Section 20: Java Networking Programming

15 July 2021 19:23

Section 20: Java Networking Programming

Network :- A network is a system of computers connected together so they can share resources and communicate with each other.

Networking :- Networking refers to how the connected computers communicate.

Host : When discussing networking, a machine is usually referred to as a **host**.

Computer on a network (which includes the internet) communicate with each other using a **transport protocol**.

Each application that needs data from the network is assigned a **port** (this includes clients connecting to a server that's on the same machine). When data arrives, the port number is used to router the data to the application that's waiting for it.

IP stands for **Internet Protocol**. WE heard the term **TCP/IP**. This refers to using the TCP protocol with IP addresses, which doesn't necessarily mean the host is connected to the internet. Two applications running on the same host can use TCP/IP to communicate with each other. When the client and server are on the same host, usually the IP address **127.0.0.1**, which is referred to as **localhost**, is used to identify the host.

TCP, which stands for **Transmission Control Protocol**, establishes a two-way connection between hosts, and this connection is reliable in the sense that the two hosts talk to each other. When used with Internet addresses, you get TCP/IP, which uses the client/server model.

When communicating using TCP/IP, the sequence of events is as follows :-

1) The client open a connection to the server
2) The clients sends a request to the server
3) The server sends a response to the client
4) The client closes the connection to the server.

Steps 2 and 3 may be repeated multiple times before the connection is closed.

When using low-level networking API, we will use **Sockets** to establish connections, send request, and receive responses. A socket is one end-point of two-way connection. The client will have a socket, and the server will have a socket.
When we have multiple clients connecting to the same server, we will use the same port number, but each client will have its own socket. We will use the **Socket Class** for the client socket, and the **ServerSocket class** for the server's socket.

Server Code :

```
try(ServerSocket serverSocket = new ServerSocket(5000))
{
        Socket socket = serverSocket.accept();
        System.out.println("Client Connected");

....
}
```

Client Code :

```
try (Socket socket = new Socket("localhost", 5000))
{
        ……
}
```

When using TCP, some handshaking has to take place between the server and the client. So the client has to connect to the server, and the server has to accept that connection. So the client sends a request and the server sends a response. It's a two way connection, and there's tight coupling between the client and the server. Now the TCP protocol also performs error checking and will resend message that don't make it to the server. It's reliable.

When using UDP, there's no handshaking at all, and the destination host, which may or may not be a server, doesn't actually send any responses to the message sender. So we use it when we don't need a reliable connection or a two way connection or when speed is essential.
UDP uses datagrams, and a datagram is a self-contained message, and it's not guaranteed to arrive at its destination. UDP is often used for time-sensitive communication and when losing he odd message or packet here or there won't matter.

URIs, URLs and URNs :-
https://www.w3.org/TR/uri-clarification/

When working with the java.net package, a **URI** is an identifier that might not provide enough information to access the resource it identifies. A **URL** is an identifier that includes information about how to access the resource it identifies.
URL can specify a relative path, but a **URL** has to be an absolute path, because when we use the **URL**, it has to contain enough information to locate and access the resource it identifies.

A **Scheme** is the part of a URI or URL that appears before colon.
For example, "http", "file" and "ftp" are all **schemes**.

When working with **Low-level API**, we used the following classes: **Socket**, **ServerSocket**, and **DatagramSocket**.
When working with the **High-Level API**, we will use the following classes: **URI**, **URL**, **URLConnection**, and **HttpURLConnection**.

A **URI** can contain **Nine Components :-**
1) Scheme
2) Scheme-specific part
3) Authority
4) User-info
5) Host
6) Port
7) Path
8) Query
9) Fragment

URI - Uniform Resource Identifier :-
https://en.wikipedia.org/wiki/Uniform_Resource_Identifier
Scheme:[//[user[:password]@]host[:port]][/path][?query][#fragment]

A URIs that specify a scheme are called **Absolute URIs**.
A URI doesn't specify the scheme, it's called a **Relative URI**.

The URI doesn't have to be valid to work with it, It only has to be valid when you wanna convert it an absolute location (by converting to url using : URL url = uri.toURL();)

Methods : GET, POST, HEAD, OPTIONS, PUT, DELETE, TRACE, CONNECT :-
https://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html

List of HTTP header fields :-
https://en.wikipedia.org/wiki/List_of_HTTP_header_fields

Download Apache HttpClient :-
http://hc.apache.org/downloads.cgi