UNIT-5 [Around 5 marks]

Network Programming

The term network programming refers to writing programs that execute accross multiple devices (computers), in which the devices are all connected to each other using a network. It is also referred as socket programming because we use the concept of socket to write programs that can communicate in the network. The java.net package provides support for the two common network protocols TCP & UDP.

⊛ Transmission Control Protocol (TCP):

TCP stands for Transmission Control Protocol. It is a transport layer protocol that ease the transmission of packets from source to destination. This protocol is used with an IP protocol, so together, they are referred to as a TCP/IP.

The main functionality of the TCP is to take the data from application layer. Then it divides the data into a several packets, provides numbering to these packets, and finally transmits these packets to the destination. The TCP, on the other side, will reassamble the packets and transmits them to the application layer. As we know that TCP is a connection-oriented protocol, so the connection will remain established until the communication is not completed between the sender and the receiver.

Features of TCP Protocol:

→ Transport Layer Protocol
→ Reliable
→ Order of data is maintained
→ Connection-oriented
→ Full duplex
→ Stream-oriented

⊛ User Datagram Protocol (UDP):

UDP is a protocol that sends independent packets of data, called datagrams, from one computer to another with no guarantees about arrival. UDP is not connection based like TCP.

⊛TCP vs. UDP: [Imp]

→ TCP is a connection-oriented protocol, whereas UDP is a connectionless protocol.
→ The speed for TCP is slower while the speed of UDP is faster.
→ TCP uses handshake protocol while UDP uses no handshake protocol.
→ TCP does error checking and also makes error recovery, on the other hand, UDP performs error checking, but it discards packets having error.
→ TCP has acknowledgement segments, but UDP does not have.
→ TCP is heavy-weight, and UDP is lightweight.

⊛When to use UDP and TCP?

→ TCP is an ideal choice, when most of the overhead is in the connection. Use TCP sockets when both client and server independently send packets. Example: Online Poker. Use TCP when occasional delay is acceptable.
→ UDP is an ideal to use with multimedia like VoIP. We should use UDP if both client and server may separately send packets. Example: Multiplayer games. Use UDP when occasional delay is not acceptable.

⊛TCP Port:

TCP port is a unique number assigned to different applications. For example, we have opened the email and game applications on our computer; In order to do these tasks, different unique numbers are assigned to email and game. TCP and UDP protocols mainly use port numbers.

A port number is a unique identifier used with an IP address. A port is a 16-bit unsigned integer, and the total number of ports available in the TCP/IP model is 65,535 ports. Therefore the range of port numbers is 0 to 65535. In case of TCP, the zero-port number is reserved and cannot be used whereas, in UDP, the zero-port number is ~~unavailable.~~ not available. IANA (Internet Assigned Numbers Authority) is a standard body that assigns the port numbers.

Example of port number:

192.168.1.100:7

In this case, 192.168.1.100 is an IP address and 7 is port number.

⊛ Why do we require port numbers?

A single client can have multiple connections with the same server or multiple servers. The client may be running multiple applications at the same time. When the client tries to access some service, then the IP address is not sufficient to access the service. To access the service from a server, the port number is required. So, the transport layer plays a major role in providing multiple communication between these applications by assigning a port number to the applications.

⊛ IP Address Network Classes in JDK:

i) Socket Class: The Java Socket class is used to create sockets when we use TCP for communication.

ii) Server Socket Class: It is used to create sockets for servers when TCP is used for communication.

iii) IntAddress Class: It represents an IP address.

iv) Datagram Socket Class: It is used to create sockets for servers when UDP is used for communication.

v) Datagram Packet Class: This class is used to create datagram packets that are exchanged between UDP clients and UDP servers.

vi) URL Class: It represents an URL(Uniform Resource Locator). It points to resource on World Wide Web.

vii) URL Connection Class: It represents a communication link between the URL and the application.

⊛ What is Socket? [Imp]

A socket is one endpoint 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 data that is to be sent to destination. Sockets provide communication mechanism between two programs using TCP.

⊛ Socket Programming using TCP:

⊛ How can you write java programs that communicate with each other using TCP Sockets? [Imp]

Solution:

Steps for Writing Client Program:

→ Open a socket
→ Open an input stream and output stream to the socket
→ Read from and write to the sockets stream.
→ Close the socket.

Steps for Writing Server Program:

→ Create the Object of ServerSocket class.
→ Accept the connection from the client.
→ Get input and output stream of socket.
→ Read/Write Data to the socket.
→ Close Streams and Socket.

last three points are same to above last three points of client program only close streams added.

