Projet AppRef“BRI”

ligne courte

CHATHIANATHAN Jeneifan | CHALONS Guillaume  
203-204

# Sommaire

1. Introduction
2. Présentation technique
3. Programmation concurrentielle
4. Protocole de communication
5. Services
6. Bilan du projet

# 

# 

# Introduction

Le but de ce projet est de créer une application client/serveur qui permet à des programmeurs de mettre à disposition de personne des services qu’ils auront développées. Ces programmeurs pourront :

* installer / désinstaller un service
* allumer / éteindre un service
* mettre à jour un service

Les services devront être mis en ligne sur un serveur FTP accessible par le serveur.

# Présentation technique

Deux types de personnes interagissent avec le serveur : les amateurs et les programmeurs. Pour une meilleure séparation entre les services, nous avons opté pour deux serveurs lancés sur deux ports différents 5000 et 5001, respectivement utilisés pour contacter le service amateur et programmeur.

Chaque service devra respecter les normes BRI c’est-à-dire :

* implémenter l'interface bri.Service
* ne pas être abstract
* être publique
* avoir un constructeur public (Socket) sans exception
* avoir un attribut Socket private final
* avoir une méthode public static String toString() sans exception
* le package source doit être le pseudo du programmeur

Les programmeurs pourront utiliser des bibliothèques. Ces dernières devront être placées dans un dossier libs et un fichier libs.loader contenant à chaque ligne le chemin relatif de la bibliothèque.

Par exemple pour un programmeur nommé john, le fichier john/libs.loader contiendra :

“libs/lib1.jar

libs/lib2.jar”

# Programmation concurrentielle

Nous avons développé notre application dans une optique de fonctionnement concurrentielle. C’est-à-dire, que nous avons utilisé des Threads pour effectuer des tâches en parallèles et pour que notre application puisse fonctionner correctement, nous avons dû protéger certaines parties de notre application soumise à des modifications venant de plusieurs threads. Nous avons donc sécurisé tout ce qui est de la modification d’un programmeur ou des services en utilisant l'objet Vector et ConcurrentHashMap.

De plus au niveau des sockets, chaque client est traité dans un nouveau thread.

# Protocole de communication

Au niveau du protocole de communication entre le client et le serveur, nous avons opté pour un formatage de message assez simple. Néanmoins, ce formatage nous permet de réduire au maximum le code côté client et donner la main mise au serveur sur ce qui doit être affiché et demander au client. Cet aspect du protocole nous permet (en situation réelle) de modifier l’affichage côté client, ainsi que les informations demandées directement côté serveur sans modifier les différents clients. La seule chose qui rend spécifique un client par rapport à un autre, c’est le port et l’ip du serveur auquel il souhaite se connecter.

Au niveau du protocole, les messages sont formatés comme cela :

« commande : argument »

Nous avons utilisé 4 différentes commandes :

- msg : qui va afficher toutes les données passées en argument dans la console

- ask : qui va faire la même chose que pour la commande msg, sauf qu’il lancera également un scanner et attendra que l’utilisateur tape quelque chose dans la console pour l’envoyer au serveur

- newLine : c’est un simple saut de ligne (pour avoir un affichage plus digeste)

- end : qui va fermer la socket coté client.

Au final, ces 4 possibilités ont été suffisantes pour pouvoir créer l’interface console des 3 différents services.

Par la même occasion, nous avons donné la possibilité que dans le futur le client puisse utiliser un autre support de communication que les sockets. Néanmoins, les 4 commandes fonctionneront toujours. Cette spécificité est implémentée par le biais d’une interface «Communicator » que nous avons développée qui spécifie les fonctions basiques d’une communication tel que nous l’avons imaginé :

- afficher un message reçu

- savoir quand on est connecté

- savoir quand on est déconnecté

- afficher une erreur

- envoyer un message

- terminer la communication

En ce qui concerne la partie serveur, de la même façon, nous avons donné la possibilité de faire évoluer le support de communication sans devoir redévelopper les 3 services. En effet, ces services héritent tous d’une même classe qui par le biais des macros basiques va communiquer avec le client au travers d’une socket. On a donc plus besoin de s’embêter avec les spécificités du support au niveau des services en eux-mêmes puisque tout est géré via cette classe parente.

