UD3 - 05. Zócalos (Sockets)

En esta unidad abordaremos de manera completa y didáctica el concepto de zócalos o **sockets** , un elemento esencial para la programación de aplicaciones en red. Los enchufes son el punto de partida para establecer comunicación entre dispositivos en una red, ya sea local o global.

---

1. Introducción

Un enchufe es un punto final para la comunicación entre dos máquinas. Es una abstracción de software que permite a las aplicaciones enviar y recibir datos a través de redes. Los enchufes son la base para implementar modelos cliente-servidor y otras arquitecturas de red.

---

2. Concepto y características

1. **Definición de socket**
    - Es una interfaz de programación que permite a dos nodos de una red comunicarse.
    - Cada socket está asociado a un puerto y una dirección IP.
2. **Características principales**
    - Facilitar la comunicación bidireccional.
    - Son independientes del lenguaje de programación.
    - Soportan protocolos como TCP (orientado a conexión) y UDP (no orientado a conexión).
3. **Relación con los protocolos**
    - **TCP:** Asegura una comunicación confiable y ordenada.
    - **UDP:** Ofrece una comunicación rápida y ligera, pero no garantiza la entrega ni el orden.

---

3. Tipos de sockets

1. **Sockets orientados a conexión (TCP)**
    - Proporcionarán un canal de comunicación confiable.
    - Garantizan que los datos se entreguen en el orden correcto.
    - Ejemplo de uso: transferencia de archivos.
2. **Enchufes no orientados a conexión (UDP)**
    - Envían datos como datagramas.
    - No garantizan la entrega ni el orden de los datos.
    - Ejemplo de uso: aplicaciones de streaming en tiempo real.

---

3. **Sockets sincrónicos vs. asíncronos**
    - **Sincrónicos:** Bloquean la ejecución hasta que se completa una operación.
    - **Asíncronos:** Permiten que la aplicación continúe mientras se procesan las operaciones de red.

---

4. Creación y uso de sockets

1. **Creación de un Socket en Java**

Para un cliente:

```
Socket socket = new Socket("127.0.0.1", 8080);
```

Para un servidor:

```
ServerSocket serverSocket = new ServerSocket(8080);
Socket cliente = serverSocket.accept();
```

    -

2. **Operaciones básicas**
    - **Apertura:** Crear un enchufe con un puerto y dirección.
    - **Enlace (binding):** Asociar el socket a un puerto local.
    - **Escucha:** Configurar el enchufe para aceptar conexiones.
    - **Lectura y escritura:** Utilizar flujos de entrada y salida para enviar y recibir datos.
    - **Cierre:** Finalizar la conexión para liberar recursos.
3. **Gestión de errores**
    - Excepciones comunes:
        - `IOException`:Error en la comunicación.
        - `BindException`:Puerto ya en uso.
    - Soluciones:
        - Manejo con bloques `try-catch`.
        - Asignar puertos dinámicos.

---

5. Ejemplos prácticos

Servidor TCP con sockets

Este servidor espera conexiones y responde a mensajes de los clientes.

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

```
import java.io.InputStream;
import java.io.OutputStream;

public class ServidorTCP {
    public static void main(String[] args) {
        try (ServerSocket serverSocket = new ServerSocket(8080)) {
            System.out.println("Servidor esperando conexiones...");
            Socket cliente = serverSocket.accept();
            System.out.println("Cliente conectado.");

            InputStream entrada = cliente.getInputStream();
            byte[] buffer = new byte[1024];
            int leido = entrada.read(buffer);
            System.out.println("Mensaje del cliente: " + new
String(buffer, 0, leido));

            OutputStream salida = cliente.getOutputStream();
            salida.write("Mensaje recibido".getBytes());

            cliente.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Cliente TCP con sockets

Este cliente envía un mensaje al servidor y recibe una respuesta.

```
import java.net.Socket;
import java.io.InputStream;
import java.io.OutputStream;

public class ClienteTCP {
    public static void main(String[] args) {
        try (Socket socket = new Socket("127.0.0.1", 8080)) {
            OutputStream salida = socket.getOutputStream();
            salida.write("Hola, servidor".getBytes());
```

```
            InputStream entrada = socket.getInputStream();
            byte[] buffer = new byte[1024];
            int leido = entrada.read(buffer);
            System.out.println("Respuesta del servidor: " + new
String(buffer, 0, leido));
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

---

6. Aplicaciones prácticas

1. **Chat en red:** Usando sockets TCP para enviar y recibir mensajes.
2. **Streaming:** Implementación de sockets UDP para transmisión de vídeo en tiempo real.
3. **Sistemas distribuidos:** Uso de sockets en aplicaciones cliente-servidor para bases de datos y sistemas de archivos.

---

7. Ventajas y limitaciones de los enchufes

- **Ventajas:**
  - Flexibilidad para implementar distintos protocolos.
  - Comunicación eficiente y directa entre aplicaciones.
  - Compatibilidad con múltiples idiomas.
- **Limitaciones:**
  - Complejidad en la gestión de conexiones múltiples.
  - Requiere un manejo explícito de errores y recursos.

---

8. Conclusión

El uso de zócalos o sockets es un pilar en la programación de aplicaciones en red. Su comprensión permite desarrollar soluciones versátiles que abarcan desde sistemas locales hasta aplicaciones distribuidas a gran escala. Dominar su funcionamiento y manejo es clave para cualquier desarrollador que desee trabajar con redes.