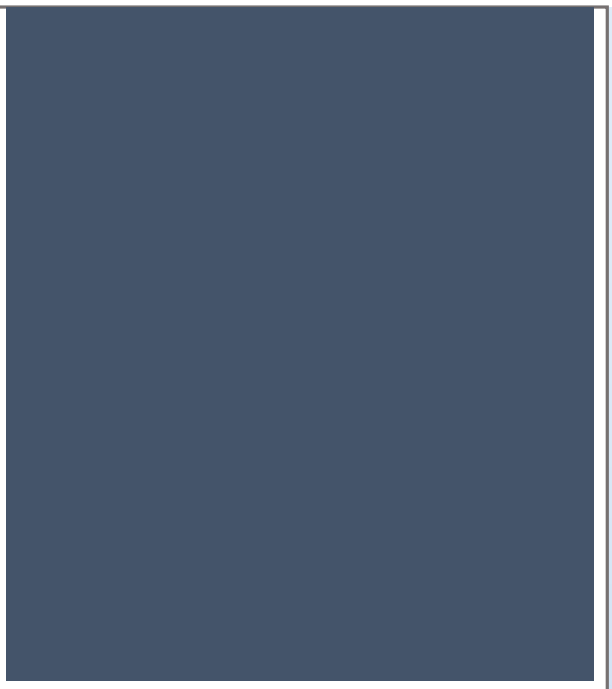

Programación de servicios y procesos

Act3.1. Intercambio de mensajes de texto entre cliente y servidor mediante TCP

Francisco José García Cutillas | 2FPGS_DAM

Índice

Ejercicio 1

Intercambio de mensajes de texto entre clientes y servidor a través de sockets TCP. Crear una aplicación cliente y una aplicación servidor que intercambien mensajes de texto basando la comunicación en sockets TCP.

Crear una aplicación cliente y una aplicación servidor que intercambien mensajes de texto basando la comunicación en sockets TCP.

Código del cliente

En esta primera parte tenemos el main del cliente. En él creamos un objeto de la clase "SocketTCPClient" desarrollada más adelante. Este objeto recibe una dirección IP y un puerto, los cuales corresponden a la IP del servidor y el puerto por el que nos vamos a comunicar con el mismo.

En este caso, iniciamos el cliente con el método ".start()", abrimos la comunicación (.abrirComunicacion()), creamos un String para guardar el mensaje que queremos enviar, creamos un objeto de tipo Scanner para introducir el mensaje por terminal y lo guardamos en el String anterior (mensajeEnviar).

Con el método ".enviarMensaje()", introduciéndole como parámetro el mensaje introducido por terminal, enviamos el mensaje al servidor.

Tras esto creamos un String "mensajeRecibido" en el que vamos a guardar, usando el método ".leerMensaje()", el mensaje que nos envíe el servidor y posteriormente lo mostramos por terminal haciendo uso del getter del InputStream del objeto SocketTCPClient y del método ".read()" de InputStream.

Para finalizar, cerramos la comunicación primero y después terminamos cerrando el objeto cliente.

Cabe destacar que debemos controlar las excepciones que pueden ocurrir, como son "UnknownHostException" e "IOException".

```java
package com.mycompany.garcia_cutillas_franciscojose_act3_1cliente;

import java.io.IOException;
import java.net.UnknownHostException;
import java.util.Scanner;

/**
 *
 * @author Fran
 */
public class Garcia_Cutillas_FranciscoJose_Act3_1Cliente {

    public static void main(String[] args) {

        SocketTCPClient cliente = new SocketTCPClient( serverIP: "10.8.168.84",  serverPort: 49171);

        try {

            cliente.start();
            cliente.abrirComunicacion();
            String mensajeEnviar = "";
            Scanner sc = new Scanner( source: System.in);
            System.out.println( x: "Mensaje:");
            mensajeEnviar = sc.nextLine();
            cliente.enviarMensaje( mensaje: mensajeEnviar);
            String mesajeRecibido = cliente.leerMensaje();
            System.out.println("Mensaje del servidor: " + cliente.getIs().read()) ;
            cliente.cerrarComunicacion();
            cliente.stop();

        } catch (UnknownHostException ex) {
            System.out.println("Error: " + ex.getMessage());
        } catch (IOException ex) {
            System.out.println("Error: " + ex.getMessage());
        }

    }
}
```

La clase creada para la parte del cliente es “SocketTCPClient”. En ella tenemos los métodos:

