UD3 - 03.- Roles Cliente y Servidor

Listado de Contenidos:

1. Definición y roles:
    - ¿Qué es un cliente?
    - ¿Qué es un servidor?
2. Características de un cliente.
3. Características de un servidor.
4. Comunicación básica entre cliente y servidor.
5. Ejemplo práctico: Implementar un cliente y un servidor básicos utilizando sockets TCP.

Desarrollo de Contenidos

1. Definición y Roles

En el contexto de las comunicaciones en red, cliente y servidor son dos roles complementarios que permiten el intercambio de datos entre aplicaciones.

- Cliente: Es una aplicación que inicia la comunicación solicitando servicios o recursos.
- Servidor: Es una aplicación que espera solicitudes de clientes y proporciona los servicios o recursos requeridos.

Ejemplo sencillo:

- Cuando navegas en internet, tu navegador web es el cliente que solicita una página a un servidor. El servidor procesa la solicitud y devuelve el contenido de la página.

---

2. Características de un Cliente

El cliente es proactivo. Sus características principales son:

1. Inicia la conexión: Es quien envía la primera solicitud al servidor.
2. Dependencia del servidor: El cliente no puede funcionar sin el servidor.
3. Temporalidad: La conexión cliente-servidor puede durar solo el tiempo necesario para completar la solicitud.
4. Interface de usuario: Por lo general, el cliente tiene una interfaz con la que interactúa el usuario.

Ejemplo de aplicaciones cliente:

- Navegadores web (Chrome, Firefox).
- Clientes de correo electrónico (Outlook, Thunderbird).

- Aplicaciones móviles que consumen servicios de backend.

---

3. Características de un Servidor

El servidor es reactivo. Sus características principales son:

1. Espera conexiones: Permanece activo, esperando solicitudes de los clientes.
2. Proporciona servicios: Responde a las solicitudes con los recursos o servicios necesarios.
3. Soporta múltiples clientes: Puede atender varias conexiones al mismo tiempo, utilizando técnicas como la programación multihilo.
4. Permanencia: Normalmente, los servidores están diseñados para estar en funcionamiento constante.

Ejemplo de servidores:

- Servidores web (Apache, Nginx).
- Servidores de bases de datos (MySQL, PostgreSQL).
- Servidores de archivos (FTP).

---

4. Comunicación Básica entre Cliente y Servidor

La comunicación entre cliente y servidor sigue un esquema general:

1. El cliente inicia la conexión: Utilizando un protocolo como TCP o UDP.
2. El servidor acepta la conexión: Configura un socket para recibir datos del cliente.
3. Intercambio de datos: El cliente envía solicitudes y el servidor responde.
4. Cierre de conexión: Una vez finalizado el intercambio, se cierra la conexión.

Esquema visual:

```
Cliente  -------->   Solicita un recurso   -------->   Servidor
         <--------   Responde con el recurso  <--------
```

---

5. Ejemplo Práctico: Cliente y Servidor Básicos

Vamos a implementar un ejemplo donde un cliente envía un mensaje al servidor, y el servidor responde.

Servidor TCP

```
import java.io.BufferedReader;

import java.io.InputStreamReader;

import java.io.OutputStream;

import java.io.PrintWriter;
```

```
import java.net.ServerSocket;
import java.net.Socket;

public class ServidorTCP {
    public static void main(String[] args) {
        try {
            // Crear un socket para escuchar en el puerto 8080
            ServerSocket servidor = new ServerSocket(8080);
            System.out.println("Servidor en espera de conexiones...");

            // Aceptar conexión de un cliente
            Socket socket = servidor.accept();
            System.out.println("Cliente conectado.");

            // Leer mensaje del cliente
            BufferedReader entrada = new BufferedReader(new InputStreamReader(socket.getInputStream()));
            String mensaje = entrada.readLine();
            System.out.println("Mensaje recibido del cliente: " + mensaje);

            // Enviar respuesta al cliente
            OutputStream output = socket.getOutputStream();
            PrintWriter escritor = new PrintWriter(output, true);
            escritor.println("Hola, cliente. Mensaje recibido: " + mensaje);

            // Cerrar conexión
            socket.close();
            servidor.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Cliente TCP

```
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.OutputStream;
```

```
import java.io.PrintWriter;
import java.net.Socket;

public class ClienteTCP {
    public static void main(String[] args) {
        try {
            // Conectar al servidor en localhost y puerto 8080
            Socket socket = new Socket("127.0.0.1", 8080);

            // Enviar mensaje al servidor
            OutputStream output = socket.getOutputStream();
            PrintWriter escritor = new PrintWriter(output, true);
            escritor.println("¡Hola, servidor!");

            // Leer respuesta del servidor
            BufferedReader entrada = new BufferedReader(new InputStreamReader(socket.getInputStream()));
            String respuesta = entrada.readLine();
            System.out.println("Respuesta del servidor: " + respuesta);

            // Cerrar conexión
            socket.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Explicación del Código

1. Servidor TCP:
    - Se crea un `ServerSocket` en el puerto `8080` que escucha conexiones entrantes.
    - Cuando un cliente se conecta, se acepta la conexión con el método `accept()`.
    - Se leen los datos enviados por el cliente y se responde con un mensaje.
2. Cliente TCP:
    - Se conecta al servidor en la dirección `127.0.0.1` (localhost) y el puerto `8080`.
    - Envía un mensaje al servidor utilizando un flujo de salida (`OutputStream`).
    - Recibe la respuesta del servidor y la imprime en la consola.

6. Ejemplo de Uso

Al ejecutar estos programas:

1. Inicia el servidor primero.
2. Luego ejecuta el cliente.
3. Observa cómo el cliente envía un mensaje al servidor y recibe una respuesta.

En resumen

Los roles de cliente y servidor son fundamentales para las comunicaciones en red. Mientras que el cliente inicia las conexiones y solicita servicios, el servidor los proporciona y gestiona múltiples conexiones. Este modelo es la base de aplicaciones modernas como navegadores web, servicios de streaming y sistemas de mensajería.