Comunicación en Red

La programación en red siempre ha sido dificultosa, el programador debía de conocer la mayoría de los detalles de la red, incluyendo el hardware utilizado, los distintos niveles en que se divide la capa de red, las librerías necesarias para programar en cada capa, etc.

Pero, la idea simplemente consiste en obtener información desde otra máquina, aportada por otra aplicación software. Por lo tanto, de cierto modo se puede reducir al mero hecho de leer y escribir archivos, con ciertas salvedades. El sistema de Entrada/Salida de Unix sigue el paradigma que normalmente se designa como Abrir-Leer-Escribir-Cerrar. Antes de que un proceso de usuario pueda realizar operaciones de entrada/salida, debe hacer una llamada a Abrir (open) para indicar, y obtener los permisos del fichero o dispositivo que se desea utilizar.

Una vez que el fichero o dispositivo se encuentra abierto, el proceso de usuario realiza una o varias llamadas a Leer (read) y Escribir (write), para la lectura y escritura de los datos.

El proceso de lectura toma los datos desde el objeto y los transfiere al proceso de usuario, mientras que el de escritura los transfiere desde el proceso de usuario al objeto. Una vez concluido el intercambio de información, el proceso de usuario llamará a Cerrar (close) para informar al sistema operativo que ha finalizado la utilización del fichero o dispositivo.

En Unix, un proceso tiene un conjunto de descriptores de entrada/salida desde donde leer y por donde escribir. Estos descriptores pueden estar referidos a ficheros, dispositivos, o canales de comunicaciones *sockets*.

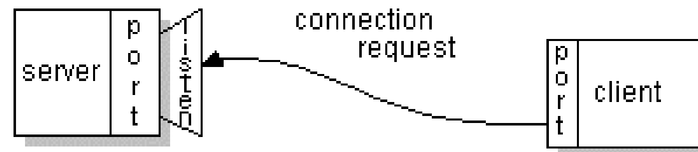
What Is a Socket?

A socket is one end-point of a two-way communication link between two programs running on the network. A socket is bound to a port number so that the TCP layer can identify the application that data is destined to be sent. Socket classes are used to represent the connection between a client program and a server program. The java.net package provides two classes--Socket and ServerSocket--that implement the client side of the connection and the server side of the connection, respectively.

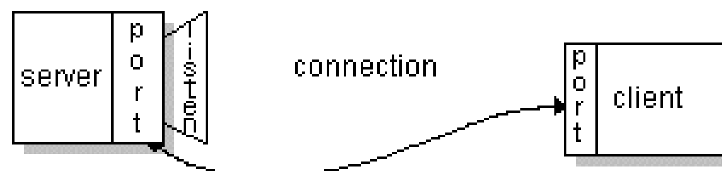
Normally, **a server** runs on a specific computer and has a socket that is bound to a specific port number. The server just waits, listening to the socket for a client to make a connection request.

On the **client-side**: The client knows the hostname of the machine on which the server is running and the port number on which the server is listening. To make a connection

request, the client tries to rendezvous with the server on the server's machine and port. The client also needs to identify itself to the server so it binds to a local port number that it will use during this connection. This is usually assigned by the system.



If everything goes well, the server accepts the connection. Upon acceptance, the server gets a new socket bound to the same local port and also has its remote endpoint set to the address and port of the client. It needs a new socket so that it can continue to listen to the original socket for connection requests while tending to the needs of the connected client.



On the client side, if the connection is accepted, a socket is successfully created and the client can use the socket to communicate with the server. The client and server can now communicate by writing to or reading from their sockets.

Endpoint

An endpoint is a combination of an IP address and a port number. Every TCP connection can be uniquely identified by its two endpoints. That way you can have multiple connections between your host and the server.

The java.net package in the Java platform provides a class, Socket, that implements one side of a two-way connection between your Java program and another program on the network. The Socket class sits on top of a platform-dependent implementation, hiding the details of any particular system from your Java program. By using the java.net.Socket class instead of relying on native code, your Java programs can communicate over the network in a platform-independent fashion.

