PRÁCTICA 1

Implementación de un chat con sockets e hilos

NOMBRES:

Javier Sanz Rozalén

Alberto Gómez García

CURSO: 2º DAM

MÓDULO: PROGRAMACIÓN DE SERVICIOS Y PROCESOS

ÍNDICE

1.- INTRODUCCIÓN

Un chat es una aplicación que lleva mucho tiempo desarrollándose y que hace uso de la implementación de sockets e hilos para crear una conexión entre cliente y servidor.

Un socket es una herramienta que establece una apertura en el servidor para que distintos clientes puedan acceder a él, con el fin de transmitir información.

Por otra parte, un hilo es la base donde se sustenta un proceso, dando opción a crear distintos hilos con el fin de que una aplicación ejecute distintos procesos al mismo tiempo, sin que se colapse el hilo principal.

2.- EXPLICACIÓN DEL FUNCIONAMIENTO

Un chat consta de dos partes principales: un servidor y uno o varios clientes. Para iniciar un chat, primero tiene que haber un servidor arrancado en un puerto específico, que será el que reciba la información de los clientes, y a su vez enviar esta misma información al resto de clientes.

Por otra parte, tenemos el lado del cliente, que tendrá que estar conectado al mismo puerto sobre el que se ha arrancado el servidor. Éste enviará información al servidor y obtendrá la respuesta que éste le haya dado.

Para que se conecten varios clientes al servidor y que todos ellos puedan enviar y recibir mensajes de forma simultánea, simulando el funcionamiento de un chat, será necesaria la implementación de hilos. En nuestro caso, hemos creado un hilo diferente por cada socket abierto en el servidor, de esta manera cada cliente que se conecte, lo hará en un hilo distinto.

3.- EXPLICACIÓN DE LAS CLASES UTILIZADAS

Como hemos comentado anteriormente, nuestra idea inicial era que cuando un cliente del chat manda un mensaje al servidor, el propio servidor redirigiera ese mensaje al resto de clientes. De esta manera, el servidor se comporta como un mero gestor de la información, y son los clientes los que reciben y mandan mensajes.

Debido a la complejidad de implementación, hemos decidido hacer una aproximación más simple del funcionamiento real de un chat. En nuestro proyecto, varios clientes se comunicarán simultáneamente con el servidor, gracias a la implementación de hilos, pero esa comunicación será anónima para el resto de clientes.

Debido a la simplificación anterior, sólo hemos utilizado 2 clases para la implementación del chat. Éstas son la clase **Servidor** y la clase **Cliente**.

El código perteneciente a la clase **Servidor** se resume a continuación:

1. Se define el puerto en el que el servidor escuchará, y el tiempo durante el cual el puerto permanecerá abierto esperando conexiones.
2. Mediante un bucle for, se llama al método **crearNuevoHilo()** tantas veces como hilos se quieran crear. El número de hilos será igual al número de clientes que podrán conectarse de manera simultánea con el servidor.
3. Se desarrolla el método **crearNuevoHilo(),** dentro del cual, además de crear un nuevo hilo, el servidor acepta la petición del cliente. Posteriormente se establece el flujo de información (en este caso, cadenas de texto) entre ellos.

El código perteneciente a la clase C**liente** es aún más sencillo que el anterior. Básicamente consiste en:

1. Se configura la dirección del puerto al que el cliente se se conectará con el servidor.
2. Posteriormente se establece el flujo de información (en este caso, cadenas de texto) entre ellos.

4.- CÓDIGO Y CAPTURAS

En este apartado vamos a mostrar capturas de nuestro código, de esta manera se podrá ver de forma práctica los pasos que se han descrito en el apartado anterior.

