

Programación de comunicaciones en red. Objetos e Hilos

Programación de servicios y procesos

Contenidos:

- 1) Envío de objetos a través de Sockets TCP
- 2) Conexión de múltiples clientes. Hilos

Envío de objetos a través de Sockets TCP

Hemos visto como los sockets podían intercambiar cadenas de caracteres entre programas cliente y servidor. Pero los stream soportan diversos tipos de datos como son los bytes, los tipos de datos primitivos, caracteres localizados y objetos.

Las clases `ObjectInputStream` y `ObjectOutputStream` nos permiten enviar objetos a través de sockets TCP. Utilizaremos los métodos `readObject()` para leer el objeto del stream y `writeObject()` para escribir el objeto al stream. Usaremos el constructor que admite un `InputStream` y un `OutputStream`. Para preparar el flujo de salida para escribir objetos escribimos:

```
ObjectOutputStream outObjeto = new ObjectOutputStream(socket.getOutputStream());
```

Para preparar el flujo de entrada para leer objetos escribimos:

```
ObjectInputStream inObjeto = new ObjectInputStream(socket.getInputStream());
```

Las clases a la que pertenecen estos objetos deben implementar la interfaz **Serializable**. Podemos ver un ejemplo de dos sockets que intercambian y modifican un objeto en los proyectos `UT4_Ejem3_SocketsObjetosClient` y `UT4_Ejem3_SocketsObjetosServer` de la carpeta de recursos.

Envío de objetos a través de Sockets UDP

Para intercambiar objetos en sockets UDP utilizaremos las clases `ByteArrayOutputStream` y `ByteArrayInputStream`. Se necesita convertir el objeto a un array de bytes. Por ejemplo, para convertir un objeto `Persona` a un array de bytes escribimos las siguientes líneas:

```
Persona persona = new Persona ("Maria", 22);
```

```
//CONVERTIMOS OBJETO A BYTES
```

```
ByteArrayOutputStream bs= new ByteArrayOutputStream ();  
ObjectOutputStream out = new ObjectOutputStream (bs);  
out.writeObject (persona); //escribir objeto Persona en el  
stream
```

```
out.close(); //cerrar stream
```

```
byte[] bytes = bs.toByteArray(); // objeto en bytes
```

```
// RECIBO DATAGRAMA
```

```
byte[] recibidos = new byte[1024];
```

```
DatagramPacket pagRecibido = new
```

```
DatagramPacket (recibidos, recibidos.length);
```

```
socket .receive (pagRecibido); //recibo el datagrama
```

```
// CONVERTIMOS BYTES A OBJETO
```

```
ByteArrayInputStream bais = new
```

```
ByteArrayInputStream(recibidos);
```

```
ObjectInputStream in = new ObjectInputStream(bais);
```

```
Persona persona = (Persona) in.readObject (); //obtengo objeto
```

```
in.close();
```

Conexión de múltiples clientes. Hilos

Hasta ahora los programas servidores que hemos creado solo son capaces de atender a un cliente en cada momento, pero lo más típico es que un programa servidor pueda atender a muchos clientes simultáneamente.

La solución para poder atender a múltiples clientes está en el multihilo, cada cliente (o clientes) será atendido en un hilo.

El esquema básico en sockets TCP sería construir un único servidor con la clase `ServerSocket` e invocar al método `accept()` para esperar las peticiones de conexión de los clientes.

Cuando un cliente (o varios, según diseñemos la aplicación) se conecta, el método `accept()` devuelve un objeto `Socket`, éste se usará para crear un hilo cuya misión es atender a este cliente/s.

Después se vuelve a invocar a `accept()` para esperar a un nuevo cliente; habitualmente la espera de conexiones se hace dentro de un bucle infinito.

El hilo creado se encargará de atender al cliente, dejando así al servidor libre para atender peticiones. Vamos a verlo en un ejemplo

Conexión de múltiples clientes. Hilos. TIC, TAC

Vamos a hacer el ejercicio de Tic, Tac con un servidor que se encargará de pintar lo que le manden los clientes y dos clientes que se conectan simultáneamente al servidor. Uno enviará TIC y el otro TAC. La clase servidor sería la siguiente:

```
public class TictacServer {  
    public static void main(String args[]) throws IOException {  
        ServerSocket servidor;  
        servidor = new ServerSocket(6000,2);  
        System.out.println("Servidor iniciado...");  
  
        while (true) {  
            Socket cliente = new Socket();  
            Socket cliente2 = new Socket();  
            cliente=servidor.accept();//esperando cliente 1  
            cliente2=servidor.accept();//esperando cliente 2  
            TictacServer_Hilo hilo = new TictacServer_Hilo(cliente, cliente2);  
            hilo.start();  
        }  
    }  
}
```

Establecemos el puerto de escucha y en este caso queremos 2 clientes para cada hilo.

