Lecture 6 - Java Networking

Guiliang Liu

The Chinese University of Hong Kong, Shenzhen

CSC-1004: Computational Laboratory Using Java
Course Page: [Click]

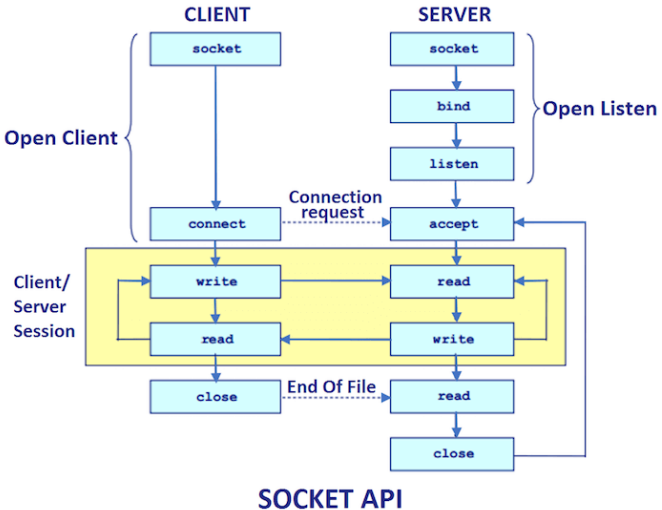
Outline

- Java Socket Programming

Java Socket Programming

- Java Socket programming is used for communication between the applications running on different JRE.
- **Socket and ServerSocket** classes are used for connection-oriented socket programming.
- **The client** in socket programming must know the IP Address of the server and port number.

Java Socket Programming



One-way client and server communication: the client sends a message to the server, and the server reads the message and prints it.

Example of Java Socket Programming

- **Creating Server:** 1) We use the 6666 port number for the communication between the client and server. 2) The accept() method waits for the client. If clients connect 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
```

香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

Example of Java Socket Programming

- **Creating Client:** We pass the IP address or hostname of the Server and a port number. Here, we are using "localhost" because our server is running on the same system.

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

Example of Java Socket Programming

- 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);}
}
}
```

- 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);}
}
}
```

The Chinese University of Hong Kong, Shenzhen

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

- 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();
}}
```

- MyClient.java

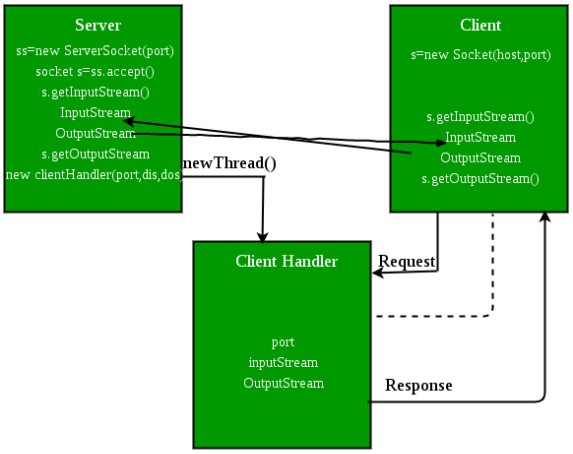
```
import java.net.*;
import java.io.*;
class MyClient{
public static void main(String args[])throws Exception{
Socket s=new Socket("localhost",3333);
DataInputStream din=new DataInputStream(s.getInputStream());
DataOutputStream dout=new DataOutputStream(s.getOutputStream());
BufferedReader br=new BufferedReader(new 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();
}}
```

Threads in Socket Programming in Java

Why use threads in network programming?

We want our architecture to support multiple clients at the same time. For this reason, we must use threads on the server side so that whenever a client request comes, a separate thread can be assigned to handle each request.

Threads in Socket Programming in Java



1. Server file contains two classes namely Server (for creating server) and Client-Handler (for handling any client using multithreading).
2. Client file contains only one public class Client (for creating a client).

