Distributed Network Programming

Prativa Nyaupane

Review

- Socket Programming using TCP and UDP

Outline

- TCP
- UDP
- IP Address
- Ports
- Socket Programming using TCP and UDP
- Working with URLs and URL Connection Class
- Email Handling using Java Mail API
- Architecture of RMI
- Creating and Executing RMI applications
- Architecture of CORBA
- RMI vs CORBA
- IDL and Simple CORBA Program

Creating own Server and Client in Java Contd...

- The client application creates a socket object and specifies the server's IP address and port number to establish a connection.
- Both the server and client applications can use input and output streams associated with the Socket object to send and reciev data.
- Communication between the server and client can be done using various protocols, such as TCP or UDP.

Networking Classes in Java

- Java provides a set of built in classes required for creating own server and client:
    - First of all, we should import java.net package
    - Networking classes in Java : Socket, ServerSocket

Client-side endpoint(Socket)	Server-side endpoint(Server Socket)
It establishes a TCP connection to a remote server.	It is used to listen for incoming client connections in a specific port.
It also provides input and output stream for sending and receiving data.	It can accept client connections using accept() method and create a separate Socket objects

- Once a client connection is accepted, a new Socket object is created on the server side to communicate with the client.

Networking Classes in Java

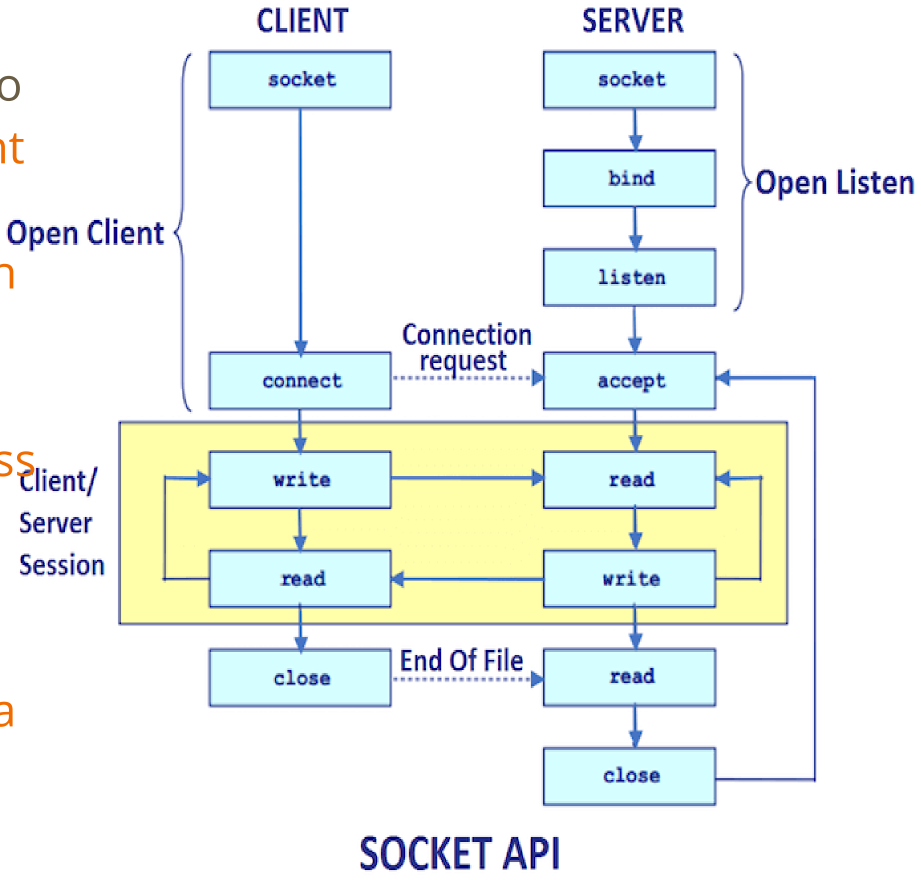
- The other networking classes are:
  - URL : In Java, the URL class represents a URL and provides functionalities to parse, construct, and work with its various components.
  - URLConnection : In Java, the URLConnection class serves as a bridge between a URL and the actual network connection.
  - InetAddress : It acts as a bridge between human-readable hostnames (like "pu.edu.np") and numerical IP addresses.

Creating own Server and Client in Java Contd...

- Java socket programming is used for communication between the applications running on different JRE.
- In Java, we can create our own Server and Client applications using the ServerSocket and Socket classes.
- The Server application typically listens on a specific port using a ServerSocket object. It accepts incoming client connections using the accept() method of the server socket class.
- Once a client connection is accepted, a new Socket object is created on the Server side to communication with the client.

