Repaso - Sockets TCP

Utilizamos esta clase para implementar tanto los clientes como los servidores con TCP.

Mas información en la documentación oficial:
https://docs.oracle.com/javase/tutorial/networking/sockets/index.html

El lado del cliente

Crear la conexión

Para conectarnos como cliente a un servidor necesitamos especificarle como mínimo el puerto y la IP del servidor. Para ellos podemos usar uno de los siguientes constructores:

- **Socket(InetAddress address, int port)** la dirección la expresamos como un objeto InetAddress
- **Socket(String host, int port)** la dirección la expresamos como un String
- **Socket(InetAddress address, int port, InetAddress localAddr, int localPort)** en este caso no solo especificamos el puerto de destino, sino tambien nuestra direccion de origen y puerto. (Los datos de origen siempre se especifican, pero en vez de dejar que sea el sistema operativo quien rellene estos datos los especificamos nosotros)

```
public class Cliente {
    public static void main(String[] args) {
        try {
            Socket socket = new Socket("localhost", 6666); // Abro conexión

            // Con este buffer leo los datos que me envia el servidor
            BufferedReader bufferEntrada = new BufferedReader(new
InputStreamReader(socket.getInputStream()));

            // Con este buffer escribo y envío al servidor
            /* El parametro true indica el autoflush, es decir, que por cada
println
             * que escriba se enviará al servidor sin necesidad de usar la funciín
flush() */
            PrintWriter bufferSalida = new PrintWriter(socket.getOutputStream(),
true);

            // Leo un dato del servidor y lo muestro por pantalla
            System.out.println(bufferEntrada.readLine());
            // Envío un dato al servidor
            bufferSalida.println("hola");

            socket.close(); // Cierro el conexión

        } catch (IOException e) {
            e.printStackTrace();
        }
```

```
    }
}
```

El lado del servidor

La clase SocketServer se utiliza para crear programa servidor que se comuniquen con un cliente a través de la red utilizando el protocolo TCP/IP. Los pasos a seguir son los siguientes:

1. Crear un **ServerSocket** y asociarlo conb un numero de puerto.
2. Escuchar la conexión y **aceptar peticiones de los clientes**. Esto proporcionara un objeto socket con el que nos comunicaremos con el cliente.
3. Enviamos y/o recibimos información del cliente
4. Cerramos la conexión.

Crear la conexión SocketServer

Para crear un socketServer utilizamos uno de los siguientes constructores.

- **ServerSocket(int port)**: Crea un socket ligado al puerto. El número máximo de conexión es 50.
- **ServerSocket(int port, int backlog)**: Igual que el anterior pero el número máximo de conexión esta determinado por el parámetro backlog.
- **ServerSocket(int port, int backlog, InetAddress bindAddr)**: Igual que el anterior, pero el parámetro bindAddr asocia el socket a una IP en particular. Un servidor puede tener más de una dirección IP.

Con el meteodo **accept()** aceptamos la petición de un cliente, cuando este abra una conexión con nuestro servidor. El método accept() bloquea el hilo de ejecución hasta que un cliente nos solicita una conexión.

Aceptar conexiones de multiples clientes

No tendria mucho sentido si un servidor solo aceptara la conexion de un cliente y finalizara su ejecución. En el siguiente ejemplo se propone un servidor que aceptara de forma indefinida clientes:

1. Abro el puerto de conexión en mi puerto 6666.
2. Cada vez que un cliente se conecte realizaré las tareas que hagan falta y luego finalizaré la conexión.
3. Cuando termine con dicho cliente cierro la conexión y vuelvo al principio del while done esperaré a que se conecte un nuevo cliente.

```
public class Servidor {
    public static void main(String[] args) {
        try {
            // Abro mi puerto 666 y este proceso recogerá las peticiones que
lleguen.
            ServerSocket servidorSocket =  new ServerSocket(6666);
            while(true){
                /* Cuando cliente se conecte me comunicaré con el a través del
socket
                   que devuelve el método accept() */
                Socket socket = servidorSocket.accept();
```

```
                // Abro los stream de comunicación
                PrintWriter salida = new PrintWriter(socket.getOutputStream(),
true);
                BufferedReader entrada = new BufferedReader(new
InputStreamReader(socket.getInputStream()));

                // Hago las tareas correspondientes a mi servidor.

                socket.close();
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Aceptar conexiones de multiples clientes de forma concurrente.

Con el módelo anterior el primer cliente que se conecte podrá trabajar con el servidor inmediatamente, pero el segundo cliente tendrá que esperar a que el primer cliente termine para trabajar con el servidor. Esto supone un problema si las tareas con el servidor conyevan varios minutos.

A continuación se propone una solución, que es que la conexión de cada cliente se maneje en un hilo de forma que puedan los dos operar con el servidor de forma concurrente.

1. La clase Servidor abre el puerto 6666.
2. Por cada cliente que se conecte creara una instancia de ManejadorClientes. Le enviará el socket o conexión con dicho cliente, e iniciara el hilo.
3. Mientras ManejadorClientes hace su trabajo, la clase servidor puede volver al principio del bucle y aceptar nuevos clientes si fuera necesario.

```
public class Servidor {
    public static void main(String[] args) {
        try {
            ServerSocket servidor = new ServerSocket(6666);
            while(true){
                Socket clienteSocket = servidor.accept();
                ManejadorClientes cliente = new ManejadorClientes(clienteSocket);
                cliente.start();
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

```
public class ManejadorClientes extends Thread{

    Socket socketCliente;

    public ManejadorClientes(Socket socketCliente) {
        this.socketCliente = socketCliente;
    }

    @Override
    public void run() {
        try {
            PrintWriter writer = new PrintWriter(socketCliente.getOutputStream(),
true);
            BufferedReader reader = new BufferedReader(new
InputStreamReader(socketCliente.getInputStream()));
            // Realizo las tareas necesarias
            socketCliente.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```