


Programmation avancée

TP n°2 : sockets & requêtes

Détails

Écrit par stéphane Domas

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

 Publication : 18 novembre 2009

 Affichages : 3856

Préambule

Le premier TP ne contient que des exercices faisant intervenir un schéma de communication basique entre un client et un serveur. Il n'y a en effet pas vraiment de notion de requête pour demander différents services au serveur. Cette notion est pourtant essentielle. Par exemple, dès lors qu'un objet connecté doit faire plus que donner son état à un serveur, il est nécessaire qu'il puisse envoyer différents type de requêtes au serveur.

Ce deuxième TP permet de découvrir "en douceur" un exemple simple de client/serveur à base de requêtes.

Exercice 1

- Créer un fichier `Position.java` grâce au code suivant.

```
1 import java.io.*;
2
3 public class Position {
4
5     private double x,y,z;
6
7     public Position() {
8         x = 0; y = 0; z = 0;
9     }
10
11     public Position(double x, double y, double z) {
12         this.x = x; this.y = y; this.z = z;
13     }
14
15
16     public String toString() {
17         String msg = x+", "+y+", "+z;
18         return msg;
19     }
20
21     public double getX() {
22         return x;
23     }
24
25     public double getY() {
26         return y;
27     }
28
29     public double getZ() {
30         return z;
31     }
32
33     public Position clone() {
34         return new Position(x,y,z);
35     }
36
37     public boolean equals(Position p, double eps) {
38         if ( (Math.abs(p.x-x)>eps)|| (Math.abs(p.y-y)>eps)|| (Math.abs(p.z-z)>eps) ) return false;
39         return true;
40     }
41
42     public double distanceTo(Position p) {
43         double d = 0.0;
44         // à compléter : calcul la distance entre la position courante et p
45         return d;
46     }
47 }
```

- Créer un fichier `PositionServer.java` qui va faire différents calculs sur les positions qu'un client lui a envoyé.
 - Principe du serveur :
 - attend une connexion,
 - instancie un `PrintStream` et un `BufferedReader` grâce à la socket de communication obtenue,
 - attend une première ligne de texte contenant le n° de la requête, puis une seconde ligne contenant les paramètres de cette requête,
 - traite la requête et renvoie le résultat (ou l'accusé de réception) au client.
 - Les requêtes que le client peut envoyer sont :
 - n° 1 : demande au serveur de mémoriser une position. Celle-ci doit être stockée au niveau du serveur sous la forme d'un objet `Position` dans un `ArrayList`. Pour cette requête :
 - le client envoie une ligne contenant : "1",
 - puis le client envoie une ligne contenant 3 valeurs séparées par des virgules, représentant les coordonnées en x, y, z de la position à stocker.
 - le serveur renvoie une ligne contenant soit "OK" si l'opération s'est bien passée, soit "ERR_REQ" s'il n'y a pas 3 valeurs fournies ou/et elles n'ont pas le bon format (i.e. nombre à virgule)
 - n°2 : demande au serveur de calculer la longueur du chemin parcouru. Pour cette requête :
 - le client envoie une ligne contenant : "2",
 - le serveur fait la somme des distances entre chaque couple de position stockée et renvoie cette somme au client qui l'affiche.
 - n°3 : demande au serveur si le client est déjà passé par une position donnée avec un facteur de proximité (cf. eps dans la classe `Position`). Pour cette requête :
 - le client envoie une ligne contenant : "3",
 - puis le client envoie une ligne contenant 4 valeurs séparées par des virgules, la première étant le facteur de proximité, puis les coordonnées en x, y, z de la position à tester.
 - le serveur vérifie si (x,y,z) correspond à une position déjà stockée, compte tenu du facteur de proximité.
 - le serveur renvoie "TRUE" si c'est le cas, "FALSE" sinon. De plus, s'il n'y a pas assez de paramètres ou bien qu'ils ne sont pas au bon format, le serveur renvoie "ERR_REQ"
 - Créer un fichier java `PositionClient.java`.
 - Principe du client :
 - se connecte au serveur,
 - instancie un `PrintStream` et un `BufferedReader` grâce à la socket de communication,
 - envoie une dizaine de requêtes n°1 au serveur, par exemple en partant de x=y=z=0, et en incrémentant leur valeur entre chaque requête,
 - envoie une requête n°2
 - envoie deux requêtes n°3 en envoyant d'abord une position correspondant à l'une des positions précédemment envoyées, et un autre qui ne correspond pas.
-

Exercice 2

- NB : dans ce TP, le serveur n'accepte qu'un seul client à la fois. Cet exercice permet de mettre en place ce qu'il est nécessaire pour gérer plusieurs clients, sans toutefois aller à la mise en place de connexions simultanées.
- Modifier l'exercice 1 pour que :
 - lorsqu'un client se connecte au serveur, ce dernier lui envoie un numéro représentant l'identifiant du client. Cet identifiant doit être incrémenté à chaque fois qu'un client se connecte afin que chacun ait un identifiant différent.
 - le serveur utilise un `HashMap<Integer, ArrayList<Position>` pour stocker les positions des clients qui se sont connectés. La clé de type `Integer` représente l'identifiant de chaque client.
 - lorsqu'un client envoie une requête, il envoie d'abord une ligne avec le n° de la requête, puis une ligne avec son identifiant, et enfin les paramètres éventuels de la requête. Cela permet au serveur de faire les traitements demandés sur les positions associées au client.
 - si le client a envoyé un identifiant incorrect, le serveur répond en envoyant "NOID_ERR". Cela permet au client de différencier le type d'erreur quand elle a lieu.

Exercice 3

- Modifier le client pour qu'il soit interactif, par exemple en utilisant le canevas de code donné en cours.
 - Les requêtes sont donc tapées au clavier par l'utilisateur et doivent avoir le format suivant :
 - requête 1 : storepos x,y,z.
 - requête 2 : pathlen
 - requête 3 : findpos facteur_proximité,x,y,z.
 - Le client doit se charger de vérifier que les lignes tapées par l'utilisateur suivent ces format et de transformer ces lignes en une requête auprès du serveur.
 - Par exemple, si l'utilisateur tape : "storepos 1.5,3,-2", le client va envoyer une requête n°1 en envoyant une ligne contenant "1", puis une ligne contenant "id_client,1.5,3,-2", où id_client doit être remplacé par l'identifiant client donné par le serveur.
-

Exercice 4 (bonus pour ceux qui vont très vite !)

- Modifier l'exercice 3 pour ajouter les requêtes suivantes :
 - n°4 : demande au serveur le dénivelé positif et négatif le long du parcours du client,
 - n°5 : demande au serveur la dernière position connue des autres clients.