La liste des macros disponibles est :

- lire()

- ecrire(String msg)

- poserUneQuestion(String msg)

- sauterUneLigne()

- terminer()

Ce protocole doit être respecté par tous les programmeurs pour pouvoir communiquer convenablement avec les clients. Nous avons choisi de définir un protocole de communication pour normaliser au maximum la communication entre les services et les clients.

# Services

Nous avons développé 2 services :

* un service simple d’inversion de mot
* un service plus complexe d’analyse de fichier XML.

Le fonctionnement du premier étant trivial, nous nous attarderons seulement sur le second.

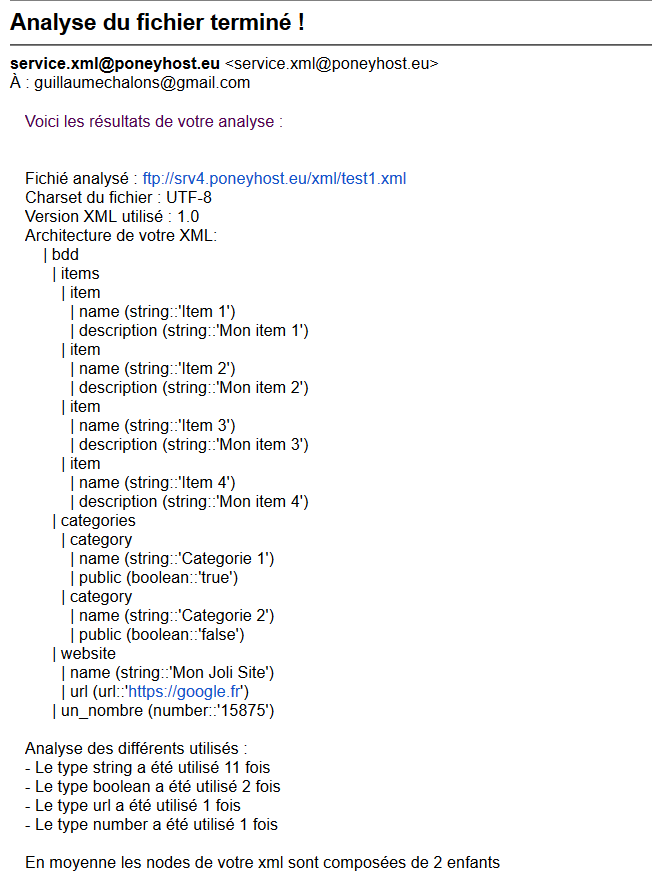
## Analyse de fichier XML

Ce service permet à partir d’un fichier xml donné d’effectuer une analyse sur ce dernier et d’envoyer les résultats par emails. Le service va s’occuper d’extraire l’architecture du fichier XML, déduire les différents types utilisés (number, boolean, url, string etc..) et afficher quelques statistiques.

Exemple d'interaction avec le service :

|  |
| --- |
| ######################################################  Bienvenue sur la plateforme de services dynamiques  BRI  ######################################################  Liste des services disponibles actuellement  #1 ServiceXML par guillaume  Merci d'indiquer le numéro du service que vous souhaitez utiliser  > 1  Bienvenue sur le service d'analyse de fichier xml  Merci d'indiquer un lien ftp  > ftp://srv4.poneyhost.eu/xml/test1.xml  Nous allons débuter l'analyse du fichier.  ftp: //srv4.poneyhost.eu/xml/test1.xml  L'analyse pouvant prendre du temps, nous allons vous envoyer les résultats par email  Merci d'indiquer votre adresse email.  > guillaumechalons@gmail.com  Très bien !  Nous allons traiter votre fichier et vous enverrons les résultats sur votre adresse email.  Vous êtes désormais deconnecté. |

Voici le mail qui a été envoyé :



# Améliorations possibles

Le déchargement des classes pourrait être amélioré. En effet, à l’heure actuelle, pour désinstaller un service nous le supprimons simplement de la liste stockant tous les services et nous surchargeons la fonction loadClass de la class URLCLassLoader ce qui permet de prendre toujours la version la plus récente de la classe sans regarder dans le contexte des parents. Néanmoins, on pourrait imaginer avertir un amateur qui serait en train d’utiliser une classe sur le point d’être mis à jour. On ne gère pas le garbage collector.

On pourrait par exemple faire en sorte qu’un service soit utilisable uniquement un certain nombre de fois ou restreindre l’utilisation d’un service à une plage d’adresse ip / pays.

Ou encore pouvoir gérer des rôles entre les programmeurs :

* restreindre certaines actions à un groupe restreint de personne