Generación de Servicios en Red

Hemos visto cómo es posible establecer comunicaciones entre dos sistemas mediante **Sockets**. Java nos ofrece una serie de clases para trabajar con ellos de forma simple; no obstante, se hace evidente que a cuanto más bajo sea el nivel del protocolo, más complicado es trabajar con él. En general, es mucho más conveniente para un programador utilizar los protocolos de la **capa de aplicación**, dado que son más próximos al usuario.

Los protocolos más conocidos de esta capa son:

- **HTTP** y **HTTPS**: El *Protocolo de Transmisión de Hipertexto* es el más utilizado para la transmisión de documentos por internet. Es fundamental para la web, pues realiza peticiones y transfiere los recursos solicitados. Funciona sobre **TCP**. Por su parte, el *Protocolo de Transmisión de Hipertexto Seguro* es una versión segura del mismo, ya que cifra la información intercambiada entre Cliente y Servidor. **[Puertos 80 / 443]**

- **FTP**: El *Protocolo de Transferencia de Ficheros* permite la transferencia de ficheros de todo tipo entre dispositivos. Funciona sobre **TCP**. Tiene su solera ya, y utiliza una arquitectura cliente / servidor. **[Puerto 21]**

- **SMTP**: Otro venerable protocolo, el *Protocolo Simple de Transferencia de Correo* se emplea para enviar correos electrónicos. Utiliza la especificación MIME (*Multipurpose Internet Mail Extensions*) para poder intercambiar todo tipo de archivos por internet. **[Puerto 25]**

- **IMAP**: El *Protocolo de Acceso a Mensajes de Internet* se utiliza para la recepción de correos electrónicos. IMAP almacena los correos en un servidor, los cuales se pueden leer desde diferentes dispositivos. **[Puertos 194 y 993]**

- **POP3**: Tan viejo como la propia Internet, el *Protocolo de Oficina de Correo*, su funcionalidad es similar a IMAP. POP3 descarga los correos y los elimina del servidor, por lo que una vez consultados, no podrán ser accedidos desde otros dispositivos. **[Puertos 110 y 995]**

- **DNS**: El *Sistema de Nombres de Dominio* permite asociar el nombre de un dispositivo a una IP. Este protocolo permite que una URL o una dirección de mail se vincule con una dirección IP en la que se encuentre el servicio relacionado. **[Puerto 53]**

- **Telnet**: Este protocolo permite acceder a través de un terminal (de texto) a un equipo remoto. No cifra la información enviada, y se le considera inseguro. **[Puerto 23]**

- **SSH**: *Secure Shell* es un protocolo cuyo objetivo es similar a Telnet, aunque al cifrar la comunicación, se le considera seguro. **[Puerto 22]**

- **LDAP**: el *Protocolo Ligero de Acceso a Datos* proporciona una estructura jerárquica, ordenada y distribuida de información, así como las herramientas de acceso a la misma. Se suele usar para almacenar credenciales y permisos. **[Puerto 389]**

- **NFS**: El *Sistema de Archivos en Red* permite distribuir ficheros en una red, y acceder a ellos desde cualquier nodo. **[Puerto 2049]**

- **SNMP**: El *Protocolo Simple de Administración de Red* permite intercambiar información entre los dispositivos de red que forman parte de su infraestructura (router, switch, etc.). Obviamente, se usa para administrar dicha red. **[Puerto 161]**

- **DHCP**: Para la asignación dinámica de IP y configuración a los diferentes equipos de una red se usa el *Protocolo de Configuración Dinámica de Host*. Cuando las IP de una red no son fijas, este protocolo se las asigna a un equipo en base a una lista de IP permitidas. **[Puerto 67]**

- **SMB y CIFS**: Estos protocolos permiten compartir recursos por red, desde ficheros a impresoras. El *Server Message Block* y la *Common Internet File System* están muy relacionados entre sí, dado que CIFS es una implementación de SMB creada por Microsoft. En la actualidad se recomienda usar SMB2 y 3. **[Puertos 139 / 445]**

