PHẦN II. KỸ THUẬT LẬP TRÌNH MẠNG VỚI SOCKET

CHƯƠNG II

LẬP TRÌNH ỨNG DỤNG MẠNG VỚI SOCKET

I. GIỚI THIỆU CHUNG

Lập trình ứng dụng mạng với socket là kỹ thuật hiện nay được sử dụng cực kỳ phổ biến trong thực tế. Các ngôn ngữ lập trình mạng hầu hết đều có thư viện hỗ trợ lập trình với socket như: Ngôn ngữ c/c^{++} có thư viện socket, VC^{++} có , VB có thư viện WinSock, C# có thư viện system.socket...Trong Java các lớp thư viện hỗ trợ lập trình với socket hầu hết nằm trong gói java.net. Khi phát triển các ứng dụng mạng thì java và .NET hỗ trợ rất mạnh đối với socket sử dụng giao thức TCP(TCPsocket) và UDP(UDPsocket), nhưng lập trình Raw socket với java thì cực kỳ phức tạp. Chính vì vậy, khi lập trình các ứng dụng tiện ích mạng như chương trình ping, tracer,.. hoặc các ứng dụng can thiệt sâu hệ thống mạng mà sử dụng raw socket thì tốt nhất sử dụng ngôn ngữ C/C^{++}(Linux), VC^{++} hoặc .NET(Windows).

Trong chương này chúng tôi sẽ tập trung lập trình ứng dụng mạng sử dụng TCPSocket, UDPSocket và sử dụng ngôn ngữ lập trình Java. Đối với các ứng dụng này, Java hỗ trợ rất mạnh trong các gói java.net, java.nio. Các lớp quan trong nhất trong gói java.net gồm 6 lớp: InetAddress, ServerSocket, Socket, DatagramPacket, DatagramSocket, URL. Với 6 lớp này Java cho phép phát triển tất cả các ứng dụng mạng từ chương trình ứng dụng đơn giản cho đến phức tạp, từ các ứng dụng cỡ nhỏ đến các ứng dụng lớn. Ngoài ra còn một số lớp khác cũng được sử dụng phổ biến như NetworkInterface... Sau đây chúng ta sẽ khảo sát những kỹ thuật lập trình mạng cơ bản nhất sử dụng socket trong Java.

II. LẬP TRÌNH THAO TÁC VỚI ĐỊA CHỈ MÁY TRẠM

1. Lập trình thao tác với địa chỉ IP

1.1. Lớp InetAddress

Java có các lớp quan trọng để thao tác với địa chỉ IP trong gói java.net. Lớp quan trọng nhất là lớp InetAddress. Lớp này cho phép lấy địa chỉ của một máy trạm bất kỳ trên mạng và cho phép dễ dàng hoán chuyển giữa địa chỉ IP và tên của một máy trạm(host). Mỗi đối tượng InetAddress chứa 2 thành phần chính của một máy trạm là hostname và địa chỉ IP của máy trạm đó. Ngoài ra còn có 2 lớp khác kết thừa trực tiếp từ lớp InetAddress dành cho các phiên bản IPv4 và IPv6 là lớp Inet4Address, Inet6Address và 2 lớp khác là lớp SocketAddress , InetSocketAddress liên quan tới địa chỉ socket .

Hình 2.1. Lớp kế thừa từ lớp InetAddress và SocketAddress

Lớp InetAddress được sử dụng phổ biến trong các lớp Socket, ServerSocket, URL, DatagramSocket, DatagramPacket và nó được kế thừa từ lớp Object:

public class InetAddress extends Object implements Serializable

Đặc điểm của lớp InetAddress là lớp không có cấu tử nên không thể tạo ra đối tượng InetAddress bằng toán tử new. Nhưng bù lại, lớp InetAddress có một số phương thức có thuộc tính static cho phép lấy địa chỉ của máy trạm bất kỳ trên mạng, cụ thể là có các phương thức sau:

Tóm tắt các phương thức của lớp InetAddress	
boolean	***equals**(Object obj)* So sánh đối tượng với đối tượng obj
byte[]	***getAddress()*** Trả về địa chỉ IP chứa trong đối tượng InetAddress dạng mảng byte
static InetAddress[]	***getAllByName**(String host)* Trả về mảng địa chỉ của tất cả các máy trạm có cùng tên trên mạng
static InetAddress	***getByAddress**(byte[] addr)* Trả về đối tượng InetAddress tương ứng với địa chỉ IP truyền cho phương thức dưới dạng mảng byte
static InetAddress	***getByAddress**(String host,byte[] addr)* Tạo đối tượng InetAddress dựa trên tên và địa chỉ IP
static InetAddress	***getByName**(String host)* Xác định địa chỉ IP của máy trạm từ tên của máy trạm(host)
String	***getCanonicalHostName()*** Lấy tên miền của địa chỉ IP
String	***getHostAddress()*** Trả về địa chỉ IP chứa trong đối tượng INetAddress là chuỗi dạng a.b.c.d
String	***getHostName()*** Trả về tên máy trạm chưa trong đối tượng
static InetAddress	***getLocalHost()*** Lấy đối tượng InetAddress của máy cục bộ
int	***hashCode()*** Trả về hashcode của địa chỉ IP cục thể
boolean	***isAnyLocalAddress()*** Kiểm tra địa chỉ InetAddress có phải địa chỉ wildcard không?
boolean	***isLinkLocalAddress()***

	Kiểm tra địa chỉ có phải là một địa chỉ link-local hay không.
boolean	***isLoopbackAddress()*** Kiểm tra địa chỉ có phải là địa chỉ Loopback không.
boolean	***isMCGlobal()*** Kiểm tra địa chỉ multicast có phạm vi toàn cục hay không?
boolean	***isMCLinkLocal()*** Kiểm tra địa chỉ multicast có phải là địa chỉ có phạm vi liên kết hay không?
boolean	***isMCNodeLocal()*** Kiểm tra địa chỉ multicast có phải là địa chỉ phạm vi nút mạng hay không?
boolean	***isMulticastAddress()*** Kiểm tra địa chỉ InetAddress có phải là địa chỉ IP multicast hay không.
String	***toString()*** Chuyển địa chỉ IP thành chuỗi.

- Phương thức *getByName():*

Phương thức này có cú pháp sau:

public static InetAddress getByName(String hostName)

throws UnknownHostException

Phương thức này cho phép trả về địa chỉ của một máy trạm bất kỳ trên mạng được chỉ ra bởi tham số hostName. Tham số này có thể PCname, là tên miền DNS hoặc địa chỉ IP. Trong trường hợp không tồn tại máy trạm có tên chỉ ra trên mạng, phương thức ném trả về ngoại lệ *UnknownHostException.* Ví dụ đoạn chương trình sau để lấy địa chỉ của máy trạm có tên miền là www.yahoo.com và hiển thị địa chỉ ra màn hình:

```
try {
  InetAddress address = InetAddress.getByName("www.yahoo.com");
  System.out.println(address);

}
catch (UnknownHostException ex) {
  System.out.println("Could not find www.yahoo.com");
}
```

Lệnh *InetAddress.getByName()* sử dụng được do phương thức *getByName()* có thuộc tính static. Nếu máy trạm với tên miền chỉ ra không tồn tại thì ngoại lệ *UnknownHostException được* ném trả về và được xử lý.

- Phương thức *getAllByName():*

Phương thức này cho phép trả về địa chỉ của tất cả các máy trạm có cùng tên trên mạng dưới dạng là một mảng đối tượng InetAddress. Phương thức có cú pháp sau:

InetAddress[] addresses = InetAddress.getAllByName(String name)

throws UnknownHostException

Ví dụ: Hãy in ra địa chỉ của tất cả các máy trạm trên mạng mà có cùng tên miền www.microsoft.com:

```
//AllAddr.java
import java.net.*;
public class AllAddr{
  public static void main (String[] args) {
    try {
      InetAddress[] addresses =
       InetAddress.getAllByName("www.microsoft.com");
      for (int i = 0; i < addresses.length; i++) {
        System.out.println(addresses[i]);
      }
    }
   catch (UnknownHostException ex) {
    System.out.println("Could not find www.microsoft.com");
   }
  }
}
```

Dịch chạy chương trình trên máy tính có kết nối mạng Internet, kết quả trả về như sau:

```
www.microsoft.com/63.211.66.123
www.microsoft.com/63.211.66.124
www.microsoft.com/63.211.66.131
www.microsoft.com/63.211.66.117
www.microsoft.com/63.211.66.116
www.microsoft.com/63.211.66.107
www.microsoft.com/63.211.66.118
www.microsoft.com/63.211.66.115
www.microsoft.com/63.211.66.110
```

- Phương thức *getLocalHost():*

Phương thức này cho phép trả về địa chỉ của máy cục bộ, nếu không tìm thấy nó cũng ném trả về ngoại lệ tượng tự như phương thức *getByName()*. Nó cũng có cú pháp:

public static InetAddress getLocalHost( ) throws UnknownHostException

Ngoài các phương thức static trên, một số phương thức khác cho phép trả về địa chỉ IP hoặc tên của một máy trạm từ đối tượng InetAddress của máy trạm sau khi đã lấy được địa chỉ của máy trạm. Các phương thức tiêu biểu là:

- Phương thức *getHosName():*Trả về tên máy trạm từ đối tượng InetAddress của máy trạm đó. Cú pháp:

  public String getHostName( )

