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

sharing resources

centralize software management

Java Networking Terminology

The widely used java networking terminologies are given below:

IP Address

Protocol

Port Number

MAC Address

Connection-oriented and connection-less protocol

Socket

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 and 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 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
2) public synchronized void close()	closes the server socket.

Example of Java Socket Programming

Let's see a simple of java 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.flush();
dout.close();

s.close();

}catch(Exception e){System.out.println(e);}

}

}
```

Java URL

The Java URL class represents an URL. URL is an acronym for Uniform Resource Locator. It points to a resource on the World Wide Web.

A URL contains many information:

Protocol: In this case, http is the protocol.

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

Port Number: It is an optional attribute. If we write http//ww.javatpoint.com:80/sonoojaiswal/ , 80 is the port number. If port number is not mentioned in the URL, it returns -1.

File Name or directory name: In this case, index.jsp is the file name.

Commonly used methods of Java URL class

The java.net.URL class provides many methods. The important methods of URL class are given below.

Method	Description
public String getProtocol()	it returns the protocol of the URL.
public String getHost()	it returns the host name of the URL.
public String getPort()	it returns the Port Number of the URL.
public String getFile()	it returns the file name of the URL.

Public URLConnection openConnection()	it returns the instance of URLConnection i.e. associated with this URL.

Example of Java URL class

```
//URLDemo.java

import java.io.*;

import java.net.*;

public class URLDemo{

public static void main(String[] args){

try{

URL url=new URL("http://www.google.com/android");

System.out.println("Protocol: "+url.getProtocol());

System.out.println("Host Name: "+url.getHost());

System.out.println("Port Number: "+url.getPort());

System.out.println("File Name: "+url.getFile());

}catch(Exception e){System.out.println(e);}

}

}
```

Output:

```
    Protocol: http

    Host Name: www.google.com

    Port Number: -1

    File Name: /android
```