Example (Client Program):

```
import java.net.*;
import java.io.*;
import java.util.*;
public class MsgClient{
    public static void main(String args[]) throws IOException{
        Socket cs=new Socket("localhost",1254);

        Scanner ins=new Scanner(cs.getInputStream());
        PrintWriter outs=new PrintWriter(cs.getOutputStream(),true);

        outs.println("Hello Server");
        String s=ins.nextLine();
        System.out.println("From Server:"+s);

        ins.close();
        outs.close();
        cs.close();
    }
}
```

1254 is port number to which connection made

Example:(Server Program):

```
import java.net.*;
import java.io.*;
import java.util.*;
class MsgServer{
    public static void main(String args[]) throws IOException{
        ServerSocket ss = new ServerSocket(1254);
        Socket cs=ss.accept();
        Scanner ins=new Scanner(cs.getInputStream());
        PrintWriter outs=new PrintWriter(cs.getOutputStream(),true);
        String s = ins.nextLine();
        System.out.println("From Client:"+s);
        outs.println("Hello Client");
        outs.close();
        ins.close();
        cs.close();
        ss.close();
    }
}
```

Same as Client program only these two start lines are only different and ss.close() added at last all other is same

⊛Socket Programming using UDP:

Steps of Writing Client Program:

→Get a datagram socket
→Send request
→Get response
→Display response
→Close Socket

Steps of Writing Server Program:

→Get a datagram socket
→Receive request
→Send response
→Close socket.

Example (Client Program):

```
import java.net.*;
import java.io.*;
public class UDPClient{
```

```
public static void main(String args[]) throws IOException{
        DatagramSocket socket = new DatagramSocket();

        byte[] buf=new byte[256];
        InetAddress address=InetAddress.getByName("localhost");
        DatagramPacket packet = new DatagramPacket (buf, buf.length,
            socket.send(packet);   address, 4445);

        packet= new DatagramPacket (buf, buf.length);
        socket.receive (packet);

        String received= new String (packet.getData());

        System.out.println ("Quote of the Moment:"+received);
        socket.close();
    }
}
```

Example (Server Program):

```
import java.net.*;
import java.io.*;
public class UDPServer{
    public static void main (String args[]) throws IOException{
            byte[] buf = new byte[256];
            DatagramSocket socket = new DatagramSocket(4445);

            DatagramPacket packet=new DatagramPacket (buf, buf.length);
            socket.receive(packet);
            InetAddress address = packet.getAddress();
            int port = packet.getPort();
            String s= "Hello";
            buf = s.getBytes();

            packet =new DatagramPacket (buf, buf.length, address, port);
            socket.send (packet);

            socket.close();
    }
}
```

⊛ Working with URL's:

https://www.javatpoint.com/java-tutorial

protocol — Host name — File

A URL contains following informations:

i) Protocol: In this case https is the protocol.

ii) IP Address or Server name: In this case, www.javatpoint.com is the Server name.

iii) Port number: It is an optional attribute. It is written with IP address to identify the resource uniquely. Default value is -1.

iv) File Name: In this case, java-tutorial is file name. It can also be path to directory.

URL Demo Example:

```
import java.net.*;
public class URLDemo{
    public static void main(String[] args){
        try {
            URL url=new URL("https://www.javatpoint.com/java-tutorial");
            System.out.println("Protocol:"+url.getProtocol());
            System.out.println("Host Name:"+url.getHost());
            System.out.println("File Name:"+url.getPort());
            System.out.println("File Name:"+url.getFile());
        }
        catch (Exception e) {System.out.println(e);}
    }
}
```

⊛ Working with URL Connection Class:

The Java URLConnection class represents a communication link between the URL and the application. This class can be used to read and write data to the specified resource referred by the URL.

The URLConnection class provides many methods, we can display all the data of a webpage by using the getInputStream() method. The getInputStream() method returns all the data of the specified URL in the stream that can be read and displayed.

⊛ Java Mail API:

The JavaMail is an API that is used to compose, write and read electronic messages (emails). This API provides protocol-independent and platform-independent framework for sending and receiving mails. The javax.mail and javax.mail.activation packages the core classes of JavaMail API.

If we use Java 2 Platform, no additional setup is required, for Enterprise Edition (J2EE) 1.3. However, if we use Standard Edition (J2EE) 1.1.7 and upwards, we need to download and install following:

→ JavaMail API
→ Java Activation Framework.

⊛ Sending and Receiving Email:

Steps (Sending): There are following three steps to send email using JavaMail. They are as follows:

Get the session object: Stores all the information of host like host name, username, password etc.

```
Properties properties=new Properties();
Session session=Session.getDefaultInstance(properties,null);
```

Compose the message: MimeMessage class is mostly used for composing.

```
MimeMessage message=new MimeMessage(session);
```

Send the message: Transport class provides method to send the message.

```
Transport.send(message);
```

Steps (Receiving):

→ Get the session object.
→ create the POP3 store object and connect with the pop server.
→ create the folder object and open it.
→ retrive the messages from the folder in an array and print it.
→ close the store and folder objects.

If my notes really helped you,then you can support me on esewa for my hardwork.

Esewa ID: 9806470952