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

1. sharing resources
2. centralize software management

Network programming refers to writing programs that execute across multiple devices (computers), in which the devices are all connected to each other using a network.

The java.net package of the J2SE APIs contains a collection of classes and interfaces that provide the low-level communication details, allowing you to write programs that focus on solving the problem at hand.

The java.net package provides support for the two common network protocols −

- **TCP** − TCP stands for Transmission Control Protocol, which allows for reliable communication between two applications. TCP is typically used over the Internet Protocol, which is referred to as TCP/IP.
- **UDP** − UDP stands for User Datagram Protocol, a connection-less protocol that allows for packets of data to be transmitted between applications

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

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.

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:

1. IP Address of Server, and
2. 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.The **java.net.Socket** class represents the socket that both the client and the server use to communicate with each other. The client obtains a Socket object by instantiating one, whereas the server obtains a Socket object from the return value of the accept() method.

The Socket class has constructors that a client uses to connect to a server −

Sr.No.	Method & Description

1	**public Socket(String host, int port) throws UnknownHostException, IOException.** This method attempts to connect to the specified server at the specified port. If this constructor does not throw an exception, the connection is successful and the client is connected to the server.
2	**public Socket(InetAddress host, int port) throws IOException** This method is identical to the previous constructor, except that the host is denoted by an InetAddress object.
3	**public Socket(String host, int port, InetAddresslocalAddress, intlocalPort) throws IOException.** Connects to the specified host and port, creating a socket on the local host at the specified address and port.
4	**public Socket()Creates an unconnected socket. Use the connect() method to connect this socket to a server.**

Important methods

Method	Description
1) public InputStreamgetInputStream()	returns the InputStream attached with this socket.
2) public OutputStreamgetOutputStream()	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 client.
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. For example:

1. http://www.javatpoint.com/java-tutorial

A URL contains many information:

1. **Protocol:** In this case, http is the protocol.
2. **Server name or IP Address:** In this case, www.javatpoint.com is the server name.
3. **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.
4. **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 URLConnectionopenConnection()	it returns the instance of URLConnection i.e. associated with this URL.

Example of Java URL class

```
1. //URLDemo.java
2. import java.io.*;
3. import java.net.*;
4. public class URLDemo{
5. public static void main(String[] args){
6. try{
7. URL url=new URL("http://www.javatpoint.com/java-tutorial");
8.
9. System.out.println("Protocol: "+url.getProtocol());
10. System.out.println("Host Name: "+url.getHost());
11. System.out.println("Port Number: "+url.getPort());
12. System.out.println("File Name: "+url.getFile());
13.
14. }catch(Exception e){System.out.println(e);}
15. }
16. }
```

Java URLConnection class

The **Java URLConnection** class represents a communication link between the URL and the application. This class can be used to read and write data to the specified resource referred by the URL.

How to get the object of URLConnection class

The openConnection() method of URL class returns the object of URLConnection class. Syntax:

```
1. public URLConnection openConnection()throws IOException{}
```

Displaying source code of a webpage by URLConnecton class

The URLConnection class provides many methods, we can display all the data of a webpage by using the getInputStream() method. The getInputStream() method returns all the data of the specified URL in the stream that can be read and displayed.

Example of Java URLConnecton class

```
1. import java.io.*;
2. import java.net.*;
3. public class URLConnectionExample {
```

```
4. public static void main(String[] args){
5. try{
6. URL url=new URL("http://www.javatpoint.com/java-tutorial");
7. URLConnection urlcon=url.openConnection();
8. InputStream stream=urlcon.getInputStream();
9. int i;
10. while((i=stream.read())!=-1){
11. System.out.print((char)i);
12. }
13. }catch(Exception e){System.out.println(e);}
14. }
15. }
```

Java InetAddress class

Java InetAddress class represents an IP address. The java.net.InetAddress class provides methods to get the IP of any host name for example www.javatpoint.com, www.google.com, www.facebook.com etc.