Clases y Librerías Java para servicios en Red

Prácticamente todos los lenguajes de programación modernos disponen de librerías para trabajar en red. En ausencia de dichas librerías, la programación se realizaría a un nivel más bajo, requiriendo el uso de sockets, transferencia de bytes y delegando muchas veces en el programador la tarea de gestionar los problemas que surjan.

java.net.URL

Representa una URL (Universal Resource Locator), un recurso de red. Si no se le proporciona una dirección bien formada, genera una *MalformedUrlException*. Su método más relevante es **openConnection ()**, que retorna una conexión **URLConnection** con el equipo destino.

```
URL url = new URL ("www.google.com");
```

java.net.URLConnection

Representa un enlace de comunicación entre la aplicación y la URL. Las instancias de esta clase permiten leer y escribir en el recurso referenciado por la URL. Su subclase más relevante es *HttpURLConnection*.

Métodos más relevantes:

getByName	Retorna la IP en base a un nombre
getLocalHost	Retorna la IP del host local
getHostAddress	Retorna la IP del host
getHostName	Retorna el alias del host
getAddress	Retorna la IP del host como array de bytes
getCanonicalHostName	Retorna el nombre del host
getInputStream	Retorna un Stream de lectura
getOutputStream	Retorna un Stream de escritura
setRequestProperty	Asigna el valor de una propiedad

Un ejemplo que lee de un recurso de la web:

```
try {

    URL url = new URL ("www.google.com");
    URLConnection myURLConnection = url.openConnection();
    InputStream inputStream = myURLConnection.getInputStream();

    int byteLeido; // Lee bytes como int de 0 a 255
    while ((byteLeido = inputStream.read()) != 1) {
        System.out.println ((char) byteLeido);
    } // read () se bloquea hasta que le llegue informacion

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

java.net.HttpURLConnection

Permite gestionar conexiones HTTP. Sus métodos más importantes son **disconnect ()** y **getResponseCode ()**, que permite obtener el código de estado del protocolo HTTP.

java.net.http.HttpClient

Esta clase permite generar peticiones HTTP y obtener respuestas. Las instancias deben de ser creadas a través de un *builder* u otro objeto de instanciación.

Métodos más relevantes de HttpClient:

newBuilder	Crea un builder
send	Envía una petición HTTP y devuelve un HttpResponse.

Métodos más relevantes de Builder:

build	Retorna un HttpClient
followsRedirect	Mecanismo para controlar las redirecciones del servidor
version	Permite especificar la versión del protocolo HTTP

java.net.http.HttpRequest

Esta clase representa una petición HTTP. Su método más relevante es **newBuilder ()**. El interfaz HttpRequest.Builder proporciona otros métodos:

build	Retorna un HttpClient
DELETE	Asigna el método DELETE al builder
GET	Asigna el método GET al builder
header	Añade un par parámetro-valor a la petición
headers	Añade varios pares parámetro-valor a la petición
POST	Asigna el método POST al builder
PUT	Asigna el método PUT al builder
setHeader	Asigna un par parámetro-valor a la petición
timeout	Permite añadir un timeout a la petición
uri	Asigna una URI
version	Permite especificar la versión del protocolo HTTP

java.net.http.HttpRequest

Representa la respuesta de una petición HTTP. Estas instancias las provee el método send () de HttpClient. Su método más importante es **statusCode ()**, que retorna el código de estado de la petición HTTP.

Dispone a demás de diversos métodos para la obtención de instancias de **BodyHandler**.

Métodos más relevantes de BodyHandler:

ofByteArray	Devuelve un BodyHandler <byte []>
ofFilefollowsRedirect	Devuelve un BodyHandler <Path>
ofInputStream	Devuelve un BodyHandler <InputStream>
ofString	Devuelve un BodyHandler <String>

Por ejemplo, este código ejecuta una petición y proporciona un String como respuesta:

```
HttpResponse<String> response = httpClient.send(request,
        HttpResponse.BodyHandlers.ofString());
```

Este otro almacena en el fichero indicado en path el contenido del cuerpo de la respuesta:

```
HttpResponse<Path> response = httpClient.send(request,
        HttpResponse.BodyHandlers.ofFile(Path.of(path)));
```