NETWORK PROGRAMMING

CHAPTER 2

CHANDAN GUPTA BHAGAT

https://me.chandanbhagat.com.np

CONTENT

- The InetAddress Class: Creating a new InetAddress Objects
- Getter Methods, Address Types, Testing Reachability and Object Methods
- Inet4Address and Inet6Address
- The Network Interface Class : Factory Method and Getter Method
- Some useful programs: SpamCheck, Processing WebServer Log Files

THE INETADDRESS CLASS

THE INETADDRESS CLASS

- 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.
  - Unicast is an identifier for a single interface
  - 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.
- Versions of IP
  - IPV4
  - IPV6

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 232 addresses 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.

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:
  - 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.
  - 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.

JAVA INETADDRESS CLASS METHODS

Methods	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.

EXAMPLE CODE

https://github.com/chandan-g-bhagat/network-programming > Chapter2

```
package Chapter2;

import java.io.*;
import java.net.*;

public class InetDemo {
    public static void main(String[] args) {
        try {
            InetAddress ip = InetAddress.getByName("chandanbhagat.com.np");

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

```
package Chapter2;

import java.net.Inet4Address;
import java.util.Arrays;
import java.net.InetAddress;

public class InetDemo2 {
    public static void main(String[] arg) throws Exception {
        InetAddress ip = Inet4Address.getByName("chandanbhagat.com.np");
        InetAddress ip1[] = InetAddress.getAllByName("chandanbhagat.com.np");
        byte addr[] = { 72, 3, 2, 12 };
        System.out.println("ip : " + ip);
        System.out.print("\nip1 : " + ip1);
        InetAddress ip2 = InetAddress.getByAddress(addr);
        System.out.print("\nip2 : " + ip2);
        System.out.print("\nAddress : " + Arrays.toString(ip.getAddress()));
        System.out.print("\nHost Address : " + ip.getHostAddress());
        System.out.print("\nisAnyLocalAddress : " + ip.isAnyLocalAddress());
        System.out.print("\nisLinkLocalAddress : " + ip.isLinkLocalAddress());
        System.out.print("\nisLoopbackAddress : " + ip.isLoopbackAddress());
        System.out.print("\nisMCGlobal : " + ip.isMCGlobal());
        System.out.print("\nisMCLinkLocal : " + ip.isMCLinkLocal());
        System.out.print("\nisMCNodeLocal : " + ip.isMCNodeLocal());
        System.out.print("\nisMCOrgLocal : " + ip.isMCOrgLocal());
        System.out.print("\nisMCSiteLocal : " + ip.isMCSiteLocal());
        System.out.print("\nisMulticastAddress : " + ip.isMulticastAddress());
        System.out.print("\nisSiteLocalAddress : " + ip.isSiteLocalAddress());
        System.out.print("\nhashCode : " + ip.hashCode());
        System.out.print("\n Is ip1 == ip2 : " + ip.equals(ip2));
    }
}
```

GETTER METHODS, ADDRESS TYPES, TESTING REACHABILITY AND OBJECT METHODS

GETTER METHODS

public String getHostName()

public String getCanonicalHostName()

public String getAddress()

public String getHostAddress()

GETTER METHODS

public String getHostName()

- This method returns a String that contains the name of the host with the IP address represented by the object
- There is no setHostName() method which means that the package outside cannot change the value of HostName
- It helps making this immutable and thread safe

GETTER METHODS

public String getCanoncialHostName()

- This method is kind of similar but a bit aggressive about contacting DNS.
- It calls DNS if it can and may replace the existing cached hostname
- It is particularly useful when we are starting with the dotted quad IP address rather than hostname

EXAMPLE CODE

https://github.com/chandan-g-bhagat/network-programming > Chapter2

```
package Chapter2;

import java.net.InetAddress;
import java.net.UnknownHostException;

public class ReverseTest {
    public static void main(String[] args) throws
UnknownHostException {
        InetAddress ip = InetAddress.getByName("XXX.XXX.XXX.XXX");
        System.out.println("Host : " + ip.getCanonicalHostName());
    }
}
```

GETTER METHODS

public String getAddress()

- Returns the IP Address
- Returns as an array of bytes in network byte order

EXAMPLE CODE

https://github.com/chandan-g-bhagat/network-programming > Chapter2