Commonly used methods of InetAddress class

Method	Description
public static InetAddressgetByName(String host) throws UnknownHostException	it returns the instance of InetAddress containing LocalHost IP and name.
public static InetAddressgetLocalHost() throws UnknownHostException	it returns the instance of InetAdddress containing local host name and address.
public String getHostName()	it returns the host name of the IP address.
public String getHostAddress()	it returns the IP address in string format.

Example of Java InetAddress class

Let's see a simple example of InetAddress class to get ip address of www.javatpoint.com website.

```
import java.io.*;
import java.net.*;
public class InetDemo{
```

```
public static void main(String[] args){
try{
InetAddress ip=InetAddress.getByName("www.javatpoint.com");

System.out.println("Host Name: "+ip.getHostName());
System.out.println("IP Address: "+ip.getHostAddress());
}catch(Exception e){System.out.println(e);}
}
}
```

Java DatagramSocket and DatagramPacket

Java DatagramSocket and DatagramPacket classes are used for connection-less socket programming.

Java DatagramSocket class

Java DatagramSocket class represents a connection-less socket for sending and receiving datagram packets.

A datagram is basically an information but there is no guarantee of its content, arrival or arrival time.

Commonly used Constructors of DatagramSocket class

- **DatagramSocket() throws SocketEeption:** it creates a datagram socket and binds it with the available Port Number on the localhost machine.
- **DatagramSocket(int port) throws SocketEeption:** it creates a datagram socket and binds it with the given Port Number.
- **DatagramSocket(int port, InetAddress address) throws SocketEeption:** it creates a datagram socket and binds it with the specified port number and host address.

Java DatagramPacket class

Java DatagramPacket is a message that can be sent or received. If you send multiple packet, it may arrive in any order. Additionally, packet delivery is not guaranteed.

Commonly used Constructors of DatagramPacket class

- **DatagramPacket(byte[] barr, int length):** it creates a datagram packet. This constructor is used to receive the packets.
- **DatagramPacket(byte[] barr, int length, InetAddress address, int port):** it creates a datagram packet. This constructor is used to send the packets.

Example of Sending DatagramPacket by DatagramSocket

```
1. //DSender.java
2. import java.net.*;
```

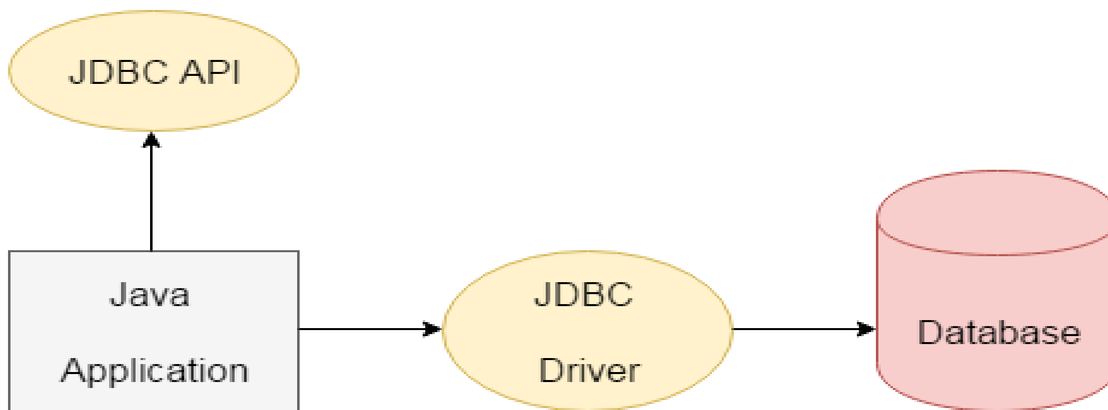
```
3. public class DSender{
4.   public static void main(String[] args) throws Exception {
5.     DatagramSocket ds = new DatagramSocket();
6.     String str = "Welcome java";
7.     InetAddress ip = InetAddress.getByName("127.0.0.1");
8.
9.     DatagramPacket dp = new DatagramPacket(str.getBytes(), str.length(), ip, 3000)
   ;
10.    ds.send(dp);
11.    ds.close();
12.  }
13.}
```

