NMK30703: Programming for Networks

Lab module 3:

Java Network and client-server socket

Mohamed Elshaikh

Faculty of Electronics Engineering Technology – UniMAP (FTKEN-UniMAP)

Objectives

- Java Network Programming
- Java InetAddress
- Java client Server Socket

1. Java Networking (java.net)

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. Java networking advantages:

- sharing resources
- centralize software management

The widely used java networking terminologies are given below:

- IP Address
- Protocol
- Port Number
- MAC Address
- Connection-oriented and connection-less protocol
- 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 communication.

2. Java InetAddress

The InetAddress class represents an IP address, both IPv4 and IPv6. Basically you create instances of this class to use with other classes such as Socket, ServerSocket, DatagramPacket and DatagramSocket. In the simplest case, you can use this class to know the IP address from a hostname, and vice-versa.

The InetAddress class doesn't have public constructors, so you create a new instance by using one of its factory methods:

1. **`getByName(String host)`**: creates an `InetAddress` object based on the provided hostname.
2. **`getByAddress(byte[] addr)`**: returns an `InetAddress` object from a byte array of the raw IP address.
3. **`getAllByName(String host)`**: returns an array of `InetAddress` objects from the specified hostname, as a hostname can be associated with several IP addresses.
4. **`getLocalHost()`**: returns the address of the `localhost`.
5. To get the IP address/hostname you can use a couple of methods below:
6. **`getHostAddress()`**: returns the IP address in text.
7. **`getHostname()`**: gets the hostname.

Note that the InetAddress class's toString() method returns both hostname and IP address, In addition, this class also provides several methods for checking the address type, which would be useful for system programmers. However we don't have to concern about those methods, most of the time. Let's see some examples that demonstrate how to use the InetAddress class.

1. Get IP address of a given domain/hostname:

The following code prints the IP address of a given hostname:

```
InetAddress address1 = InetAddress.getByName("www.codejava.net");
System.out.println(address1.getHostAddress());
```

2. Get hostname from IP address:

The following code finds out the hostname from an IP address:

```
InetAddress address2 = InetAddress.getByName("8.8.8.8");
System.out.println(address2.getHostName());
```

3. List all IP addresses associate with a hostname/domain:

The following code prints all the IP addresses associated with the hostname google.com:

```
InetAddress[] google = InetAddress.getAllByName("google.com");
for (InetAddress addr : google) {
    System.out.println(addr.getHostAddress());
}
```

4. Get the localhost address:

And the following code gets the localhost address:

```
InetAddress localhost = InetAddress.getLocalHost();
System.out.println(localhost);
```

5. Inet4Address and Inet6Address:

These are subclasses of the InetAddress class. Inet4Address and Inet6Address represent IPv4 and IPv6 addresses, respectively. However, when writing network applications, you don't have to concern about IPv4 or IPv6 as Java hides all the details.

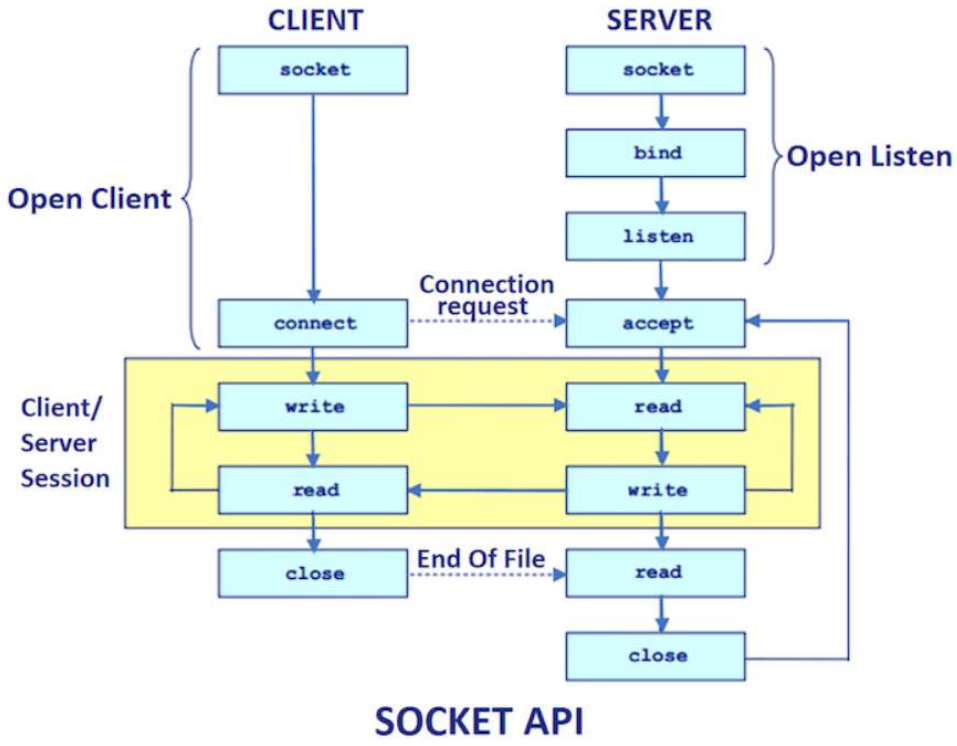
The InetAddress can refer to either Inet4Address or Inet6Address so most of the time, using InetAddress is enough.

