Java Networking

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

Java socket programming provides facility to share data between different computing devices.

Advantage of Java Networking

1. sharing resources
2. centralize software management

Java Networking Terminology

The widely used java networking terminologies are given below:

1. IP Address
2. Protocol
3. Port Number
4. MAC Address
5. Connection-oriented and connection-less protocol
6. Socket

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.

5) Connection-oriented and connection-less 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.

But, 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.

6) Socket

A socket is an endpoint between two way communications.

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
- **DatagramSocket** and **DatagramPacket** classes are used for connection-less socket programming.

The client in socket programming must know two information:

- IP Address of Server, and
- 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.

Important methods

Method	Description
1) public InputStream getInputStream()	returns the InputStream attached with this socket.
2) public OutputStream getOutputStream()	returns the OutputStream attached with this socket.
3) public synchronized void close()	closes this socket

ServerSocket

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
1) public Socket accept()	returns the socket and establish a connection between server and client.
2) public synchronized void close()	closes the server socket.

Example: socket programming in which client sends a text and server receives it.

File: MyServer.java

```
import java.io.*;
import java.net.*;
public class MyServer {
public static void main(String[] args){
try{
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();
}catch(Exception e){System.out.println(e);}
}
}
```

File: MyClient.java

```
import java.io.*;
import java.net.*;
public class MyClient {
public static void main(String[] args) {
try{
Socket s=new Socket("localhost",6666);
DataOutputStream dout=new DataOutputStream(s.getOutputStream());
dout.writeUTF("Hello Server");
dout.close();
s.close();
}catch(Exception e){System.out.println(e);}
}
}
```

Process: To execute this program open two command prompts and execute each program at each command prompt.

After running the client application, a message will be displayed on the server console.

Example of Java Socket Programming (Read-Write both side)

In this example, client will write first to the server then server will receive and print the text. Then server will write to the client and client will receive and print the text. The step goes on.

File: MyServer.java

```
import java.net.*;
import java.io.*;
class MyServer{
public static void main(String args[])throws Exception{
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();
}}
```

File: MyClient.java

```
import java.net.*;
import java.io.*;
class MyClient{
public static void main(String args[])throws Exception{
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();
}}
```

Java DatagramSocket and DatagramPacket

Java DatagramSocket and DatagramPacket classes are used for connection-less socket programming.

DatagramSocket class

Java DatagramSocket class represents a connection-less socket for sending and receiving datagram packets.
A datagram is basically information but there is no guarantee of its content, arrival or arrival time.

Commonly used Constructors of DatagramSocket class

- DatagramSocket() throws SocketException: it creates a datagram socket and binds it with the available Port Number on the localhost machine.
- DatagramSocket(int port) throws SocketException: it creates a datagram socket and binds it with the given Port Number.
- DatagramSocket(int port, InetAddress address) throws SocketException: it creates a datagram socket and binds it with the specified port number and host address.

DatagramPacket class

Java DatagramPacket is a message that can be sent or received. If you send multiple packet, it may arrive in any order. Additionally, packet delivery is not guaranteed.

Commonly used Constructors of DatagramPacket class

- DatagramPacket(byte[] barr, int length): it creates a datagram packet. This constructor is used to receive the packets.
- DatagramPacket(byte[] barr, int length, InetAddress address, int port): it creates a datagram packet. This constructor is used to send the packets.

Example: //DSender.java

```
import java.net.*;

public class DSender{

  public static void main(String[] args) throws Exception {

    DatagramSocket ds = new DatagramSocket();

    String str = "Welcome java";

    InetAddress ip = InetAddress.getByName("127.0.0.1");  //”localhost”

   DatagramPacket dp = new DatagramPacket(str.getBytes(), str.length(), ip, 3000);

    ds.send(dp);

    ds.close();   } }
```

```
//DReceiver.java
import java.net.*;
public class DReceiver{
  public static void main(String[] args) throws Exception {
    DatagramSocket ds = new DatagramSocket(3000);
    byte[] buf = new byte[1024];
    DatagramPacket dp = new DatagramPacket(buf, 1024); //buf.length
    ds.receive(dp);
    String str = new String(dp.getData(), 0, dp.getLength());
    System.out.println(str);
    ds.close();
  }
}
```

Public String(byte[] bytes,int offset,int length)

Bytes- the bytes to be decoded into characters.

Offet- the index of the first byte to decode.

Length- the number of bytes to decode.