  Ví dụ: Cho địa chỉ, in ra tên máy trạm:

```
import java.net.*;
public class ReverseTest {
  public static void main (String[] args) {
    try {
      InetAddress ia = InetAddress.getByName("208.201.239.37");
      System.out.println(ia.getHostName( ));
    }
    catch (Exception ex) {
      System.err.println(ex);
    }
  }
}
```

- Phương thức *getHostAddress():* Trả về địa chỉ IP của máy trạm từ đối tượng InetAddress tương ứng là chuỗi địa chỉ dạng a.b.c.d. Phương thức có cú pháp:

  public String getHostAddress( )

  Ví dụ: In ra địa chỉ IP của máy cục bộ

```
import java.net.*;
public class MyAddress {
  public static void main(String[] args) {
    try {
      InetAddress me = InetAddress.getLocalHost( );
      String dottedQuad = me.getHostAddress( );
      System.out.println("My address is " + dottedQuad);
    }
    catch (UnknownHostException ex) {
      System.out.println("I'm sorry. I don't know my own address.");
    }
  }
}
```

- Phương thức *getAddress():* Trả về địa chỉ IP của máy trạm từ đối tượng InetAddress của máy trạm tương ứng dưới dạng mảng byte. Phương thức có cú pháp:

  public byte[] getAddress( )

Ví dụ: Phương thức getVersion() lấy phiên bản địa chỉ IP của máy cục bộ:

```
import java.net.*;
```

```
public class AddressTests {
  public static int getVersion(InetAddress ia) {
    byte[] address = ia.getAddress( );
    if (address.length == 4) return 4;
    else if (address.length == 16) return 6;
    else return -1;
  }
```

Lưu ý: Khi in ra các byte địa chỉ IP, nếu giá trị của byte địa chỉ mà vượt qua 127 thi phải cộng với 256 để ra giá trị đúng(vì kiểu byte chỉ có giá trị trong khoảng từ 0128 đến +127), nếu không nó sẽ có giá trị âm. Ví dụ với mảng address trong ví dụ trên:

```
for(int i=0;i<address.length;i++)
    System.out.println((address[i]>0)?address[i]: (address[i]+256));
```

Các phương thức khác của lớp InetAddress:

public boolean isAnyLocalAddress( ): Phương thức này trả về giá trị true với địa chỉ wildcard, false nếu không phải. Địa chỉ wildcard tương hợp với bất cứ địa chỉ nào của máy cục bộ. Phương thức này quan trọng nếu hệ thống cục bộ có nhiều card giao tiếp mạng, nhất là đối với server và gateway. Trong IPv4, địa chỉ wildcard là 0.0.0.0, trong IPv6 là 0:0:0:0:0:0:0:0.

public boolean isLoopbackAddress( ): Phương thức này kiểm tra một địa chỉ có phải là địa chỉ loopback hay không, nếu không phải trả về false. Địa chỉ loopback kết nối trực tiếp trong máy trạm trong lớp IP mà không sử dụng bất kỳ phần cứng vật lý nào. Với IPv4, địa chỉ loopback là 127.0.0.1, với IPv6 là 0:0:0:0:0:0:0:1.

public boolean isLinkLocalAddress( ): Phương thức này trả về giá trị true nếu một địa chỉ là địa chỉ link-local IPv6, nếu không phải thì trả về giá trị false. Địa chỉ link-local là địa chỉ chỉ được hỗ trợ trong mạng IPv6 để tự cấu hình, tương tự như DHCP trên mạng IPv4 nhưng không cần server. Bộ định tuyến sẽ không cho phép truyền qua các gói tin có địa chỉ này ra khỏi mạng con cục bộ. Tất cả địa chỉ link-local đều bắt đầu với 8 byte:

FE80:0000:0000:0000

8 byte tiếp theo sẽ là địa chỉ cục bộ thường là địa chỉ lấy từ địa chỉ MAC trong thẻ Ethernet(NIC).

public boolean isMulticastAddress( ): Trae về true nếu địa chỉ là địa chỉ multicast, nếu không trả về giá trị false. Trong IPv4, địa chỉ multicast nằm trong dải địa chỉ IP: 224.0.0.0->239.255.255.255(lớp D), trong IPv6 thì chúng được bắt đầu với byte có giá trị FF.

1. 2. Ví dụ sử dụng các phương thức lớp InetAddress

Chương trình sau cho phép sử dụng các phương thức của lớp InetAddresss để hiển thị các đặc trưng của một địa chỉ IP được nhập vào từ trên dòng lệnh. Mã chương trình ví dụ được thể hiện như sau:

```
//IPCharacteristics.java
import java.net.*;
public class IPCharacteristics {
  public static void main(String[] args) {
```

```
try {
 InetAddress address = InetAddress.getByName(args[0]);
 if (address.isAnyLocalAddress( )) {
   System.out.println(address + " is a wildcard address.");
 }
 if (address.isLoopbackAddress( )) {
   System.out.println(address + " is loopback address.");
 }
 if (address.isLinkLocalAddress( )) {
   System.out.println(address + " is a link-local address.");
 }
 else if (address.isSiteLocalAddress( )) {
   System.out.println(address + " is a site-local address.");
 }
 else {
   System.out.println(address + " is a global address.");
 }
 if (address.isMulticastAddress( )) {
   if (address.isMCGlobal( )) {
     System.out.println(address + " is a global multicast address.");
   }
   else if (address.isMCOrgLocal( )) {
     System.out.println(address
      + " is an organization wide multicast address.");
   }
   else if (address.isMCSiteLocal( )) {
     System.out.println(address + " is a site wide multicast
                        address.");
   }
   else if (address.isMCLinkLocal( )) {
     System.out.println(address + " is a subnet wide multicast
                        address.");
   }
   else if (address.isMCNodeLocal( )) {
     System.out.println(address
      + " is an interface-local multicast address.");
   }
   else {
     System.out.println(address + " is an unknown multicast
                       address type.");
   }
 }
 else {
```

```
        System.out.println(address + " is a unicast address.");
      }
    }
    catch (UnknownHostException ex) {
      System.err.println("Could not resolve " + args[0]);
    }
  }
}
```

Sau khi biên dịch chương trình, chạy chương trình với lệnh:

java *IPCharacteristics* <*addresss*> [Enter]

III. LẬP TRÌNH ỨNG DỤNG MẠNG VỚI TCPSOCKET

1. Giao thức TCP và cơ chế truyền thông của TCP

<Tham khảo giáo trình mạng máy tính>

2. Một số lớp Java hỗ trợ lập trình với TCPSocket

2.1. Lớp Socket

Lớp Socket dùng để tạo đối tượng socket cho phép truyền thông với giao thức TCP hoặc UDP. (Với giao thức UDP người ta thường sử dụng lớp DatagramSocket thay vì lớp Socket).

2.1.1. Các cấu tử:

- *public Socket(String host, int port)*

  throws UnknownHostException, IOException

Cấu tử này cho phép tạo ra đối tượng Socket truyền thông với giao thức TCP và thực hiện kết nối với máy trạm từ xa có địa chỉ và số cổng được chỉ ra bởi tham số host và port tương ứng. Tham số host có thể là tên máy trạm, tên miền hoặc địa chỉ IP. Nếu không tìm thấy máy trạm từ xa hoặc đối tượng Socket không được mở thì nó ném trả về ngoại lệ *UnknownHostException* hoặc *IOException.* Ví dụ đoạn chương trình sau cho phép mở socket và kết nối tới máy trạm từ xa có tên miền *www.yahoo.com* và số cổng là 80.

```
try {
  Socket toYahoo = new Socket("www.yahoo.com", 80);
  // Hoạt động gửi /nhận dữ liệu
}
catch (UnknownHostException ex) {
  System.err.println(ex);
}
catch (IOException ex) {

  System.err.println(ex);
}
```

- *public Socket(InetAddress host, int port) throws IOException*

Cấu tử này tương tự như cấu tử trên, nhưng tham số thứ nhất là đối tượng InetAddress của máy trạm từ xa. Đối tượng InetAddress của máy trạm từ xa có thể lấy được bằng phương thức getByName() của lớp InetAddress.

- *public Socket(String host, int port, InetAddress interface, int localPort) throws IOException, UnknownHostException*

Cấu tử này cho phép tạo ra đối tượng Socket và kết nối với máy trạm từ xa. Hai tham số đầu là tên và số cổng của máy trạm từ xa, 2 tham số sau là giao tiếp mạng vật lý(NIC) hoặc ảo và số cổng được sử dụng trên máy cục bộ. Nếu số cổng cục bộ localPort mà bằng 0 thì Java sẽ chọn sử dụng một số cổng cho phép ngẫu nhiên trong khoảng 1024 đến 65535.

- *public Socket(InetAddress host, int port, InetAddress interface, int localPort) throws IOException*

Tương tự như cấu tử trên, nhưng tham số thứ nhất là đối tượng InetAddress của máy trạm từ xa.

- *protected Socket( )*

Cấu tử này tạo đối tượng socket mà không kết nối với máy trạm từ xa. Cấu tử này được sử dụng khi chương trình có các socket lớp con.