```
package Chapter2;

import java.net.InetAddress;

public class Addresses {
    public static void main(String[] args) throws Exception {
        InetAddress ip = InetAddress.getLocalHost();
        byte[] bytes = ip.getAddress();
        if (bytes.length == 4) {
            System.out.println("IPv4");
        } else if (bytes.length == 16) {
            System.out.println("IPv6");
        } else {
            System.out.println("Neither");
        }

    }
}
```

GETTER METHODS

public String getHostAddress()

- Returns the IP Address of localhost

EXAMPLE CODE

https://github.com/chandan-g-bhagat/network-programming > Chapter2

```
package Chapter2;

import java.net.InetAddress;
import java.net.UnknownHostException;

public class Localhost {
    public static void main(String[] args) throws UnknownHostException {
        InetAddress ip = InetAddress.getLocalHost();
        String address = ip.getHostAddress();
        System.out.println("Address : " + address);

    }
}
```

ADDRESS TYPES

public boolean isAnyLocalAddress()

public boolean isLoopbackAddress()

public boolean isLinkLocalAddress()

public boolean isSiteLocalAddress()

public boolean isMulticastAddress()

public boolean isMCGlobal()

public boolean isMCNodeLocal()

public boolean isMCLinkLocal()

public boolean isMCSiteLocal()

public boolean isMCOrgLocal()

ADDRESS TYPES

public boolean isAnyLocalAddress()

- Returns true if the address is a wildcard address, false otherwise
- Useful when the system has multiple network interface (e.g. Ethernet Cards, 802.11 Wifi Adapter)
- In IPV4 the wildcard address is 0.0.0.0
- In IPV6 the wildcard address is 0:0:0:0:0:0:0:0 or ::

ADDRESS TYPES

public boolean isLoopbackAddress()

- Returns true if the address is a loopback address, false otherwise
- The loopback address connects to the same machine in the IP layer without using any physical hardware
- In IPV4 the wildcard address is 127.0.0.1
- In IPV6 the wildcard address is 0:0:0:0:0:0:0:1 or ::1

ADDRESS TYPES

public boolean isLinkLocalAddress()

- Returns true if the address is a IPV6 link-local, false otherwise
- Helps IPV6 network to self configure
- All link local address begin with the 8 bytes FE80:0000:0000:0000, next are filled with the local address often copied from MAC address.

ADDRESS TYPES

public boolean isSiteLocalAddress()

- Returns true if the address is a IPV6 site-local, false otherwise
- Site local address are similar to the link-local address except that they may be forwarded by routers within the site but not beyond that
- All link local address begin with the 8 bytes FEC0:0000:0000:0000, next are filled with the local address often copied from MAC address.

ADDRESS TYPES

public boolean IsMulticastAddress()

- Returns true if the address is a multicast address, false otherwise
- Multicast broadcasts content to all the subscribed computers rather than one.
- In IPV4 multicast address all fall in the range 224.0.0.0 to 239.255.255.255
- In IPV6 they all begin with FF

ADDRESS TYPES

public boolean isMCGlobal()

- Returns true if the address is a global multicast address, false otherwise
- The global multicast address may have the subscriber around the world.
- In IPV4 all multicast address have a global scope
- In IPV6 they all begin with FF0E or FF1E depending on whether the multicast address is a well-known permanently assigned address or a transient address

ADDRESS TYPES

public boolean isMCOrgLocal()

- Returns true if the address is an organization-wide multicast address, false otherwise
- The organization-wide multicast address may have the subscriber within the site of organization.
- In IPV6 they all begin with FF08 or FF18 depending on whether the multicast address is a well-known permanently assigned address or a transient address

ADDRESS TYPES

public boolean isMCSiteLocal()

- Returns true if the address is a site-wide multicast address, false otherwise
- Packets addressed to the site-wide multicast address will only be transmitted within their local site.
- In IPV6 they all begin with FF05 or FF15 depending on whether the multicast address is a well-known permanently assigned address or a transient address

ADDRESS TYPES

public boolean isMCLinkLocal()

- Returns true if the address is a subnet-wide multicast address, false otherwise
- Packets addressed to the subnet-wide multicast address will only be transmitted within their subnet.
- In IPV6 they all begin with FF02 or FF12 depending on whether the multicast address is a well-known permanently assigned address or a transient address

ADDRESS TYPES

public boolean isMCNodeLocal()

- Returns true if the address is a interface-wide multicast address, false otherwise
- Packets addressed to the subnet-wide multicast address are not sent beyond their network interface.
- In IPV6 they all begin with FF01 or FF11 depending on whether the multicast address is a well-known permanently assigned address or a transient address

TESTING REACHABILITY

public Boolean isReachable(int timeout) throws IOException

public Boolean isReachable(NetworkInterface interface, int ttl, int timeout) throws IOException

OBJECT METHODS

public boolean equals (Object o)

public int hashCode()

public String toString()

INET4ADDRESS AND INET6ADDRESS

INET4ADDRESS AND INET6ADDRESS

public final class Inet4Address extends InetAddress

public final class Inet6Address extends InetAddress

INET4ADDRESS

- This class represents an Internet Protocol version 4 (IPv4) address. *Defined by RFC 790: Assigned Numbers, RFC 1918: Address Allocation for Private Internets, and RFC 2365: Administratively Scoped IP Multicast*
- This class extends the InetAddress class and represents an IPv4 address. It provides methods to interpret and display useful information about IP addresses.
- Methods of this class take input in 4 formats:
  - d.d.d.d : Each of the given values are assigned to 4 bytes of the IP address from left to right.
  - d.d.d : The last part is interpreted as a 16-bit number and assigned to the rightmost 2 bytes as the host address. This is generally used for specifying a class-B address.
  - d.d : The last part is interpreted as a 24-bit number and assigned to the rightmost 3 bytes as the host address. This is generally used for specifying a class-A address.
  - d : The given value is directly stored as a network address without any rearrangement.

INET4ADDRESS

