Programmation avancée

canevas de base client/server (en Java)

Détails
Écrit par stéphane Domas
Catégorie : Programmation avancée (/index.php/menu-lpsil/objets-connectes)
Publication : 18 novembre 2009
Affichages : 1225

Préambule

On suppose que :

- le protocole est défini pour des communication en mode ligne de texte,
- les requêtes sont toutes de la forme : `id_requête id_client param1 param2 ...` c'est-à-dire des lignes de textes avec des "mots" séparés par des espaces.
- les requêtes sont toutes identifiées par un simple numéro 1, 2, ... excepté la requête QUIT qui permet au client d'arrêter la connexion.
- juste après la connexion, le serveur envoie au client un identifiant unique, qui doit ensuite être envoyé pour toute requête, juste avant les paramètres de la requête.
- les réponses d'erreur sont au format : `ERR message`

1°/ Serveur mono-clients successifs

- On suppose que le serveur est décomposé en 2 classes : `Server` and `ServerTCP`
- la communication de ligne de texte utilise les flux de type `PrintStream` et `BufferedReader`

`Server.java` (le point d'entrée) :

```java
import java.net.*;
import java.io.*;

class Server {

  public static void usage() {
    System.err.println("usage : java Server port");
    System.exit(1);
  }

  public static void main(String[] args) {

    if (args.length != 1) {
      usage();
    }

    ServerTCP server = null;
    try {
      int port = Integer.parseInt(args[0]);
      server = new ServerTCP(port);
      server.mainLoop();
    }
    catch(NumberFormatException e) {
      System.err.println("invalid port number");
      System.exit(1);
    }
    catch(IOException e) {
      System.err.println("cannot start server");
      System.exit(1);
    }
  }
}
```

`ServerTCP.java` (le serveur réel) :

```
import java.io.*;
import java.net.*;
import java.util.*;

class ServerTCP  {

  BufferedReader br;
  PrintStream ps;
  ServerSocket waitClient;
  Socket sockComm;
  int idClient;
  // other attributes, usefull to process request

  public ServerTCP(int port) throws IOException {
    idClient = 0;
    waitClient = new ServerSocket(port);
  }

  public void mainLoop() {
    while(true) {
      try {
        sockComm = waitClient.accept();
        br = new BufferedReader(new InputStreamReader(sockComm.getInputStream()));
        ps = new PrintStream(sockComm.getOutputStream());
        idClient += 1;
        ps.println(idClient); // send its id to the client
        requestLoop();
        br.close(); // close stream => close connection
        ps.close();
      }
      catch(IOException e) {
        System.out.println("client disconnected");
      }
    }
  }

  public void requestLoop() throws IOException {

    String req = "";
    boolean stop = false;

    while(!stop) {
      req = br.readLine();
      if (req == null) return; // stop loop : suppose that client disconnected
      if (req.isEmpty()) continue; // empty req => do nothing special
      String[] reqParts = req.split(" ");

      // check if id is correct
      boolean invalidId = false;
      int idRecv = 0;
      if (reqParts.length == 1) invalidId = true;
```

```
      try {
        idRecv = Integer.parseInt(reqParts[1]);
        if (idRecv != idClient) invalidId = true;
      }
      catch(NumberFormatException e) {
        invalidId = true;
      }
      if (invalidId) {
        ps.println("ERR invalid id");
        continue;
      }

      if ("1".equals(reqParts[0])) {
        if (reqParts.length != ...) { // check number of params
          ps.println("ERR invalid number of parameters");
        }
        else {
          processRequest1(reqParts[2], ... ); // replace ... by a list of reqParts[x]
        }
      }
      else if ("2".equals(reqParts[0])) {
        if (reqParts.length != ...) {
          ps.println("ERR invalid number of parameters");
        }
        else {
          processRequest2(reqParts[2], ... ); // replace ... by a list of reqParts[x]
        }
      }
      /* etc. with other requests */
      else if ("quit".equals(reqParts[0])) {
        stop = true;
      }
    }
  }

  public void processRequest1(String param1, ...) throws IOException {
    // tests on parameters : in range ? good type ? ...
    // send an error message if params are not ok
    ps.println("OK");     // if all ok,
    // process the request and send the result to the client
  }

  // etc. with other processRequestX() methods
}
```

2°/ Client interactif en mode console

- On suppose que le client s'exécute dans un terminal et que les requêtes sont tapées au clavier par l'utilisateur.
- Pour ce dernier, envoyer une requête consiste à exprimer textuellement son besoin sous la forme d'une ou plusieurs ligne de texte avec un format bien spécifié.
- ATTENTION ! Ce format n'est pas forcément celui qui est utilisé pour les échanges entre le client et le serveur. Dans ce cas, le programme client doit d'abord transformer les ordres de l'utilisateur en un format compatible avec le protocole de communication avec le serveur.

- Pour le code ci-dessous, on suppose que l'utilisateur tape des lignes avec un format un peu différent de celui attendu par le serveur : nom_requête param1 param2 ...,
- De plus, juste après la connexion, le serveur envoie au client un identifiant unique, qui doit ensuite être envoyé pour toute requête, juste avant les paramètres de la requête.

- Le canevas est séparé en 2 classes: `Client` et `ClientTCP`

`Client.java` (le point d'entrée) :

```
import java.net.*;
import java.io.*;

class Client {

  public static void usage() {
    System.err.println("usage : java Client ip_server port");
    System.exit(1);
  }

  public static void main(String[] args) {

    if (args.length != 2) {
      usage();
    }

    ClientTCP client = null;
    try {
      int port = Integer.parseInt(args[1]);
      client = new ClientTCP(args[0], port);
      client.requestLoop();
    }
    catch(NumberFormatException e) {
      System.err.println("invalid port number");
      System.exit(1);
    }
    catch(IOException e) {
      System.err.println("cannot communicate with server");
      System.exit(1);
    }
  }
}
```

```
import java.io.*;
import java.net.*;
import java.util.*;

class ClientTCP  {

  Socket sockComm;
  BufferedReader br; PrintStream ps;
  String id; // the client id sent by the server with request 0

  public ClientTCP(String serverIp, int serverPort) throws IOException {
    sockComm = new Socket(serverIp, serverPort);
    ps = new PrintStream(sockComm.getOutputStream());
    br = new BufferedReader(new InputStreamReader(sockComm.getInputStream()));
  }

  public void requestLoop() throws IOException {

    boolean stop = false;
    String reqLine = null;
    BufferedReader consoleIn = null; // to read from keyboard
    String[] reqParts = null;

    // get my id
    id = br.readLine();

    consoleIn = new BufferedReader(new InputStreamReader(System.in));

    while (!stop) {
      System.out.print("Client> ");
      reqLine = consoleIn.readLine(); // read user's input
      reqParts = reqLine.split(" "); // split the user's request thanx spaces

      if ("req1".equals(reqParts[0])) {
        String req = "1 "+id+" ";
        // adding to req parameters obtained from reqParts[1] reqParts[2], ...)
        ps.println(req);
        String answer = br.readLine();
        // processing answer
      }
      else if ("req2".equals(reqParts[0])) {
        String req = "2 "+id+ " ";
        // adding to req parameters obtained from reqParts[1] reqParts[2], ...)
        ps.println(req);
        String answer = br.readLine();
        // processing answer
      }
      /* etc. with other requests */
       else if ("quit".equals(reqParts[0])) {
        ps.println("quit");
        stop = true;
```