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

Do You Know ?

* How to perform connection-oriented Socket Programming in networking ?
* How to display the data of any online web page ?
* How to get the IP address of any host name e.g. www.google.com ?
* How to perform connection-less socket programming in networking ?

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

Java Networking Terminology

The widely used Java networking terminologies are given below:

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

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

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 communications.

Visit next page for Java socket programming.

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.

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
* InterfaceAddress
* JarURLConnection
* MulticastSocket
* InetSocketAddress
* InetAddress
* Inet4Address
* Inet6Address
* IDN
* HttpURLConnection
* HttpCookie
* NetPermission
* NetworkInterface
* PasswordAuthentication
* Proxy
* ProxySelector
* ResponseCache
* SecureCacheResponse
* ServerSocket
* Socket
* SocketAddress
* SocketImpl
* SocketPermission
* StandardSocketOptions
* URI
* URL
* URLClassLoader
* URLConnection
* URLDecoder
* URLEncoder
* URLStreamHandler

**List of interfaces available in java.net package:**

* ContentHandlerFactory
* CookiePolicy
* CookieStore
* DatagramSocketImplFactory
* FileNameMap
* SocketOption<T>
* SocketOptions
* SocketImplFactory
* URLStreamHandlerFactory
* ProtocolFamily

What we will learn in Networking Tutorial

* Networking and Networking Terminology
* Socket Programming (Connection-oriented)
* InetAddress class
* DatagramSocket and DatagramPacket (Connection-less)

Java Socket Programming

Java Socket programming is used for communication between the applications running on different JRE.

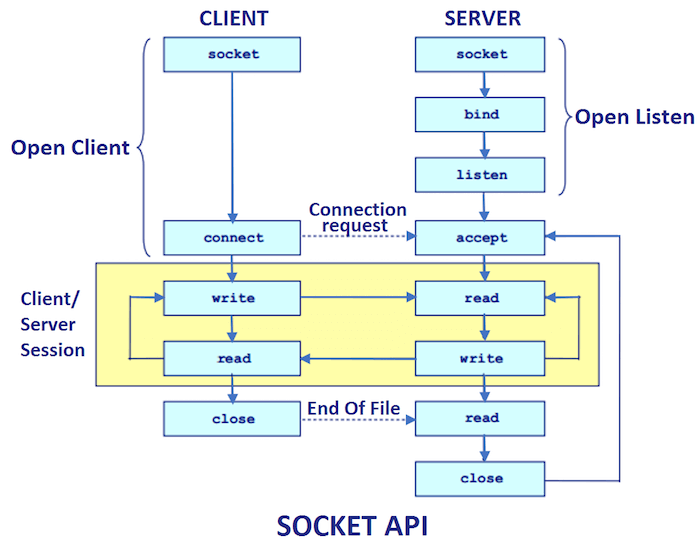
Java Socket programming can be connection-oriented or connection-less.

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.

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.



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 |

Important methods

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

**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

**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);

Let's see a simple of Java socket programming where client sends a text and server receives and prints it.

*File: MyServer.java*

1. **import** java.io.\*;
2. **import** java.net.\*;
3. **public** **class** MyServer {
4. **public** **static** **void** main(String[] args){
5. **try**{
6. ServerSocket ss=**new** ServerSocket(6666);
7. Socket s=ss.accept();//establishes connection
8. DataInputStream dis=**new** DataInputStream(s.getInputStream());
9. String  str=(String)dis.readUTF();
10. System.out.println("message= "+str);
11. ss.close();
12. }**catch**(Exception e){System.out.println(e);}
13. }
14. }

*File: MyClient.java*

1. **import** java.io.\*;
2. **import** java.net.\*;
3. **public** **class** MyClient {
4. **public** **static** **void** main(String[] args) {
5. **try**{
6. Socket s=**new** Socket("localhost",6666);
7. DataOutputStream dout=**new** DataOutputStream(s.getOutputStream());
8. dout.writeUTF("Hello Server");
9. dout.flush();
10. dout.close();
11. s.close();
12. }**catch**(Exception e){System.out.println(e);}
13. }
14. }

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.

# 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.

An IP address is represented by 32-bit or 128-bit unsigned number. An instance of InetAddress represents the IP address with its corresponding host name. There are two types of addresses: Unicast and Multicast. The Unicast is an identifier for a single interface whereas Multicast is an identifier for a set of interfaces.

Moreover, InetAddress has a cache mechanism to store successful and unsuccessful host name resolutions.

## IP Address

* An IP address helps to identify a specific resource on the network using a numerical representation.
* Most networks combine IP with TCP (Transmission Control Protocol). It builds a virtual bridge among the destination and the source.

There are two versions of IP address:

### 1. IPv4

IPv4 is the primary Internet protocol. It is the first version of IP deployed for production in the ARAPNET in 1983. It is a widely used IP version to differentiate devices on network using an addressing scheme. A 32-bit addressing scheme is used to store 232addresses that is more than 4 million addresses.

**Features of IPv4:**

* It is a connectionless protocol.
* It utilizes less memory and the addresses can be remembered easily with the class based addressing scheme.
* It also offers video conferencing and libraries.

### 2. IPv6

IPv6 is the latest version of Internet protocol. It aims at fulfilling the need of more internet addresses. It provides solutions for the problems present in IPv4. It provides 128-bit address space that can be used to form a network of 340 undecillion unique IP addresses. IPv6 is also identified with a name IPng (Internet Protocol next generation).

**Features of IPv6:**

* It has a stateful and stateless both configurations.
* It provides support for quality of service (QoS).
* It has a hierarchical addressing and routing infrastructure.

## TCP/IP Protocol

* TCP/IP is a communication protocol model used connect devices over a network via internet.
* TCP/IP helps in the process of addressing, transmitting, routing and receiving the data packets over the internet.
* The two main protocols used in this communication model are:
  1. TCP i.e. Transmission Control Protocol. TCP provides the way to create a communication channel across the network. It also helps in transmission of packets at sender end as well as receiver end.
  2. IP i.e. Internet Protocol. IP provides the address to the nodes connected on the internet. It uses a gateway computer to check whether the IP address is correct and the message is forwarded correctly or not.

|  |  |
| --- | --- |
| **Method** | **Description** |
| public static InetAddress getByName(String host) throws UnknownHostException | It returns the instance of InetAddress containing  LocalHost IP and name. |
| public static InetAddress getLocalHost() 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. |

## Java InetAddress Class Methods

## Example of Java InetAddress Class

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

**InetDemo.java**

1. **import** java.io.\*;
2. **import** java.net.\*;
3. **public** **class** InetDemo{
4. **public** **static** **void** main(String[] args){
5. **try**{
6. InetAddress ip=InetAddress.getByName("www.javatpoint.com");
8. System.out.println("Host Name: "+ip.getHostName());
9. System.out.println("IP Address: "+ip.getHostAddress());
10. }**catch**(Exception e){System.out.println(e);}
11. }
12. }

[**Test it Now**](https://www.javatpoint.com/opr/test.jsp?filename=InetDemo)