Example of Receiving DatagramPacket by DatagramSocket

```
1. //DReceiver.java
2. import java.net.*;
3. public class DReceiver{
4.   public static void main(String[] args) throws Exception {
5.     DatagramSocket ds = new DatagramSocket(3000);
6.     byte[] buf = new byte[1024];
7.     DatagramPacket dp = new DatagramPacket(buf, 1024);
8.     ds.receive(dp);
9.     String str = new String(dp.getData(), 0, dp.getLength());
10.    System.out.println(str);
11.    ds.close();
12.  }
13.}
```

Java Database Connectivity

Java JDBC is a java API to connect and execute query with the database. JDBC API uses jdbc drivers to connect with the database.



Why use JDBC

Before JDBC, ODBC API was the database API to connect and execute query with the database. But, ODBC API uses ODBC driver which is written in C language (i.e. platform dependent and unsecured). That is why Java has defined its own API (JDBC API) that uses JDBC drivers (written in Java language).

What is API

API (Application programming interface) is a document that contains description of all the features of a product or software. It represents classes and interfaces that software programs can follow to communicate with each other. An API can be created for applications, libraries, operating systems, etc

JDBC Driver

1. JDBC-ODBC bridge driver
2. Native-API driver
3. Network Protocol driver
4. Thin driver

JDBC Driver is a software component that enables java application to interact with the database.There are 4 types of JDBC drivers:

1. JDBC-ODBC bridge driver
2. Native-API driver (partially java driver)
3. Network Protocol driver (fully java driver)
4. Thin driver (fully java driver)

1) JDBC-ODBC bridge driver

: JDBC-ODBC bridge driver uses ODBC driver to connect to the database. The JDBC-ODBC lge driver converts JDBC method calls into the ODBC function calls. This is now discouraged ause of thin driver.

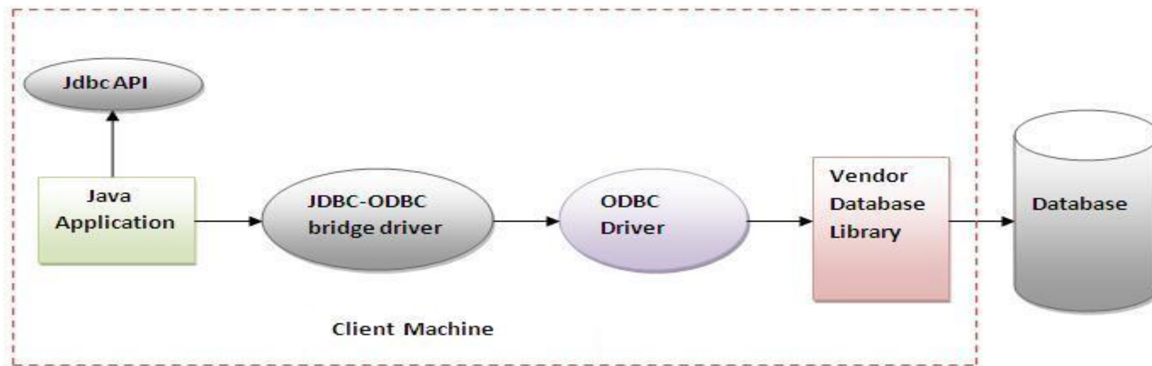


Figure- JDBC-ODBC Bridge Driver

Advantages:

- easy to use.
- can be easily connected to any database.

Disadvantages:

- Performance degraded because JDBC method call is converted into the ODBC function calls.
- The ODBC driver needs to be installed on the client machine.

