

Java Sockets

Antonio Espín Herranz

SOCKETS

- Conceptos sobre Redes, Sockets y Datagramas.
- Librería java.net.
- Protocolo http en aplicaciones Java.
- Creación y uso de datagramas y sockets.
- Cliente / Servidor.
- Envío de Correo Electrónico y ficheros con sockets.

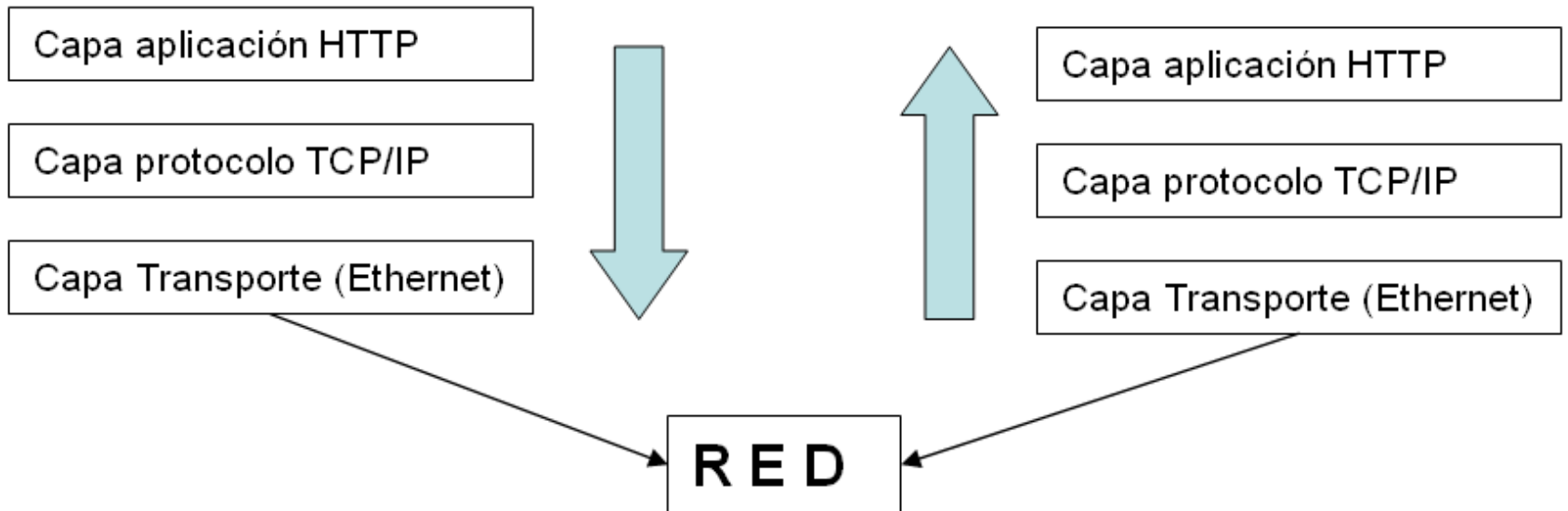
Comunicación entre Procesos

- En las aplicaciones distribuidas y de red es necesario que los componentes que se ejecutan en un proceso se comuniquen con los componentes de otro proceso.
- Por ejemplo, un cliente que manda información para escribir en una BD de un servidor.
- El lenguaje que utilizarán los procesos recibe el nombre de protocolo.

Protocolos

- Nos abstraen del nivel físico de la red.
- La arquitectura OSI proporciona una abstracción de algunas de las capas de comunicación de redes.
- El modelo OSI los podemos simplificar en 3 capas: Aplicación (HTTP, FTP), protocolo (TCP, UDP) y transporte.

Comunicación entre Procesos



Sockets

- Socket → Conector de red.
- Se asemeja a un enchufe en la red eléctrica.
- Es el punto final de la conexión entre dos máquinas.
- Existen dos tipos de Socket:
 - UDP: Proporciona un servicio de Datagramas y envían paquetes discontinuos. Es un protocolo sin conexión.
 - TCP: Proporciona un servicio fiable de flujo de bytes. Garantiza la entrega de todos los paquetes enviados.
- Protocolo → Son una serie de normas para la conexión de dos equipos a través de la red.