**Output:**

Host Name: www.javatpoint.com

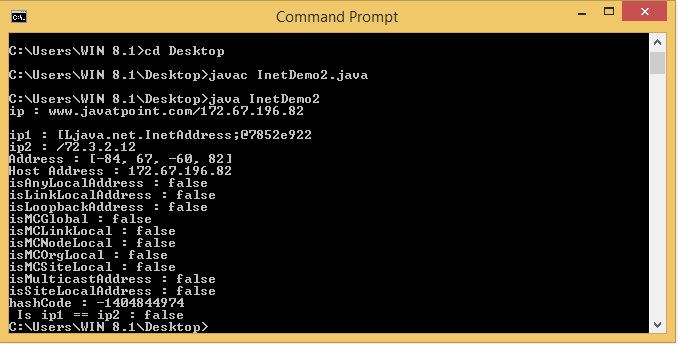
IP Address: 172.67.196.82

### Program to demonstrate methods of InetAddress class

**InetDemo2.java**

1. **import** java.net.Inet4Address;
2. **import** java.util.Arrays;
3. **import** java.net.InetAddress;
4. **public** **class** InetDemo2
5. {
6. **public** **static** **void** main(String[] arg) **throws** Exception
7. {
8. InetAddress ip =  Inet4Address.getByName("www.javatpoint.com");
9. InetAddress ip1[] = InetAddress.getAllByName("www.javatpoint.com");
10. **byte** addr[]={72, 3, 2, 12};
11. System.out.println("ip : "+ip);
12. System.out.print("\nip1 : "+ip1);
13. InetAddress ip2 =  InetAddress.getByAddress(addr);
14. System.out.print("\nip2 : "+ip2);
15. System.out.print("\nAddress : " +Arrays.toString(ip.getAddress()));
16. System.out.print("\nHost Address : " +ip.getHostAddress());
17. System.out.print("\nisAnyLocalAddress : " +ip.isAnyLocalAddress());
18. System.out.print("\nisLinkLocalAddress : " +ip.isLinkLocalAddress());
19. System.out.print("\nisLoopbackAddress : " +ip.isLoopbackAddress());
20. System.out.print("\nisMCGlobal : " +ip.isMCGlobal());
21. System.out.print("\nisMCLinkLocal : " +ip.isMCLinkLocal());
22. System.out.print("\nisMCNodeLocal : " +ip.isMCNodeLocal());
23. System.out.print("\nisMCOrgLocal : " +ip.isMCOrgLocal());
24. System.out.print("\nisMCSiteLocal : " +ip.isMCSiteLocal());
25. System.out.print("\nisMulticastAddress : " +ip.isMulticastAddress());
26. System.out.print("\nisSiteLocalAddress : " +ip.isSiteLocalAddress());
27. System.out.print("\nhashCode : " +ip.hashCode());
28. System.out.print("\n Is ip1 == ip2 : " +ip.equals(ip2));
29. }
30. }

**Output:**



In the above Java code, the various boolean methods of InetAdress class are demonstrated.



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*

1. **import** java.net.\*;
2. **import** java.io.\*;
3. **class** MyServer{
4. **public** **static** **void** main(String args[])**throws** Exception{
5. ServerSocket ss=**new** ServerSocket(3333);
6. Socket s=ss.accept();
7. DataInputStream din=**new** DataInputStream(s.getInputStream());
8. DataOutputStream dout=**new** DataOutputStream(s.getOutputStream());
9. BufferedReader br=**new** BufferedReader(**new** InputStreamReader(System.in));
11. String str="",str2="";
12. **while**(!str.equals("stop")){
13. str=din.readUTF();
14. System.out.println("client says: "+str);
15. str2=br.readLine();
16. dout.writeUTF(str2);
17. dout.flush();
18. }
19. din.close();
20. s.close();
21. ss.close();
22. }}

*File: MyClient.java*

1. **import** java.net.\*;
2. **import** java.io.\*;
3. **class** MyClient{
4. **public** **static** **void** main(String args[])**throws** Exception{
5. Socket s=**new** Socket("localhost",3333);
6. DataInputStream din=**new** DataInputStream(s.getInputStream());
7. DataOutputStream dout=**new** DataOutputStream(s.getOutputStream());
8. BufferedReader br=**new** BufferedReader(**new** InputStreamReader(System.in));
10. String str="",str2="";
11. **while**(!str.equals("stop")){
12. str=br.readLine();
13. dout.writeUTF(str);
14. dout.flush();
15. str2=din.readUTF();
16. System.out.println("Server says: "+str2);
17. }
19. dout.close();
20. s.close();
21. }}

Java DatagramSocket and DatagramPacket

Java DatagramSocket and DatagramPacket classes are used for connection-less socket programming using the UDP instead of TCP.

