BÀI GIẢNG MÔN HỌC

LẬP TRÌNH MẠNG

Chương 5 :
LẬP TRÌNH SOCKET VỚI TCP

1. Mô hình Client/Server
2. Các kiến trúc Client/Server
3. Mô hình truyền tin socket
4. Socket cho Client
5. Lớp ServerSocket
6. Cài đặt chương trình Client
7. Cài đặt chương trình Server
8. Đa tuyến trong lập trình Java

1. MÔ HÌNH CLIENT/SERVER

+ Thuật ngữ Client/Server xuất hiện vào đầu thập niên 80

+ Dạng phổ biến của mô hình ứng dụng phân tán

+ Mô hình mạng cơ bản nhất hiện nay

+ Đa số các ứng dụng mạng dựa theo mô hình này

+ Một số ứng dụng Client/Server phổ biến

- Email
- FTP
- Web
- …..

1. MÔ HÌNH CLIENT/SERVER

+ Mô hình client/server cung cấp một cách tiếp cận tổng quát để chia sẻ tài nguyên trong các hệ thống phân tán.

+ Cả tiến trình client và tiến trình server đều có thể chạy trên cùng một máy tính.

+ Một tiến trình server có thể sử dụng dịch vụ của một server khác.

+ Mô hình truyền tin client/server hướng tới việc cung cấp dịch vụ.

1. MÔ HÌNH CLIENT/SERVER

Quá trình trao đổi dữ liệu bao gồm:

+Bước 1 : Truyền một yêu cầu từ tiến trình client tới tiến trình server

+ Bước 2 : Yêu cầu được server xử lý

+ Bước 3 : Truyền đáp ứng cho client

1. MÔ HÌNH CLIENT/SERVER

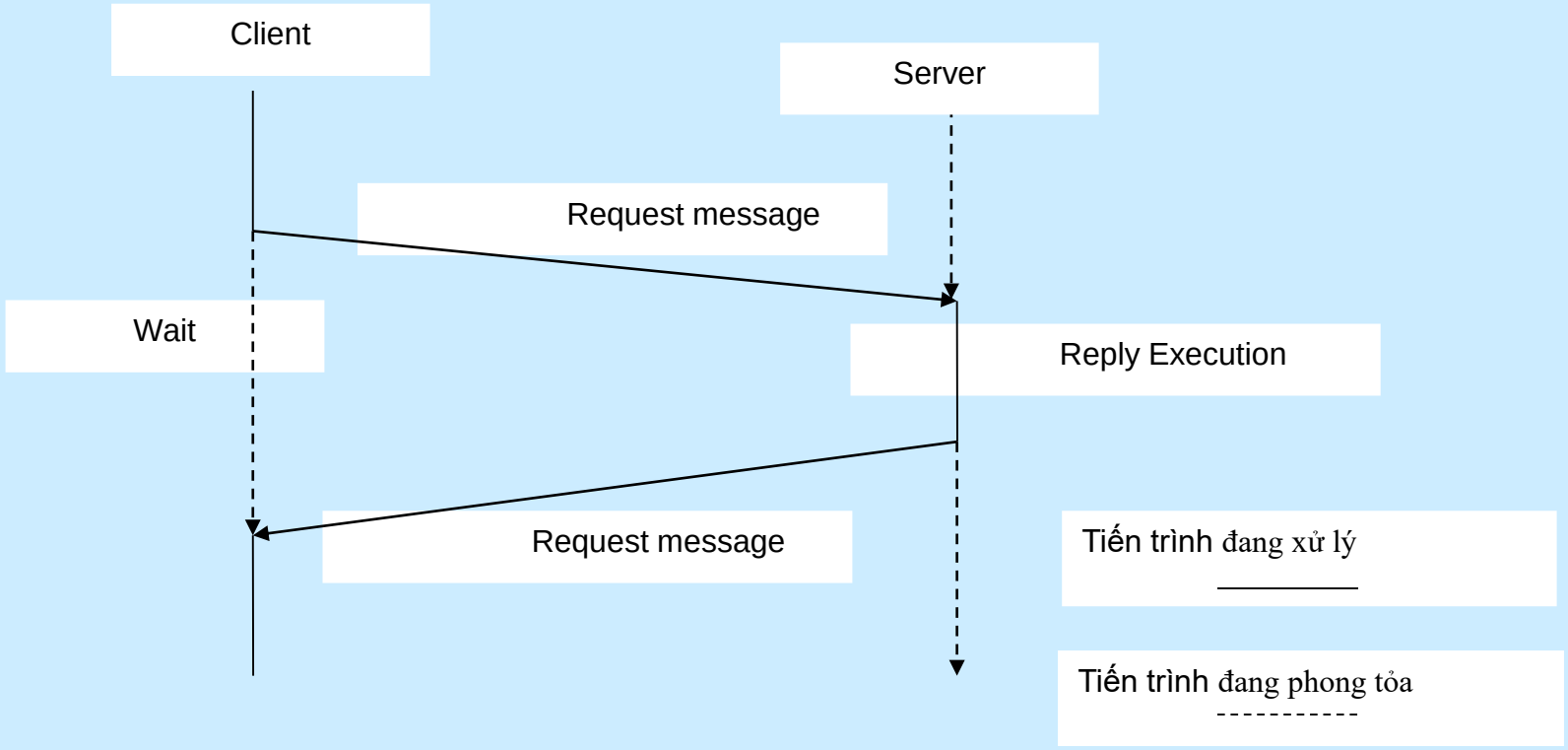
+ Mô hình truyền tin này liên quan đến việc truyền hai thông điệp và được đồng bộ hóa giữa client và server.

+ Ở bước 1, tiến trình server phải nhận thức được thông điệp được yêu cầu ngay khi nó đến và hành động phát ra yêu cầu trong client phải được tạm dừng.

+ Ở bước 3, tiến trình client ở trạng thái chờ cho đến khi nó nhận được đáp ứng do server gửi về.