香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

Threads in Socket Programming in Java

```java
// Server class
public class Server
{
    public static void main(String[] args) throws IOException
    {
        // server is listening on port 5056
        ServerSocket ss = new ServerSocket(5056);

        // running infinite loop for getting
        // client request
        while (true)
        {
            Socket s = null;

            try
            {
                // socket object to receive incoming client requests
                s = ss.accept();

                System.out.println("A new client is connected : " + s);

                // obtaining input and out streams
                DataInputStream dis = new DataInputStream(s.getInputStream());
                DataOutputStream dos = new DataOutputStream(s.getOutputStream());

                System.out.println("Assigning new thread for this client");

                // create a new thread object
                Thread t = new ClientHandler(s, dis, dos);

                // Invoking the start() method
                t.start();

            }
            catch (Exception e){
```

Server Side: Server class.

1. Establishing the Connection.
2. Obtaining the Streams.
3. Creating a handler object.
4. Invoking the start() method.

香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

Threads in Socket Programming in Java

```
// ClientHandler class
class ClientHandler extends Thread
{
    DateFormat fordate = new SimpleDateFormat("yyyy/MM/dd");
    DateFormat fortime = new SimpleDateFormat("hh:mm:ss");
    final DataInputStream dis;
    final DataOutputStream dos;
    final Socket s;


    // Constructor
    public ClientHandler(Socket s, DataInputStream dis, DataOutputStream dos)
    {
        this.s = s;
        this.dis = dis;
        this.dos = dos;
    }

    @Override
    public void run()
    {
        String received;
        String toreturn;
        while (true)
        {
            try {

                // Ask user what he wants
                dos.writeUTF("What do you want?[Date | Time]..\n"+
                            "Type Exit to terminate connection.");

                // receive the answer from client
                received = dis.readUTF();
```

Server Side: ClientHandler class.

1. Build a class to extend the Thread class.
2. Build a constructor that takes a Socket, a DataInputStream to read, and a DataOutputStream to write.
3. Override the run() method.

香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

Threads in Socket Programming in Java

```
if(received.equals("Exit"))
{
    System.out.println("Client " + this.s + " sends exit...");
    System.out.println("Closing this connection.");
    this.s.close();
    System.out.println("Connection closed");
    break;
}

// creating Date object
Date date = new Date();

// write on output stream based on the
// answer from the client
switch (received) {

    case "Date" :
        toreturn = fordate.format(date);
        dos.writeUTF(toreturn);
        break;

    case "Time" :
        toreturn = fortime.format(date);
        dos.writeUTF(toreturn);
        break;

    default:
        dos.writeUTF("Invalid input");
        break;
```

Server Side: ClientHandler class.

1. Build a class to extend the Thread class.
2. Build a constructor that takes a Socket, a DataInputStream to read, and a DataOutputStream to write.
3. Override the run() method.

香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

Threads in Socket Programming in Java

```java
// Client class
public class Client
{
    public static void main(String[] args) throws IOException
    {
        try
        {
            Scanner scn = new Scanner(System.in);

            // getting localhost ip
            InetAddress ip = InetAddress.getByName("localhost");

            // establish the connection with server port 5056
            Socket s = new Socket(ip, 5056);

            // obtaining input and out streams
            DataInputStream dis = new DataInputStream(s.getInputStream());
            DataOutputStream dos = new DataOutputStream(s.getOutputStream());

            // the following loop performs the exchange of
            // information between client and client handler
            while (true)
            {
                System.out.println(dis.readUTF());
                String tosend = scn.nextLine();
                dos.writeUTF(tosend);
```

Client Side:

1. Establish a Socket Connection.
2. Communication.

香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

Threads in Socket Programming in Java

```
    while (true)
    {
        System.out.println(dis.readUTF());
        String tosend = scn.nextLine();
        dos.writeUTF(tosend);

        // If client sends exit,close this connection
        // and then break from the while loop
        if(tosend.equals("Exit"))
        {
            System.out.println("Closing this connection : " + s);
            s.close();
            System.out.println("Connection closed");
            break;
        }

        // printing date or time as requested by client
        String received = dis.readUTF();
        System.out.println(received);
    }

    // closing resources
    scn.close();
    dis.close();
    dos.close();
}catch(Exception e){
    e.printStackTrace();
}
```

Client Side:

1. Establish a Socket Connection.
2. Communication.

香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

Multi-threaded chat Application in Java

Please check the following examples:

- Server Side Programming (Server.java)
- Client Side Programming (Client.java)

香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen