Networking

Java Networking is a concept of connecting two or more computing devices together so that we can share resources.

When computing devices such as laptops, desktops, servers, smartphones, and tablets and an eternally-expanding arrangement of IoT gadgets such as cameras, door locks, doorbells, refrigerators, audio/visual systems, thermostats, and various sensors are sharing information and data with each other is known as networking.

Java Socket Programming provides facility to share data between different computing devices.

Advantage of Java Networking : - sharing resources - centralized software management

The java.net Package Supports 2 Protocols: - **TCP:** Transmission Control Protocol provides reliable communication between the sender and receiver. TCP is used along with the Internet Protocol referred as TCP/IP. - **UDP:** User Datagram Protocol provides a connection-less protocol service by allowing packet of data to be transferred along two or more nodes

Java Networking Terminology

Some of the widely used java networking terminologies are as follows:

1. IP Address - IP address is a unique number assigned to a node of a network e.g. 192.168.0.1. It is composed of octets that range from 0 to 255. It is a logical address that can be changed.

2. Protocol - A protocol is a set of rules basically that is followed for communication. For example: TCP, FTP, Telnet, SMTP, POP, etc.

3. Port Number - The port number is used to uniquely identify different applications. It acts as a communication endpoint between applications. The port number is associated with the IP address for communication between two applications.

4. MAC Address - MAC (Media Access Control) Address is a unique identifier of NIC (Network Interface Controller). A network node can have multiple NIC but each with unique MAC. The main difference between MAC and IP address is that, MAC Address is used to ensure the physical address of computer. It uniquely identifies the devices on a network. While IP address are used to uniquely identifies the connection of network with that device take part in a network.

5. Socket - A socket is one endpoint of a two-way communication connection between the two applications running on the network. The socket mechanism presents a method of inter-process communication (IPC) by setting named contact points between which the communication occurs. A socket is tied to a port number so that the TCP layer can recognize the application to which the data is intended to be sent.

6. Connection-oriented protocol - In connection-oriented protocol, acknowledgement is sent by the receiver. So, it is reliable but slow. The example of connection-oriented protocol is TCP.

7. Connection-less protocol - In connection-less protocol, acknowledgement is not sent by the receiver. So, it is not reliable but fast. The example of connection-less protocol is UDP.

Java Networking Classes

The java.net package of the Java programming language includes various classes that provide an easy-to-use means to access network resources. The classes covered in the java.net package are given as follows –

Authenticator	CacheRequest	CacheResponse	ContentHandler	CookieHandler
CookieManager	DatagramPacket	DatagramSocket	DatagramSocketImpl	InterfaceAddress
JarURLConnection	MulticastSocket	InetSocketAddress	InetAddress	Inet4Address
Inet6Address	IDN	HttpURLConnection	HttpCookie	NetPermission
NetworkInterface	PasswordAuthentication	Proxy	ProxySelector	ResponseCache
SecureCacheResponse	ServerSocket	Socket	SocketAddress	SocketImpl
SocketPermission	StandardSocketOptions	URI	URL	URLClassLoader
URLConnection	URLDecoder	URLEncoder	URLStreamHandler	

1. CacheRequest - The CacheRequest class is used in java whenever there is a need to store resources in ResponseCache. The objects of this class provide an edge for the OutputStream object to store resource data into the cache.

2. CookieHandler - The CookieHandler class is used in Java to implement a callback mechanism for securing up an HTTP state management policy implementation inside the HTTP protocol handler. The HTTP state management mechanism specifies the mechanism of how to make HTTP requests and responses.

3. CookieManager - The CookieManager class is used to provide a precise implementation of CookieHandler. This class separates the storage of cookies from the policy surrounding accepting and rejecting cookies. A CookieManager comprises a CookieStore and a CookiePolicy.

4. DatagramPacket - The DatagramPacket class is used to provide a facility for the connectionless transfer of messages from one system to another. This class provides tools for the production of datagram packets for connectionless transmission applying the datagram socket class.

5. InetAddress - The InetAddress class is used to provide methods to get the IP address of any hostname. An IP address is expressed by a 32-bit or 128-bit unsigned number. InetAddress can handle both IPv4 and IPv6 addresses.

6. Server Socket - The ServerSocket class is used for implementing system-independent implementation of the server-side of a client/server Socket Connection. The constructor for ServerSocket class throws an exception if it can't listen on the specified port. For example – it will throw an exception if the port is already being used.

7. Socket - The Socket class is used to create socket objects that help the users in implementing all fundamental socket operations. The users can implement various networking actions such as sending, reading data, and closing connections. Each Socket object built using java.net.Socket class has been connected exactly with 1 remote host; for connecting to another host, a user must create a new socket object.

8. DatagramSocket - The DatagramSocket class is a network socket that provides a connection-less point for sending and receiving packets. Every packet sent from a datagram socket is individually routed and delivered. It can further be practiced for transmitting and accepting broadcast information. Datagram Sockets is Java's mechanism for providing network communication via UDP instead of TCP.

9. Proxy - A proxy is a changeless object and a kind of tool or method or program or system, which serves to preserve the data of its users and computers. It behaves like a wall between computers and internet users. A Proxy Object represents the Proxy settings to be applied with a connection.

10. URL - The URL class in Java is the entry point to any available sources on the internet. A Class URL describes a Uniform Resource Locator, which is a signal to a "resource" on the World Wide Web. A source can denote a simple file or directory, or it can indicate a more difficult object, such as a query to a database or a search engine.

11. URLConnection - The URLConnection class in Java is an abstract class describing a connection of a resource as defined by a similar URL. The URLConnection class is used for assisting two distinct yet interrelated purposes. Firstly it provides control on interaction with a server(especially an HTTP server) than a URL class. Furthermore, with a URLConnection, a user can verify the header transferred by the server and can react consequently. A user can also configure header fields used in client requests using URLConnection.

Java Networking Interfaces

The java.net package includes various interfaces also that provide an easy-to-use means to access network resources. The interfaces included in the java.net package are as follows –

1. CookiePolicy - The CookiePolicy interface in the java.net package provides the classes for implementing various networking applications. It decides which cookies should be accepted and which should be rejected. In CookiePolicy, there are three pre-defined policy implementations, namely ACCEPT_ALL, ACCEPT_NONE, and ACCEPT_ORIGINAL_SERVER.

2. CookieStore - A CookieStore is an interface that describes a storage space for cookies. CookieManager combines the cookies to the CookieStore for each HTTP response and recovers cookies from the CookieStore for each HTTP request.

3. FileNameMap - The FileNameMap interface is an uncomplicated interface that implements a tool to outline a file name and a MIME type string. FileNameMap charges a filename map (known as a mimetable) from a data file.

4. SocketOption - The SocketOption interface helps the users to control the behavior of sockets. Often, it is essential to develop necessary features in Sockets. SocketOptions allows the user to set various standard options.

5. SocketImplFactory - The SocketImplFactory interface defines a factory for SocketImpl instances. It is used by the socket class to create socket implementations that implement various policies.

6. ProtocolFamily - This interface represents a family of communication protocols. The ProtocolFamily interface contains a method known as name(), which returns the name of the protocol family.

Java Socket Programming

Java Socket programming is used for communication between the applications running on different JRE. Java Socket programming can be connection-oriented or connection-less.

Socket and ServerSocket classes are used for connection-oriented socket programming and DatagramSocket and DatagramPacket classes are used for connection-less socket programming.

The client in socket programming must know two information: 1. IP Address of Server 2. Port number.

Socket Class :

A socket is simply an endpoint for communications between the machines. The Socket class can be used to create a socket.