1. MÔ HÌNH CLIENT/SERVER

Mô hình client/server thường được cài đặt dựa trên các thao tác cơ bản là gửi (send) và nhận (receive).



1. MÔ HÌNH CLIENT/SERVER

Chế độ bị phong tỏa (blocked):

+ Khi tiến trình client hoặc server phát ra lệnh gửi dữ liệu (send), việc thực thi của tiến trình sẽ bị tạm ngừng cho tới khi tiến trình nhận phát ra lệnh nhận dữ liệu (receive).

+ Tương tự đối với tiến trình nhận dữ liệu, nếu tiến trình client hoặc server phát ra lệnh nhận dữ liệu, mà tại thời điểm đó chưa có dữ liệu gửi tới thì việc thực thi của tiến trình cũng sẽ bị tạm ngừng cho tới khi có dữ liệu gửi tới.

1. MÔ HÌNH CLIENT/SERVER

Chế độ không bị phong tỏa (non-blocked)

+ Trong chế độ này, khi tiến trình client hay server phát ra lệnh gửi dữ liệu thực sự, *việc thực thi của tiến trình vẫn được tiến hành mà không quan tâm đến việc có tiến trình nào phát ra lệnh nhận dữ liệu đó hay không*.

+ Tương tự cho trường hợp nhận dữ liệu, khi tiến trình phát ra lệnh nhận dữ liệu, *nó sẽ nhận dữ liệu hiện có, việc thực thi của tiến trình vẫn được tiến hành* mà không quan tâm đến việc có tiến trình nào phát ra lệnh gửi dữ liệu tiếp theo hay không.

2. CÁC KIẾN TRÚC CLIENT/SERVER

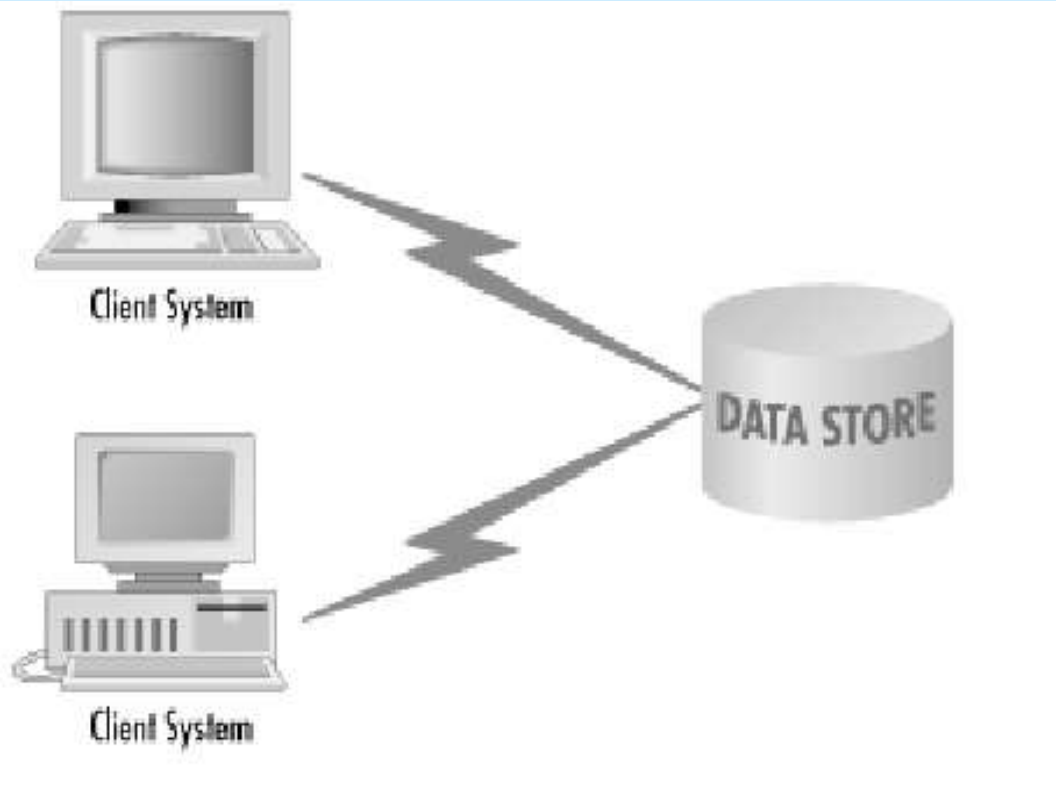
2.1. Client/Server hai tầng

2.2. Client/Server ba tầng

2.3. Kiến trúc n-tầng

2. CÁC KIẾN TRÚC CLIENT/SERVER

2.1. Client/Server hai tầng :



2. CÁC KIẾN TRÚC CLIENT/SERVER

2.1. Client/Server hai tầng:

+ Trong ứng dụng hai tầng truyền thống, *khối lượng công việc xử lý được dành cho phía client*

+ Khi đó *server chỉ đóng vai trò như là chương trình kiểm soát luồng vào ra giữa ứng dụng và dữ liệu*.

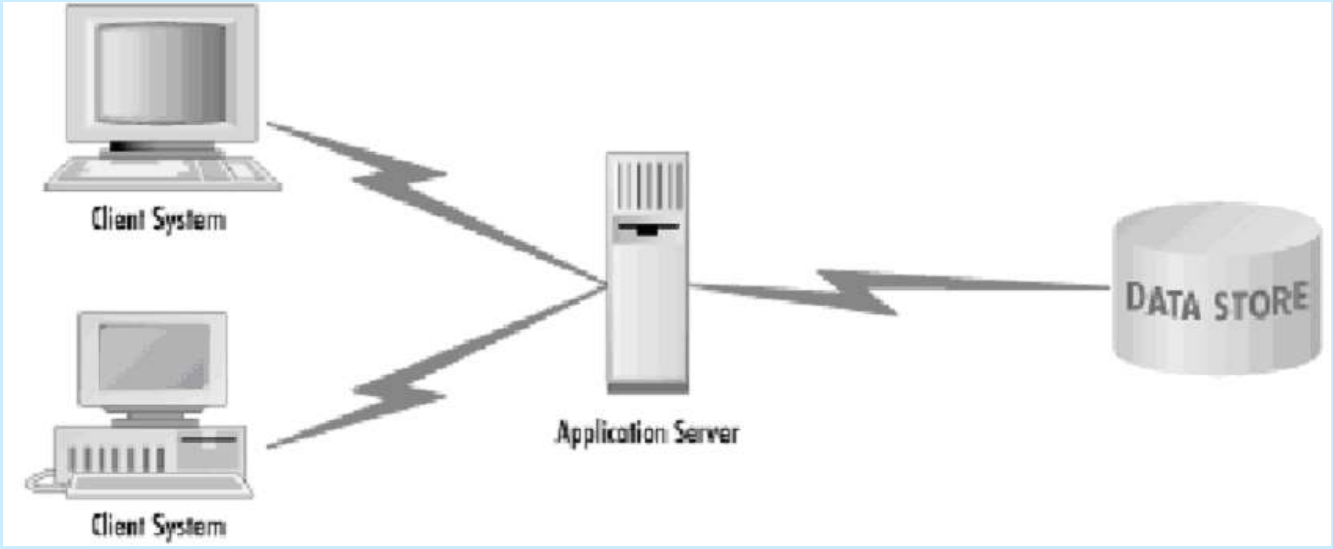
+ Hiệu năng của ứng dụng bị giảm đi do tài nguyên hạn chế của PC và khối lượng dữ liệu truyền đi trên mạng cũng tăng theo

2. CÁC KIẾN TRÚC CLIENT/SERVER

2.2. Client/Server ba tầng

+ Ta có thể tránh được các vấn đề của kiến trúc client/server hai tầng bằng cách mở rộng kiến trúc thành ba tầng.

+ Một kiến trúc ba tầng có thêm một tầng mới tách biệt việc xử lý dữ liệu ở vị trí trung tâm.



2. CÁC KIẾN TRÚC CLIENT/SERVER

2.2. Client/Server ba tầng

+ Theo kiến trúc ba tầng, một ứng dụng được chia thành ba tầng tách biệt nhau:

- Tầng đầu tiên : là tầng trình diễn thường bao gồm các giao diện đồ họa. Tầng trình diễn nhận dữ liệu và định dạng nó để hiển thị.
- Tầng thứ hai : là tầng trung gian hay tầng tác nghiệp.
- Tầng thứ ba : chứa dữ liệu cần thiết cho ứng dụng. Tầng thứ ba về cơ bản là chương trình thực hiện các lời gọi hàm để tìm kiếm dữ liệu cần thiết.

+ Sự tách biệt giữa chức năng xử lý với giao diện đã tạo nên sự linh hoạt cho việc thiết kế ứng dụng.

2. CÁC KIẾN TRÚC CLIENT/SERVER

2.3. Kiến trúc n-tầng

Kiến trúc n-tầng được chia thành như sau:

+ Tầng giao diện người dùng: quản lý tương tác của người dùng với ứng dụng

+ Tầng logic trình diễn: Xác định cách thức hiển thị giao diện người dùng và các yêu cầu của người dùng được quản lý như thế nào.

+ Tầng logic tác nghiệp: Mô hình hóa các quy tắc tác nghiệp

+ Tầng các dịch vụ hạ tầng: Cung cấp chức năng bổ trợ cần thiết cho ứng dụng.

3. MÔ HÌNH TRUYỀN TIN SOCKET

Một socket có thể thực hiện bảy thao tác cơ bản:

+ Kết nối với một máy ở xa (để gửi và nhận dữ liệu)
+ Gửi dữ liệu
+ Nhận dữ liệu
+ Ngắt liên kết
+ Gán cổng
+ Nghe dữ liệu đến
+ Chấp nhận liên kết từ các máy ở xa trên cổng được gán

3. MÔ HÌNH TRUYỀN TIN SOCKET

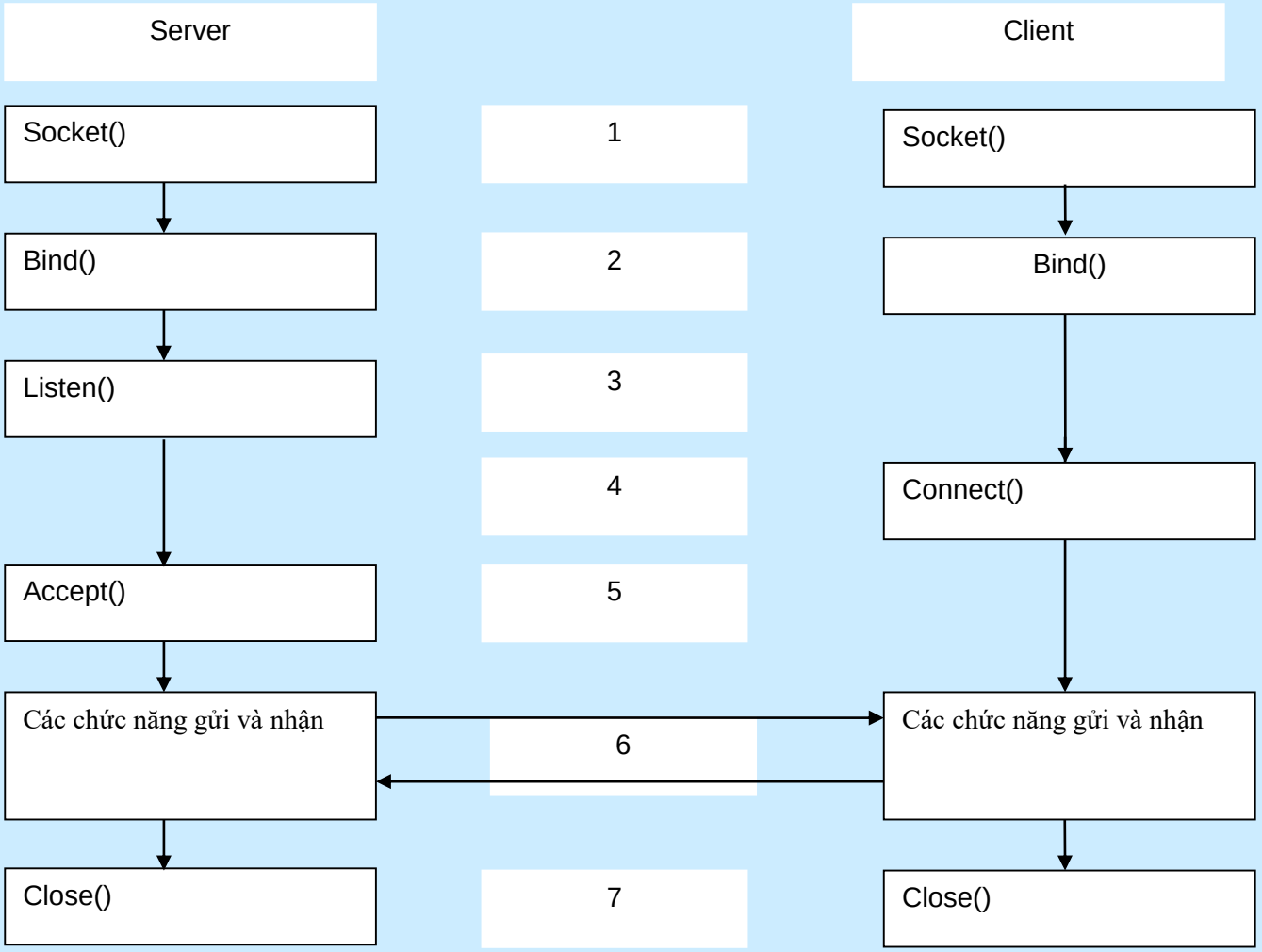
+ Lớp Socket của Java được sử dụng cả client và server.

+ Có các phương thức tương ứng với bốn thao tác đầu tiên.

+ Ba thao tác cuối chỉ cần cho server để chờ các client liên kết với chúng.

+ Các thao tác này được cài đặt bởi lớp ServerSocket.

3. MÔ HÌNH TRUYỀN TIN SOCKET



3. MÔ HÌNH TRUYỀN TIN SOCKET

Các socket cho client thường được sử dụng theo mô hình sau:

- Sử dụng hàm Socket() để tạo một socket mới.
- Socket cố gắng liên kết với một host ở xa.
- Mỗi khi liên kết được thiết lập, các host ở xa nhận các luồng vào và ra từ socket, và sử dụng các luồng này để gửi dữ liệu cho nhau.
- Khi việc truyền dữ liệu hoàn thành, một hoặc cả hai phía ngắt liên kết.

Ví dụ : HTTP đòi hỏi mỗi liên kết phải bị đóng sau khi yêu cầu được phục vụ.

4. SOCKET CHO CLIENT

4.1. Các constructor

4.2. Nhận các thông tin về Socket

4.3. Đóng Socket

4.4. Thiết lập các tùy chọn cho Socket

4.5. Các phương thức của lớp Object

4.6. Các ngoại lệ Socket

4.7. Các lớp SocketAddress

4. SOCKET CHO CLIENT

4.1. Các Constructor

public Socket(String host, int port) throws UnknownHostException, IOException

Hàm này tạo một socket TCP với host và cổng xác định và thực hiện liên kết với host ở xa.

Ví dụ:

```
try{
Socket sk = new Socket( “www.ud.edu.vn”,80);
}catch(UnknownHostException e){System.err.println(e);}
catch(IOException e) {System.err.println(e);}
```

4. SOCKET CHO CLIENT

4.1. Các Constructor

Ví dụ: Viết chương trình để kiểm tra những cổng nào đang có server hoạt động

```
import java.net.*;
import java.io.*;
class PortScanner { public static void main(String[] args)
    { String host="localhost";
     if(args.length>0){ host=args[0]; }
     for(int i=0;i<1024;i++)
            { try{ Socket sk=new Socket(host,i);
     System.out.println("Co mot server dang hoat dong  tren
                                    cong:"+i);
              } catch(UnknownHostException e){
System.err.println(e);          }
                catch(IOException e){  System.err.println(e);  }
     }}}
```

4. SOCKET CHO CLIENT

4.1. Các Constructor

public Socket(InetAddress host, int port)throws IOException

Constructor này tạo một socket TCP với thông tin là địa chỉ của một host được xác định bởi một đối tượng InetAddres và số hiệu cổng port, sau đó nó thực hiện kết nối tới host.

Constructor đưa ra ngoại lệ trong trường hợp không kết nối được tới host.

4. SOCKET CHO CLIENT

4.1. Các Constructor

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

+ Constructor này tạo ra một socket với thông tin là địa chỉ IP được biểu diễn bởi một đối tượng String và một số hiệu cổng và thực hiện kết nối tới host đó.

+ Socket kết nối tới host ở xa thông qua một giao tiếp mạng và số hiệu cổng cục bộ.

+ Nếu localPort bằng 0 thì Java sẽ lựa chọn một cổng ngẫu nhiên có sẵn nằm trong khoảng từ 1024 đến 65535.

4. SOCKET CHO CLIENT

4.1. Các Constructor

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

Constructor chỉ khác constructor trên ở chỗ *địa chỉ của host* lúc này được *biểu diễn bởi một đối tượng InetAddress*

4. SOCKET CHO CLIENT

4.2. Nhận các thông tin về Socket

+ Đối tượng Socket có một số trường thông tin riêng mà ta có thể truy nhập tới chúng thông qua các phương thức trả về các thông tin này.

public InetAddress getInetAddress()

+ Phương thức getInetAddress() cho ta biết host ở xa mà Socket kết nối tới, hoặc liên kết đã bị ngắt thì nó cho biết host ở xa mà Socket đã kết nối tới

public int getPort()

+ Phương thức này cho biết số hiệu cổng mà Socket kết nối tới trên host ở xa.

4. SOCKET CHO CLIENT

4.2. Nhận các thông tin về Socket

public int getLocalPort()

+ Thông thường một liên kết thường có hai đầu: host ở xa và host cục bộ. Để tìm ra số hiệu cổng ở phía host cục bộ ta gọi phương thức getLocalPort().

public InetAddress getLocalAddress()

+ Phương thức này cho ta biết giao tiếp mạng nào mà một socket gắn kết với nó.

4. SOCKET CHO CLIENT

4.2. Nhận các thông tin về Socket

public InputStream getInputStream() throws IOException

+ Phương thức getInputStream() trả về một luồng nhập để đọc dữ liệu từ một socket vào chương trình.

+ Để tâng cao hiệu năng, ta có thể đệm dữ liệu bằng cách gắn kết nó với luồng lọc BufferedInputStream hoặc BufferedReader.

4. SOCKET CHO CLIENT

4.2. Nhận các thông tin về Socket

public OutputStream getOutputStream() throws IOException

+ Phương thức getOutputStream() trả về một luồng xuất để ghi dữ liệu từ ứng dụng ra đầu cuối của một socket.

+ Thông thường, ta sẽ gắn kết luồng này với một luồng tiện lợi hơn như lớp DataOuputStream hoặc OutputStreamWriter trước khi sử dụng nó.

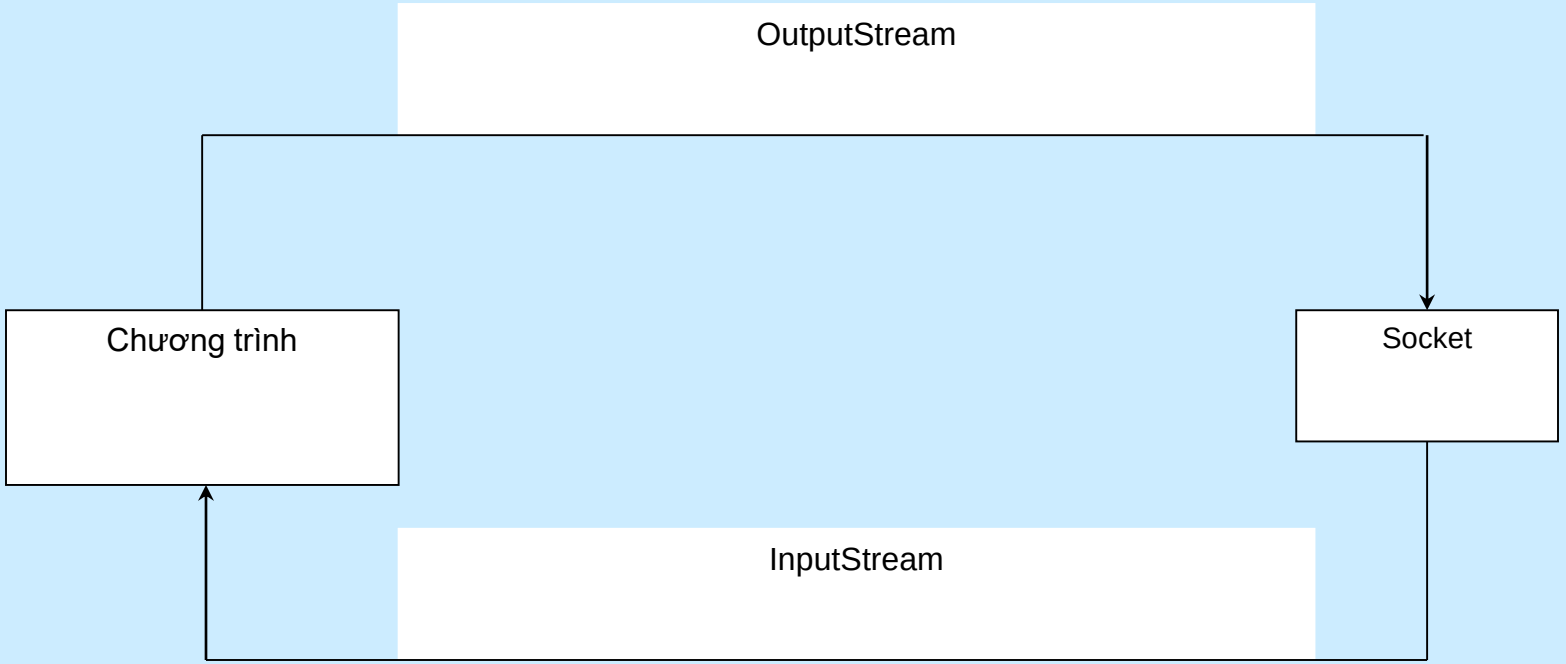
4. SOCKET CHO CLIENT

4.2. Nhận các thông tin về Socket

+ Hai phương thức getInputStream() và getOutputStream() là các phương thức lấy về các luồng dữ liệu nhập và xuất.

+ Để nhận dữ liệu từ một máy ở xa ta nhận về một luồng nhập từ socket và đọc dữ liệu từ luồng đó.

+ Để ghi dữ liệu lên một máy ở xa ta nhận về một luồng xuất từ socket và ghi dữ liệu lên luồng.



4. SOCKET CHO CLIENT

4.3. Đóng Socket

public void close() throws IOException

+ Các socket được đóng một cách tự động khi một trong hai luồng đóng lại, hoặc khi chương trình kết thúc, hoặc khi socket được thu hồi.

+ Mỗi khi một Socket đã bị đóng lại, ta vẫn có thể truy xuất tới các trường thông tin InetAddress, địa chỉ cục bộ, và số hiệu cổng cục bộ thông qua các phương thức getInetAddress(), getPort(), getLocalHost(), và getLocalPort().

4. SOCKET CHO CLIENT

4.3. Đóng Socket

Các socket đóng một nửa (Half-closed socket)

+ Phương thức close() đóng cả các luồng nhập và luồng xuất từ socket.

+ Trong một số trường hợp ta chỉ muốn đóng một nửa kết nối, hoặc là luồng nhập hoặc là luồng xuất.

public void shutdownInput() throws IOException

public void shutdownOutput() throws IOException

4. SOCKET CHO CLIENT

4.4. Thiết lập các tùy chọn cho Socket

4.4.1. TCP_NODELAY

public void setTcpNoDelay(boolean on) throws SocketException

public boolean getTcpNoDelay() throws SocketException

+ Thiết lập giá trị TCP_NODELAY là true để đảm bảo rằng các gói tin được gửi đi nhanh nhất có thể mà không quan tâm đến kích thước của chúng.

+ Trước khi gửi đi một gói tin khác, host cục bộ đợi để nhận các xác thực của gói tin trước đó từ hệ thống ở xa.

4. SOCKET CHO CLIENT

4.4. Thiết lập các tùy chọn cho Socket

4.4.2. SO_LINGER

public void setSoLinger(boolean on, int seconds) throws SocketException

public int getSoLinger() throws SocketException

+ Tùy chọn SO_LINGER xác định phải thực hiện công việc gì với datagram vẫn chưa được gửi đi khi một socket đã bị đóng lại.

+ Nếu SO_LINGER được thiết lập bằng 0, các gói tin chưa được gửi đi bị phá hủy khi socket bị đóng lại.

+ Nếu SO_LINGER lớn hơn 0, thì phương thức close() phong tỏa để chờ cho dữ liệu được gửi đi và nhận được xác thực từ phía nhận.

+ Khi hết thời gian qui định, socket sẽ bị đóng lại và bất kỳ phần dữ liệu còn lại sẽ không được gửi đi.

4. SOCKET CHO CLIENT

4.4. Thiết lập các tùy chọn cho Socket

4.4.3. SO_TIMEOUT

public void setSoTimeout(int milliseconds) throws SocketException

public int getSoTimeout() throws SocketException

Thông thường khi ta đọc dữ liệu từ một socket, lời gọi phương thức phong tỏa cho tới khi nhận đủ số byte.

Bằng cách thiết lập phương thức SO_TIMEOUT, ta sẽ đảm bảo rằng lời gọi phương thức sẽ không phong tỏa trong khoảng thời gian quá số giây quy định.

4. SOCKET CHO CLIENT

4.5. Các phương thức của lớp Object

Lớp Socket nạp chồng phương thức chuẩn của lớp java.lang.Object, toString().

Vì các socket là các đối tượng tạm thời và thường chỉ tồn tại khi liên kết tồn tại.

public String toString()

Phương thức toString() tạo ra một xâu ký tự.

4.6. Các ngoại lệ Socket

Hầu hết các phương thức của lớp Socket được khai báo đưa ra ngoại lệ IOException, hoặc lớp con của lớp IOExcepton là lớp SocketException.

4. SOCKET CHO CLIENT

4.7. Các lớp SocketAddress

+ Lớp SocketAddress là một lớp trừu tượng mà không có phương thức nào ngoài construtor mặc định.

+ Lớp này có thể được sử dụng cho cả các socket TCP và socket không phải là TCP.

+ Mục đích chính của lớp SocketAddress là cung cấp một nơi lưu trữ các thông tin liên kết socket tạm thời (như địa chỉ IP và số hiệu cổng) có thể được sử dụng lại để tạo ra socket mới.

```
public SocketAddress getRemoteSocketAddress()
public SocketAddress getLocalSocketAddress()
```

Cả hai phương thức này trả về giá trị null nếu socket vẫn chưa kết nối tới.

5. Lớp ServerSocket

5.1. Các constructor

5.2. Chấp nhận và ngắt liên kết

5. Lớp ServerSocket

5. Lớp ServerSocket

Lớp ServerSocket có đủ mọi thứ ta cần để viết các server bằng Java.

+ Nó có các constructor để tạo các đối tượng ServerSocket mới

+ Có các phương thức để lắng nghe các liên kết trên một cổng xác định, và các phương thức trả về một Socket khi liên kết được thiết lập

Vì vậy ta có thể gửi và nhận dữ liệu.

5. Lớp ServerSocket

5. Lớp ServerSocket

Vòng đời của một server

+ Một ServerSocket mới được tạo ra trên một cổng xác định bằng cách sử dụng một constructor ServerSocket.

+ ServerSocket lắng nghe liên kết đến trên cổng đó bằng cách sử dụng phương thức accept().

- Phương thức accept() phong tỏa cho tới khi một client thực hiện một liên kết

- Phương thức accept() trả về một đối tượng Socket mà liên kết giữa client và server.

+ Tùy thuộc vào kiểu server, hoặc phương thức getInputStream(), getOutputStream() hoặc cả hai được gọi để nhận các luồng vào ra để truyền tin với client.

+ Server và client tương tác theo một giao thức thỏa thuận sẵn cho tới khi ngắt liên kết.

+ Server, client hoặc cả hai ngắt liên kết

+ Server trở về bước hai và đợi liên kết tiếp theo.

5. Lớp ServerSocket

5.1. Các constructor

public ServerSocket(int port) throws IOException, BindException

+ Constructor này tạo một socket cho server trên cổng xác định. Nếu port bằng 0, hệ thống chọn một cổng ngẫu nhiên cho ta.

+ Cổng do hệ thống chọn đôi khi được gọi là cổng vô danh vì ta không biết số hiệu cổng.

+ Với các server, các cổng vô danh không hữu ích lắm vì các client cần phải biết trước cổng nào mà nó nối tới.

5. Lớp ServerSocket

5.1. Các constructor

Ví dụ: Để tạo một server socket cho cổng 80

```
try{
ServerSocket httpd = new ServerSocket(80);
} catch(IOException e)
{System. err.println(e);}
```

Constructor đưa ra ngoại lệ IOException nếu ta không thể tạo và gán Socket cho cổng được yêu cầu. Ngoại lệ IOException phát sinh khi:

+ *Cổng đã được sử dụng*
+ *Không có quyền hoặc cố liên kết với một cổng nằm giữa 0 và 1023.*

5. Lớp ServerSocket

5.1. Các constructor

Ví dụ:

```
import java.net.*;
import java.io.*;
public class congLocalHost  {
        public static void main(String[] args) {
                ServerSocket ss;
                for(int i=0;i<1024;i++){
                        try{    ss= new ServerSocket(i);
                                ss.close();
                        }catch(IOException e)
                        { System.out.println("Co mot server tren cong "+i);}
                }
        }
}
```

5. Lớp ServerSocket

5.1. Các constructor

public ServerSocket(int port, int queuelength, InetAddress bindAddress)throws IOException

Constructor này tạo một đối tượng ServerSocket trên cổng xác định với chiều dài hàng đợi xác định. ServerSocket chỉ gán cho địa chỉ IP cục bộ xác định.

Constructor này hữu ích cho các server chạy trên các hệ thống có nhiều địa chỉ IP.

5. Lớp ServerSocket

5.2. Chấp nhận và ngắt liên kết

+ Một đối tượng ServerSocket hoạt động trong một vòng lặp chấp nhận các liên kết.

+ Mỗi lần lặp nó gọi phương thức accept(). Phương thức này trả về một đối tượng Socket biểu diễn liên kết giữa client và server.

+ Khi giao tác hoàn thành, server gọi phương thức close() của đối tượng socket.

+ Nếu client ngắt liên kết trong khi server vẫn đang hoạt động, các luồng vào ra kết nối server với client sẽ đưa ra ngoại lệ InterruptedException trong lần lặp tiếp theo

```
public Socket accept() throws IOException
```

5. Lớp ServerSocket

5.2. Chấp nhận và ngắt liên kết

+ Khi bước thiết lập liên kết hoàn thành, và ta sẵn sàng để chấp nhận liên kết, cần gọi phương thức accept() của lớp ServerSocket.

+ Phương thức này phong tỏa; nó dừng quá trình xử lý và đợi cho tới khi client được kết nối.

+ Khi client thực sự kết nối, phương thức accept() trả về đối tượng Socket.

+ Ta sử dụng các phương thức getInputStream() và getOutputStream() để truyền tin với client.

5. Lớp ServerSocket

5.2. Chấp nhận và ngắt liên kết

Ví dụ:

```
try{
   ServerSocket   ser = new ServerSocket(5776);
while(true)
            {
            Socket sk = ser.accept();
            PrintStream ps = new PrintStream(sk.getOutputStream());
            ps.println("Da ket noi den Server");
            sk.close();
            }
            }
            catch(IOException e){    System.err.println(e);}
```

5. Lớp ServerSocket

5.2. Chấp nhận và ngắt liên kết

public void close() throws IOException

+ Nếu đã hoàn thành công việc với một ServerSocket, ta cần phải đóng nó lại, đặc biệt nếu chương trình của ta tiếp tục chạy.

+ Điều này nhằm tạo điều kiện cho các chương trình khác muốn sử dụng nó. Đóng một ServerSocket không đồng nhất với việc đóng một Socket.

5. Lớp ServerSocket

5.2. Chấp nhận và ngắt liên kết

public InetAddress getInetAddress()

+ Phương thức này trả về địa chỉ được sử dụng bởi server (localhost). Nếu localhost có địa chỉ IP, địa chỉ này được trả về bởi phương thức InetAddress.getLocalHost()

Ví dụ:

```
try{
  ServerSocket sk = new ServerSocket(80);
  InetAddress ia = sk.getInetAddress();
}catch(IOException e){}
```

public int getLocalHost()

+ Các contructor ServerSocket cho phép ta nghe dữ liệu trên cổng không định trước bằng cách gán số 0 cho cổng. Phương thức này cho phép ta tìm ra cổng mà server đang nghe.

6.6. Cài đặt chương trình Client

Sau khi đã tìm hiểu các lớp và các phương thức cần thiết để cài đặt chương trình Socket. Các bước để cài đặt chương trình Client

Bước 1:Tạo một đối tượng Socket

Socket client =new Socket(“hostname”,portName);

Bước 2:Tạo một luồng xuất để có thể sử dụng để gửi thông tin tới Socket

PrintWriter out=new PrintWriter(client.getOutputStream(),true);

Bước 3:Tạo một luồng nhập để đọc thông tin đáp ứng từ server

BufferedReader in=new BufferedReader(new InputStreamReader(client.getInputStream()));

6.6. Cài đặt chương trình Client

Sau khi đã tìm hiểu các lớp và các phương thức cần thiết để cài đặt chương trình Socket.Các bước để cài đặt chương trình Client

Bước 4:Thực hiện các thao tác vào/ra với các luồng nhập và luồng xuất

Đối với các luồng xuất, PrintWriter, ta sử dụng các phương thức print và println, tương tự như System.out.println.

Đối với luồng nhập, BufferedReader, ta có thể sử dụng phương thức read() để đọc một ký tự, hoặc một mảng các ký tự, hoặc gọi phương thức readLine() để đọc vào một dòng ký tự. Cần chú ý rằng phương thức readLine() trả về null nếu kết thúc luồng.

Bước 5: Đóng socket khi hoàn thành quá trình truyền tin

6.7 Cài đặt chương trình Server

Để cài đặt chương trình Server bằng ServerSocket ta thực hiện các bước sau:

Bước 1 : Tạo một đối tượng ServerSocket

ServerSocket ss=new ServerSocket(port)

Bước 2: Tạo một đối tượng Socket bằng cách chấp nhận liên kết từ yêu cầu liên kết của client. Sau khi chấp nhận liên kết, phương thức accept() trả về đối tượng Socket thể hiện liên kết giữa Client và Server.

```
while(condion)
        {
        Socket s=ss.accept();
        doSomething(s);
        }
```

Người ta khuyến cáo rằng chúng ta nên giao công việc xử lý đối tượng s cho một tuyến đoạn nào đó.

