Escuela Politécnica superior
Alcala de henares

Memoria Practica 3
Programación Avanzada

Pedro Barquin Ayuso

05-04-2014

Objetivo:

Con estos ejercicios se basa en aprender a usar las posibles alternativas de las que disponemos para la conexión cliente/servidor pudiendo así realizar una comunicación entre 2 maquinas o entre el mismo ordenador, en esta practica usaremos Sockets y RMI.

Ejercicio Sockets:

Servidor:

- En nuestra practica el servidor es iniciado en la clase colores donde se instancia y se crea, y en la clase Servidor es donde se encuentra el grueso del código.

```
public static void main(String args[]) {
    java.awt.EventQueue.invokeLater(new Runnable() {
        public void run() {
            new Colores().setVisible(true);
            Servidor Servidor=new Servidor(paso);
        }

    });

}
```

- Al servidor se le envía el paso desde el apartado anterior para poder parar la ejecución de los colores cuando sea necesario, para ello primero creamos el servidor con el puerto que queremos, después dentro de un bucle infinito comprobamos , la conexión una vez este realizada creamos los canales de transmisión y en este caso recibimos el mensaje con el cual dependiendo de lo que recibamos haremos la acción de parar o reanudar los colores, para después cerrar la conexión.

```
public class Servidor extends Thread {
    Paso paso=new Paso();

    public Servidor(Paso paso){
        this.paso=paso;
        start();
    }

    public void run(){
            ServerSocket servidor;
            Socket conexion; //socket
            DataInputStream  entrada;
            DataOutputStream salida;
            String msg;
            int num = 0;
            try {
            servidor = new ServerSocket(5000); // Creamos un ServerSocket EN EL PUERTO 5000
            System.out.println("Servidor Arrancado....");
            while (true) {
            conexion = servidor.accept();     // Esperamos una conexión y creamos el socket
            num++;
            System.out.println("Conexión n."+num+" desde: "+conexion.getInetAddress().getHostName());
            entrada = new DataInputStream(conexion.getInputStream());  // Abrimos los canales de E/S
            salida  = new DataOutputStream(conexion.getOutputStream());
            String mensaje = entrada.readUTF();    //Leemos el mensaje del cliente
            if(mensaje.equals("Detener")) {
                paso.cerrar();
            }
            else if (mensaje.equals("Reanudar")) {
                paso.abrir();
            }
            salida.writeUTF("Servidor avisa recibido " + mensaje);
            msg=mensaje;
            System.out.println("Conexión n."+num+" mensaje recivido de Modulo de control: "+mensaje);
            conexion.close();                          // Y cerramos la conexión
            }
        } catch (IOException e) {
            System.out.println("Error: " + e.getMessage());
        }
}
}
```

Cliente:

Este apartado se encuentra dentro de la clase modulo de control ya que en función del botón que pulsemos en la interface de esta clase sera necesario que envié un mensaje u otro al servidor.

- Cuando se pulsa uno de los botones se crean la conexión y las entrada salida con el servidor que tendrá una ip y un puerto definido en este caso (127,0,0,1)(5000), dependiendo del caso se manda un mensaje u otro para que el servidor proceda acorde al mensaje, después se cierra la conexión.

```
    private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
        try {
            cliente = new Socket(InetAddress.getByAddress(ip),5000);   //Creamo
            entrada = new DataInputStream(cliente.getInputStream());  //Creamos
            salida = new DataOutputStream(cliente.getOutputStream());
            mensaje="Detener";
            salida.writeUTF(mensaje);                                  // Enviam
            respuesta = entrada.readUTF();                             // Leemos
            System.out.println("Enviado a Servidor "+mensaje);
            System.out.println(respuesta);
            cliente.close();                                           // Cerram
        } catch (IOException e) {
            System.out.println("Error: " + e.getMessage());
        }
    }

    private void jButton2ActionPerformed(java.awt.event.ActionEvent evt) {
        try {
            cliente = new Socket(InetAddress.getByAddress(ip),5000);   //Creamo
            entrada = new DataInputStream(cliente.getInputStream());  //Creamos
            salida = new DataOutputStream(cliente.getOutputStream());
            mensaje="Reanudar";
            salida.writeUTF(mensaje);                                  // Enviam
            respuesta = entrada.readUTF();                             // Leemos
            System.out.println("Enviado a Servidor "+mensaje);
            System.out.println(respuesta);
            cliente.close();                                           // Cerram
        } catch (IOException e) {
            System.out.println("Error: " + e.getMessage());
        }
    }
```

Ejercicio RMI:

Servidor:
El servidor esta incluido en la clase colores que es donde se encuentra el código del servidor por lo que se inicia una vez iniciamos la clase colores.