```java
package Chapter2;

import java.net.Inet4Address;
import java.net.InetAddress;
import java.net.UnknownHostException;
import java.util.Arrays;

public class Inet4Add {
    public static void main(String args[]) throws UnknownHostException {
        String url = "www.geeksforgeeks.org";
        Inet4Address ip1 = (Inet4Address) Inet4Address.getByName(url);
        Inet4Address ip2 = (Inet4Address) InetAddress.getByName("www.yahoo.com"
);

        System.out.println("Address : " + Arrays.toString(ip1.getAddress()));
        System.out.println("Host Address : " + ip1.getHostAddress());
        System.out.println("isAnyLocalAddress : " + ip1.isAnyLocalAddress());
        System.out.println("isLinkLocalAddress : " + ip1.isLinkLocalAddress());
        System.out.println("isLoopbackAddress : " + ip1.isLoopbackAddress());
        System.out.println("isMCGlobal : " + ip1.isMCGlobal());
        System.out.println("isMCLinkLocal : " + ip1.isMCLinkLocal());
        System.out.println("isMCNodeLocal : " + ip1.isMCNodeLocal());
        System.out.println("isMCOrgLocal : " + ip1.isMCOrgLocal());
        System.out.println("isMCSiteLocal : " + ip1.isMCSiteLocal());
        System.out.println("isMulticastAddress : " + ip1.isMulticastAddress());
        System.out.println("isSiteLocalAddress : " + ip1.isSiteLocalAddress());
        System.out.println("hashCode : " + ip1.hashCode());
        System.out.println("ip1==ip2 : " + ip1.equals(ip2));
    }
}
```

INET6ADDRESS

- This class represents IPv6 address and extends the InetAddress class. Methods of this class provide facility to represent and interpret IPv6 addresses.
- Methods of this class takes input in the following formats:
- x:x:x:x:x:x:x:x –This is the general form of IPv6 address where each x can be replaced with a 16 bit hexadecimal value of the address. Note that there must be a value in place of every ‘x’ when using this format. For example, 4B0C:0:0:0:880C:99A8:4B0:4411
- When the address contains multiple set of 8 bits as ‘0’, a special format can be used to compress the address. In such cases ‘::’ is replaced in place of 0’s to make the address shorter. For example, the address in previous example can be written as- 4B0C::880C:99A8:4B0:4411
- x:x:x:x:x:x:d.d.d.d –A third format is used when hybrid addressing(IPv6 + IPv4) has to be taken care of. In such cases the first 12 bytes are used for IPv6 addressing and remaining 4 bytes are used for IPv4 addressing. For example, F334::40CB:152.16.24.142
- ::FFFF:d.d.d.d – This type of addressing is known as IPv4-mapped addressing. It is used to aid in the deployment of IPv6 addressing. It allows the use of same structure and socket to communicate via both IPv6 and IPv4 connected network. First 80 bits are filled with 0’s represented by ‘::’. Next 32 bits are all ‘1’ and remaining 32 bits represent the IPv4 address. For example, ::FFFF:152.16.24.123

INET6ADDRESS