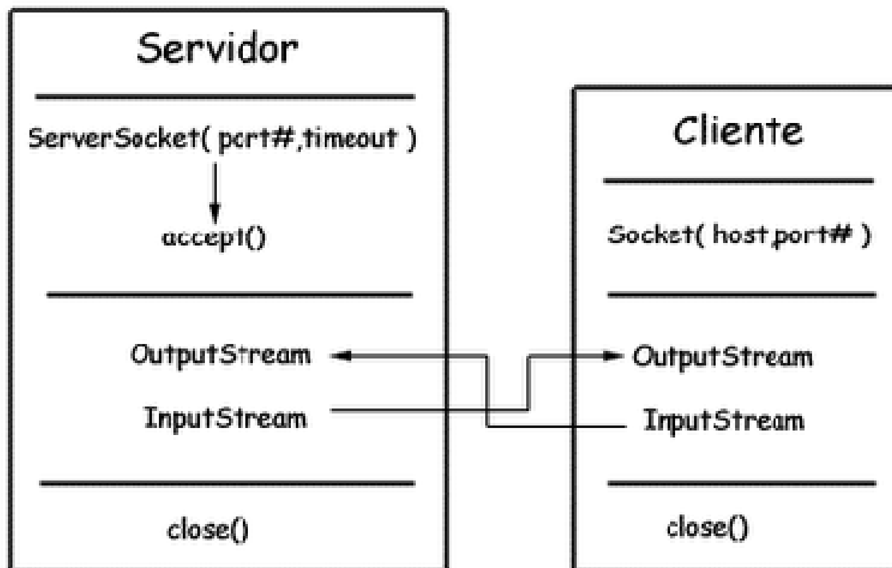
Additionally, java.net includes the ServerSocket class, which implements a socket that servers can use to listen for and accept connections to clients. This lesson shows you how to use the Socket and ServerSocket classes.

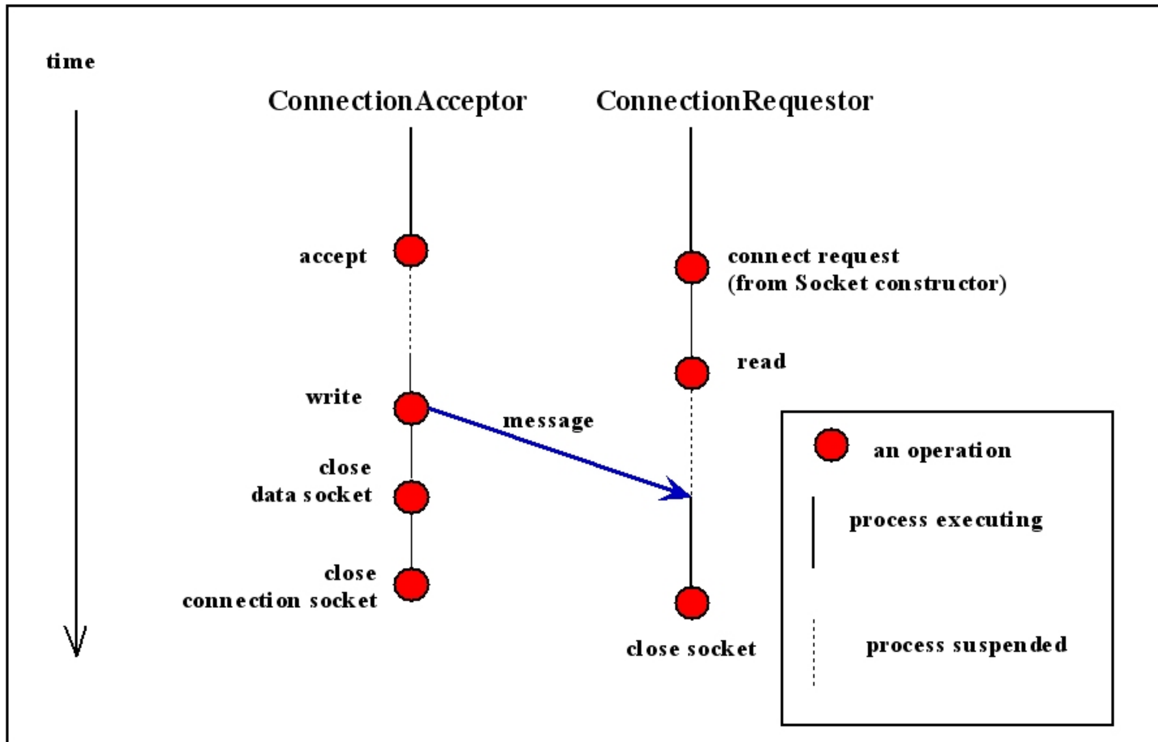
If you are trying to connect to the Web, the URL class and related classes (URLConnection, URLEncoder) are probably more appropriate than the socket classes. In fact, URLs are a relatively high-level connection to the Web and use sockets as part of the underlying implementation.

