SYR1 Réseau – TP 2, 3 et 4
Communication réseau en Java

Léo Noël-Baron & Thierry Sampaio

3/12/2015

Résumé

Ces trois séances de TP ont pour but de pratiquer la communication réseau à l'aide des sockets proposés par Java, en implémentant diverses applications reposant sur les protocoles UDP, TCP et HTTP.

1 Le protocole UDP

1.1 Client simple et serveur d'écho

On souhaite développer une application client-serveur en mode UDP, réalisant un écho (le serveur renvoie tout message reçu au client qui l'a émis).

```java
import java.io.*;
import java.net.*;
public class UDPServer {
    private static byte[] buffer = new byte[1024];
    public static void main(String args[]) throws IOException {
        // Mise en écoute sur le port 8080
        DatagramSocket server = new DatagramSocket(8080);
        System.out.println("UDP server running on 8080...");
        // Boucle d'écoute
        while (true) {
            // Réception d'un paquet UDP
            DatagramPacket request = new DatagramPacket(buffer, buffer.length);
            server.receive(request);
            // Récupération des informations client
            InetAddress ip = request.getAddress();
            int port = request.getPort();
            System.out.println(new String(request.getData()));
            // Renvoi de l'écho
            server.send(new DatagramPacket(buffer, buffer.length, ip, port));
        }
    }
}
```

```java
import java.io.*;
import java.net.*;
public class UDPClient {
    private static byte[] buffer = new byte[1024];
    public static void main(String args[]) throws IOException {
```

```
6         // Création du socket client
7         DatagramSocket client = new DatagramSocket();
8         // Consitution du paquet UDP et envoi
9         InetAddress ip = InetAddress.getByName("localhost");
10        System.out.print("Message (enter to send): ");
11        String message =
12                new BufferedReader(new InputStreamReader(System.in)).readLine();
13        buffer = message.getBytes();
14        client.send(new DatagramPacket(buffer, buffer.length, ip, 8080));
15        // Réception de la réponse
16        DatagramPacket answer = new DatagramPacket(buffer, buffer.length);
17        client.receive(answer);
18        System.out.println("From server: " + new String(answer.getData()));
19        client.close();
20    }
21 }
```

Les communications doivent s'effectuer sur un port supérieur à 1024 ; en effet, les ports inférieurs ne peuvent être utilisés que par l'administrateur (et ce pour des raisons historiques et de sécurité, voir cette discussion sur StackExchange). La communication entre client et serveur ne peut s'établir que si les deux s'exécutent en même temps et sur le même port ; cependant, dans le cas contraire, les deux programmes ne plantent pas et l'ordre de lancement n'est pas important. L'IP de retour du client est contenue dans le paquet envoyé au serveur, qui l'extrait pour pouvoir adresser sa réponse.

1.2 Application ping

On souhaite maintenant, en utilisant un serveur fourni, développer un client qui réalise un ping (envoi d'un ou plusieurs messages simples vers une machine du réseau, en vue d'analyser et de diagnostiquer son état). Le serveur simule un taux de perte de paquets UDP qui devra être pris en compte (en utilisant un timeout).

```
1  import java.io.*;
2  import java.net.*;
3  import java.util.Date;
4  public class PingClient {
5      private static byte[] buffer = new byte[1024];
6      public static void main(String[] args) throws IOException {
7          DatagramSocket socket = new DatagramSocket(); // Socket client
8          socket.setSoTimeout(1000); // Timeout d'une seconde
9          InetAddress ip = InetAddress.getByName("localhost"); // Adresse
10         int cpt = 0; // Nombre de pings reçus
11         int tt = 0; // Temps de voyage total
12         for (int i=0; i<10; i++) {
13             // Envoi d'un ping
14             Date then = new Date();
15             buffer = ("PING "+i+" "+(then.getTime())+" \n").getBytes();
16             socket.send(new DatagramPacket(buffer, buffer.length, ip, 8080));
17             // Réception de la réponse
18             DatagramPacket answer = new DatagramPacket(buffer, buffer.length);
19             try {
20                 socket.receive(answer);
21             } catch (SocketTimeoutException e) {
22                 System.out.println("Timeout");
```

```
23                continue;
24            }
25            // Analyse de la réponse
26            String[] infos = (new String(answer.getData())).split(" ");
27            int seq = Integer.parseInt(infos[1]);
28            int rtt = (int) ((new Date()).getTime() - Long.parseLong(infos[2]));
29            System.out.println("Server at "+ip+" answered for request "
30                    +seq+" in "+rtt+"ms");
31            cpt++;
32            tt += rtt;
33        }
34        socket.close();
35        // Statistiques
36        System.out.println("10 packets transmitted, "+cpt+" received, "
37                +((10-cpt)*10)+"% lost\nRTT average: "+((float) tt/cpt));
38    }
39 }
```

2 Le protocole TCP

On implémente à nouveau l'application écho vue précédemment, mais en s'appuyant cette fois sur le protocole TCP.

```
1  import java.io.*;
2  import java.net.*;
3  public class TCPServer {
4      public static void main(String[] args) throws IOException {
5          // Mise en écoute sur le port 8080
6          ServerSocket socket = new ServerSocket(8080);
7          System.out.println("TCP server running on 8080...");
8          // Boucle d'écoute
9          while (true) {
10             // Réception d'un nouveau message
11             Socket client = socket.accept();
12             String message = new BufferedReader(new InputStreamReader(
13                     client.getInputStream())).readLine();
14             System.out.println(message);
15             // Renvoi de l'écho
16             OutputStream output = client.getOutputStream();
17             output.write(message.getBytes());
18             client.close();
19         }
20     }
21 }
```

```
1  import java.io.*;
2  import java.net.*;
3  public class TCPClient {
4      public static void main(String[] args) throws Exception {
5          // Connexion au serveur distant
6          Socket socket = new Socket("localhost", 8080);
7          // Envoi d'un message
8          System.out.print("Message (enter to send): ");
```

```
9           String message = new BufferedReader(new InputStreamReader(
10                  System.in)).readLine() + "\n";
11          OutputStream output = socket.getOutputStream();
12          output.write(message.getBytes());
13          // Réception de l'écho
14          BufferedReader input = new BufferedReader(new InputStreamReader(
15                  socket.getInputStream()));
16          System.out.println("From server: " + input.readLine());
17      }
18  }
```

Si l'on essaie de lancer le client avant le serveur, celui-ci plante (connexion refusée). La différence avec le protocole UDP est que les deux programmes essaient de créer une connexion avant de communiquer (avec UDP, les paquets sont envoyés sans certitude que le destinataire puisse les recevoir) ; TCP est ainsi plus sûr mais plus lourd alors qu'UDP est plus simple mais pas fiable.

3 Le protocole HTTP

3.1 Client simple

On crée un client HTTP simplifié, réalisant une unique requête vers une page précise.

```
1   import java.io.*;
2   import java.net.*;
3   public class HTTPClient {
4       public static void main(String[] args) throws Exception {
5           // Connexion au serveur
6           Socket socket = new Socket("localhost", 8080);
7           // Requête HTTP
8           OutputStream output = socket.getOutputStream();
9           output.write("GET /quelquechose HTTP/1.0 \n\n".getBytes());
10          // Réponse du serveur
11          BufferedReader input = new BufferedReader(new InputStreamReader(
12                  socket.getInputStream()));
13          String response = "", line = "";
14          while ((line = input.readLine()) != null)
15              response += line + "\n";
16          System.out.println(response);
17      }
18  }
```

Comme attendu, on reçoit une réponse HTTP de statut 200 OK, dont les dernières lignes contiennent bien la page demandée.

3.2 Serveur simple

On crée maintenant un serveur HTTP basique capable de traiter les requêtes GET, et de distribuer une unique page d'exemple à un seul client à la fois.

```
1   import java.io.*;
2   import java.net.*;
3   import java.text.SimpleDateFormat;
4   import java.util.Date;
5   import java.util.Locale;
```

```java
public class HTTPServer {
    public static void main(String[] args) throws IOException {
        ServerSocket socket = new ServerSocket(8080);
        System.out.println("HTTP server running on 8080...");
        // Boucle d'écoute
        while (true) {
            // Lecture d'une requête
            Socket client = socket.accept();
            String request = new BufferedReader(new InputStreamReader(
                    client.getInputStream())).readLine();
            String reqs[] = request.split(" ");
            if (!reqs[0].equals("GET")) { // On n'accepte que les GET
                System.out.println("Can't accept non-GET requests");
                continue;
            }
            // Réponse HTTP (page unique)
            OutputStream output = client.getOutputStream();
            SimpleDateFormat dateFormat = new SimpleDateFormat(
                    "EEE, d MMM yyyy HH:mm:ss Z", Locale.ENGLISH);
            String response = "HTTP/1.0 200 OK \n"
                    + "Date: "+dateFormat.format(new Date())+"\n"
                    + "Server: TPsyr 1.0\n\n"
                    + "<html>h1>Hello world.</h1>"
                    + "I'm a web server, how can I help you?</html>";
            output.write(response.getBytes());
            client.close();
        }
    }
}
```

Ce serveur fonctionne et distribue correctement sa page, que ce soit au client vu précédemment ou à un navigateur classique ; quelques valeurs ont été rajoutées dans l'en-tête HTTP mais d'autres encore seraient envisageables.

3.3 Serveur réaliste

On souhaite enfin améliorer le serveur précédent pour lui permettre de distribuer les pages situées dans un répertoire données, selon l'adresse spécifiée dans la requête GET. Cette amélioration doit en outre prendre en compte le cas où le fichier demandé n'existe pas (code HTTP 404).

```java
import java.io.*;
import java.net.*;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.Locale;
public class HTTPBetterServer {
    public static void main(String[] args) throws IOException {
        ServerSocket socket = new ServerSocket(8080);
        System.out.println("HTTP server running on 8080...");
        // Boucle d'écoute
        while (true) {
            // Lecture d'une requête
            Socket client = socket.accept();
            String request = new BufferedReader(new InputStreamReader(
```

```
15                  client.getInputStream())).readLine();
16              String reqs[] = request.split(" ");
17              if (!reqs[0].equals("GET")) {
18                  System.out.println("Can't accept non-GET requests");
19                  continue;
20              }
21              // Réponse HTTP
22              OutputStream output = client.getOutputStream();
23              BufferedReader file = null;
24              try { // On tente de récupérer le fichier demandé
25                  System.out.println("Fetching file "+reqs[1]);
26                  file = new BufferedReader(new FileReader("www"+reqs[1]));
27              } catch (Exception e) { // S'il n'existe pas, 404
28                  output.write("HTTP/1.0 404 Not Found\n\n"
29                          + "<html><h1>404 Not Found</h1></html>".getBytes());
30                  client.close();
31                  continue;
32              }
33              // Le fichier a été récupéré, on le sert
34              SimpleDateFormat dateFormat = new SimpleDateFormat(
35                      "EEE, d MMM yyyy HH:mm:ss Z", Locale.ENGLISH);
36              String response = "HTTP/1.0 200 OK \n"
37                      + "Date: "+dateFormat.format(new Date())+"\n"
38                      + "Server: TPsyr1.0\n\n";
39              String line = "";
40              while ((line = file.readLine()) != null)
41                  response += line + "\n";
42              output.write(response.getBytes());
43              client.close();
44          }
45      }
46  }
```

Ce serveur permet effectivement d'accéder aux fichiers contenus dans le sous-répertoire `www` du répertoire de travail courant ; nous avons là la base d'un serveur HTTP fonctionnel bien que pour l'instant limité.