2.1.2. Một số phương thức quan trọng của lớp Socket

- *public InetAddress getInetAddress( ):* Phương thức cho phép trả về địa chỉ của máy trạm từ xa hiện đang kết nối với socket.
- *public int getPort( ):* Trả về số cổng trên máy trạm từ xa mà hiện đang kết nối với socket.
- *public int getLocalPort( )*: Trả về số cổng trên máy cục bộ
- *public InputStream getInputStream( ) throws IOException:* Trả về luồng nhập của socket là đối tượng InputStream.
- *public OutputStream getOutputStream( ) throws IOException*: Trả về luồng xuất của socket là đối tượng OutputStream.
- *public void close( ) throws IOException*: Đóng socket

2.1.3. Thiết đặt các tuỳ chọn Socket

Tuỳ chọn socket chỉ ra làm thế nào lớp Java Socket có thể gửi /nhận dữ liệu trên native socket. Socket két có các tuỳ chọn sau:

- TCP_NODELAY
- SO_BINDADDR
- SO_TIMEOUT
- SO_LINGER
- SO_SNDBUF (Java 1.2 and later)
- SO_RCVBUF (Java 1.2 and later)
- SO_KEEPALIVE (Java 1.3 and later)
- OOBINLINE (Java 1.4 and later)

Để thiết lập các tuỳ chọn và trả về trạng thái các tuỳ chọn, lớp socket có các phương thức tương ứng. Ví dụ để thiết đặt và trả về trạng thái tuỳ chọn TCP_NODELAY, lớp Socket có các phương thức sau:

public void setTcpNoDelay(boolean on) throws SocketException

public boolean getTcpNoDelay( ) throws SocketException

2.2. Lớp ServerSocket

Lớp ServerSocket cho phép tạo đối tượng socket phía server và truyền thông với giao thức TCP. Sau khi được tạo ra, nó được đặt ở trạng thái lắng nghe(trạng thái thụ động) chờ tín hiệu kết nới gửi tới từ client.

2.2.1 Các cấu tử

- *public ServerSocket(int port) throws BindException, IOException*

Cấu tử này cho phép tạo ra đối tượng ServerSocket với số cổng xác định được chỉ ra bởi tham số port. Nếu số cổng port=0 thì nó cho phép sử dụng một số cổng cho phép nào đó(anonymous port). Cấu tử sẽ ném trả về ngoại lệ khi socket không thể tạo ra được. Socket được tạo bởi cấu tử này cho phép đáp ứng cực đại tới 50 kết nối đồng thời.

- *public ServerSocket(int port, int queueLength)*

  throws IOException, BindException

Tương tự như cấu tử trên nhưng cho phép chỉ ra số kết nối cực đại mà socket có thể đáp ứng đồng thời bởi tham số queueLenth.

- *public ServerSocket( ) throws IOException*

Cấu tử này cho phép tạo đối tượng ServerSocket nhưng không gắn kết thực sự socket với một số cổng cụ thể nào cả. Và như vậy nó sẽ không thể chấp nhận bất cứ kết nối nào gửi tới. Nó sẽ được gắn kết địa chỉ sau sử dụng phương thức bind(). Ví dụ:

```
ServerSocket ss = new ServerSocket( );
// set socket options...
SocketAddress  http = new InetSocketAddress(80);
ss.bind(http);
```

2.2.2. Phương thức

- Phương thức accept()

Phương thức này có cú pháp sau:

public Socket accept( ) throws IOException

Phương thức này khi thực hiện nó đặt đối tượng ServerSocket ở trạng thái "nghe" tại số cổng xác định chờ tín hiệu kết nối gửi đến từ client. Khi có tín hiệu kết nối gửi tới phương thức sẽ trả về đối tượng Socket mới để phục vụ kết nối đó. Khi xảy ra lỗi nhập/xuất, phương thức sẽ ném trả về ngoại lệ IOException. Ví dụ:

```
ServerSocket server = new ServerSocket(5776);

while (true) {

  Socket connection = server.accept( );

  OutputStreamWriter out

   = new OutputStreamWriter(connection.getOutputStream( ));
```

```
    out.write("You've connected to this server. Bye-bye now.\r\n");

  connection.close( );

}
```

- Phương thức close()

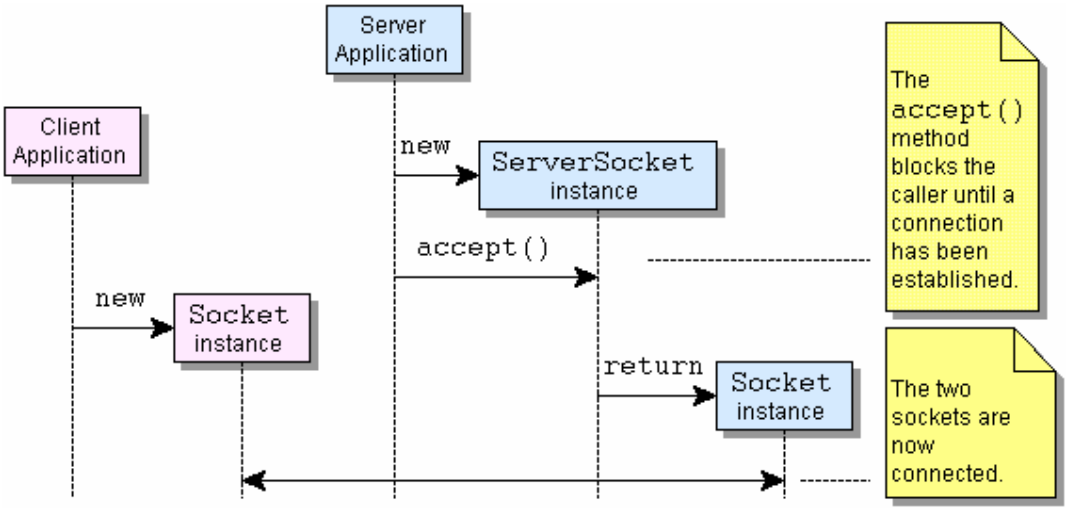
Phương thức close() có cú pháp sau:

public void close( ) throws IOException

Phương thức này cho phép đóng soccket và giải phóng tài nguyên cấp cho socket.

3. Kỹ thuật lập trình truyền thông với giao thức TCP

Trong chương trình ứng dụng mạng xây dựng theo mô hình client/server, để chương trình client và chương trình server có thể truyền thông được với nhau thì mỗi phía phải thực hiện tối thiểu các thao tác cơ bản sau đây(Hình 2.2):



Hình 2.2. Quá trình khởi tạo truyền thông với TCPSocket

3.1. Chương trình phía server:

- Tạo đối tượng ServerSocket với một số hiệu cổng xác định
- Đặt đối tượng ServerSocket ở trạng thái nghe tín hiệu đến kết nối bằng phương thức accept(). Nếu có tín hiệu đến kết nối phương thức accept() tạo ra đối tượng Socket mới để phục vụ kết nối đó.
- Khai báo luồng nhập/xuất cho đối tượng Socket mới(tạo ra ở bước trên). Luồng nhập/xuất có thể là luồng kiểu byte hoặc kiểu char.
- Thực hiện truyền dữ liệu với client thông qua luồng nhập/xuất
- Server hoặc client hoặc cả 2 đóng kết nối
- Server trở về bước 2 và đợi kết nối tiếp theo.

3.2. Chương trình client

- Tạo đối tượng Socket và thiết lập kết nối tới server bằng cách chỉ ra các tham số của server.
- Khai báo luồng nhập/xuất cho Socket. Luồng nhập/xuất có thể là luồng kiểu byte hoặc kiểu char.
- Thực hiện truyền dữ liệu qua mạng thông qua luồng nhập/xuất
- Đóng Socket, giải phóng các tài nguyên khác, kết thúc chương trình nếu cần.

Lưu ý:

- Bình thường chương trình server luôn chạy trước chương trình client
- Một chương trình server có thể phục vụ nhiều client đồng thời hoặc lặp.

Ví dụ:

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

public class EchoClient {
    public static void main(String[] args) throws IOException {

        Socket echoSocket = null;
        PrintWriter out = null;
        BufferedReader in = null;

        try {
            echoSocket = new Socket("taranis", 7);
            out   =   new   PrintWriter(echoSocket.getOutputStream(),
true);
            in = new BufferedReader(new InputStreamReader(

echoSocket.getInputStream()));
        } catch (UnknownHostException e) {
            System.err.println("Don't know about host: taranis.");
            System.exit(1);
        } catch (IOException e) {
            System.err.println("Couldn't get I/O for "
                               + "the connection to: taranis.");
            System.exit(1);
        }

  BufferedReader stdIn = new BufferedReader(
                                    new InputStreamReader(System.in));
  String userInput;

  while ((userInput = stdIn.readLine()) != null) {
      out.println(userInput);
```

```
            System.out.println("echo: " + in.readLine());
        }

        out.close();
        in.close();
        stdIn.close();
        echoSocket.close();
          }}
```

3.3. Luồng nhập/xuất mạng và đọc/ghi dữ liệu qua luồng nhập/xuất

Luồng nhập/xuất mạng cho phép chương trình client và server trao đổi dữ liệu với nhau qua mạng. Luồng nhập/xuất của socket có thể là luồng kiểu byte hoặc kiểu ký tự. Ở đây chúng tôi nêu lên một cách thông dụng nhất tạo luồng kiểu byte và kiểu ký tự để chuong trình thực hiện đọc ghi dữ liệu với mạng.

- **Luồng kiểu byte**

Giả sử đối tượng Socket được tạo ra với biến tham chiếu là cl.

- Với luồng nhập:

+ Tạo luồng nhập cho socket:

InputStream inp=cl.getInputStream();

+ Đọc dữ liệu: Có ba cách

-/ Đọc mỗi lần một byte: *inp.read()*

-/Đọc một khối dữ liệu và cất vào mảng b:

byte b=new byte[1024];

inp.read(b) hoặc inp.read(b,offset, len)

- Với luồng xuất:

+Tạo luồng xuất:

OutputStream outp=cl.getOutputStream();

+ Viết dữ liệu:

-/Viết mỗi lần một byte b: *outp.write(b);*

-/ Viết cả khối dữ liệu chứa trong mảng b kiểu byte:

//byte[] b;

outp.write(b) hoặc *outp.write(b,offset,len);*

- **Luồng kiểu char:**

- Với luồng nhập:

+Tạo luồng nhập:

BufferedReader inp=new BuferedReader(

new InputStreamReader(cl.getInputStream()));

+ Đọc dữ liệu:

-/Đọc từng ký tự: *int ch=inp.read()*

-/ Đọc chuỗi*:* *String s=inp.readLine();*

- Với luồng xuất:

+ Tạo luồng xuất:

PrintWriter outp=new PrintWriter(cl.getOutputStream(),true);

+ Viết dữ liệu:

outp.println(<data>);

4. Một số ví dụ

4.1. Chương trình quét cổng sử dụng Socket