Bước 3: Tạo một luồng nhập để đọc dữ liệu từ client

```
BufferedReader in=new BufferedReader(new
InputStreamReader(s.getInputStream()));
```

6.7 Cài đặt chương trình Server

Để cài đặt chương trình Server bằng ServerSocket ta thực hiện các bước sau:

Bước 4: Tạo một luồng xuất để gửi dữ liệu trở lại cho client

PrintWriter pw=new PrintWriter(s.getOutputStream(),true);

Trong đó tham số true được sử dụng để xác định luồng sẽ được tự động đẩy ra.

Bước 5: Thực hiện các thao tác vào ra với các luồng nhập và luồng xuất

Bước 6: Đóng socket s khi đã truyền tin xong. Việc đóng socket cũng đồng nghĩa với việc đóng các luồng.

6.8 Đa tuyến trong lập trình Java

+ Các server như đã viết ở trên chỉ quản lý được một client tại một thời điểm.

+ Khi khối lượng công việc mà server cần xử lý một yêu cầu của client là quá lớn và không biết trước được thời điểm hoàn thành công việc xử lý thì các server này là không thể chấp nhận được.

+ Để khắc phục điều này, người ta quản lý mỗi phiên của client bằng một tuyến đoạn riêng, cho phép các server làm việc với nhiều client đồng thời.

+ Server này được gọi là server tương tranh và tạo ra một tuyến đoạn để quản lý từng yêu cầu, sau đó tiếp tục lắng nghe các client khác.

6.8 Đa tuyến trong lập trình Java

+ Các server đơn tuyến đoạn chỉ quản lý được một liên kết tại một thời điểm.

+ Trong thực tế một server có thể phải quản lý nhiều liên kết cùng một lúc.

+ Để thực hiện điều này server chấp nhận các liên kết và chuyển các liên kết này cho từng tuyến xử lý.

+ Trong phần dưới đây chúng ta sẽ xem xét cách tiến hành cài đặt một chương trình client/server đa tuyến.

6.8 Đa tuyến trong lập trình Java

Chương trình phía Server

```
import java.io.*;
import java.net.*;
class EchoServe extends Thread
{
  private Socket socket;
  private BufferedReader in;
  private PrintWriter out;
  public EchoServe (Socket s) throws IOException
  {
    socket = s;
    System.out.println("Serving: "+socket);
    in = new BufferedReader(new InputStreamReader(socket.getInputStream()));
    out = new PrintWriter(new BufferedWriter(new OutputStreamWriter(
          socket.getOutputStream())), true);
    start();
  }
```

6.8 Đa tuyến trong lập trình Java

Chương trình phía Server

```
private static String VuaHoaVuaThuong(String s3)
   {
          int k =0;
          int i;
          char c;
          String st3="";
          k = s3.length();
          for(i=0;i<k;i++)
          {
                    c = s3.charAt(i);
                    if(c>='A'&& c<='Z')  c = (char) (c + 32);
                    else if (c>='a'&& c<='z') c=(char) (c-32);
                    // Noi vao chuoi moi st3
                              st3=st3+c;
          }
          return st3;
   }
```

6.8 Đa tuyến trong lập trình Java

Chương trình phía Server

```
public void run()
  { try { while (true){
            System.out.println("....Server is waiting...");
                  String str = in.readLine();
                  if (str.equals(“exit”) ) break;
                  System.out.println("Received: " + str);
                  System.out.println("From: "+ socket);
                   String upper=VuaHoaVuaThuong(str);
                  // gửi lại cho client
                  out.println(upper);
               } System.out.println("Disconnected with.."+socket);
               }     catch (IOException e) {}
               finally   {
                   try  {socket.close(); }  catch(IOException e) {}
               }    }   }
```

6.8 Đa tuyến trong lập trình Java

Chương trình phía Server

```
public class TCPServer1
{    static int PORT=0; .
    public static void main(String[] args) throws IOException
    {
      if (args.length == 1) {
              PORT=Integer.parseInt(args[0]); // Nhập số hiệu cổng từ đối dòng lệnh  }
    ServerSocket s = new ServerSocket(PORT);
    InetAddress  addrs= InetAddress.getLocalHost();
    System.out.println("TCP/Server running on : "+ addrs +" ,Port "+s.getLocalPort());
     try {  while(true) {
                      Socket socket = s.accept();
                    try  {    new EchoServe(socket);
              } catch(IOException e) {  socket.close(); }}}
            finally {
            s.close();
            } } }
```

6.8 Đa tuyến trong lập trình Java

Chương trình phía Client

```
import java.net.*;
import java.io.*;
public class TCPClient1
{
  public static void main(String[] args) throws IOException
  {
      if (args.length != 2)  {
              System.out.println("Sử dụng: java TCPClient  hostid port#");
              System.exit(0); }
      try  {
             InetAddress addr = InetAddress.getByName(args[0]);
             Socket socket = new Socket(addr, Integer.parseInt(args[1]));
             try {
                  System.out.println("socket = " + socket);
                  BufferedReader in = new BufferedReader(new InputStreamReader(
              socket.getInputStream()));
             PrintWriter out =new PrintWriter(new BufferedWriter(
              new OutputStreamWriter(socket.getOutputStream())),true);
```

6.8 Đa tuyến trong lập trình Java

Chương trình phía Client

```
// Đọc dòng ký tự từ bàn phím
DataInputStream strin = new DataInputStream(new BufferedInputStream(System.in));
try   {
        for(;;)
       {System.out.println("Exit to terminate the program.");
    String str=strin.readLine();
  if (str.equals("exit")) break;
  else
                out.println(str);          // Send the message
                String strout = in.readLine();      // Recive it back
                if ( str.length()==strout.length())
                              { System.out.println("Received: "+strout);}
                else
                System.out.println("Error"+ strout);
                      }
       } catch (IOException e) { e.printStackTrace();}
```

6.8 Đa tuyến trong lập trình Java

Chương trình phía Client

```
    } finally       {
            System.out.println("EOF...exit");
            socket.close();
            }       }
    catch(UnknownHostException e){
            System.err.println("Can't find host");
            System.exit(1);
    }
    catch (SocketException e)
    {       System.err.println("Can't open socket");
            e.printStackTrace();
            System.exit(1);
} } }
```

Hết *!!!*