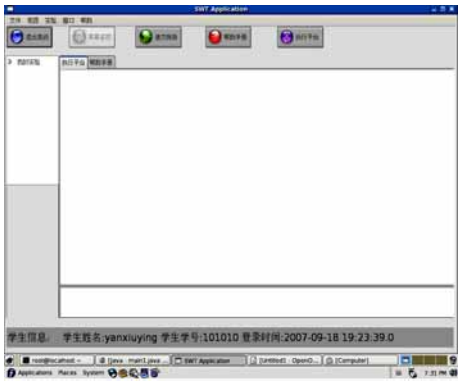

图 4 服务器端正在运行的应用程序界面(被监控端)

（下转第 78 页）