---

2) Native-API driver

: Native API driver uses the client-side libraries of the database. The driver converts JDBC :hod calls into native calls of the database API. It is not written entirely in java.

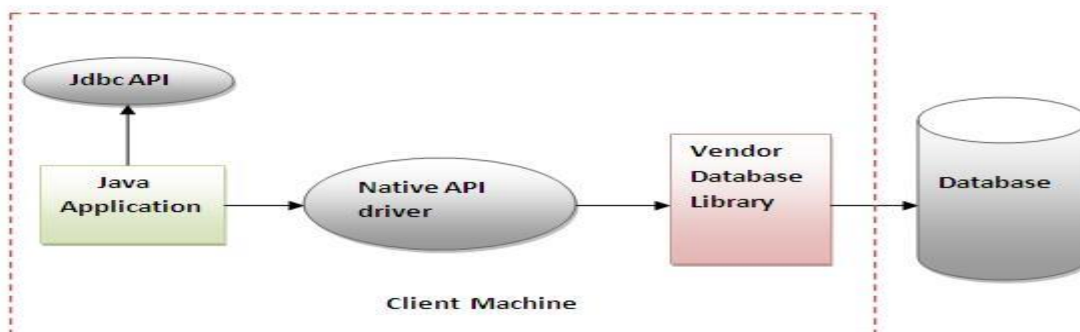


Figure- Native API Driver

Advantage:

- performance upgraded than JDBC-ODBC bridge driver.

Disadvantage:

- The Native driver needs to be installed on the each client machine.
- The Vendor client library needs to be installed on client machine.

---

3) Network Protocol driver

The Network Protocol driver uses middleware (application server) that converts JDBC calls directly or indirectly into the vendor-specific database protocol. It is fully written in java.

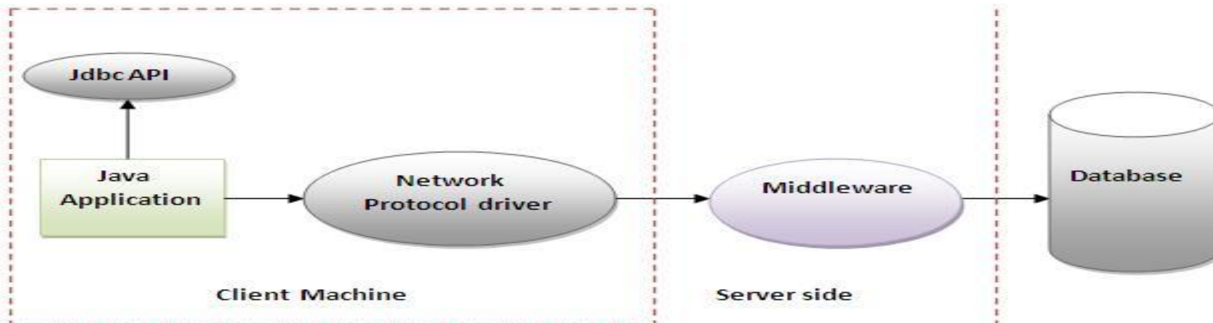


Figure- Network Protocol Driver

Advantage:

- No client side library is required because of application server that can perform many tasks like auditing, load balancing, logging etc.

Disadvantages:

- Network support is required on client machine.
- Requires database-specific coding to be done in the middle tier.
- Maintenance of Network Protocol driver becomes costly because it requires database-specific coding to be done in the middle tier.

---

4) Thin driver

ıe thin driver converts JDBC calls directly into the vendor-specific database protocol. That is why it is known as thin driver. It is fully written in Java language.