```java
package Chapter2;

import java.net.Inet6Address;
import java.net.UnknownHostException;
import java.util.Arrays;

public class Inet6Add {
    public static void main(String[] args) throws UnknownHostException {
        String host = "localhost";
        byte add[] = { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1 };
        Inet6Address ip1 = Inet6Address.getByAddress(host, add, 5);
        Inet6Address ip2 = Inet6Address.getByAddress(null, add, 6);

        System.out.println("Scope Id : " + ip1.getScopeId());
        System.out.println("Scoped Interface : " + ip1.getScopedInterface());
        System.out.println("Address : " + Arrays.toString(ip1.getAddress()));
        System.out.println("Host Address : " + ip1.getHostAddress());
        System.out.println("isAnyLocalAddress : " + ip1.isAnyLocalAddress());
        System.out.println("isLinkLocalAddress : " + ip1.isLinkLocalAddress());
        System.out.println("isLoopbackAddress : " + ip1.isLoopbackAddress());
        System.out.println("isMCGlobal : " + ip1.isMCGlobal());
        System.out.println("isMCLinkLocal : " + ip1.isMCLinkLocal());
        System.out.println("isMCNodeLocal : " + ip1.isMCNodeLocal());
        System.out.println("isMCOrgLocal : " + ip1.isMCOrgLocal());
        System.out.println("isMCSiteLocal : " + ip1.isMCSiteLocal());
        System.out.println("isMulticastAddress : " + ip1.isMulticastAddress());
        System.out.println("isSiteLocalAddress : " + ip1.isSiteLocalAddress());
        System.out.println("hashCode : " + ip1.hashCode());
        System.out.println("ip1==ip2 : " + ip1.equals(ip2));
    }
}
```

THE NETWORK INTERFACE CLASS : FACTORY METHOD AND GETTER METHOD

NETWORK INTERFACE CLASS

- Represents a local IP address
- Can be physical or virtual interface
- Provides methods to enumerate all the local address
- Used to create the sockets, server sockets and so on.
- Used in cases when we want to specifically use a particular interface for transmitting our packet on a system with multiple NICs.

NETWORK INTERFACE CLASS : EXAMPLE

- https://github.com/chandan-g-bhagat/network-programming/blob/main/Chapter2/NetInterface.java

FACTORY METHODS

- public static NetworkInterface getByName(String name) throws SocketException
- public static NetworkInterface getByInetAddress(InetAddress address) throws SocketException
- public static Enumeration getNetworkInterface() throws SocketException

GETBYNAME(STRING NAME)

- public static NetworkInterface getByName(String name) throws SocketException
- The getByName method returns the NetworkInterface object representing the network interface with the particular name
- No interface associated with that name, it returns null.