Protocolos

- **IP:** *Internet Protocol*, protocolo de enrutamiento de bajo nivel que divide los datos en pequeños paquetes y los manda a una dirección a través de la red pero que no garantiza la llegada de los paquetes.
- **TCP:** *Protocolo de Control de la Transmisión*. De alto nivel se ocupa de unir los paquetes de forma robusta, ordenándolos y retransmitiéndolos.
- **UDP:** *Protocolo de datagramas de usuario*. Envía de forma rápida pero no garantiza la llegada de los mismos.

Conectores Reservados

- TCP/IP: reserva los 1024 primeros puertos.
- Los mas conocidos:
 - 21 → FTP.
 - 23 → Servicio de Telnet.
 - 25 → SMTP: Correo electrónico.
 - 80 → HTTP: protocolo de Internet.

Direcciones de Internet

- Cada ordenador en Internet tiene una dirección.
- Están formadas por 32 bits.
- 4 números de 0 a 255 separados por .
 - 192.168.0.1

Servicio de nombres de Dominio (DNS)

- Hace mas fácil la navegación por Internet.
- Hacen una traducción entre el nombre de un dominio y una dirección IP.
 - www.osborne.com → <http://192.9.9.1/>
- Así podemos navegar con nombres de dominio y no utilizar las direcciones IP.

Datagramas

- Un datagrama es un fragmento de paquete que es enviado con la suficiente información como para que la red pueda simplemente encaminar el fragmento hacia el ordenador receptor, de manera independiente a los fragmentos restantes. Esto puede provocar una recomposición desordenada o incompleta del paquete en el ordenador destino.
- La estructura de un datagrama es: cabecera y datos.
- Protocolos basados en datagramas: IPX, UDP, IPoAC. CL Los datagramas tienen cabida en los servicios de red no orientados a la conexión o Datagrama Agrupación lógica de información que se envía como una unidad de capa de red a través de un medio de transmisión sin establecer con anterioridad un circuito virtual.
- Los datagramas IP son las unidades principales de información de Internet. Los términos trama, mensaje, paquete y segmento también se usan para describir las agrupaciones de información lógica en las diversas capas del modelo de referencia OSI y en los diversos círculos tecnológicos.

java.net

- **Socket:** Representa un cliente Socket para enviar y recibir datos a través de conexiones TCP.
- **DatagramSocket:** Clases cliente y servidor para enviar y recibir datos a través de UDP.
- **ServletSocket:** Servidores TCP, cuando el cliente conecta, esta clase devuelve una clase Socket para enviar y recibir datos.
- **InetSocketAddress:** Representa una dirección IP junto con un número de puerto. Por ejemplo:
www.ejemplo.com:8080.

Programación Cliente

- Utiliza las clases: Socket y InetAddress.
- Creamos un objeto dirección indicando Servidor (www.ejemplo.com o 127.0.0.1) y puerto.
- ¡OJO! Capturar las excepciones la red puede estar caída.
- Crear el socket y conectar a la dirección creada.
- Si se establece la conexión podemos enviar y recibir objetos.

Esquema del Cliente

- Crear la dirección donde queremos conectar.
- Crear el socket y conectar a esta dirección. O podemos crear el socket indicando ya la máquina y el puerto.
- Habilitar un stream de entrada y otro de salida para poder escribir y leer del servidor.
- Cerrar los canales de entrada / salida y la conexión, es decir, el Socket.