```
//PortScanner.java
import java.net.*;
import java.io.*;
public class PortScanner {
  public static void main(String[] args) {
     String host = "localhost";
    if (args.length > 0) {
      host = args[0];
    }
    try {
      InetAddress theAddress = InetAddress.getByName(host);
      for (int i = 1; i < 65536; i++) {
        Socket connection = null;
        try {
          connection = new Socket(host, i);
          System.out.println("There is a server on port "
           + i + " of " + host);
        }
        catch (IOException ex) {
          // must not be a server on this port
        }
        finally {
          try {
            if (connection != null) connection.close( );
          }
          catch (IOException ex) {}
        }
      } // end for

    } // end try
    catch (UnknownHostException ex) {
      System.err.println(ex);
    }
  }  // end main
```

```
        }  // end PortScanner
```

4.2. Chương trình quét cổng cục bộ dùng lớp ServerSocket

```
import java.net.*;
import java.io.*;
public class LocalPortScanner {
  public static void main(String[] args) {

    for (int port = 1; port <= 65535; port++) {
      try {
        // the next line will fail and drop into the catch block if
        // there is already a server running on the port
        ServerSocket server = new ServerSocket(port);
      }
      catch (IOException ex) {
        System.out.println("There is a server on port " + port + ".");
      } // end catch
    } // end for
  }
}
```

4.3. Chương trình finger client

Finger là một giao thức truyền thẳng theo RFC 1288, client tạo kết nối TCP tới server với số cổng 79 và gửi một truy vấn on-line tới server. Server đáp ứng truy vấn và đóng kết nối.

```
import java.net.*;
import java.io.*;
public class FingerClient {
  public final static int DEFAULT_PORT = 79;
  public static void main(String[] args) {
    String hostname = "localhost";
    try {
      hostname = args[0];
    }
    catch (ArrayIndexOutOfBoundsException ex) {
      hostname = "localhost";
    }
    Socket connection = null;
    try {
      connection = new Socket(hostname, DEFAULT_PORT);
      Writer out = new OutputStreamWriter(
       connection.getOutputStream( ), "8859_1");
      for (int i = 1; i < args.length; i++) out.write(args[i] + " ");
      out.write("\r\n");
      out.flush( );
      InputStream raw = connection.getInputStream( );
```

```
      BufferedInputStream buffer = new BufferedInputStream(raw);
      InputStreamReader in = new InputStreamReader(buffer, "8859_1");
      int c;
      while ((c = in.read( )) != -1) {
 // filter non-printable and non-ASCII as recommended by RFC 1288
      if ((c >= 32 && c < 127) || c == '\t' || c == '\r' || c == '\n')
        {
          System.out.write(c);
        }
      }
    }
    catch (IOException ex) {
      System.err.println(ex);
    }
    finally {
      try {
        if (connection != null) connection.close( );
      }
      catch (IOException ex) {}
    }
  }
}
```

4.4. Chương trình cho phép lấy thời gian server về client.

//TimeClient.java

```
import java.net.*;
import java.io.*;
import java.util.*;
public class TimeClient {
  public final static int    DEFAULT_PORT = 37;
  public final static String DEFAULT_HOST = "time.nist.gov";
  public static void main(String[] args) {
    String hostname = DEFAULT_HOST ;
    int port = DEFAULT_PORT;
   if (args.length > 0) {
      hostname = args[0];
    }
    if (args.length > 1) {
     try {
        port = Integer.parseInt(args[1]);
      }
      catch (NumberFormatException ex) {
        // Stay with the default port
      }
```

```
    }
     // The time protocol sets the epoch at 1900,
    // the Java Date class at 1970. This number
    // converts between them.
      long differenceBetweenEpochs = 2208988800L;
      // If you'd rather not use the magic number, uncomment
    // the following section which calculates it directly.

   /*
    TimeZone gmt = TimeZone.getTimeZone("GMT");
    Calendar epoch1900 = Calendar.getInstance(gmt);

    epoch1900.set(1900, 01, 01, 00, 00, 00);
    long epoch1900ms = epoch1900.getTime( ).getTime( );
    Calendar epoch1970 = Calendar.getInstance(gmt);
    epoch1970.set(1970, 01, 01, 00, 00, 00);
    long epoch1970ms = epoch1970.getTime( ).getTime( );
      long differenceInMS = epoch1970ms - epoch1900ms;
    long differenceBetweenEpochs = differenceInMS/1000;
    */
      InputStream raw = null;
    try {
      Socket theSocket = new Socket(hostname, port);
      raw = theSocket.getInputStream( );
          long secondsSince1900 = 0;
      for (int i = 0; i < 4; i++) {
        secondsSince1900 = (secondsSince1900 << 8) | raw.read( );
      }
      long secondsSince1970
       = secondsSince1900 - differenceBetweenEpochs;
      long msSince1970 = secondsSince1970 * 1000;
      Date time = new Date(msSince1970);
           System.out.println("It is " + time + " at " + hostname);
     }  // end try
    catch (UnknownHostException ex) {
      System.err.println(ex);
    }
    catch (IOException ex) {
      System.err.println(ex);
    }
    finally {
      try {
        if (raw != null) raw.close( );
```

```
      }
      catch (IOException ex) {}
    }
  }  // end main
} // end TimeClient
//TimeServe.java
import java.net.*;
import java.io.*;
import java.util.Date;
public class TimeServer {
 public final static int DEFAULT_PORT = 37;
  public static void main(String[] args) {
   int port = DEFAULT_PORT;
   if (args.length > 0) {
     try {
        port = Integer.parseInt(args[0]);
        if (port < 0 || port >= 65536) {
          System.out.println("Port must between 0 and 65535");
          return;
        }
     }
     catch (NumberFormatException ex) {}
   }
   // The time protocol sets the epoch at 1900,
   // the Date class at 1970. This number
   // converts between them.
   long differenceBetweenEpochs = 2208988800L;
   try {
     ServerSocket server = new ServerSocket(port);
       while (true) {
         Socket connection = null;
         try {
           connection = server.accept( );
           OutputStream out = connection.getOutputStream( );
           Date now = new Date( );
           long msSince1970 = now.getTime( );
           long secondsSince1970 = msSince1970/1000;
           long secondsSince1900 = secondsSince1970
            + differenceBetweenEpochs;
          byte[] time = new byte[4];
     time[0]= (byte) ((secondsSince1900 & 0x00000000FF000000L) >> 24);
     time[1 = (byte) ((secondsSince1900 & 0x0000000000FF0000L) >> 16);
     time[2] = (byte) ((secondsSince1900 & 0x000000000000FF00L) >> 8);
```

```
            time[3] = (byte) (secondsSince1900 & 0x00000000000000FFL);
            out.write(time);
           out.flush( );
          } // end try
          catch (IOException ex) {
          } // end catch
          finally {
           if (connection != null) connection.close( );
         }
        }  // end while
    }  // end try
    catch (IOException ex) {
      System.err.println(ex);
    } // end catch
  } // end main
} // end TimeServer
```

IV. LẬP TRÌNH ỨNG DỤNG MẠNG VỚI UDPSOCKET

1. Giao thức UDP và cơ chế truyền thông của UDP

<Tham khảo giáo trình mạng máy tính>

2. Một số lớp Java hỗ trợ lập trình với UDPSocket

2.1. Lớp DatagramPacket

Lớp này cho phép tạo gói tin truyền thông với giao thức UDP. Lớp này kết thừa trực tiếp từ lớp Object.

public final class DatagramPacket extends Object

Gói tin là đối tượng của lớp này chứa 4 thành phần quan trọng: Địa chỉ, dữ liệu truyền thật sự, kích thước của gói tin và số hiệu cổng chứa trong gói tin.

2.1.1. Cấu tử

Lớp này có các cấu tử tạo gói tin gửi và gói tin nhận khác nhau:

* Cấu tử tạo gói tin nhận từ mạng:

public DatagramPacket(byte[] inBuffer, int length)

Tham số:

- inBuffer: Bộ đệm nhập, chứa dữ liệu của gói tin nhận
- length: kích cỡ của dữ liệu của gói tin nhận, nó thường được xác định bằng lệnh: *length= buffer.length*.

Ví dụ tạo gói tin nhận:

byte[] inBuff=new byte[512];//bộ đệm nhập

DatagramPacket inData=new DatagramPacket(inBuf, inBuff.length);

* Cấu tử tạo gói tinh gửi:

public DatagramPacket(byte[] outBuffer , int length,

InetAddress destination, int port)

Tham số:

- outBuffer: Bộ đệm xuất chưa dữ liệu của gói tin gửi
- length: kích cỡ dữ liệu của gói tin gửi tính theo số byte và thường bằng outBuffer.length.
- destination: Địa chỉ nơi nhận gói tin
- port: Số hiệu cổng đích, nơi nhận gói tin.

Ví dụ:

```
      String s=" Hello World!";
//Bộ đệm xuất và gán dữ liệu cho bộ đệm xuất
      byte[] outBuff=s.getBytes();
//Địa chỉ đích
      InetAddress addrDest=InetAddress.getByName("localhost");
//Số cổng đích
      int   portDest=3456;
//Tạo gói tin gửi
      DatagramPacket outData=new DatagramPacket(outBuff,
                              outBuff.length, addrDest, portDest);
```

2.1.2. Phương thức

- *public InetAddress getAddress( ):* Phương thức này trả về đối tượng InetAddress của máy trạm từ xa chứa trong gói tin nhận.
- *public int getPort( ):* Trả về số hiệu cổng của máy trạm từ xa chứa trong gói tin.
- *public byte[] getData( )*: Trả về dữ liệu chứa trong gói tin dưới dạng mảng byte.
- *public int getLength( ):* Trả về kích cỡ của dữ liệu chưa trong gói tin tính theo số byte.

Tương ứng với 4 phương thức getXXXX..(), lớp DatagramPacket có 4 phương thức setXXXX..() để thiết lập 4 tham số cho gói tin gửi.

2.2. Lớp DatagramSocket

Lớp DatagramSocket cho phép tạo ra đối tượng socket truyền thông với giao thức UDP. Socket này cho phép gửi/nhận gói tin DatagramPacket. Lớp này được khai báo kế thừa từ lớp Object.

public class DatagramSocket extends Object

2.2.1. Các cấu tử (phương thức khởi tạo)

- *public DatagramSocket( ) throws SocketException:*

Cấu tử này cho phép tạo ra socket với số cổng nào đó(anonymous) và thường được sử dụng phía chương trình client. Nếu tạo socket không thành công, nó ném trả về ngoại lệ *SocketException.* Ví dụ:

```
       try {

         DatagramSocket client = new DatagramSocket( );
```

```
      // send packets...

    }

    catch (SocketException ex) {

      System.err.println(ex);

    }
```

- *public DatagramSocket(int port) throws SocketException:*

Cấu tử này cho phép tạo socket với số cổng xác định và chờ nhận gói tín truyền tới. Cấu tử này được sử dụng phía server trong mô hình client/server. Ví dụ chương trình sau sẽ cho phép hiển thị các cổng cục bộ đã được sử dụng:

```
//UDPPortScanner.java
import java.net.*;
public class UDPPortScanner {
  public static void main(String[] args) {
    for (int port = 1024; port <= 65535; port++) {
      try {
        // the next line will fail and drop into the catch block if
        // there is already a server running on port i
       DatagramSocket server = new DatagramSocket(port);
        server.close( );
      }
      catch (SocketException ex) {
        System.out.println("There is a server on port " + port +
".");
      } // end try
    } // end for
  }
  }
```

2.2.2. Các phương thức

- *public void send(DatagramPacket dp) throws IOException:*

Phương thức này cho phép gửi gói tin UDP qua mạng. Ví dụ chương trình sau nhận một chuỗi từ bàn phím, tạo gói tin gửi và gửi tới server.

```
//UDPDiscardClient.java
import java.net.*;
import java.io.*;
public class UDPDiscardClient {
  public final static int DEFAULT_PORT = 9;
  public static void main(String[] args) {
    String hostname;
    int port = DEFAULT_PORT;
```

```
        if (args.length > 0) {
          hostname = args[0];
          try {
                  port = Integer.parseInt(args[1]);
          }
          catch (Exception ex) {
            // use default port
          }
        }
        else {
          hostname = "localhost";
        }
        try {
          InetAddress server = InetAddress.getByName(hostname);
          BufferedReader userInput
           = new BufferedReader(new InputStreamReader(System.in));
          DatagramSocket theSocket = new DatagramSocket( );
          while (true) {
            String theLine = userInput.readLine( );
            if (theLine.equals(".")) break;
            byte[] data = theLine.getBytes( );
            DatagramPacket theOutput
             = new DatagramPacket(data, data.length, server, port);
            theSocket.send(theOutput);
          }  // end while
        }  // end try
        catch (UnknownHostException uhex) {
          System.err.println(uhex);
        }
        catch (SocketException sex) {
          System.err.println(sex);
        }
        catch (IOException ioex) {
          System.err.println(ioex);
        }
      }  // end main
    }
```

- *public void receive(DatagramPacket dp) throws IOException:*

Phương thức nhận gói tin UDP qua mạng. Ví dụ chương trình sau sẽ tạo đối tượng DatagramSocket với số cổng xác định, nghe nhận gói dữ liệu gửi đến, hiển thị nội dung gói tin và địa chỉ, số cổng của máy trạm gửi gói tin.

//UDPDiscardServer.java

```
import java.net.*;
import java.io.*;
public class UDPDiscardServer {
  public final static int DEFAULT_PORT = 9;
  public final static int MAX_PACKET_SIZE = 65507;
  public static void main(String[] args) {
    int port = DEFAULT_PORT;
  byte[] buffer = new byte[MAX_PACKET_SIZE];
    try {
     port = Integer.parseInt(args[0]);
    }
    catch (Exception ex) {
      // use default port
    }
    try {
      DatagramSocket server = new DatagramSocket(port);
      DatagramPacket packet = new DatagramPacket(buffer, buffer.length);
      while (true) {
        try {
          server.receive(packet);
          String s = new String(packet.getData( ), 0, packet.getLength( ));
          System.out.println(packet.getAddress( ) + " at port "
           + packet.getPort( ) + " says " + s);
          // reset the length for the next packet
          packet.setLength(buffer.length);
        }
        catch (IOException ex) {
          System.err.println(ex);
        }
       } // end while
    }  // end try
    catch (SocketException  ex) {
      System.err.println(ex);

    }  // end catch
  }  // end main
}
```

- *public void close( ):* Phương thức đóng socket.

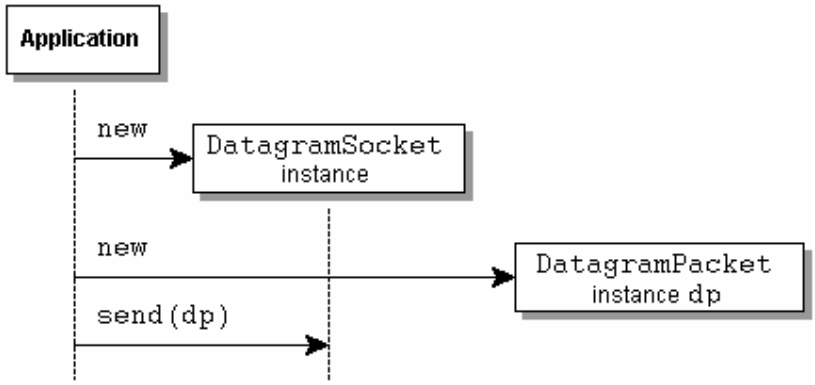
Các phương thức khác thể hiện trong bảng sau:

Một số phương thức của lớp DatagramSocket	
`void`	***bind*** *(SocketAddress addr)*

	Gắn kết DatagramSocket với địa chỉ và số cổng cụ thể
`void`	***connect*** *(InetAddress `address,int port`)* Kết nối socket với địa chỉ máy trạm từ xa
`void`	***connect*** *(SocketAddress `addr`)* Kết nối socket với địa chỉ socket từ xa.
`void`	***disconnect*** *()* Huỷ bỏ kết nối
`boolean`	***isBound*** *()* Trả về trạng thái kết nối của socket.
`boolean`	***isClosed*** *()* Kiểm tra socket đã đóng hay chưa
`boolean`	***isConnected*** *()* Kiểm tra trạng thái kết nối

3. Kỹ thuật lập trình truyền thông với giao thức UDP

Trong mô hình client/server, để chương trình client và server có thể truyền thông được với nhau, mỗi phía phải thực hiện một số thao tác cơ bản sau đây(Hình 2.3)



Hình 2.3. Quá trình khởi tạo truyền thông UDPSocket

3.1. Phía server:

- Tạo đối tượng DatagramSocket với số cổng xác định được chỉ ra
- Khai báo bộ đệm nhập /xuất inBuffer/outBuffer dạng mảng kiểu byte
- Khai báo gói tin nhận gửi inData/outData là đối tượng DatagramPacket.
- Thực hiện nhận/gửi gói tin với phương thức receive()/send()
- Đóng socket, giải phóng các tài nguyên khác, kết thúc chương trình nếu cần, không quay về bước 3.

3.2. Phía client

- Tạo đối tượng DatagramSocket với số cổng nào đó
- Khai báo bộ đệm xuất/nhập outBuffer/inBuffer dạng mảng kiểu byte
- Khai báo gói tin gửi/nhận outData/inData là đối tượng DatagramPacket.
- Thực hiện gửi /nhận gói tin với phương thức send()/receive()
- Đóng socket, giải phóng các tài nguyên khác, kết thúc chương trình nếu cần, không quay về bước 3.

3.3. Một số lưu ý:

- Chương trình server phải chạy trước chương trình client và chương trình client phải gửi gói tin đến server trước. Để từ gói tín nhận được phía server, server mới tách được địa chỉ và số hiệu cổng phía client , từ đó mới tạo gói tin gửi cho client.
- Chương trình server có thể phục vụ nhiều máy khách kiểu lặp.

4. Một số chương trình ví dụ

4.1. Chương trình minh hoạ