El código más significativo de la clase **Servidor** es el siguiente:

```
public class Servidor  {

    // Variables de clase
    private static Socket socket_cli=null;
    private static ServerSocket socket;


    public static void main(String args[]){

            // Declaramos un bloque try y catch para controlar la ejecución del subprograma
             try {
                 // Instanciamos un ServerSocket con la dirección del destino y el
                 // puerto que vamos a utilizar para la comunicación
                 socket = new ServerSocket(8000);
                 // Tiempo de espera
                 socket.setSoTimeout(40000);
                 System.out.println("Esperando conexiones");

                 // Se crean los hilos para los clientes
                 for(int i=0; i<5; i++){
                     crearNuevoHilo();
                 }



             }
            // utilizamos el catch para capturar los errores que puedan surgir
            catch (Exception e) {
                // si existen errores los mostrará en la consola y después saldrá del programa
                 System.err.println(e.getMessage());
                 System.out.println("Has superado el número de conexiones");
                 System.exit(1);
             }

    }
```

```
    private static void crearNuevoHilo(){
        Thread hilo = new Thread (new Runnable() {

            @Override
            public void run() {
                try{
                    Socket skCliente = socket.accept(); // Crea objeto
                    System.out.println("Sirvo al cliente");
                    // envío de datos al cliente
                    OutputStream aux = skCliente.getOutputStream();
                    DataOutputStream flujo = new DataOutputStream(aux);
                    flujo.writeUTF("Hola cliente");
                    //skCliente.close();


                    // Declaramos e instanciamos el objeto DataInputStream que nos valdrá para recibir datos del cliente
                     DataInputStream in = new DataInputStream(skCliente.getInputStream());

                    // Creamos un bucle do while en el que recogemos el mensaje
                    // que nos ha enviado el cliente y después lo mostramos
                    // por consola

                    aux = skCliente.getOutputStream();
                    flujo = new DataOutputStream(aux);
                    BufferedReader ent = new BufferedReader(new InputStreamReader(System.in));

                    do {
                        //mostrar mensajes del cliente
                        String mensaje ="";
                        mensaje = in.readUTF();
                        System.out.println(mensaje);

                        //lectura de texto
                        flujo.writeUTF(ent.readLine());
                     } while (1>0);
                }
                catch(Exception e){
                    e.printStackTrace();
                }
            }
        });
        hilo.start();
    }
}
```

El código más significativo de la clase **Cliente** es el siguiente:

```
public class Cliente {

    //método principal de la clase
    public static void main(String argv[]) {

        //Creamos una instancia BuffererReader en la
        //que guardamos los datos introducido por el usuario
        BufferedReader in = new BufferedReader(new InputStreamReader(System.in));

        //declaramos un objeto socket para realizar la comunicación
        Socket socket;

        //declaramos una variable de tipo string
        String mensaje="";

        //Declaramos un bloque try y catch para controlar la ejecución del subprograma
        try {

            //Instanciamos un socket con la dirección del destino y el
            //puerto que vamos a utilizar para la comunicación
            socket = new Socket("127.0.0.1",8000);

            //Declaramos e instanciamos el objeto DataOutputStream
            //que nos valdrá para enviar datos al servidor destino
            DataOutputStream out = new DataOutputStream(socket.getOutputStream());
            DataInputStream inServ = new DataInputStream(socket.getInputStream());
```

```
            //Creamos un bucle do while en el que enviamos al servidor el mensaje
            //los datos que hemos obtenido despues de ejecutar la función
            //"readLine" en la instancia "in"
            do {
                mensaje = in.readLine();
                //enviamos el mensaje codificado en UTF
                out.writeUTF(mensaje);
                //mientras el mensaje no encuentre la cadena fin, seguiremos ejecutando el bucle do-while

                //mostrar mensajes del servidor
                String mensajeServidor ="";
                mensajeServidor = inServ.readUTF();
                System.out.println(mensajeServidor);

            } while (!mensaje.startsWith("fin"));
        }
        //utilizamos el catch para capturar los errores que puedan surgir
        catch (Exception e) {
            //si existen errores los mostrará en la consola y después saldrá del
            //programa
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }
}
```

5.- BIBLIOGRAFÍA

Además de las transparencias y otro material del profesor, las webs utilizadas como consulta para realizar esta práctica son la siguientes:

http://programandointentandolo.com/2013/04/ejemplo-chat-en-java-usando-sockets-e-hilos.html

http://netosolis.com/chat-en-java-sockets-threads/