Datagram

Datagrams are collection of information sent from one device to another device via the established network. When the datagram is sent to the targeted device, there is no assurance that it will reach to the target device safely and completely. It may get damaged or lost in between. Likewise, the receiving device also never know if the datagram received is damaged or not. The UDP protocol is used to implement the datagrams in Java.

Java DatagramSocket class

**Java DatagramSocket** class represents a connection-less socket for sending and receiving datagram packets. It is a mechanism used for transmitting datagram packets over network.`

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 DatagramSocket Class

|  |  |
| --- | --- |
| **Method** | **Description** |
| void bind(SocketAddress addr) | It binds the DatagramSocket to a specific address and port. |
| void close() | It closes the datagram socket. |
| void connect(InetAddress address, int port) | It connects the socket to a remote address for the socket. |
| void disconnect() | It disconnects the socket. |
| boolean getBroadcast() | It tests if SO\_BROADCAST is enabled. |
| DatagramChannel getChannel() | It returns the unique DatagramChannel object associated with the datagram socket. |
| InetAddress getInetAddress() | It returns the address to where the socket is connected. |
| InetAddress getLocalAddress() | It gets the local address to which the socket is connected. |
| int getLocalPort() | It returns the port number on the local host to which the socket is bound. |
| SocketAddress getLocalSocketAddress() | It returns the address of the endpoint the socket is bound to. |
| int getPort() | It returns the port number to which the socket is connected. |
| int getReceiverBufferSize() | It gets the value of the SO\_RCVBUF option for this DatagramSocket that is the buffer size used by the platform for input on the DatagramSocket. |
| boolean isClosed() | It returns the status of socket i.e. closed or not. |
| boolean isConnected() | It returns the connection state of the socket. |
| void send(DatagramPacket p) | It sends the datagram packet from the socket. |
| void receive(DatagramPacket p) | It receives the datagram packet from the socket. |

Java DatagramPacket Class

**Java DatagramPacket** is a message that can be sent or received. It is a data container. 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.

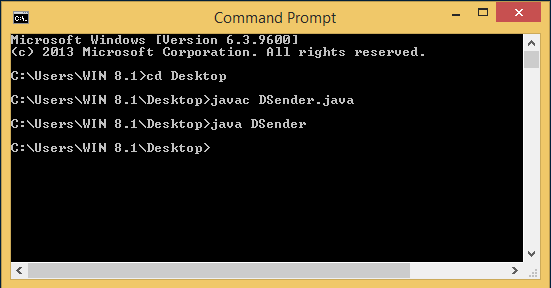
Java DatagramPacket Class Methods

|  |  |
| --- | --- |
| **Method** | **Description** |
| 1) InetAddress getAddress() | It returns the IP address of the machine to which the datagram is being sent or from which the datagram was received. |
| 2) byte[] getData() | It returns the data buffer. |
| 3) int getLength() | It returns the length of the data to be sent or the length of the data received. |
| 4) int getOffset() | It returns the offset of the data to be sent or the offset of the data received. |
| 5) int getPort() | It returns the port number on the remote host to which the datagram is being sent or from which the datagram was received. |
| 6) SocketAddress getSocketAddress() | It gets the SocketAddress (IP address + port number) of the remote host that the packet is being sent to or is coming from. |
| 7) void setAddress(InetAddress iaddr) | It sets the IP address of the machine to which the datagram is being sent. |
| 8) void setData(byte[] buff) | It sets the data buffer for the packet. |
| 9) void setLength(int length) | It sets the length of the packet. |
| 10) void setPort(int iport) | It sets the port number on the remote host to which the datagram is being sent. |
| 11) void setSocketAddress(SocketAddress addr) | It sets the SocketAddress (IP address + port number) of the remote host to which the datagram is being sent. |

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");
9. DatagramPacket dp = **new** DatagramPacket(str.getBytes(), str.length(), ip, 3000);
10. ds.send(dp);
11. ds.close();
12. }
13. }

**Output:**



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. }

**Output:**