- start(). Necesario para establecer la conexión con el servidor. En este método establecemos un Socket con una IP y un puerto determinados, además de establecer sus canales de entrada y salida de información.
- stop(). Con este método somos capaces de cerrar la comunicación. En él cerramos primero los canales de entrada y salida del socket, para finalmente cerrar el socket y finalizar la conexión.
- abrirComunicacion(). Método necesario para leer el flujo de datos entre el servidor y el cliente.
- cerrarComunicacion(). Con este método finalizamos el flujo de datos abierto con “abrirComunicacion()”.
- leerMensaje(). Este método nos va a devolver una cadena con el mensaje recibido del servidor.
- enviarMensaje(). Para enviar un mensaje del cliente al servidor.

- Métodos getter y setter. Con estos métodos somos capaces de acceder desde fuera de la clase a todos los atributos de la misma, puesto que éstos son privados.

```java
package com.mycompany.garcia_cutillas_franciscojose_act3_lcliente;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.OutputStream;
import java.io.PrintWriter;
import java.net.Socket;
import java.net.UnknownHostException;

/**
 *
 * @author Fran
 */
public class SocketTCPClient {

    private String serverIP;
    private int serverPort;
    private Socket socket;
    private InputStream is;
    private OutputStream os;
    private InputStreamReader isr;
    private BufferedReader br;
    private PrintWriter pw;

    public SocketTCPClient (String serverIP, int serverPort){
        this.serverIP = serverIP;
        this.serverPort = serverPort;
    }

    public void start() throws UnknownHostException, IOException{

        System.out.println( x: "(Cliente) Estableciendo conexion...");
        socket = new Socket( host: serverIP,  port: serverPort);
        os = socket.getOutputStream();
        is = socket.getInputStream();

        System.out.println( x: "(Cliente) Conexión establecida.");

    }

    public void stop() throws IOException{

        System.out.println( x: "(Cliente) Cerrando conexión...");
        is.close();
        os.close();
        socket.close();
        System.out.println( x: "(Cliente) Conexiones cerradas.");

    }

    public void abrirComunicacion(){

        isr = new InputStreamReader( in: is);
        br = new BufferedReader( in: isr);

        pw = new PrintWriter( out: os,  autoFlush: true);

    }

```

```java
    public void cerrarComunicacion() throws IOException{

        br.close();
        isr.close();
        pw.close();

    }

    public String leerMensaje() throws IOException{

        String mensaje = br.readLine();
        return mensaje;

    }

    public void enviarMensaje(String mensaje){

        pw.println( x: mensaje);

    }

    public String getServerIP() {
        return serverIP;
    }

    public void setServerIP(String serverIP) {
        this.serverIP = serverIP;
    }
```

```java
    public int getServerPort() {
        return serverPort;
    }

    public void setServerPort(int serverPort) {
        this.serverPort = serverPort;
    }

    public Socket getSocket() {
        return socket;
    }

    public void setSocket(Socket socket) {
        this.socket = socket;
    }

    public InputStream getIs() {
        return is;
    }

    public void setIs(InputStream is) {
        this.is = is;
    }

    public OutputStream getOs() {
        return os;
    }

    public void setOs(OutputStream os) {
        this.os = os;
    }

}
```

```
-----< com.mycompany:Garcia_Cutillas_FranciscoJose_Act3_1Cliente >------
Building Garcia_Cutillas_FranciscoJose_Act3_1Cliente 1.0-SNAPSHOT
--------------------------------[ jar ]---------------------------------

--- exec-maven-plugin:3.0.0:exec (default-cli) @ Garcia_Cutillas_FranciscoJose_Act3_1Cliente ---
(Cliente) Estableciendo conexion...
(Cliente) Conexión establecida.
Mensaje:
Pepe
(Cliente) Cerrando conexión...
(Cliente) Conexiones cerradas.
(Cliente) Estableciendo conexion...
(Cliente) Conexión establecida.
Mensaje:
Hola
(Cliente) Cerrando conexión...
(Cliente) Conexiones cerradas.
(Cliente) Estableciendo conexion...
(Cliente) Conexión establecida.
Mensaje:
Juan
(Cliente) Cerrando conexión...
(Cliente) Conexiones cerradas.
(Cliente) Estableciendo conexion...
(Cliente) Conexión establecida.
Mensaje:
Antonio
(Cliente) Cerrando conexión...
(Cliente) Conexiones cerradas.
(Cliente) Estableciendo conexion...
(Cliente) Conexión establecida.
```

Código del servidor

En el caso del servidor, sólo es necesario establecer el puerto de comunicación a la hora de crear el objeto, iniciamos el mismo e iniciamos la conexión.