```
//UDPEchoClient.java
import java.net.*;
import java.io.*;
public class UDPEchoClient {
  public final static int DEFAULT_PORT = 7;
  public static void main(String[] args) {
    String hostname = "localhost";
    int port = DEFAULT_PORT;
    if (args.length > 0) {
     hostname = args[0];
    }
    try {
      InetAddress ia = InetAddress.getByName(hostname);
      Thread sender = new SenderThread(ia, DEFAULT_PORT);
      sender.start( );
      Thread receiver = new ReceiverThread(sender.getSocket( ));
      receiver.start( );
    }
    catch (UnknownHostException ex) {
      System.err.println(ex);
    }
    catch (SocketException ex) {
      System.err.println(ex);
    }
  }  // end main
}
```

```
//UDPEchoServer.java
import java.net.*;
```

```
import java.io.*;
public class  extends UDPServer {
  public final static int DEFAULT_PORT = 7;
  public UDPEchoServer( ) throws SocketException {
   super(DEFAULT_PORT);
  }
  public void respond(DatagramPacket packet) {
    try {
      DatagramPacket  outgoing  =  new  DatagramPacket(packet.getData(
),
       packet.getLength(  ),  packet.getAddress(  ),  packet.getPort(
));
      socket.send(outgoing);
    }
    catch (IOException ex) {
      System.err.println(ex);
    }
  }
  public static void main(String[] args) {
   try {
     UDPServer server = new UDPEchoServer( );
     server.start( );
  }
   catch (SocketException ex) {
     System.err.println(ex);
   }
  }
}
```

V. LẬP TRÌNH VỚI THẺ GIAO TIẾP MẠNG(NIC)

1. Giới thiệu về thẻ giao tiếp mạng(*network interface card-NIC)*

Thẻ giao tiếp mạng là điểm liên kết giữa máy tính với mạng riêng hoặc mạng công cộng. Giao tiếp mạng nới chung là một thẻ giao tiếp mạng(NIC) nhưng nó cũng có thể không phải giao tiếp vật lý. Mà thay vào đó giao tiếp mạng có thể được thực hiện trong dạng phần mềm. Ví dụ giao tiếp loopback(127.0.0.1 đối với IPv4 và ::1 đối với IPv6) không phải là dạng thiết bị vật lý mà là một phần mềm phỏng theo giao tiếp mạng vật lý. Giao tiếp loopback noi chung được sử dụng trong môi trường thử nghiệm.

2. Lớp NetworkInterface

Lớp này dùng cho cả thẻ giao tiếp vật lý như Ethernet Card hoặc thẻ giao tiếp ảo mà được tạo ra tương tự giống như thẻ giao tiếp vật lý. Lớp NetworkInterface cung cấp các phương thức để liệt kê tất cả các địa chỉ cục bộ và tạo ra đối tượng InetAddress từ chúng. Các đối tượng InetAddress này có thể được sử dụng để tạo các socket, server socket...

Đối tượng NetworkInterface thể hiện phần cứng vật lý hoặc địa chỉ ảo và chúng không thể được xây dựng tuỳ ý. Cũng tương tự như lớp InetAdddress, nó cũng có một số phương thức

có thuộc tính static cho phép trả về đối tượng NetworkInterface gắn kết với bộ giao tiếp mạng cụ thể. Sau đây chúng ta sẽ khảo sát một số phương thức quan trọng của lớp NetworkInterface.

2.1. Các phương thức static

- Phương thức *getByName():*

Cú pháp:

public static NetworkInterface getByName(String name)

throws SocketException

Phương thức này trả về đối tượng NetworkInterface biểu diễn một bộ giao tiếp mạng với tên cụ thể. Nếu không có tên đó thì nó trả về giá trị null. Nếu các tầng mạng nền tảng xẩy ra vấn đề, phương thức trả về ngoại lệ *SocketException.* Dạng tên giao tiếp mạng phụ thuộc vào nền cụ thể. Với hệ điều hành Unix, tên của giao tiếp Ethernet có dạng eth0, eth1,...Địa chỉ loopback cục bộ có thể đặt tên chẳng hạn như "lo". Đối với hệ điều hành Windows, tên là các chuỗi "CE31", "ELX100" mà được lấy từ các nhà cung cấp và mô hình phần cứng trên phần cứng giao tiếp mạng đó. Ví dụ đoạn chương trình sau thực hiện tìm giao tiếp mạng Etthernet cơ sở trên hệ điều hành Unix:

```
try {

  NetworkInterface ni = NetworkInterface.getByName("eth0");

  if (ni == null) {

    System.err.println("No such interface:  eth0" );

  }

}
catch (SocketException ex) {

 System.err.println("Could not list sockets." );
}
```

- Phương thức *getByInetAddress():*

Cú pháp:

public static NetworkInterface getByInetAddress(InetAddress address)

throws SocketException

Phương thức này trả về đối tượng NeworkInterface biểu diễn giao tiếp mạng được gắn với với một địa chỉ IP cụ thể, Nếu không có giao tiếp mạng gắn với địa chỉ IP đó trên máy trạm cục bộ thì nó trả về null. Khi xẩy ra lỗi nó ném trả về ngoại lệ SocketException. ví dụ đoạn chương trình sau minh hoạ cách sử dụng phương thức để tìm giao tiếp mạng đối với địa chỉ loopback cục bộ:

```
 try {
```

```
        InetAddress local = InetAddress.getByName("127.0.0.1");
        NetworkInterface ni = NetworkInterface.getByName(local);
        if (ni == null) {
          System.err.println("That's weird. No local loopback address.");
        }
      }
      catch (SocketException ex) {
        System.err.println("Could not list sockets." );
      }
      catch (UnknownHostException ex) {
        System.err.println("That's weird. No local loopback address.");
      }
```

- Phương thức *getNetworkInterfaces( ):*

Cú pháp:

public static Enumeration getNetworkInterfaces( ) throws SocketException

Phương thức này trả về đối tượng java.util.Enumeration là một danh sách liệt kê tất cả các giao tiếp mạng có trên máy cục bộ. Chương trình ví dụ sau minh hoạ cách sử dụng phương thức để đưa ra một danh sách tất cả các giao tiếp mạng trên máy cục bộ:

```
//InterfaceLister.java
import java.net.*;
import java.util.*;
public class InterfaceLister {
    public static void main(String[] args) throws Exception {
      Enumeration interfaces = NetworkInterface.getNetworkInterfaces( );
  while (interfaces.hasMoreElements( )) {
   NetworkInterface          ni          =          (NetworkInterface)
    interfaces.nextElement( );
        System.out.println(ni);
      }
    }
}
```

2.2. Các phương thức khác:

- *public Enumeration getInetAddresses( )*: Phương thức này trả về đối tượng `java.util.Enumeration` chứa đối tượng InetAddress đối với mỗi địa chỉ IP mà giao tiếp mạng được với nó. Mà mỗi giao tiếp mạng đơn có thể gắn với các địa chỉ IP khác nhau. Ví dụ sau hiển thị tất cả các địa chỉ IP gắn với giao diện mạng eth0:

```
NetworkInterface eth0 = NetworkInterrface.getByName("eth0");
Enumeration addresses = eth0.getInetAddresses( );
while (addresses.hasMoreElements( )) {
  System.out.println(addresses.nextElement( ));
}
```

- *public String getName( )*: Phương thức này trả về tên của đối tượng NetworkInterface cụ thể, chẳng hạn như eth0 hoặc lo.
- *public String getDisplayName( )*:

Phương thức trả về tên "thân thiện" hơn của một giao tiếp mạng cụ thể. Trong mạng Unix, nó trả về chuỗi giống như phương thức *getName(),* Trong mạng Windows, no trả về chuỗi tên "thân thiện" như "Local Area Connection" hoặc "Local Area Connection 2".

Ngoài ra trong lớp NetworkInterface còn định nghĩa các phương thức equals(), hashCode(), toString().

3. Lập trình với giao tiếp mạng(NIC)

Lớp NetworkInterface thể hiện cả 2 kiểu giao diện vật lý và giao tiếp mềm. Lớp này đầy hữu ích đối với các hệ thống multihome có nhiều NIC. Với lớp này, chương trình có thể chỉ ra NIC cho một hoạt động mạng cụ thể.

Để gửi dữ liệu, hệ thống xác định giao tiếp nào sẽ được sử dụng. Nhưng cũng có thể truy vấn hệ thống đối với các giao tiếp phù hợp và tìm một địa chỉ trên giao tiếp muốn sử dụng. Khi chương trình tạo ra một socket và gắn nó với địa chỉ đó, hệ thống sẽ sử dụng giao tiếp được gắn kết đó. Ví dụ:

```
NetworkInterface nif = NetworkInterface.getByName("bge0");
  Enumeration nifAddresses = nif.getInetAddresses();
  Socket soc = new java.net.Socket();
  soc.bind(nifAddresses.nextElement());
  soc.connect(new InetSocketAddress(address, port));
```

Người sử dụng cũng có thể sử dụng NetworkInterface để nhận biết giao tiếp cục bộ mà một nhóm multicast được ghép nối, ví dụ:

```
NetworkInterface nif = NetworkInterface.getByName("bge0");
MulticastSocket() ms = new MulticastSocket();
ms.joinGroup(new InetSocketAddress(hostname, port) , nif);
```

3.1. Lấy các giao tiếp mạng

Lớp NetworkInterface không có cấu tử public. Do đó không thể tạo được đối tượng với toán tử new. Thay vào đó nó có các phương thức static(giống InetAddress) cho phép lấy được các chi tiết giao tiếp từ hệ thống: `getByInetAddress()`, `getByName()` và `getNetworkInterfaces()`. Hai phương thức đầu tiên được sử dụng khi có sẵn địa chỉ IP hoặc tên của giao tiếp mạng cục thể. Phương thức thứ 3, `getNetworkInterfaces( ),` trả về một danh sách đầy đủ các giao tiếp mạng trên máy tính.

Giao tiếp mạng cũng có thể tổ chức theo kiểu phân cấp. Lớp NetworkInterface sử dụng 2 phương thức *getParent()* và *getSubInterface()* đối với cấu trúc giao tiếp mạng phân cấp. Nếu giao tiếp mạng là giao tiếp con, *getParent()* trả về giá trị none-null. Phương thức *getSubInterfaces()* sẽ trả về tất cả các giao tiếp con của giao tiếp mạng. Ví dụ sau đây sẽ hiển thị tên của tất cả các giao tiếp mạng và giao tiếp con(n nếu nó tồn tại) trên một máy:

```
//ListNIFs.java
import java.io.*;
import java.net.*;
import java.util.*;
import static java.lang.System.out;
```

```
public class ListNIFs
{
    public static void main(String args[]) throws SocketException {
        Enumeration<NetworkInterface> nets =
NetworkInterface.getNetworkInterfaces();

        for (NetworkInterface netIf : Collections.list(nets)) {
            out.printf("Display name: %s\n", netIf.getDisplayName());
            out.printf("Name: %s\n", netIf.getName());
            displaySubInterfaces(netIf);
            out.printf("\n");
        }
    }

    static void displaySubInterfaces(NetworkInterface netIf) throws
SocketException {
        Enumeration<NetworkInterface> subIfs =
netIf.getSubInterfaces();

        for (NetworkInterface subIf : Collections.list(subIfs)) {
            out.printf("\tSub Interface Display name: %s\n",
subIf.getDisplayName());
            out.printf("\tSub Interface Name: %s\n",
subIf.getName());
        }
     }
}
```

Kết quả chạy trên máy tính của chúng tôi hiện ra như sau:

```
Display name: bge0
Name: bge0
    Sub Interface Display name: bge0:3
    Sub Interface Name: bge0:3
    Sub Interface Display name: bge0:2
    Sub Interface Name: bge0:2
    Sub Interface Display name: bge0:1
    Sub Interface Name: bge0:1

Display name: lo0
Name: lo0
```

3.2. Lấy danh sách địa chỉ giao tiếp mạng

Một phần thông tin cực kỳ hữu ích mà người sử dụng cần lấy được từ giao tiếp mạng là danh sách địa chỉ IP mà được gắn cho các giao tiếp mạng. Người sử dụng có thể thu được thông tin từ một thể hiện NetworkInterface bằng cách sử dụng một trong 2 phương thức sau: Phương thức *getInetAddresses()* trả về một *Enumeration* của các đối tượng *InetAddress*, còn phương thức *getInterfaceAddresses()* trả về một danh sách của các thể hiện *java.net.InterfaceAddress*. Phương thứcc này được sử dụng khi người sử dụng cần thông tin nhiều hơn về địa chỉ giao tiếp ngoài địa chỉ IP của nó. Ví dụ, khi bạn cần thông tin bổ sung về mặt nạ mạng con và địa chỉ broardcast khi địa chỉ là một địa chỉ IPv4 và chiều dài prefix mạng trong địa chỉ IPv6. Ví dụ sau đây hiển thị danh sách tất cả các giao tiếp mạng và địa chỉ của chúng trên một máy:

```
import java.io.*;
import java.net.*;
import java.util.*;
import static java.lang.System.out;

public class ListNets
{
    public static void main(String args[]) throws SocketException {
```

```
            Enumeration<NetworkInterface> nets =
        NetworkInterface.getNetworkInterfaces();
            for (NetworkInterface netint : Collections.list(nets))
                displayInterfaceInformation(netint);
        }

        static void displayInterfaceInformation(NetworkInterface netint)
        throws SocketException {
            out.printf("Display name: %s\n", netint.getDisplayName());
            out.printf("Name: %s\n", netint.getName());
            Enumeration<InetAddress> inetAddresses =
        netint.getInetAddresses();
            for (InetAddress inetAddress :
        Collections.list(inetAddresses)) {
                out.printf("InetAddress: %s\n", inetAddress);
            }
            out.printf("\n");
         }
        }
```

Kết quả chạy chương trình trên máy tính của chúng tôi như sau:

```
Display name: bge0
Name: bge0
InetAddress: /fe80:0:0:0:203:baff:fef2:e99d%2
InetAddress: /121.153.225.59
Display name: lo0
Name: lo0
InetAddress: /0:0:0:0:0:0:0:1%1
InetAddress: /127.0.0.1
```

3.3. Truy cập các tham số giao tiếp mạng

Người sử dụng có thể truy cập các tham số về giao tiếp mạng ngoài tên và địa chỉ IP gán cho nó. Và chương trình có thể phát hiện giao tiếp mạng đang chạy với phương thức *isUp()*. các phương thức sau chỉ thị kiểu giao tiếp mạng:

- *isLoopback():* chỉ thị giao tiếp mạng là một giao tiếp loopback.
- *isPointToPoint()* chỉ thị nếu giao tiếp là giao tiếp point-to-point.
- *isVirrtual():* chỉ thị nếu giao tiếp là giao tiếp ảo(giao tiếp mềm).

Phương thức *supportsMulticast()* chỉ thị một khi giao tiếp mạng hỗ trợ multicast. Phương thức *getHardwareAddress()* trả về địa chỉ phần cứng vật lý của giao tiếp mạng, địa chỉ MAC, khi nó có khả năng. Phương thức *getMTU()* trả về đơn vị truyền cực đại(MTU) là kích cỡ gói tin lớn nhất. Ví dụ sau mở rộng của ví dụ trên bằng cách thêm các tham số mạng bổ sung:

```
//ListNetsEx.java
import java.io.*;
import java.net.*;
import java.util.*;
import static java.lang.System.out;

public class ListNetsEx
{
    public static void main(String args[]) throws SocketException {
        Enumeration<NetworkInterface> nets =
NetworkInterface.getNetworkInterfaces();
        for (NetworkInterface netint : Collections.list(nets))
            displayInterfaceInformation(netint);
    }

    static void displayInterfaceInformation(NetworkInterface netint) throws
SocketException {
```

```
        out.printf("Display name: %s\n", netint.getDisplayName());
        out.printf("Name: %s\n", netint.getName());
        Enumeration<InetAddress> inetAddresses = netint.getInetAddresses();

        for (InetAddress inetAddress : Collections.list(inetAddresses)) {
            out.printf("InetAddress: %s\n", inetAddress);
        }

        out.printf("Up? %s\n", netint.isUp());
        out.printf("Loopback? %s\n", netint.isLoopback());
        out.printf("PointToPoint? %s\n", netint.isPointToPoint());
        out.printf("Supports multicast? %s\n", netint.supportsMulticast());
        out.printf("Virtual? %s\n", netint.isVirtual());
        out.printf("Hardware address: %s\n",
                   Arrays.toString(netint.getHardwareAddress()));
        out.printf("MTU: %s\n", netint.getMTU());

        out.printf("\n");

    }
}
```

Kết quả chạy chương trình trên máy tính của chúng tôi như sau:

```
Display name: bge0
Name: bge0
InetAddress: /fe80:0:0:0:203:baff:fef2:e99d%2
InetAddress: /129.156.225.59
Up? true
Loopback? false
PointToPoint? false
Supports multicast? false
Virtual? false
Hardware address: [0, 3, 4, 5, 6, 7]
MTU: 1500

Display name: lo0
Name: lo0
InetAddress: /0:0:0:0:0:0:0:1%1
InetAddress: /127.0.0.1
Up? true
Loopback? true
PointToPoint? false
Supports multicast? false
Virtual? false
Hardware address: null
MTU: 8232
```

4. Một số chương trình ví dụ minh hoạ sử dụng lớp NetworkInterface và InetAddress

```
//InetExample.java
 import java.util.Enumeration;
 import java.net.*;
 public class InetExample {
public static void main(String[] args) {
 // Get the network interfaces and associated addresses for this host
try {
```