En este fragmento en el método main, definimos el código para crear el objeto remoto que se quiere compartir y hacer el objeto remoto visible para los clientes, mediante la clase Naming y su método rebind(...). El objeto se crea con el paso añadido , también se realiza la conexión del servicio rmi de forma directa con la linea (Registr registry=LocateRegistry(puerto) sin necesidad de acceder desde consola.

```
public static void main(String args[]) {
    java.awt.EventQueue.invokeLater(new Runnable() {
        public void run() {
            new Colores().setVisible(true);
             try {
                RMI obj = new RMI(paso); //Crea una instancia de sí mismo co
                Registry registry = LocateRegistry.createRegistry(1099); //A
                Naming.rebind("//127.0.0.1/ObjetoEstado",obj);      //conect
                System.out.println("El Objeto Estado ha quedado registrado");
            } catch (Exception e) {
            System.out.println(" Error: " + e.getMessage());
            e.printStackTrace();
            }
        }
    });
```

Cliente:
Este apartado se encuentra dentro de la clase modulo de control ya que en función del botón que pulsemos en la interface de esta clase sera necesario que envié un mensaje u otro al servidor.

- Dependiendo del caso se enviara un mensaje u otro, pero básicamente el programa lo que hace en principio es obtener los objetos remotos necesarios .
- Para ello simplemente consiste en buscar el objeto remoto en el registro RMI de la máquina remota. Para ello usamos la clase Naming y su método lookup(...). Y se envía el mensaje al objeto remoto el cual en nuestro caso da un mensaje de retorno con la respuesta dependiendo del mensaje enviado.

```
private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
    String respuesta;
    try {
        String nombre ="Detener";       //mensaje predefinido para este boton
        InterfaceRMI obj = (InterfaceRMI) Naming.lookup("//127.0.0.1/ObjetoEstado'
        respuesta = obj.mensaje(nombre);     //Se manda el mensaje para hacer la pa
        System.out.println(respuesta);       //Se muestra la respuesta del mensaje
    } catch (Exception e) {
        System.out.println("Excepción : " + e.getMessage());
        e.printStackTrace();
    }
}

private void jButton2ActionPerformed(java.awt.event.ActionEvent evt) {
    String respuesta = "";
    try {
        String nombre="Reanudar";       //mensaje predefinido para este boton
        InterfaceRMI obj = (InterfaceRMI) Naming.lookup("//127.0.0.1/ObjetoEstado'
        respuesta = obj.mensaje(nombre);     //Se manda el mensaje para hacer la re
        System.out.println(respuesta);       //Se muestra la respuesta del mensaje
    } catch (Exception e) {
        System.out.println("Excepción : " + e.getMessage());
        e.printStackTrace();
    }
}
```

Interface:
Este apartado es donde se crea la interface del objeto remoto en nuestro caso RMI que es con el que vamos a realizar las conexiones y con el que trabajamos para las comunicaciones entre las partes

- La interfaz debe ser pública.
- Debe heredar de la interfaz java.rmi.Remote, para indicar que puede llamarse desde cualquier máquina virtual Java.
- Cada método remoto debe lanzar la excepción java.rmi.RemoteException en su cláusula throws, además de las excepciones que pueda manejar.

```
package RMI_Colores;

import java.rmi.Remote;
import java.rmi.RemoteException;

/**
 *
 * @author pedro.barquin
 */
public interface InterfaceRMI extends Remote {

    String mensaje(String estado) throws RemoteException;
}
```

RMI:
En esta parte del programa es donde esta la clase remota que se creado en la parte servidor.

- Como se puede observar, la clase Remota implementa la interfaz Interfaz Remota que hemos definido previamente. Junto con el paso que se a añadido al crear que se usara para cerrar o reanudar los colores.
- Luego, dentro de la clase, definimos un constructor (que lanza la excepción RemoteException , y los métodos de la interfaz implemente.
- En nuestro caso dependiendo del mensaje que se reciba se realizara una acción u otra.

```
package RMI_Colores;

/**
 *
 * @author pedro.barquin
 */
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;

public class RMI extends UnicastRemoteObject implements InterfaceRMI
        Paso paso=new Paso();

        public RMI(Paso paso) throws RemoteException {
        this.paso=paso;         //recibimos el paso para poder parar
        }

        public String mensaje(String estado) throws RemoteException
            switch (estado) {                       //Segun lo que no
                case "Detener":
                    paso.cerrar();
                    return "Ejecucion Parada";
                case "Reanudar":
                    paso.abrir();
                    return "Ejecucion Abierta";
                default:
                    return "ok";
            }
```

Conclusiones:

La conclusión de este ejercicio es aprender a usar los tipos diferentes que hay en java para realizar comunicación entre dos partes y las diferencias entre ambos métodos.

- Ejemplo de como queda el programa final: