Network Programming

Introduction

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.

Introduction

The java.net package supports two protocols,

1.TCP: Transmission Control Protocol provides reliable communication between the sender and receiver. TCP is used along with the Internet Protocol referred as TCP/IP.

2.UDP: User Datagram Protocol provides a connection-less protocol service by allowing packet of data to be transferred along two or more nodes

Introduction

Networking Terminology

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

Introduction

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.

Introduction

2) Protocol

A protocol is a set of rules basically that is followed for communication. For example:

- TCP
- FTP
- Telnet
- SMTP
- POP etc.

Introduction

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.

Introduction

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 address.

For example, an ethernet card may have a **MAC** address of 00:0d:83::b1:c0:8e.

Introduction

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.

Introduction

6) Socket

A socket is an endpoint between two way communications.

Introduction

java.net package

The java.net package can be divided into two sections:

1.A Low-Level API: It deals with the abstractions of addresses i.e. networking identifiers, Sockets i.e. bidirectional data communication mechanism and Interfaces i.e. network interfaces.

2.A High Level API: It deals with the abstraction of URIs i.e. Universal Resource Identifier, URLs i.e. Universal Resource Locator, and Connections i.e. connections to the resource pointed by URLs.

Introduction

The java.net package provides many classes to deal with networking applications in Java. A list of these classes is given below:

- Authenticator
- CacheRequest
- CacheResponse
- ContentHandler
- CookieHandler
- CookieManager
- DatagramPacket
- DatagramSocket
- DatagramSocketImpl

Introduction

List of interfaces available in java.net package:

- ContentHandlerFactory
- CookiePolicy
- CookieStore
- DatagramSocketImplFactory

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.

Java Socket Programming

The client in socket programming must know two information:

1.IP Address of Server, and
2.Port number.

Java Socket Programming

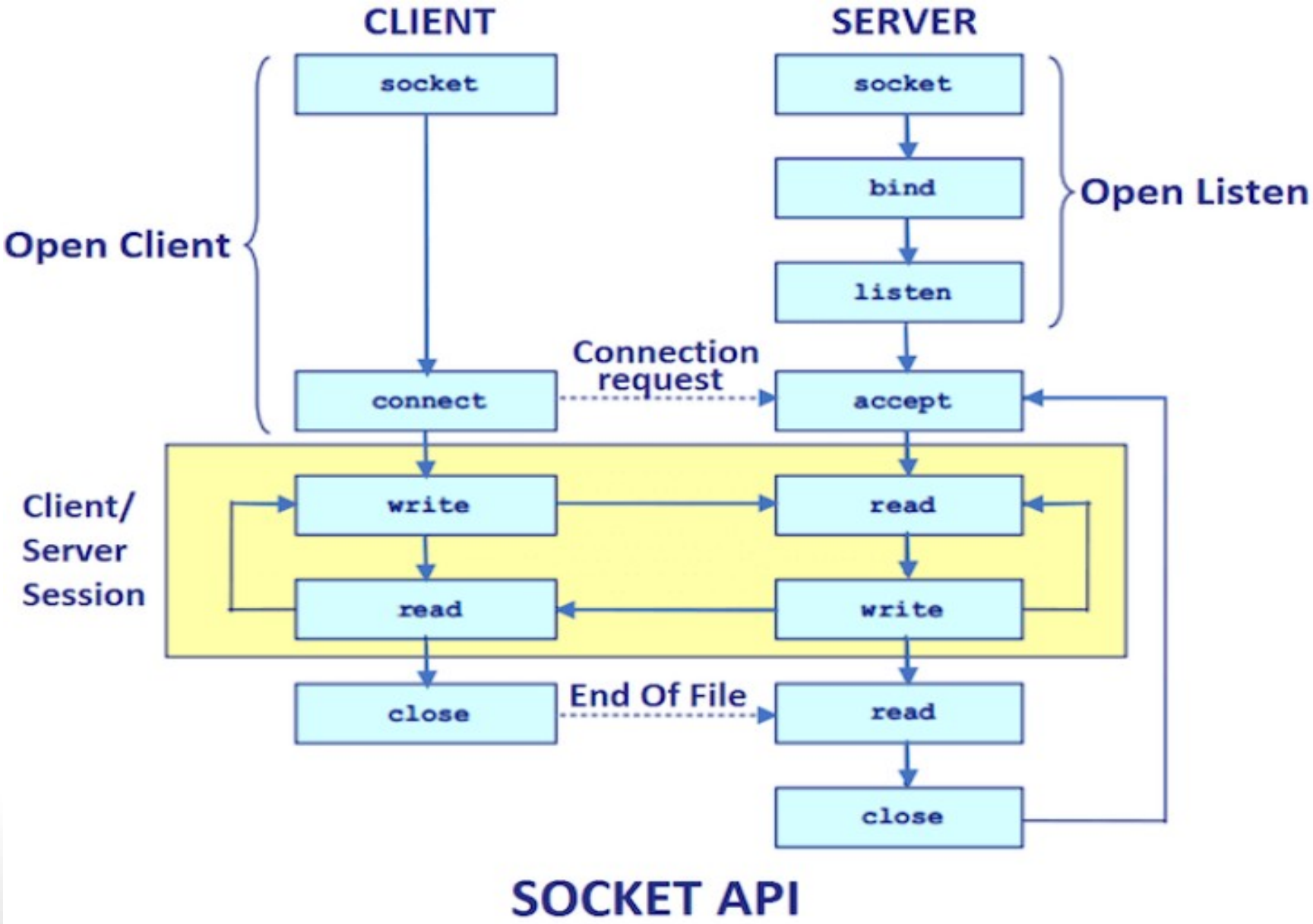
Here, we are going to make one-way client and server communication. In this application, client sends a message to the server, server reads the message and prints it.

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.

Java Socket Programming



Java Socket Programming

Socket class

A socket is simply an endpoint for communications between the machines. The Socket class can be used to create a socket.

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

Java Socket Programming

ServerSocket class

The ServerSocket class can be used to create a server socket. This object is used to establish communication with the clients.

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.

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.

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

Java Socket Programming

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.

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

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.getOutput
Stream());
dout.writeUTF("Hello Server");
dout.flush();
dout.close();
s.close();
}catch(Exception e){System.out.println(e);}
}
}
```

Java **DataOutputStream** class allows an application to write primitive Java data types to the output stream in a machine-independent way.

getOutputStream() method of Java Socket class returns an output stream for the given socket.

writeUTF Writes a string to the underlying output stream