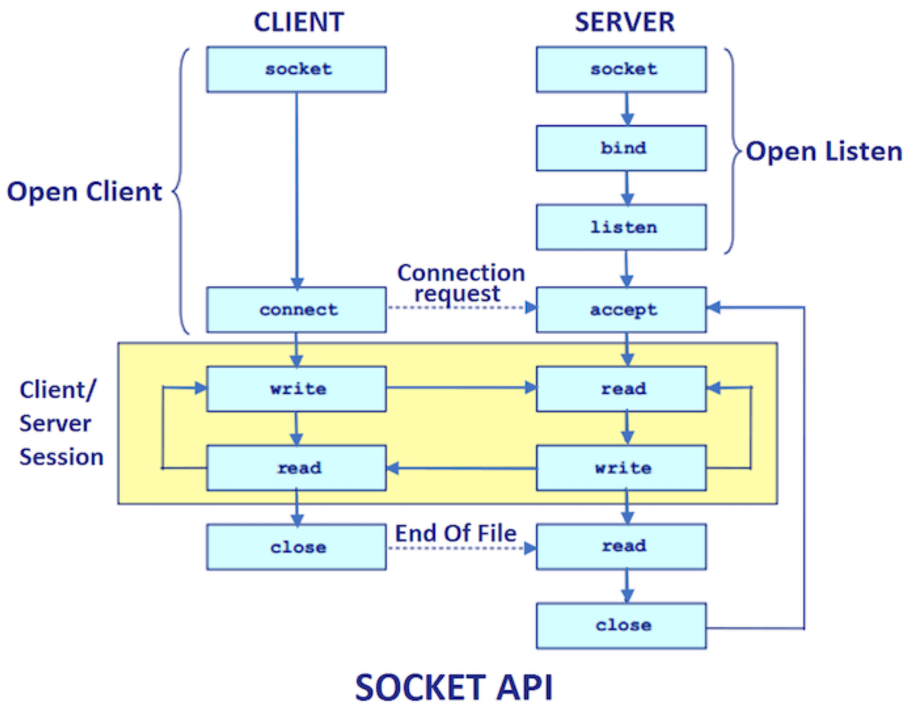

Figure 1: socket-programming

Important methods : |Method|Description| |-|-| |void connect(SocketAddress host, int timeout)|connects the socket to the particularized host| |int getPort()|returns the port to which the socket is pinned on the remote machine| |InetAddress getInetAddress()|returns the location of the other computer to which the socket is connected| |int getLocalPort()|returns the port to which the socket is joined on the local machine| |SocketAddress getRemoteSocketAddress()|returns the location of the remote socket| |InputStream getInputStream()|returns the input stream of the socket| |OutputStream getOutputStream()|returns the output stream of the socket| |synchronized void close()|closes the socket|

ServerSocket Class :

The ServerSocket class can be used to create a server socket. This object is used to establish communication with the clients.

Important methods : |Method|Description| |-|-| |int getLocalPort()|returns the port that the server socket is monitoring on| |void setSoTimeout(int timeout)|sets the time-out in which the server socket pauses for a client during the accept() method| |Socket accept()|waits for an incoming client; returns the socket and establish a connection between server and client| |void bind(SocketAddress host, int backlog)|bind the socket to the particularized server and port in the object of SocketAddress| |synchronized void close()|closes the server socket|

Example of Java Socket Programming

Creating Server :

```
ServerSocket ss = new ServerSocket(6666);
Socket s = ss.accept(); //establishes connection and waits for the client
```

Creating Client :

```
Socket s = new Socket("localhost",6666);
```

Simple Socket example where client sends a text; the server receives and prints it.

SERVER

```
ServerSocket ss = new ServerSocket(6666);
Socket s=ss.accept(); //establishes connection
DataInputStream dis = new DataInputStream(s.getInputStream());
String  str = (String)dis.readUTF();
System.out.println("message= "+str);
ss.close();
```

CLIENT

```
Socket s = new Socket("localhost",6666);
DataOutputStream dout = new DataOutputStream(s.getOutputStream());
dout.writeUTF("Hello Server");
dout.flush();
dout.close();
s.close();
```

Open two command prompts and execute each program After running the client application, a message will be displayed on the server console.

More complex Socket example where

- First, client will write to the server; server will receive and print.
- Then, server will write to the client; client will receive and print.

SERVER

```
ServerSocket ss = new ServerSocket(3333);
Socket s = ss.accept();
DataInputStream din = new DataInputStream(s.getInputStream());
DataOutputStream dout = new DataOutputStream(s.getOutputStream());
BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

String str = "", str2 = "";
while (!str.equals("stop")) {
    str = din.readUTF();
    System.out.println("client says: " + str);
    str2 = br.readLine();
    dout.writeUTF(str2);
    dout.flush();
}
din.close();
s.close();
ss.close();
```

CLIENT

```
Socket s = new Socket("localhost", 3333);
DataInputStream din = new DataInputStream(s.getInputStream());
DataOutputStream dout = new DataOutputStream(s.getOutputStream());
BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

String str = "", str2 = "";
while (!str.equals("stop")) {
    str = br.readLine();
    dout.writeUTF(str);
    dout.flush();
    str2 = din.readUTF();
    System.out.println("Server says: " + str2);
}

dout.close();
s.close();
```