Nos creamos una variable de tipo String para guardar el texto recibido del cliente, y lo leemos con el método “leerTexto()”, para posteriormente mostrarlo por terminal.

Al finalizar de mostrar el mensaje, cerramos la conexión “cerrarConexion()” y paramos el servicio “stop()”.

Debemos controlar la excepción IOException.

```
public static void main(String[] args) {
    try{
        while(true){
        Servidor ser= new Servidor( puerto: 49171);
        ser.start();
        ser.conexion();
        String mensajeRec=ser.leerTexto();
        System.out.println("Mensaje del cliente: "+mensajeRec);
        ser.enviarTexto( mensaje: "Mensaje desde el servidor");
        ser.cerrarConexion();
        ser.stop();
        }

    }catch (IOException ioe){

        ioe.printStackTrace();
    }



}
```

En la clase Servidor, tenemos los métodos:

- start(). En el primero vamos a establecer un socket de conexión, para después asignarle sus canales de entrada y salida.
- Stop(). Método para cerrar la conexión y finalizar el servicio.
- conexion(). Método para leer el flujo de datos entre el servidor y cliente.
- cerrarConexion(). Este método es necesario para finalizar el flujo de datos una vez que éstos ya se hayan leído.
- leerTexto(). Método utilizado para leer el texto que venga del cliente.
- enviarTexto(). Este método es utilizado para enviar una cadena de texto al cliente.

```java
public class Servidor {
    private ServerSocket serverSocket;
    private Socket socket;
    private InputStream is;
    private OutputStream os;

    private InputStreamReader isr;
    private BufferedReader br;
    private PrintWriter pw;


    public Servidor(int puerto)throws IOException{
        serverSocket= new ServerSocket( port: puerto);
    }

    public void start() throws IOException {
        System.out.println( x: "(Servidor) Esperando conexiones...");
        socket = serverSocket.accept();
        is= socket.getInputStream();
        os=socket.getOutputStream();
        System.out.println( x: "(Servidor) Conexión establecida");
    }

    public void stop() throws IOException{

        System.out.println( x: "(Servidor) Cerrando conexiones...");
        is.close();
        os.close();
        socket.close();
        serverSocket.close();
        System.out.println( x: "(Servidor) Conexiones cerradas");

    }
```

```java
public void conexion(){

    System.out.println( x: "Abriendo conexion de texto");
    isr=new InputStreamReader( in: is);
    br=new BufferedReader( in: isr);
    pw= new PrintWriter( out: os,  autoFlush: true);
    System.out.println( x: "Conexion de texto abierta");
}

public void cerrarConexion() throws IOException{
    System.out.println( x: "Cerrando conexion de texto");
    br.close();
    isr.close();
    pw.close();
    System.out.println( x: "Conexión de texto cerrada");


}

public String leerTexto() throws IOException{

    System.out.println( x: "Leyendo Mensaje");
    String mensaje=br.readLine();
    System.out.println( x: "Mensaje leido");
    return mensaje;

}

public void enviarTexto(String mensaje){

    System.out.println( x: "Enviando Mensaje");
    pw.println( x: mensaje);
    System.out.println( x: "Mensaje enviado");


}
```

```
Mensaje del cliente: hola
Enviando Mensaje
Mensaje enviado
Cerrando conexion de texto
Conexión de texto cerrada
(Servidor) Cerrando conexiones...
(Servidor) Conexiones cerradas
(Servidor) Esperando conexiones...
(Servidor) Conexión establecida
Abriendo conexion de texto
Conexion de texto abierta
Leyendo Mensaje
Mensaje leido
Mensaje del cliente: Pepe
Enviando Mensaje
Mensaje enviado
Cerrando conexion de texto
Conexión de texto cerrada
(Servidor) Cerrando conexiones...
(Servidor) Conexiones cerradas
(Servidor) Esperando conexiones...
(Servidor) Conexión establecida
Abriendo conexion de texto
Conexion de texto abierta
Leyendo Mensaje
Mensaje leido
Mensaje del cliente: Juan
Enviando Mensaje
Mensaje enviado
Cerrando conexion de texto
```