```
1    try {
2        NetworkInterface ni = NetworkInterface.getByName("eth0");
3        if (ni == null) {
4            System.out.println("Interface not found");
5        }
6    } catch (SocketException e) {
7        System.out.println(e);
8    }
```

GETBYINETADDRESS(INETADDRESS ADDRESS)

- public static NetworkInterface getByInetAddress(InetAddress address) throws SocketException
- The getByInetAddress method returns a NetworkInterface object bound to the IP

```java
try {
    InetAddress local=InetAddress.getByName("127.0.0.1");
    NetworkInterface ni=NetworkInterface.getByInetAddress(local);
    if(ni==null){
        System.out.println("No Network Interface Found");
    }
    else{
        System.out.println("Network Interface Found");
    }
} catch (SocketException e) {
    System.out.println(e);
} catch(UnknownHostException e){
    System.out.println(e);
}
```

GETNETWORKINTERFACE()

- public static Enumeration getNetworkInterface() throws SocketException

```
1  Enumeration<NetworkInterface> interfaces = NetworkInterface.
   getNetworkInterfaces();
2      while (interfaces.hasMoreElements()) {
3          NetworkInterface ni = interfaces.nextElement();
4          System.out.println("Interface Name: " + ni.getDisplayName());
5      }
```

GETTER METHODS

- public Enumeration getInetAddresses()
- public String getName()

SOME USEFUL PROGRAMS

SPAM CHECK

- A number of services monitor spammers, and inform clients whether a host attempting to connect to them is a known spammer or not. These real-time blackhole lists need to respond to queries extremely quickly, and process a very high load. Thousands, maybe millions, of hosts query them repeatedly to find out whether an IP address attempting a connection is or is not a known spammer.SpamCheck
- The nature of the problem requires that the response be fast, and ideally it should be cacheable. Furthermore, the load should be distributed across many servers, ideally ones located around the world. Although this could conceivably be done using a web server, SOAP, UDP, a custom protocol, or some other mechanism, this service is in fact cleverly implemented using DNS and DNS alone.
- To find out if a certain IP address is a known spammer, reverse the bytes of the address, add the domain of the blackhole service, and look it up. If the address is found, it's a spammer. If it isn't, it's not. For instance, if you want to ask sbl.spamhaus.org if 207.87.34.17 is a spammer, you would look up the hostname 17.34.87.207.sbl.spamhaus.org. (Note that despite the numeric component, this is a hostname ASCII string, not a dotted quad IP address.)

SPAM CHECK

- If the DNS query succeeds (and, more specifically, if it returns the address 127.0.0.2), then the host is known to be a spammer. If the lookup fails-that is, it throws an UnknownHostException-it isn’t.
- If you use this technique, be careful to stay on top of changes to blackhole list policies and addresses. For obvious reasons, blackhole servers are frequent targets of DDOS and other attacks, so you want to be careful that if the blackhole server changes its address or simply stops responding to any queries, you don't begin blocking all traffic.
- Further note that different blackhole lists can follow slightly different protocols. For example, a few lis return 127.0.0.1 for spamming IPs instead of 127.0.0.2.

PROCESSING WEB SERVER LOGFILES

- Web server logs track the hosts that access a website. By default, the log reports the IP addresses of the sites that connect to the server. However, you can often get more information from the names of those sites than from their IP addresses.
- Most web servers have an option to store hostnames instead of IP addresses but this can hurt performance because the server needs to make a DNS request for each hit. It is much more efficient to log the IP addresses and convert them to hostnames at a later time, when the server isn't busy or even on another machine completely.
- Example: a program called Weblog that reads a web server logfile and prints each line with IP addresses converted to hostnames.
- Most web servers have standardized on the common logfile format. A typical line in the common logfile format looks like this:
  285.160.186.76 unknown [17/Jun/2020:22:53:58 -0500]"GET /bgs/greenbg.gif HTTP 1.0 200 58

PROCESSING WEB SERVER LOGFILES

- This line indicates that a web browser at IP address 205.160.186.76 requested the file /bg/greenbg.gif from 35 this web server at 11:53 PM (and 58 seconds) on June 17, 2020. The file was found (response code 200) and 50 bytes of data were successfully transferred to the browser.
- The first field is the IP address or, if DNS resolution is turned on, the hostname from which the connection was made. This is followed by a space. Therefore, for our purposes, parsing the logfile is easy: everything before the first space is the IP address, and everything after it does not need to be changed.
- The dotted quad format IP address is converted into a hostname using the usual methodsof java.net.InetAddress.

THANK YOU

YOU CAN FIND THIS SLIDES IN

https://chandanbhagat.com.np/docs/network-programming/

AND CODES IN

https://github.com/chandan-g-bhagat/network-programming