Fundamentos

Los sockets son un sistema de comunicación entre procesos de diferentes máquinas de una red. Más exactamente, un socket es un punto de comunicación por el cual un proceso puede emitir o recibir información. Fueron popularizados por Berckley Software Distribution, de la universidad norteamericana de Berkley. Los sockets han de ser capaces de utilizar el protocolo de streams TCP (Transfer Contro Protocol) y el de datagramas UDP (User Datagram Protocol).

Utilizan una serie de primitivas para establecer el punto de comunicación, para conectarse a una máquina remota en un determinado puerto que esté disponible, para escuchar en él, para leer o escribir y publicar información en él, y finalmente para desconectarse. Con todas primitivas se puede crear un sistema de diálogo muy completo.





Notas:

- Ambos programas (servidor y cliente) no necesitan estar programados en Java, es posible programarlos en lenguajes de programación diferentes, o inclusive programar un servidor en java y utilizar un cliente ya existente que pueda conectarse a un puerto especificado.
- El cliente debe de conocer tanto el puerto a utilizar como la IP o dominio del servidor, mientras el servidor solo debe conocer el puerto de conexión

CLASES PARA LAS COMUNICACIONES DE RED EN JAVA: java.net

En las aplicaciones en red es muy común el paradigma cliente-servidor. El servidor es el que espera las conexiones del cliente (en un lugar claramente definido) y el cliente es el que lanza las peticiones a la maquina donde se está ejecutando el servidor, y al lugar donde está esperando el servidor (el puerto(s) específico que atiende). Una vez establecida la conexión, ésta es tratada como un *stream* (flujo) típico de entrada/salida.

Cuando se escriben programas Java que se comunican a través de la red, se está programando en la capa de aplicación. Típicamente, no se necesita trabajar con las capas TCP y UDP, en su lugar se puede utilizar las clases del paquete **java.net**. Estas clases proporcionan comunicación de red independiente del sistema.

A través de las clases del paquete **java.net**, los programas Java pueden utilizar TCP o UDP para comunicarse a través de Internet. Las clases **URL**, **URLConnection**, **Socket**, y **SocketServer** utilizan TCP para comunicarse a través de la Red. Las clases **DatagramPacket** y **DatagramServer** utilizan UDP.

TCP proporciona un canal de comunicación fiable punto a punto, lo que utilizan para comunicarse las aplicaciones cliente-servidor en Internet. Las clases Socket y ServerSocket del paquete java.net proporcionan un canal de comunicación independiente del sistema utilizando TCP, cada una de las cuales implementa el lado del cliente y el servidor respectivamente.

What Is a URL?

URL is an acronym for *Uniform Resource Locator* and is a reference (an address) to a resource on the Internet.

If you've been surfing the Web, you have undoubtedly heard the term URL and have used URLs to access HTML pages from the Web. It's often easiest, although not entirely accurate, to think of a URL as the name of a file on the World Wide Web because most URLs refer to a file on some machine on the network.
However, remember that URLs also can point to other resources on the network, such as database queries and command output.

A URL has two main components:

- Protocol identifier: For the URL http://example.com, the protocol identifier is http.
- Resource name: For the URL http://example.com, the resource name is example.com.

Note that the protocol identifier and the resource name are separated by a colon and two forward slashes. The protocol identifier indicates the name of the protocol to be used to fetch the resource. The example uses the Hypertext Transfer Protocol (HTTP), which is typically used to serve up hypertext documents. HTTP is just one of many different protocols used to access different types of resources on the net. Other protocols include File Transfer Protocol (FTP), Gopher, File, and News.

The resource name is the complete address to the resource. The format of the resource name depends entirely on the protocol used, but for many protocols, including HTTP, the resource name contains one or more of the following components:

Host Name

The name of the machine on which the resource lives.

Filename

The pathname to the file on the machine.

Port Number

The port number to which to connect (typically optional).

Reference

A reference to a named anchor within a resource that usually identifies a specific location within a file (typically optional).

For many protocols, the host name and the filename are required, while the port number and reference are optional. For example, the resource name for an HTTP URL must specify a server on the network (Host Name) and the path to the document on that machine (Filename); it also can specify a port number and a reference.

Fuentes:

http://www.infor.uva.es/~fdiaz/sd/doc/java.net.pdf
http://docs.oracle.com/javase/tutorial/networking/urls/index.html
http://docs.oracle.com/javase/tutorial/networking/sockets/definition.html
http://www.dlsi.ua.es/asignaturas/sid/JSockets.pdf