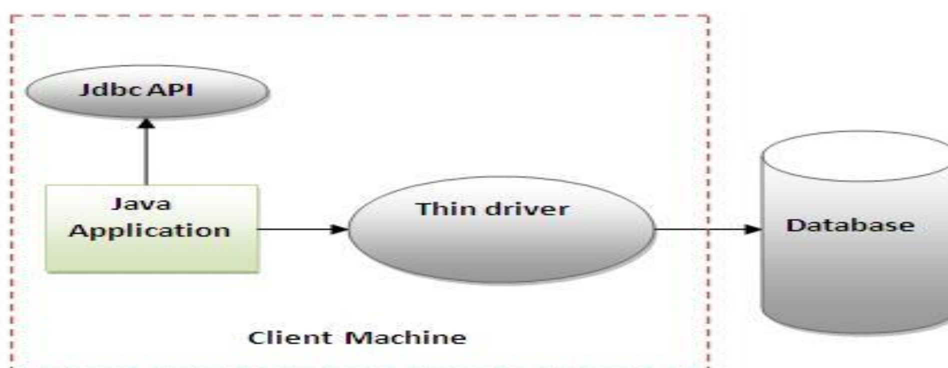


Figure- Thin Driver

Advantage:

- Better performance than all other drivers.
- No software is required at client side or server side.

Disadvantage:

- Drivers depends on the Database.

Steps to connect to the database in java

1. Register the driver class
2. Create the connection object
3. Create the Statement object
4. Execute the query
5. Close the connection object

---

1) Register the driver class

The forName() method of Class class is used to register the driver class. This method is used to dynamically load the driver class.

Syntax of forName() method

1. **public static void** forName(String className)**throws** ClassNotFoundException

Example to register the OracleDriver class

1. Class.forName("oracle.jdbc.driver.OracleDriver");

---

2) Create the connection object

The getConnection() method of DriverManager class is used to establish connection with the database.

Syntax of getConnection() method

1. 1) **public static** Connection getConnection(String url)**throws** SQLException
2. 2) **public static** Connection getConnection(String url,String name,String password)
3. **throws** SQLException

Example to establish connection with the Oracle database

1. Connection con=DriverManager.getConnection(
2. "jdbc:oracle:thin:@localhost:1521:xe","system","password");

---

3) Create the Statement object

The createStatement() method of Connection interface is used to create statement. The object of statement is responsible to execute queries with the database.

Syntax of createStatement() method

```
1. public Statement createStatement()throws SQLException
```

Example to create the statement object

```
1. Statement stmt=con.createStatement();
```

---

4) Execute the query

The executeQuery() method of Statement interface is used to execute queries to the database. This method returns the object of ResultSet that can be used to get all the records of a table.

Syntax of executeQuery() method

```
1. public ResultSet executeQuery(String sql)throws SQLException
```

Example to execute query

```
1. ResultSet rs=stmt.executeQuery("select * from emp");
2.
3. while(rs.next()){
4. System.out.println(rs.getInt(1)+" "+rs.getString(2));
5. }
```

---

5) Close the connection object

closing connection object statement and ResultSet will be closed automatically. The close() :hod of Connection interface is used to close the connection.

Syntax of close() method

```
1. public void close()throws SQLException
```

Example to close connection

```
1. con.close();
```

Example Program

```
1. import java.sql.*;
2. class OracleCon{
3. public static void main(String args[]){
4. try{
5. //step1 load the driver class
6. Class.forName("oracle.jdbc.driver.OracleDriver");
7.
8. //step2 create  the connection object
9. Connection con=DriverManager.getConnection(
10. "jdbc:oracle:thin:@localhost:1521:xe","system","oracle");
11.
12. //step3 create the statement object
13. Statement stmt=con.createStatement();
14.
```

```
15. //step4 execute query
16. ResultSet rs=stmt.executeQuery("select * from emp");
17. while(rs.next())
18. System.out.println(rs.getInt(1)+"  "+rs.getString(2)+"  "+rs.getString(3));
19.
20. //step5 close the connection object
21. con.close();
22.
23. }catch(Exception e){ System.out.println(e);}
24.
25. }
26. }
```