3. Java Client Server Socket

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.

Here, we are going to make one-way client and server communication. In this application, client sends a message to the server, server reads the message and prints it. Here, two classes are being used: Socket and ServerSocket. The Socket class is used to communicate client and server. Through this class, we can read and write message. The ServerSocket class is used at server-side. The accept() method of ServerSocket class blocks the console until the client is connected. After the successful connection of client, it returns the instance of Socket at server-side.



1. 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

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

3. Example of Java Socket Programming

Creating Server:

To create the server application, we need to create the instance of ServerSocket class. Here, we are using 6666 port number for the communication between the client and server. You may also choose any other port number. The accept() method waits for the client. If clients connects with the given port number, it returns an instance of Socket.

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

Creating Client:

To create the client application, we need to create the instance of Socket class. Here, we need to pass the IP address or hostname of the Server and a port number. Here, we are using "localhost" because our server is running on same system.

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

Let's see a simple of Java socket programming where client sends a text and server receives and prints 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.flush();
dout.close();
s.close();
}catch(Exception e){System.out.println(e);}
}
}
```

To execute this program open two command prompts and execute each program at each command prompt as displayed in the below figure. After running the client application, a message will be displayed on the server console.

4. 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();
}}
```

TASKs 1:

Answer all questions.

1. What are the differences between IPv4 and IPv6? List your answers and give examples for both addresses.
2. Discuss the reasons why there is a need for network protocols.
3. Describe how hosts from two different LANs can connect to each other. Include figure to demonstrate the idea.
4. Explain why we need both TCP and IP protocols.
5. Discuss the significance of using layered network model. Write the importance of each layer.
6. For each of the following IP addresses, determine its class, default subnet mask, network ID, broadcast ID, and range.

    a. 216.254.85.74
    b. 10.250.1.1
    c. 117.89.56.45
    d. 95.0.21.90
    e. 199.155.77.56

TASKs 2:

1. Write a Java program to print the IP address of "www.google.com" using the `InetAddress` class.
2. Write a Java program to display the localhost's IP address and hostname.
3. Write a Java program to list all IP addresses associated with "facebook.com".
4. Write a Java program to perform a reverse DNS lookup for the IP address "8.8.8.8" and print the hostname.
5. Write a Java program to check if "www.github.com" and "github.com" resolve to the same IP address(es). Print "Same" or "Different" accordingly.
6. Write a Java program to classify the following IP addresses into their respective classes (A, B, C, D, or E):
    a. 192.168.1.1
    b. 10.0.0.1
    c. 224.0.0.1
    d. 172.16.254.1
7. Write a Java program to check if ports 80 (HTTP) and 443 (HTTPS) are open on "www.unimap.edu.my". Use `Socket` class and handle exceptions gracefully.
8. Create a server that listens on port 1234 and a client that sends the message "Hello Server" to the server. The server should print the received message.
9. Extend the client-server example to allow bidirectional communication. The server should reply to the client's message with "Message Received", and the client should print the server's response.