Projet d’Application Répartie

Sommaire

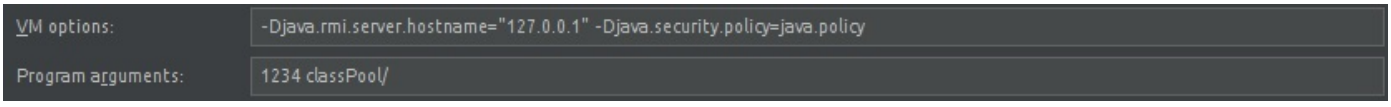
Configuration

Afin de bien faire communiquer le client au RMIRegistry ainsi qu’aux différents serveurs, ces derniers nécessitent certaines configurations de leur JVM.

Configuration du ClassServer

Le programme necessite comme argument :

- Aucun argument
- un seul argument representant le port
- deux arguments répresentant le port est le lien vers les .class de l’application



Configuration du Servor

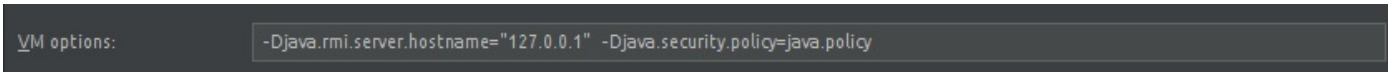
La JVM du serveur nécessite la configuration suivante: `-Djava.security.policy=java.policy` , `-Djava.rmi.server.codebase=http://xxx.xxx.xxx.xxx:1234`, `-Djava.rmi.server.hostname="xxx.xxx.xxx.xxx"` (en general `localhost`) qui permet de configurer l'adresse IP à laquelle le serveur sera affecté.

Configuration du ClientRMI

La JVM du client nécessite la configuration suivante : `-Djava.rmi.server.hostname="xxx.xxx.xxx.xxx"`, `-Djava.security.policy=java.policy` , `-Djava.rmi.server.codebase=http://xxx.xxx.xxx.xxx:1234` qui permet de configurer la gestion des connexions aux serveurs. De base tout le monde est accepté puisque le fichier java.policy contient `grant { permission java.security.AllPermission; } ;`.

Configuration du Serveur

La JVM du serveur nécessite la configuration suivante: `-Djava.security.policy=java.policy` , `-Djava.rmi.server.hostname="xxx.xxx.xxx.xxx"` (en general `localhost`) qui permet de configurer l'adresse IP à laquelle le serveur sera affecté.



Configuration du RMIRegistry

Le rmiregistry doit être lancé en se plaçant dans le repertoire parent du package “app” avec la commande suivante permettant de le lancer sur le port 2000 :

```
> rmiregistry 2000
```

Explication de la solution

Comme expliqué précédemment il y a donc quatre éléments distincts :

- Un serveur de class permettant le téléchargement dynamique de classes
- Un UniversalRegistry qui contient une hashmap de services et de données
- Un serveur qui peut déposer des services et des données
- Un client qui peut utiliser ces services et ces données

Le serveur de classe

Le serveur de classe permet aux différents producteurs/consommateurs de télécharger les stubs des classes manquantes pour leur exécution. Nous devons le paramétrer afin qu’il pointe vers un dossier contenant nos `.class` (dans notre projet le dossier porte le nom de “classPool/”).

Le servor

Le servor permet de stocker les services ainsi que les données dans une Map. A chaque appel d'un client pour obtenir un service ou une donnée, il suffira alors d'incrémenter le bon enregistrement pour les garder trier, avant de lui envoyer la référence du service ou la copie de la donnée.

Le Serveur

Le serveur implémente l'interface `IServorCommunication` et peut déposer des objets, des services, et meme des queues JMS dans la HastmMap du Servor en utilisant la méthode `rebind` fournie par l'interface (ce qui ecrasera l'objet s'il existe deja). Le serveur est donc un producteur.

Le Client

Le client implémente l'interface `IServorCommunication` et peut recupérer des objets, des services, et meme s'inscrire dans une queues JMS afin d'en lire les messages se trouvant dans la HashMap du Servor en utilisant la méthode `lookup` fournie par l'interface. Le serveur est donc un consommateur.

Problemes Rencontrés

Un des premiers points clefs a été la vision du projet. En effet cela a pris du temps de passer à une architecture, concrète apres avoir levé

l’abstraction du sujet.

Une deuxième difficulté aura été le debuggage des RemoteExceptions. Ces exceptions pouvant avoir des causes différentes et variées il est souvent long d’en trouver la cause.

Equipe

Projet réalisé dans le cours d’Application répartie par :

- DAHMOUL Salah
- SARROCHE Nicolas