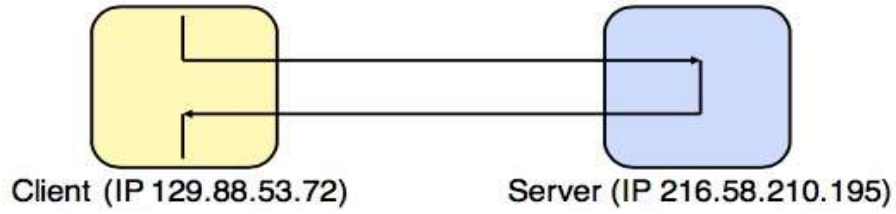
Bài thực hành số 2: Java Sockets

I. Tài liệu tham khảo

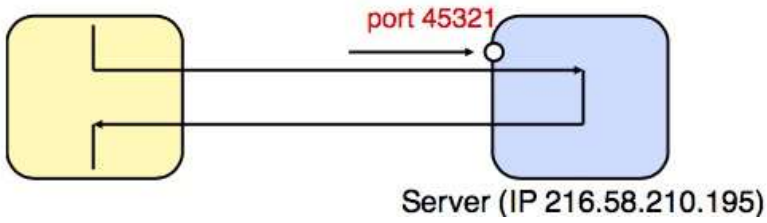
[1] https://docs.oracle.com/javase/tutorial/networking/sockets/
[2] https://docs.oracle.com/javase/tutorial/essential/exceptions/tryResourceClose.html

II. Giới thiệu socket

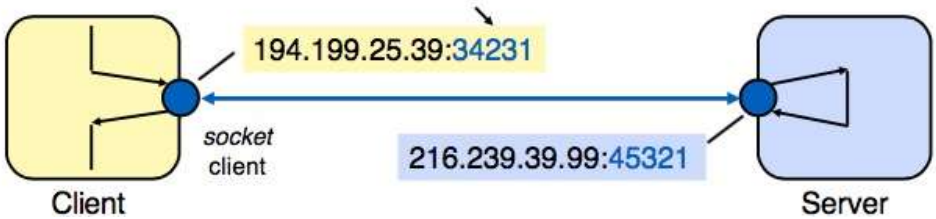
Tại tầng mạng, client và server sử dụng các giao thức mạng như TCP hoặc UDP để liên lạc với nhau. Client và Server được xác định thông qua địa chỉ IP. Địa chỉ IP tương ứng với tên ký hiệu được xác định thông qua DNS.



Thông thường, trên server chạy đồng thời nhiều dịch vụ khác nhau. Ví dụ như ssh, ftp, mail, web server vân vân. Để phân biệt các dịch vụ này, chúng ta sử dụng các cổng. Số hiệu cổng được mã hóa 16 bit; các số cổng từ 0 đến 1023 đã được đặt cho các dịch vụ riêng.



Socket cung cấp một giao tiếp đơn giản phục vụ việc liên lạc qua giao thức TCP và UDP ở tầng giao vận. Socket xác định một điểm liên lạc (điêm gửi hoặc điểm nhận) thông qua địa chỉ IP và số hiệu cổng.



Nếu sử dụng giao thức TCP, đầu tiên cần thiết lập một kết nối giữa client và server; tất cả thông điệp trao đổi giữa client và server sẽ sử dụng kết nối này. Trong Java, server được khai báo bởi lớp **ServerSocket**, còn client là lớp **Socket**.

Nếu sử dụng giao thức UDP thì không cần thiết lập kết nối trước; các thông điệp được trao đổi từng cái một. Trong Java, cả server và client đều được khai báo thông qua lớp **DatagramSocket.**

Với giao thức TCP, tuần tự thiết lập kết nối như sau:

1. Server: khởi động server. Server tạo socket server chờ kết nối từ client.
2. Client: khởi tạo client. Client kết nối tới server.
3. Server: nhận và chấp nhận yêu cầu kết nối; tạo một kênh socket riêng để trao đổi dữ liệu với client.
4. Client: nếu kết nối thành công, client và server trao đổi dữ liệu thông qua việc đọc/ghi socket.

III. Ví dụ: Echo Server

Trong ví dụ này, server nhận một xâu từ client và gửi trả lại.

Code của server và client như dưới. Yêu cầu:

- Chạy chương trình
- Chuyện gì xảy ra nếu dừng server ?
- Chuyện gì xảy ra nếu dừng client ?

```
1. public class EchoServer {
2.     public static void main(String[] args) throws IOException {
3.
4.         if (args.length != 1) {
5.             System.err.println("Usage: java EchoServer <port number>");
6.             System.exit(1);
7.         }
8.
9.         int portNumber = Integer.parseInt(args[0]);
10.
11.        try (
12.             ServerSocket serverSocket =
13.                 new ServerSocket(Integer.parseInt(args[0]));
14.             Socket clientSocket = serverSocket.accept();
15.             PrintWriter out =
16.                 new PrintWriter(clientSocket.getOutputStream(), true);
17.             BufferedReader in = new BufferedReader(
18.                 new InputStreamReader(clientSocket.getInputStream()));
19.         ) {
20.             String inputLine;
21.             while ((inputLine = in.readLine()) != null) {
22.                 out.println(inputLine);
23.             }
24.         } catch (IOException e) {
25.             System.out.println("Exception caught when trying to listen on port "
26.                 + portNumber + " or listening for a connection");
27.             System.out.println(e.getMessage());
28.         }
29.     }
30.}
```

```
1. public class EchoClient {
2.     public static void main(String[] args) throws IOException {
3.
4.         if (args.length != 2) {
5.             System.err.println(
6.                 "Usage: java EchoClient <host name> <port number>");
7.             System.exit(1);
8.         }
9.
10.        String hostName = args[0];
11.        int portNumber = Integer.parseInt(args[1]);
12.
13.        try (
14.            Socket echoSocket = new Socket(hostName, portNumber);
15.            PrintWriter out =
16.                new PrintWriter(echoSocket.getOutputStream(), true);
17.            BufferedReader in =
18.                new BufferedReader(
19.                    new InputStreamReader(echoSocket.getInputStream()));
20.            BufferedReader stdIn =
21.                new BufferedReader(
22.                    new InputStreamReader(System.in))
```

IV. Bài tập

a. Calculator server

Thực thi server cung cấp dịch vụ tính toán với giao tiếp như sau:

```
interface Calculator_itf {
	public int plus(int, int);
	public int minus (int, int);
	public int divide(int, int);
	public int multiply(int, int);
}
```

Yêu cầu:

- Thiết kế giao thức trao đổi giữa client-server để thực thi các phép toán.
- Thực thi server
- Thực thi client cho phép hiển thị menu để lựa chọn thực thi các phép toán trên

b. Danh bạ điện thoại – Gửi đối tượng qua socket

Thực thi server cung cấp dịch vụ danh bạ điện thoại với giao tiếp như sau:

```
interface Registry_itf {
	public void add(Person p);
	public String getPhone(String name);
	public Iterable<Person> getAll();
	public Person search(String name);
}
```

Yêu cầu:

- Thiết kế giao thức trao đổi giữa client và server.
- Định nghĩa lớp Person.

- Thực thi server; không sử dụng database, có thể cho dữ liệu vào mảng, danh sách …
- Thực thi client cho phép hiển thị menu lựa chọn chức năng tương ứng để thực hiện.

c. Tiếp tục hoàn thành bài kiểm tra tại lớp.

d. Liên lạc điểm-điểm

Thực thi ứng dụng client-server:

- Có một server trung tâm
- Client kết nối với server
- Client có thể yêu cầu server danh sách client đang kết nối
- 2 Client bất kỳ có thể trao đổi thông tin với nhau.
- 1 client bất kỳ có thể gửi thông điệp đến tất cả client còn lại.

Gợi ý:

- Để 2 client bất kỳ có thể kết nối được với nhau: mỗi client phải được định danh.
- 2 client có thể liên lạc thông qua server
- 2 client có thể liên lạc trực tiếp với nhau, sửa code một client để trở thành server.