1. Establish a Socket Connection:

- To enable communication between two machines there must be socket present at both endpoints.
    - Each socket must be assigned with IP address and port number.
- After creation of socket, the bind function binds the socket to the address and port number.
- listen(): It puts the server socket in a passive mode, where it waits for the client to approach the server to make a connection.
- accept(): This method waits for the client. If client connects with the given port number, it returns an instance of a



After the connection is established...

2. Communication in Client/Server:

- In the figure, client will first write to the server i.e send data to the server.
  - To write data: DataOutputStream Class is used.
- Server will then read the data ie. receive the data.
  - To read data: DataInputStream Class is used.

The new socket object maintains direct communication, hence, we can write and read messages to and from the server.

3. Closing the Connection:

- Finally, once the purpose is fulfilled the client sends a request to the server to terminate the connection between the client socket and server socket.

DataOutputStream

- A data output stream lets an application write primitive Java data types to an output stream in a portable way.
- To use this, first import
  - import java.io.*;
- When we want to send data over a network connection, we use SocketOutputStream.
  - `OutputStream outputStream =clientSocket.getOutputStream();`
- We then pass the OutputStream to DataOutputStream, because DataOutputStream so that we can add primitive data type to the socket's output stream.

```
Socket clientSocket = new Socket("localHost"
,6666);
OutputStream outputStream =clientSocket.getOutputStream();
DataOutputStream dataOutputStream = new
DataOutputStream(outputStream );
 dataOutputStream.writeUTF("Hello!");
 dataOutputStream.flush();
 dataOutputStream.close();
 clientSocket.close();
```

DataOutputStream Contd...

writeUTF(“`Hello`”) - writes String to the DataOutputStream.

flush() - flush() makes sure any buffered data is written to the output stream.

close() - It is important to close output stream to free up the resources and ensure proper cleanup.

DataInputStream

```
 InputStream inputStream = clientSocket.getInputStream();
DataInputStream dataInputStream = new DataInputStream(inputStream);
String stringFromInputStream = (String) dataInputStream.readUTF();
```

- Java DataInputStream allows an application to read primitive data from the input stream in a machine-independent way.
- InputStream is used to read data from a source into a java program.
- DataInputStream is wrapped around Socket’s InputStream so that the client/server can read primitive data types sent over the network using DataOutputStream.
- readUTF - reads a String thats stored in a modified UTF-8 format from the inputstream.

- Create a TCP client server application where the client sends a string and the server responds by echoing the same string in uppercase

```
import java.io.*;
import java.net.*;
import java.util.Scanner;
public class TCPClient {
    public static void main(String[] args) {
        try {
            Socket socket = new Socket("locahost", 1234);

            DataOutputStream out = new DataOutputStream(socket.getOutputStream());//pass data between
//two applications

            out.writeUTF("Hello!");//write string in a network-compatible way
                  out.flush();           //force buffered data to the output stream
                  out.close();           //free up the resource

            // Receive and print the response from the server
            DataInputStream in = new DataInputStream(socket.getInputStream());
            String response = in.readUTF();
            System.out.println("Server response: " + response);

            socket.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

```
import java.io.*;
import java.net.*;

public class TCPServer {
    public static void main(String[] args) {
        final int PORT = 1234;
        try {
            ServerSocket serverSocket = new ServerSocket(PORT); //Server is started and listening in 1234
//port
            Socket clientSocket = serverSocket.accept(); //client is connected to server

            DataInputStream in = new DataInputStream(clientSocket.getInputStream());//get data from client
            // Receive the string from the client
            String receivedString = in.readUTF(); //read client’s data
            System.out.println("Received from client: " + receivedString);

            DataOutputStream out = new DataOutputStream(clientSocket.getOutputStream());
           // Echo back the received string in uppercase
            String upperCaseString = receivedString.toUpperCase();
            out.writeUTF(upperCaseString);
                       out.flush();
                       out.close();

             clientSocket.close();
            serverSocket.close(); // Close the server socket after processing one client
        } catch (IOException e) {
            e.printStackTrace();
        } }}
```

Past Questions

- Create a TCP client server application where the client sends a string and the server responds by echoing the same string in uppercase
- Write a Java program that send messages with each other using TCP socket.
- Write a program to send “Message from Pokhara University” from client to server using java socket programming.
- Create a TCP client/server program in which multiple user can send and receive message from each other
- TCP vs UDP