


Programmation avancée


TP n°1 : manipulations simples des sockets

Détails

Écrit par stéphane Domas

Catégorie : Programmation avancée (/index.php/menu-lpsil/objets-connectes)

 Publication : 18 novembre 2009

 Affichages : 3533

Préambule

La communication entre deux objets connectés ou bien entre un objet et un serveur passe forcément par des protocoles proches du matériel, tels que le bluetooth, wifi, ethernet, ou un peu au-dessus IP/TCP. Bien entendu, cette utilisation peut être masquée par l'utilisation de protocoles de niveau applicatifs qui s'appuient sur ces derniers, comme quand par exemple un capteur avec wifi envoie son résultat à un serveur web via HTTP. Un des avantages de ces protocoles est qu'ils sont généralement utilisables dans bons nombres d'objets connectés fonctionnant avec un minimum de système d'exploitation (comme avec un raspberry), ou disons de fonctionnalités similaires (comme avec un arduino).

Ce TP est l'occasion de se familiariser avec TCP, l'établissement d'une connexion et la communication entre deux applications Java. Pour faire ces exercices, pas besoin de s'encombrer d'un raspberry : un seul ordinateur suffit.

Exercice 1

NB : cet exercice est un simple tutoriel permettant de voir une application client/serveur simple en Java

- Copiez/collez le code suivant dans un fichier `EchoServer.java`

```
1  import java.io.*;
2  import java.net.*;
3
4  class EchoServer {
5
6      public static void main(String []args) {
7
8          BufferedReader br = null; // pour lire du texte sur la socket
9          PrintStream ps = null; // pour envoyer du texte sur la socket
10         String line = null; // la ligne reçue/envoyée
11         ServerSocket conn = null;
12         Socket sock = null;
13         int port = -1;
14
15         if (args.length != 1) {
16             System.out.println("usage: Server port");
17             System.exit(1);
18         }
19
20         try {
21             port = Integer.parseInt(args[0]); // récupération du port sous forme int
22             conn = new ServerSocket(port); // création socket serveur
23         }
24         catch(IOException e) {
25             System.out.println("problème création socket serveur : "+e.getMessage());
26             System.exit(1);
27         }
28
29         try {
30             sock = conn.accept(); // attente connexion client
31             br = new BufferedReader(new InputStreamReader(sock.getInputStream())); // creation flux lecture lignes de textes
32             ps = new PrintStream(sock.getOutputStream()); // création flux écriture lignes de texte
33
34             line = br.readLine(); // réception d'une ligne
35             System.out.println("le client me dit : "+line); // affichage debug
36
37             ps.println(line); // envoi de la ligne précédemmetn reçue
38             br.close();
39             ps.close();
40         }
41         catch(IOException e) {
42             System.out.println(e.getMessage());
43         }
44     }
45 }
```

- Principe du serveur :
 - attend une connexion,
 - instancie un `BufferedReader` et un `PrintStream` grâce à la socket de communication obtenue,

- attend une ligne de texte envoyée par le client et l'affiche à l'écran,
 - renvoie cette ligne au client.
- Copiez/collez le code ci-dessous dans un fichier `EchoClient.java`

```

1  import java.io.*;
2  import java.net.*;
3
4  class EchoClient {
5
6      public static void main(String []args) {
7
8          BufferedReader br = null; // pour lire du texte sur la socket
9          PrintStream ps = null; // pour écrire du texte sur la socket
10         String line = null;
11         Socket sock = null;
12         int port = -1;
13
14         if (args.length != 3) {
15             System.out.println("usage: EchoClient ip_server port message");
16             System.exit(1);
17         }
18
19         try {
20             port = Integer.parseInt(args[1]); // récupération du port sous forme int
21             sock = new Socket(args[0],port); // création socket client et connexion au serveur donné en args[0]
22         }
23         catch(IOException e) {
24             System.out.println("problème de connexion au serveur : "+e.getMessage());
25             System.exit(1);
26         }
27
28         try {
29             br = new BufferedReader(new InputStreamReader(sock.getInputStream())); // création flux lecture lignes de texte
30             ps = new PrintStream(sock.getOutputStream()); // création flux écriture lignes de texte
31
32             ps.println(args[2]); // envoi du texte donné en args[2] au serveur
33             line = br.readLine(); // lecture réponse serveur
34             System.out.println("le serveur me répond : "+line); // affichage debug
35             br.close();
36             ps.close();
37         }
38         catch(IOException e) {
39             System.out.println(e.getMessage());
40         }
41     }
42 }
```

