Home of The JavaSpecialists' Newsletter

Menu

091 Controlling Machines Remotely with Java

Posted: 2004-07-08 | Category: GUI | Java Version: 1.5+ | Dr. Heinz M. Kabutz

Abstract:

Welcome to the 91st edition of **The Java(tm) Specialists' Newsletter**. I am currently in Deutschland (Germany), where I have spent the last two weeks training a company on how to program in Java. I was fortunate on this trip that I could stay with an old friend, instead of a hotel. I will choose a mattress on the floor at a friend's house ANY day over a five star hotel. Thanks, Markus :-)

Great minds think alike - or is it - fools seldom differ? Whilst I was putting together this newsletter, Michael Abernethy and Kulvir Singh Bhogal from IBM published a newsletter on Java Pro on the same topic. Have a look at the JavaPro website for their article. This is the first time that such a thing has happened, and I decided to publish this newsletter nonetheless, since both our newsletters were totally original. Pick the best from both approaches and you will have a great system.

Controlling Machines Remotely with Java

The biggest obstacle in training is losing the enthusiasm that you had when you first started teaching. The joy of watching people get excited about Java, can easily become so common, that it appears dull. I have counted that this is my 58th course in 5 years. However, the ideas and questions of the students, tend to stimulate my imagination greatly. This then flows into my work, into my newsletter, and back into my courses. On a course a few years ago, someone asked me if it was possible to write a program that transferred a screen dump over the network. The result is this little program, which I use to watch the way that my students solve their problems.

You can gladly use this program to remotely control computers, but please don't abuse the technology, ok? Don't use it as spyware, or to annoy your colleagues. Use it for good, not harm.

This program, in under 300 lines, allows me to start a server to which students can connect. It requests screen shots from the students, and can simulate mouse moves and button clicks. The possibilities are almost endless - and the architecture would allow you to also send keystrokes, read the disk, and do all sorts of other weird and wonderful things. Whilst I have not tried it out, it should also work cross platform without any modifications.

The program contains only code that would work with JDK 1.3, so that it is possible to control computers that are using older versions of Java. I have put these classes into the package com.maxoft.teacher. Please put them into your own packages if you decide to use them.

The first piece of code is an interface RobotAction that represents a command that is executed on the client machine (i.e. the student's machine).

```java
package com.maxoft.teacher;

import java.awt.*;
import java.io.*;

public interface RobotAction extends Serializable {
  Object execute(Robot robot) throws IOException;
}
```

On the client machine, you would run the Student class, which would read in the RobotAction objects, one by one, and run them by passing a handle to the Robot on the client machine. If the action produces a result, we send that back to the teacher. In this simple program, the only reaction could be a screendump, so the teacher program only has to cater for a `byte[]`. However, it could easily be expanded to allow actions to be sent back to the teacher, such as the HelpAction!

```
package com.maxoft.teacher;

import java.awt.*;
import java.io.*;
import java.net.Socket;

public class Student {
  private final ObjectOutputStream out;
  private final ObjectInputStream in;
  private final Robot robot;

  public Student(String serverMachine, String studentName)
      throws IOException, AWTException {
    Socket socket = new Socket(serverMachine, Teacher.PORT);
    robot = new Robot();
    out = new ObjectOutputStream(socket.getOutputStream());
    in = new ObjectInputStream(
      new BufferedInputStream(socket.getInputStream()));
    out.writeObject(studentName);
    out.flush();
  }

  public void run() throws ClassNotFoundException {
    try {
      while (true) {
        RobotAction action = (RobotAction) in.readObject();
        Object result = action.execute(robot);
        if (result != null) {
          out.writeObject(result);
          out.flush();
          out.reset();
        }
      }
    } catch (IOException ex) {
      System.out.println("Connection closed");
    }
  }

  public static void main(String[] args) throws Exception {
    Student student = new Student(args[0], args[1]);
    student.run();
  }
}
```

The first RobotAction is MoveMouse. We specify where the mouse must move to, then send the coordinates over the network to the student. The student program then calls `execute`, causing his mouse to runs across his monitor to the specified position.

```
package com.maxoft.teacher;

import java.awt.*;
import java.awt.event.MouseEvent;

public class MoveMouse implements RobotAction {
  private final int x;
  private final int y;

  public MoveMouse(Point to) {
    x = (int)to.getX();
    y = (int)to.getY();
  }
  public MoveMouse(MouseEvent event) {
    this(event.getPoint());
  }
  public Object execute(Robot robot) {
    robot.mouseMove(x, y);
    return null;
  }
  public String toString() {
    return "MoveMouse: x=" + x + ", y=" + y;
  }
}
```

The next RobotAction is ClickMouse. We specify which mouse button to click, and how many times. The student's mouse now clicks without his intervention. With mouse move and mouse click, I can control enough of his computer to help him when he gets stuck.

```
package com.maxoft.teacher;

import java.awt.*;
import java.awt.event.MouseEvent;

public class ClickMouse implements RobotAction {
  private final int mouseButton;
  private final int clicks;

  public ClickMouse(int mouseButton, int clicks) {
    this.mouseButton = mouseButton;
    this.clicks = clicks;
  }
  public ClickMouse(MouseEvent event) {
    this(event.getModifiers(), event.getClickCount());
  }
  public Object execute(Robot robot) {
    for (int i = 0; i < clicks; i++) {
      robot.mousePress(mouseButton);
      robot.mouseRelease(mouseButton);
    }
    return null;
  }
  public String toString() {
    return "ClickMouse: " + mouseButton + ", " + clicks;
  }
}
```

The last action is the ScreenShot, where the Robot takes a snapshot of what was on the screen and sends it back as a JPEG. The JPEG Image Encoder writes the output into an OutputStream, so we pass it a ByteArrayOutputStream, which we then convert to a byte[] and return to the caller of execute. I did some tests to compare the space saving, and it is huge! If I were to simply send an ImageIcon (my first approach), then my screen (1680 x 1050) would take 7'056'254 bytes. With the compressed JPG, it now only takes 272'536 bytes. Still a bit slow on a DSL connection, but alright on a LAN. Further gains could be achieved by scaling the image first.

```
package com.maxoft.teacher;

import com.sun.image.codec.jpeg.*;

import java.awt.*;
import java.io.*;

public class ScreenShot implements RobotAction {
  public Object execute(Robot robot) throws IOException {
    Toolkit defaultToolkit = Toolkit.getDefaultToolkit();
    Rectangle shotArea = new Rectangle(
        defaultToolkit.getScreenSize());
    ByteArrayOutputStream bout = new ByteArrayOutputStream();
    JPEGImageEncoder encoder = JPEGCodec.createJPEGEncoder(bout);
    encoder.encode(robot.createScreenCapture(shotArea));
    return bout.toByteArray();
  }
  public String toString() {
    return "ScreenShot";
  }
}
```

That's it for the actions. You could define just about any action for the client machine, and the most obvious one would be to open up a pop-up window with some suggestions of what to do next.

The next class is a simple blocking queue for the RobotAction objects in the Teacher program. JDK 1.5 has a built-in blocking queue, but I want to make sure that this program also works with older versions of Java. So, here goes:

```
package com.maxoft.teacher;

import java.util.LinkedList;

public class RobotActionQueue {
  private final LinkedList jobs = new LinkedList();

  public RobotAction next() throws InterruptedException {
    synchronized (jobs) {
      while (jobs.isEmpty()) {
        jobs.wait();
      }
      return (RobotAction) jobs.removeFirst();
    }
  }

  public void add(RobotAction action) {
    synchronized (jobs) {
      jobs.add(action);
      System.out.println("jobs = " + jobs);
      jobs.notifyAll();
    }
  }
}
```

