Université Lille 1 *IEEA*

Rapport de Travaux Pratiques

Construction d'Applications Réparties (CAR)

TP1 : Serveur FTP

Auteur :

Anis TELLO
Salla DIAGNE

Professeur :

Laurence DUCHIEN

17 Février 2014

Introduction

Ce TP est une implémentation en JAVA d'un serveur FTP basique.

Listing des dossiers et fichiers du projet

conf/ : contient les fichiers de propriétés nécessaires à configurer le serveur

lib/ : contient les librairies (`.jar`) dont le projet est dépendant

src/ : contient les fichiers sources (`.java`) du projet

test/ : contient les fichiers de tests

ftpServer.jar : archive exécutable du projet

Utilisation

Pour lancer le serveur, il suffit d'ouvrir un terminal et taper la commande :

- `java -jar ftpServer.jar [port] [baseDirectory]` avec :

  port : le numéro du port TCP sur lequel écoutera le serveur

  baseDirectory : le répertoire auquel auront accès les clients potentiels

Architecture

- *package ftp*

  `FTPDatabase` : classe représentant la base de données du projet. Elle contient les informations relatives aux comptes authentifiés, l'adresse IP du serveur ou encore la liste des codes de retour et des messages.
  `FTPLoggable` : classe abstraite représentant une entité capable de logger des messages sur la sortie standard ou dans un fichier.
  `FTPMessageSender` : classe abstraite représentant un "envoyeur de messages". Toute classe héritant de cette classe est capable d'envoyer des commandes ou des données à travers les sockets.
  `FTPRequest` : classe représentant une requête FTP. Une requête FTP est définie par une commande (obligatoire) et un argument (optionnel).

`FTPRequestHandler` : classe représentant un handler de requête FTP. Une instance de cette classe sera lancée dans un nouveau Thread à chaque fois qu'un client est connecté.
`FTPServer` : classe représentant le serveur du projet.
`Main` : classe principale.
- *package command*
  `FTPCdupCommand` : classe représentant la commande CDUP.
  `FTPCommand` : interface représentant une commande (polymorphisme).
  `FTPCommandManager` : classe représentant le manager des commandes.
  C'est cette classe qui se chargera d'exécuter la commande adéquate
  quand le serveur en reçoit une.
  `FTPCwdCommand` : classe représentant la commande CWD.
  `FTPListCommand` : classe représentant la commande LIST.
  `FTPNotImplementedCommand` : classe représentant une commande inconnue.
  `FTPPassCommand` : classe représentant la commande PASS.
  `FTPPasvCommand` : classe représentant la commande PASV.
  `FTPPortCommand` : classe représentant la commande PORT.
  `FTPPwdCommand` : classe représentant la commande PWD
  `FTPQuitCommand` : classe représentant la commande QUIT.
  `FTPRetrCommand` : classe représentant la commande RETR.
  `FTPStorCommand` : classe représentant la commande STOR.
  `FTPSystCommand` : classe représentant la commande SYST.
  `FTPTypeCommand` : classe représentant la commande TYPE.
  `FTPUserCommand` : classe représentant la commande USER.
- *package configuration*
  `FTPClientConfiguration` : classe représentant la configuration d'un client.
  `FTPServerConfiguration` : classe représentant la configuration du serveur.

Parcours de code

En premier lieu, une base de données est mise en place en créant une instance de la classe `FTPDatabase`, instance que l'on injectera en constructeur de la classe `FTPServer` avec un numéro de port (> 1023) et un répertoire de base. La configuration du client (`FTPClientConfiguration`) sera paramétrée dans `FTPServer`.

```
public FTPServer(int port, String baseDirectory, FTPDatabase database) {
```

```
    super(database);
    _configuration = new FTPServerConfiguration(port, baseDirectory);
}
```

Comme tout bon serveur qui se respecte, le serveur sera en écoute continue sur le port. Une fois connecté à un client, le serveur "traite" le client en lançant un processus en parallèle, et ceci après lui avoir envoyé un message de bienvenuee. Ce Runnable se verra injecter la base de données, la configuration du serveur et le manager de commandes qui contiendra les commandes acceptées par le serveur.

```
final FTPCommandManager commandManager = new FTPCommandManager();
commandManager.addCommand(...);
...
while (true) {
    ftpServer.connectToClient();
    System.out.println("--> New client connected on this server.");
    ...
    final FTPRequestHandler requestHandler = new FTPRequestHandler(ftpDatabase,
    serverConfiguration, commandManager);
    new Thread(requestHandler).start();
}
```

La configuration client sera paramétrée dans `FTPRequestHandler`. À chaque commande reçue, le command manager se chargera d'exécuter le bon traitement.

```
FTPRequestHandler {
    public synchronized void processRequest(FTPRequest request) {
        \_commandManager.execute(request.getCommand(),
        request.getArgument(), \_clientConfiguration);
}
```

Le design pattern Command est ici dans le command manager. Ainsi, chaque commande sera visitée pour savoir si elle doit être exécutée ou pas. La requête est donc encapsulée et son mode de traitement est inconnu du `FTPRequestHandler`.

```
public void execute(String commandString, String argument,
FTPClientConfiguration clientConfiguration) {
```

```
    for (final FTPCommand command : \_commands) {
        if (command.accept(commandString)) {
            command.execute(argument, clientConfiguration);
            break;
        }
    }
}
```

Chaque commande a un type commun `FTPCommand`.

```
public interface FTPCommand {

    boolean accept(String command);

    void execute(String argument, FTPClientConfiguration clientConfiguration);

}
```

La commande `FTPNotImplementedCommand` acceptera toutes les commandes.