- Principe du client :
 - se connecte au serveur,
 - instancie un `BufferedReader` et un `PrintStream`, grâce à la socket de communication,
 - envoie une ligne de texte au serveur,
 - attend une ligne de texte au serveur et l'affiche.
- Pour exécuter le client et le serveur :
 - ouvrez deux terminaux, et pour chacun, allez dans le répertoire où se trouve vos deux fichiers java.
 - dans un des deux terminaux, compilez les 2 fichiers (i.e. `javac EchoClient.java` et `javac EchoServer.java`)
 - dans le premier terminal, lancez le serveur en choisissant un n° de port (par ex. 12345) : `java EchoServer 12345`
 - dans le deuxième, lancez le client en donnant comme adresse IP du serveur localhost, le même port et un message : `java EchoClient localhost 12345 "bonjour ca va?"`
- Normalement, le serveur affiche le message du client et lui renvoie.

Exercice 2

- Le serveur du l'exercice 1 n'est pas écrit comme un serveur habituel : il s'arrête après avoir traité une seule requête, d'un seul client.
- Modifiez le code de `EchoServer` afin que le serveur puisse répondre à un nombre potentiellement infini de clients successifs (= tant que l'on arrête pas le serveur).

Remarques :

- comme dans l'exercice 1, le client n'envoie qu'un seul message avant de se terminer.
- tant que le serveur est en train de répondre à un client, il ne peut pas accepter de connexion de la part d'un autre.

Exercice 3

- Modifiez le code de `EchoClient` pour que :
 - il n'y ait plus de troisième paramètre,
 - le programme demande à l'utilisateur de taper au clavier un message, qu'il envoie au serveur puis affiche la réponse du serveur (qui devrait être le message lui-même)
 - cette saisie/envoi soit répétée tant que le message n'est pas une ligne vide ou bien que l'utilisateur appuie sur ctrl+d.
- Modifiez le code de `EchoServer` de l'exercice 2 pour que :

- le programme puisse recevoir plusieurs message successifs d'un même client,
- que cette suite s'arrête quand le client envoie une ligne vide ou bien se déconnecte,

Remarque : quand le serveur a fini de servir un client, il faut qu'il continue d'attendre des connexions d'autres clients.

Exercice 4

- En vous inspirant du code de `EchoClient`, créez un fichier `SommeClient.java` pour que :
 - lors de chaque saisie clavier, l'utilisateur tape une ligne de texte contenant une série de nombre entiers séparés par des virgules. Cette série peut comporter un nombre variable d'entiers. Par ex. : 32,4,1,89
 - le client envoie cette ligne au serveur qui va la découper pour retrouver les entiers, en faire la somme et renvoyer le résultat au client. Ce dernier l'affiche sauf s'il reçoit un message d'erreur (cf. ci-dessous) auquel cas le client affiche un message d'erreur du style "requête malformée".
 - la saisie ne s'arrête que lorsque la liste est vide (ou bien appui sur ctrl+d)
- En vous inspirant du code de `EchoServer`, créez un fichier `SommeServer.java` pour que :
 - après connexion d'un client, le serveur attend une ligne de texte.
 - si celle-ci n'est pas vide ou null, le serveur découpe la chaîne en sous-chaînes en se servant du caractère virgule comme séparateur.
 - il convertit chaque sous chaîne en un entier, fait la somme de tous ces entiers et renvoie le résultat au client, sous forme texte. Si l'une des sous chaînes ne peut pas être convertie en entier, il s'agit d'une requête malformée et dans ce cas, le serveur renvoie au client le message "REQ_ERR".
 - il attend ensuite une autre ligne et reproduit le même processus tant qu'elle n'est pas vide, auquel cas, il rompt la connexion avec le client.

Remarques :

- Pour séparer une chaîne en morceaux, allez voir du côté de la méthode `split()` de `String`. Cette méthode prend en paramètre un séparateur et renvoie un tableau de `String`.
-

Exercice 5

- L'exercice 4 pose un problème classique : qui doit vérifier la validité du format d'une requête, le client ou bien le serveur ? Dans l'exo 4, c'est le serveur qui prend en charge la vérification.
 - L'exercice courant propose de faire une vérification au niveau client. C'est donc ce dernier qui doit :
 - découper la ligne saisie au clavier en morceaux,
 - vérifier que chaque morceau se convertit en int.
 - si c'est le cas, envoyer chaque entier au serveur et sinon ne rien faire.
 - Ensuite le client attend la somme calculée par le serveur.
 - De son côté le serveur se contente de recevoir les entiers sous forme texte, de les convertir puis de renvoyer leur somme au client.
-
- Cette version pose cependant un nouveau problème : comment le serveur sait combien d'entiers il doit additionner avant de renvoyer la somme ?
 - Modifiez le client et le serveur afin de faire la vérification du côté client ET de trouver une solution au problème ci-dessus.