Programmation avancée

canevas serveur multithreadé

Détails

Écrit par stéphane Domas

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

Publication : 18 novembre 2009

Affichages : 1038

Serveur multi-threadé

- Lorsqu'un serveur doit recevoir des requêtes simultanées, il devient nécessaire de créer un thread par client.
- Il faut donc modifier le canevas de base donné dans canevas de base client/serveur (/index.php/menu-lpsil/objets-connectes/2498-1canevas-de-base) avec l'ajout d'une classe représentant les threads du serveur.
- Dans le code ci-dessous, on garde les hypothèses sur le protocole de communication : envoi/réception de lignes de texte avec des requêtes données dans une seul ligne.

- Globalement, le canevas multi-threadé consiste a déplacer tout la partie communication avec le client dans la classe de thread.
- Cependant, selon les traitements nécessaires pour traiter les requêtes, les threads vont peut être avoir besoin ou non d'accéder aux mêmes données. Si c'est le cas, il y a un partage de ces données en mémoire.
- Dans l'exemple ci-dessous, on suppose que tout ce qui est partagé entre les threads est regroupé dans une seule classe `ServerData`, objet que le serveur principal va instancier puis donner en paramètre au constructeur des threads.

Remarques :

- Si les threads sont amenés à modifier cet objet partagé de façon concurrente, il faudra certainement utilisé des mutex, voire des attentes sur conditions pour régler les conflits d'accès.
- Si les threads sont mal programmés et qu'un d'entre eux provoque une erreur critique, le serveur crashe, et pas seulement le thread fautif.

- On aboutit aux canevas suivants :

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

class ServerTCP {
  private ServerSocket sockConn;
  private int id;
  private ServerData data; // l'objet partagé entre les threads

  public ServerTCP(int port) throws IOException {
    sockConn = new ServerSocket(port);
    data = new ServerData( ... );
    id = 0;
  }
  public void mainLoop() {
    while(true) {
      try {
        Socket sockComm = sockConn.accept();
        id += 1;
        ThreadServer t = new ThreadServer(id, sockComm, data);
        t.start();
      }
      catch(IOException e) { ... }
    }
  }
}
```

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

public ServerThread extends Thread {
  private Socket sockComm;
  BufferedReader br; PrintStream ps;
  private int id;
  private ServerData data;
  // autres attributs nécessaires aux traitements des requête

  public ServerThread(int id, Socket sockComm, ServerData) {
    this.id = id;
    this.sockComm = sockComm;
    this.data = data;
    // init. autres attributs
  }

  public void run() {
    try {
      br = new BufferedReader(new InputStreamReader(sockComm.getInputStream()));
      ps = new PrintStream(sockComm.getOutputStream());
      requestLoop();
    }
    catch(IOException e) { // erreur => fin thread }
  }

  public void requestLoop() throws IOException {
    // code identique à celui du canevas serveur mono-threadé
  }

  public void processRequest1(String param1, ...) throws IOException {
    // code adapté du canevas serveur mono-threadé en utilisant
    // l'attribut data pour traiter les requêtes.
  }

  // etc. avec les autres méthodes processRequestX()
}
```