The main class is the Teacher. When a student connects, he reads the name of the student over the network. Then he creates one thread for reading screen shots from the student, another thread for writing the RobotActions to the student, and yet another timer to periodically request screen shots from the student. Some code thrown in to capture mouse clicks and send them back to the client as RobotActions, and voila! a perfectly functional spyware^H^H^H^H^H^Hremote administration program. The keystrokes are left out on purpose, and I only move the mouse once I have clicked it. This prevents the mouse from zooting over the screen whilst my students are trying to work.

```
package com.maxoft.teacher;

import com.sun.image.codec.jpeg.*;
import javax.swing.*;
import java.awt.event.*;
import java.awt.image.BufferedImage;
import java.awt.*;
import java.io.*;
import java.net.*;
import java.util.Timer;
import java.util.*;

public class Teacher extends JFrame {
  public static final int PORT = 5555;
  private static final long SCREEN_SHOT_PERIOD = 2000;
  private static final int WINDOW_HEIGHT = 400;
  private static final int WINDOW_WIDTH = 500;

  private final ObjectInputStream in;
  private final ObjectOutputStream out;
  private final String studentName;
  private final JLabel iconLabel = new JLabel();
  private final RobotActionQueue jobs = new RobotActionQueue();
  private final Thread writer;
  private final Timer timer;
  private volatile boolean running = true;

  public Teacher(Socket socket)
      throws IOException, ClassNotFoundException {
    out = new ObjectOutputStream(socket.getOutputStream());
    in = new ObjectInputStream(new BufferedInputStream(socket.getInputStream()));
    studentName = (String) in.readObject();
    setupUI();

    createReaderThread();
    timer = createScreenShotThread();
    writer = createWriterThread();
    addWindowListener(new WindowAdapter() {
      public void windowClosing(WindowEvent e) {
        timer.cancel();
      }
    });
    addWindowListener(new WindowAdapter() {
      public void windowClosing(WindowEvent e) {
        try {
          out.close();
        } catch (IOException ex) { }
        try {
          in.close();
        } catch (IOException ex) { }
      }
    });
    System.out.println("finished connecting to " + socket);
  }

  private Timer createScreenShotThread() {
    Timer timer = new Timer();
    timer.schedule(new TimerTask() {
      public void run() {
        jobs.add(new ScreenShot());
      }
    }, 1, SCREEN_SHOT_PERIOD);
    return timer;
  }

  private void setupUI() {
    setTitle("Screen from " + studentName);
    getContentPane().add(new JScrollPane(iconLabel));
    iconLabel.addMouseListener(new MouseAdapter() {
      public void mouseClicked(MouseEvent e) {
        if (running) {
          jobs.add(new MoveMouse(e));
          jobs.add(new ClickMouse(e));
          jobs.add(new ScreenShot());
        } else {
          Toolkit.getDefaultToolkit().beep();
        }
      }
    });
    setSize(WINDOW_WIDTH, WINDOW_HEIGHT);
    setVisible(true);
  }

  private Thread createWriterThread() {
    Thread writer = new Thread("Writer") {
      public void run() {
        try {
          while (true) {
            RobotAction action = jobs.next();
            out.writeObject(action);
            out.flush();
          }
        } catch (Exception e) {
          System.out.println("Connection to " + studentName +
              " closed (" + e + ')');
          setTitle(getTitle() + " - disconnected");
        }
      }
    };
    writer.start();
    return writer;
  }

  private void showIcon(byte[] byteImage) throws IOException {
    ByteArrayInputStream bin = new ByteArrayInputStream(byteImage);
    JPEGImageDecoder decoder = JPEGCodec.createJPEGDecoder(bin);
    final BufferedImage img = decoder.decodeAsBufferedImage();
    SwingUtilities.invokeLater(new Runnable() {
      public void run() {
        iconLabel.setIcon(new ImageIcon(img));
      }
    });
  }

  private void createReaderThread() {
    Thread readThread = new Thread() {
      public void run() {
        while (true) {
          try {
            byte[] img = (byte[]) in.readObject();
            System.out.println("Received screenshot of " +
                img.length + " bytes from " + studentName);
            showIcon(img);
          } catch (Exception ex) {
            System.out.println("Exception occurred: " + ex);
            writer.interrupt();
            timer.cancel();
            running = false;
            return;
          }
        }
      }
    };
    readThread.start();
```