Cliente en Java

// Crear la conexión con el Servidor:

Socket miCliente;

try {

 miCliente = new Socket("maquina", numeroPuerto);

} catch(IOException e) {

 System.out.println(e);

} catch (UnknownHostException e){

 System.out.println(e);

}

Cliente en Java

// Para leer del Servidor:

```
DataInputStream entrada;
```

```
try {  
    entrada = new DataInputStream(  
        miCliente.getInputStream() );  
    String texto = entrada.readUTF();
```

```
} catch( IOException e ) {  
    System.out.println( e );  
}
```

// Para escribir en el Servidor.

```
DataOutputStream salida;
```

```
try {  
    salida = new DataOutputStream(  
        miCliente.getOutputStream() );  
    salida.writeUTF("texto");
```

```
} catch( IOException e ){  
    System.out.println( e );  
}
```


Cliente en Java

// Cerrar los canales y el Socket:

```
try {  
    salida.close();  
    entrada.close();  
    miCliente.close();  
} catch( IOException e ) {  
    System.out.println( e );  
}
```

Programación Servidor

- Se utiliza la clase `ServerSocket` que monitoriza un determinado puerto y queda a la espera de una conexión con el cliente.
- Y aceptará las peticiones del cliente mediante el método `accept()`.
- Si necesitamos crear un servidor que acepte varias conexiones simultáneas tendremos que usar hilos.

Esquema del Servidor

- Crear el `ServerSocket` especificando el puerto por el que va a escuchar.
- Crear un `Socket` por el que va a aceptar a los posibles clientes.
- Crear un stream para poder leer / escribir en el cliente.
- Cerrar los canales de comunicación y la conexión.

Servidor en Java

// Crear la conexión en el servidor:

```
ServerSocket miServicio;  
try {  
    miServicio = new ServerSocket(  
        numeroPuerto );  
} catch( IOException e ) {  
    System.out.println( e );  
}
```

// Para atender a los clientes:

```
Socket socketServicio = null;  
try { socketServicio =  
    miServicio.accept();  
} catch( IOException e ) {  
    System.out.println( e );  
}
```

// Para atender múltiples peticiones:

```
while (true){  
    Socket cliente =  
        serverSocket.accept();  
    Thread hiloCliente = new Thread(new  
        UnaClaseConRunnable(cliente));  
    hiloCliente.start();  
}
```

Servidor en Java

// Para leer del cliente:

```
DataInputStream entrada;  
  
try {  
    entrada = new DataInputStream(  
        socketServicio.getInputStream  
        () );  
    String texto = entrada.readUTF();  
} catch( IOException e ){  
    System.out.println( e );  
}
```

// Para escribir en el cliente:

```
DataOutputStream salida;  
  
try {  
    salida = new DataOutputStream (   
        socketServicio.getOutputStream()  
        );  
    salida.writeUTF("texto");  
} catch( IOException e ) {  
    System.out.println( e );  
}
```

Servidor en Java

// Cerrar las conexiones:

```
try {  
    salida.close();  
    entrada.close();  
    socketServicio.close();  
    miServicio.close();  
  
} catch( IOException e ) {  
    System.out.println( e );  
}
```

Transmitir Objetos Propios

- Nuestra clase tendría que implementar el interfaz serializable.
- Con las clases `DataInputStream` y `DataOutputStream` podemos escribir datos primitivos y `String`.
- Para escribir Objetos usaremos `ObjectOutputStream` y `ObjectInputStream` con los métodos `writeObject` y `readObject`.

Transmitir Objetos Propios

- Habilitar los canales de E/S en este orden:
- Cliente:
 - // Inicializar canales de E/S con el socket.
 - `canalSalida = new ObjectOutputStream(cliente.getOutputStream());`
 - `canalEntrada = new ObjectInputStream(cliente.getInputStream());`
- Servidor:
 - `cliente = server.accept();`
 - `canalEntrada = new ObjectInputStream(cliente.getInputStream());`
 - `canalSalida = new ObjectOutputStream(cliente.getOutputStream());`

Envío de ficheros con Sockets

- **Por parte del Servidor:**

- Crear el ServerSocket.
- accept() para aceptar al cliente.
- Necesita un canal de salida para mandar datos al cliente, podemos utilizar ObjectOutputStream (si mandamos un Object) y DataOutputStream (si mandamos un tipo primitivo).
- Y para leer el fichero: FileInputStream.
- Leer y enviar todos los datos del Fichero.
- Cerrar los canales de comunicación, el fichero y los Sockets.