Bucle infinito de espera.

Una vez aceptamos a los clientes se los pasamos a la clase de TicTac hilo que hemos hecho arrancamos el hilo.

Conexión de múltiples clientes. Hilos. TIC, TAC

Vemos comentada la clase TictacServer_Hilo. Lo primero es una clase debe extender de hilo

```
public TictacServer_Hilo(Socket miSocket1, Socket miSocket2){  
    this.miSocket1 = miSocket1;  
    this.miSocket2 = miSocket2;  
    // se crean flujos de entrada  
    try {  
        entrada = this.miSocket1.getInputStream();  
        flujoEntrada = new DataInputStream(entrada);  
        entrada2 = this.miSocket2.getInputStream();  
        flujoEntrada2 = new DataInputStream(entrada2);  
    } catch (IOException e) {  
        // TODO Auto-generated catch block  
        e.printStackTrace();  
    }  
}
```

En el constructor recibimos los sockets y creamos los flujos de entrada

Conexión de múltiples clientes. Hilos. TIC, TAC

Vemos comentada la clase TictacServer_Hilo

```
public void run() {// tarea a realizar con el cliente
    String cadena = "";
    String cadena2 = "";
    while (!cadena.trim().equals("*")&&!cadena2.trim().equals("*")) {
        try {
            cadena = flujoEntrada.readUTF();
            System.out.println(cadena);
            cadena2 = flujoEntrada2.readUTF();
            System.out.println(cadena2);
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    } // fin while
}
```

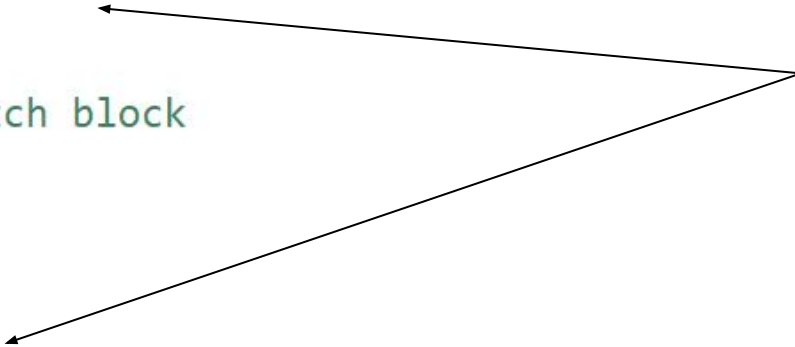
Leemos el flujo de entrada hasta que nos lleguen *. Pintamos por consola lo que nos llega.

Conexión de múltiples clientes. Hilos. TIC, TAC

Vemos comentada la clase TictacServer_Hilo

```
System.out.println("FIN CON: " + this.miSocket1.toString());
System.out.println("FIN CON: " + this.miSocket2.toString());
try {
    flujoEntrada.close();
    flujoEntrada2.close();
} catch (IOException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}
try {
    this.miSocket1.close();
    this.miSocket2.close();
} catch (IOException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}
```

Cerramos el flujo de entrada y el socket



Conexión de múltiples clientes. Hilos. TIC, TAC

Vemos ahora comentada la clase cliente TIC (TAC es igual, pero envía TAC en lugar de TIC)

```
public class Tic {  
    public static void main(String[] args) throws IOException {  
        String Host = "localhost";  
        int Puerto = 6000; // puerto remoto  
        Socket Cliente = new Socket(Host, Puerto);  
        OutputStream salida;  
        // CREO FLUJO DE SALIDA AL SERVIDOR  
        salida = Cliente.getOutputStream();  
        DataOutputStream flujoSalida = new DataOutputStream(salida);  
        for(int i=0; i<100; i++) {  
            flujoSalida.writeUTF("TIC");  
            flujoSalida.flush();  
            try {  
                Thread.sleep(500);  
            } catch (InterruptedException e) {  
                // TODO Auto-generated catch block  
                e.printStackTrace();  
            }  
        }  
    }  
}
```

Establecemos el host y el puerto al que me conecto

Creamos el flujo de salida del socket

Escribimos 100 veces tic, esperando medio segundo y cerramos

Conexión de múltiples clientes. Hilos. TIC, TAC

Vemos ahora comentada la clase cliente TIC (TAC es igual, pero envía TAC en lugar de TIC)

```
flujoSalida.writeUTF("*");  
flujoSalida.close();
```

```
//
```

Mandamos el carácter de FIN y cerramos todo

```
System.out.println("Fin del envio... ");  
Cliente.close();  
}//
```