```
Enumeration<NetworkInterface> interfaceList =
NetworkInterface.getNetworkInterfaces();
if (interfaceList == null) {
System.out.println("--No interfaces found--");
 } else {
while (interfaceList.hasMoreElements()) {
 NetworkInterface iface = interfaceList.nextElement();
 System.out.println("Interface " + iface.getName() + ":");
Enumeration<InetAddress> addrList = iface.getInetAddresses();
if (!addrList.hasMoreElements()) {
System.out.println("\t(No addresses for this interface)");
 }
while (addrList.hasMoreElements()) {
 InetAddress address = addrList.nextElement();
 System.out.print("\tAddress " + ((address instanceof Inet4Address ?
"(v4)"
 : (address instanceof Inet6Address ? "(v6)" : "(?)"))));
 System.out.println(": " + address.getHostAddress());
}
 }
}
} catch (SocketException se) {
 System.out.println("Error getting network interfaces:" +
se.getMessage());
 }
 // Get name(s)/address(es) of hosts given on command line
 for (String host : args) {
 try {
System.out.println(host + ":");
 InetAddress[] addressList = InetAddress.getAllByName(host);
 for (InetAddress address : addressList) {
System.out.println("\t" + address.getHostName() + "/" +
address.getHostAddress());
 }
 } catch (UnknownHostException e) {
System.out.println("\tUnable to find address for " + host);
}
}
}
 }
```

VI. LẬP TRÌNH TRUYỀN THÔNG MULTICAST

1. Giới thiệu truyền thông multicast và lớp MulticastSocket

Trong truyền thông multicast cho phép truyền gói tin tới một nhóm client nhờ sử dụng địa chỉ multicast của lớp D từ địa chỉ 224.0.0.0 đến 239.255.255.255. Truyền thông multicast có nhiều ứng dụng trong thực tế như:

- Videoconferencing
- Usenet news
- Computer configuration

Các địa chỉ multicast:

- 224.0.0.1: Tất cả các hệ thống ở trên mạng con cục bộ
- 224.0.0.2 : Tất cả các router tren mạng con cục bộ.
- 224.0.0.11: Các tác tử di động(agent) trên mạng con cục bộ
- 224.0.1.1 : Giao thức định thời mạng
- 224.0.1.20: Thử nghiệm mà không cho vượt ra khỏi mạng con cục bộ
- 224.2.X.X (Multicast Backbone on the Internet (MBONE)): Được sử dụng cho audio và video quảng bá trên mạng Internet .

Java hỗ trợ lớp MulticastSocket cho phép tạo ra socket thực hiện truyền thông kiểu này. Lớp MulticastSocket được kế thừ từ lớp DatagramSocket

*public class **MulticastSocket** extends DatagramSocket*

MuticastSocket là một DatagramSocket mà thêm khả năng ghép nối gộp nhóm các máy trạm multicast trên mạng Internet. Một nhóm multicast được chỉ ra bởi địa chỉ lớp D và một địa chỉ cổng UDP chuẩn. Lớp MulticastSocket được sử dụng phía bên nhận. Các cấu tử và phương thức của lớp MulticastSocket được trình bày tóm tắt trong bảng sau:

Cấu tử lớp MulticastSocket
MulticastSocket*()* Tạo socket muticast
MulticastSocket*(int port)* Tạo socket muticast và gắn với socket đó một địa chỉ cổng cụ thể.
MulticastSocket*(SocketAddress bindaddr)* Tạo socket muticast và gắn với socket đó một địa chỉ socket cụ thể.

Các phương thức của lớp MulticastSocket	
InetAddress	***getInterface****()* Lấy địa chỉ giao tiếp mạng được sử dụng cho các gói tin multicast
boolean	***getLoopbackMode****()* Lấy chuỗi thiết đặt đối local loopback của gói tin multicast
NetworkInterface	***getNetworkInterface****()* Lấy tập giao tiếp mạng multicast
int	***getTimeToLive****()* Lấy tham số time to live mặc định của các gói tin multicast gửi

	ra socket
byte	***getTTL****()* Lấy tham số time- to -live
void	***joinGroup****(InetAddress mcastaddr)* Ghép nhóm multicast
void	***joinGroup****(SocketAddress mcastaddr, NetworkInterface netIf)* Ghép nhóm multicast cụ thể tại giao tiếp mạng cụ thể
void	***leaveGroup****(InetAddress mcastaddr)* Loại bỏ một nhóm multicast
void	***leaveGroup****(SocketAddress mcastaddr, NetworkInterface netIf)* Loại bỏ một nhóm multicast trên giao tiếp mạng cục bộ được chỉ ra.
void	***send****(DatagramPacket p, byte ttl)* Gửi gói tin
void	***setInterface****(InetAddress inf)* Đặt giao tiếp mạng multicast được sử dụng bởi phương thức mà hành vi của nó bị ảnh hưởng bởi giá trị của gao tiếp mạng.
void	***setLoopbackMode****(boolean disiao tiếp mạngable)* Cho phép hoặc làm mất hiệu lực vòng phản hồi cục bộ của lược đồ dữ liệu multicast
void	***setNetworkInterface****(NetworkInterface netIf)* Chỉ ra giao tiếp mạng để gửi các lược đồ dữ liệu multicast qua
void	***setTimeToLive****(int ttl)* Thiết đặt tham số TTL mặc định cho các gói tin multicast gửi trên MulticastSocket nhằm mục đích điều khiển phạm vi multicast.
void	***setTTL****(byte ttl)* Thiết đặt tham số TTL

Để tạo ra kết nối một nhóm multicast, đầu tiên phải tạo ra đối tượng MulticastSocket với một địa chỉ cổng xác định bằng cách gọi phương thức jointGroup() của lớp MulticastSocket. Ví dụ:

```
// Kết nối một nhóm multicast và gửi lời chào tới nhóm ...
 String msg = "Hello";
 InetAddress group = InetAddress.getByName("228.5.6.7");
 MulticastSocket s = new MulticastSocket(6789);
 s.joinGroup(group);
 DatagramPacket hi = new DatagramPacket(msg.getBytes(), msg.length(),
                          group, 6789);
 s.send(hi);
 // Nhận đáp ứng của chúng
 byte[] buf = new byte[1000];
 DatagramPacket recv = new DatagramPacket(buf, buf.length);
```

```
s.receive(recv);
...
// OK, I'm done talking - leave the group...
s.leaveGroup(group);
```

Khi gửi thông điệp tới group, tất cả các máy trạm phía nhận là các thành viên của nhóm sẽ nhận được gói tin, để loại bỏ nhóm, phương thức leaveGroup() sẽ được gọi.

2. Một số ví dụ gửi/nhận dữ liệu multicast

2.1. Ví dụ gửi dữ liệu multicast

```
import java.net.*;
// Which port should we send to
int port = 5000;
// Which address
String group = "225.4.5.6";
// Which ttl
int ttl = 1;
// Create the socket but we don't bind it as we are only going to send
data
MulticastSocket s = new MulticastSocket();
// Note that we don't have to join the multicast group if we are only
// sending data and not receiving
// Fill the buffer with some data
byte buf[] = byte[10];
for (int i=0; i<buf.length; i++) buf[i] = (byte)i;
// Create a DatagramPacket
DatagramPacket pack = new DatagramPacket(buf, buf.length,
                                    InetAddress.getByName(group), port);
// Do a send. Note that send takes a byte for the ttl and not an int.
s.send(pack,(byte)ttl);
// And when we have finished sending data close the socket
s.close();
```

2.2. Ví dụ nhận dữ liệu multicast

```
import java.net.*;
// Which port should we listen to
int port = 5000;
// Which address
String group = "225.4.5.6";
// Create the socket and bind it to port 'port'.
MulticastSocket s = new MulticastSocket(port);
// join the multicast group
s.joinGroup(InetAddress.getByName(group));
// Now the socket is set up and we are ready to receive packets
// Create a DatagramPacket and do a receive
byte buf[] = byte[1024];
DatagramPacket pack = new DatagramPacket(buf, buf.length);
s.receive(pack);
// Finally, let us do something useful with the data we just received,
// like print it on stdout :-)
System.out.println("Received data from: " + pack.getAddress().toString()
+
                ":" + pack.getPort() + " with length: " +
                pack.getLength());
```

```
System.out.write(pack.getData(),0,pack.getLength());
System.out.println();
// And when we have finished receiving data leave the multicast group
and
// close the socket
s.leaveGroup(InetAddress.getByName(group));
s.close();
```

2.3. Một số ví dụ khác

//MulticastJoin.java

```
import java.net.*;
import java.io.*;
public class MulticastJoin {
public static void main(String [ ] args){
try {
MulticastSocket mSocket = new MulticastSocket(4001);
InetAddress mAddr = InetAddress.getByName("224.0.0.1");
mSocket.joinGroup(mAddr);
byte [ ] buffer = new byte[512];
while (true) {
DatagramPacket dp = new DatagramPacket(buffer,
buffer.length);
mSocket.receive(dp);
String str = new String(dp.getData(), "8859_1");
System.out.println(str);
}//end of while
}//end of try
catch (SocketException se){
System.out.println("Socket Exception : " + se); }
catch (IOException e) { System.out.println("Exception : " + e); }
}//end of main
}// end of class definition
```

//MulticastListener.java

```
import java.net.*;
import java.io.*;
public class MulticastListener {
public static void main( String [ ] args) {
InetAddress mAddr=null;
MulticastSocket mSocket=null;
final int PORT_NUM= 4001;
try {
mAddr = InetAddress.getByName("audionews.mcast.net");
mSocket = new MulticastSocket(PORT_NUM);
String hostname = InetAddress.getLocalHost().getHostName();
byte [ ] buffer = new byte[8192];
mSocket.joinGroup(mAddr);
System.out.println("Listening from " + hostname + " at " +
mAddr.getHostName());
while (true){
DatagramPacket dp = new DatagramPacket(buffer, buffer.length);
mSocket.receive(dp);
String str = new String(dp.getData(), "8859_1");
System.out.println(str);
```

```
    }//end of while
    }
    catch (SocketException se) {
    System.out.println("Socket Exception : " + se);
    }
    catch (IOException e) {
    System.out.println("Exception : " + e);
    }
    finally {
    if (mSocket != null){
    try {
    mSocket.leaveGroup(mAddr);
    mSocket.close();
    }
    catch (IOException e){ }
    }//end of if
    }//end of finally
    }//end of main
}
```

VII. KẾT LUẬN

Trong chương này chúng ta đã nghiên cứu các kỹ thuật lập trình mạng cơ bản sử dụng socket: TCP Socket, UDP Socket. Sau đó chúng ta đã nghiên cứu cách lập trình với địa chỉ mạng, với giao tiếp mạng và kỹ thuật lập trình truyền thông multicacst. Trong chương tiếp theo chúng ta sẽ mở rộng kiến thức trong chương này để phát triển các chương trình server phục vụ đồng thời nhiều chương trình máy khác cũng như tuần tự.