- **Por parte del Cliente:**

- Crear el Socket para comunicar con el Servidor.
- Necesita un canal de Entrada para poder leer del Servidor, podemos usar ObjectInputStream (si leemos un Object) y DataInputStream (si leemos un tipo primitivo).
- Y para escribir el fichero FileOutputStream.
- Leer los datos que envía el Servidor y grabarlos en el Fichero.
- Cerrar los canales de comunicación, el fichero y el Socket.

Envío de correo electrónico con Sockets

- Crear un socket cliente.
- Conectar con el servidor de correo electrónico. Servidor: servidorSMTP puerto:25.
- Habilitar el canal de entrada y de salida.
- Dialogar con el protocolo SMTP, utilizando los comandos apropiados.
- Cerrar las conexiones.

Envío correo electrónico

- Consultar comandos en: (especificaciones del protocolo)
https://www.dsi.uclm.es/personal/miguelfgraciani/mikicuri/docencia/LenguajesInternet0910/web_LI/Teoria/Protocolos%20de%20nivel%20de%20aplicacion/Material/Comandos%20del%20protocolo%20SMTP.htm
- Servidor SMTP: (desde la ventana del DOS):
 - telnet servidor_de_correo 25
 - HELO servidor_de_correo
 - MAIL FROM: <dir_email@servidor>
 - RCPT TO: <dir_email@servidor>
 - DATA
 - Cabecera.
 - Datos que quiero enviar
 - .
 - QUIT

URL

- Para trabajar con Internet. Desglosa una dirección de Internet.
- Constructor:
 - `URL(String unaURL)`
- Métodos:
 - `String getProtocol();` → Protocolo.
 - `int getPort();` → Puerto.
 - `String getHost();` → Host.
 - `String getFile();` → El nombre del fichero que forma parte de la url.
 - `String toString();`

Ejemplo

```
import java.net.*;
class URLEDemo {
    public static void main(String args[]) throws
        MalformedURLException {
        URL hp = new URL("http://www.osborne/download");
        System.out.println("Protocol: " + hp.getProtocol());
        System.out.println("Port: " + hp.getPort());
        System.out.println("Host: " + hp.getHost());
        System.out.println("File: " + hp.getFile());
    }
}
```

URLConnection

- Basada en URL, para inspeccionar las propiedades y el contenido del documento.
- Constructor:
 - No hay.
 - Capturamos a partir de la clase URL.
 - Ejemplo:
 - `URL hp = new URL("http://www.barrabes.es");`
 - `URLConnection uc = hp.openConnection();`
- Métodos:
 - `long getDate()` → La fecha
 - `String getContentType()` → Contenido de la página.
 - `long getExpiration()` → La caducidad de la página.
 - `long getLastModified()` → Última modificación.
 - `int getContentLength()` → Longitud en bytes del contenido.
 - `InputStream getInputStream()` → Un `InputStream` para leer el contenido.

Ejemplo

```
import java.net.*;
import java.io.*;
import java.util.Date;
public class UCDemo {
    public static void main(String args[]) throws Exception {
        int c;
        URL hp = new URL("http://www.barrabes.com");
        URLConnection hpCon = hp.openConnection();
        System.out.println("Date: " + new Date(hpCon.getDate()));
        System.out.println("Content-Type: " + hpCon.getContentType());
        System.out.println("Expires: " + hpCon.getExpiration());
        System.out.println("Last-Modified: " +
            new Date(hpCon.getLastModified()));
        int len = hpCon.getContentLength();
        System.out.println("Content-Length: " + len);
        if (len > 0) {
            System.out.println("=== Content ===");
            InputStream input = hpCon.getInputStream();
            int i = len;
            while (((c = input.read()) != -1) && (--i > 0)) {
                System.out.print((char) c);
            }
            input.close();
        } else {
            System.out.println("No Content Available");
        }
    }
}
```