Tutorial : RMI

CREATION : 2010/04/22
MISE A JOUR : 2010/11/08

RMI a permis de poser les bases des applications distribuées en formalisant la communication entre un client et un serveur. Il est possible d'échanger des objets entre des machines virtuelles différentes (pas forcément sur la même machine). Le serveur crée des **objets distants** (ou *remote object*, objet qui transite par RMI), rend accessible des références sur ces objets et attend qu'un client appelle une méthode de ces objets. Le client obtient une référence sur ces objets distants et invoque les méthodes de ces objets. On appelle **stub** une référence sur un objet distant. C'est une sorte de proxy dans la mesure où on ne peut appeler toutes les méthodes de l'objet mais qu'on se limite à celles définies dans l'interface distante.

La localisation des objets se fait grâce à un programme dédié appelé le registre RMI (le service JNDI peut aussi convenir). La communication entre le client et le serveur est transparente pour le développeur : tout se passe comme si les méthodes de l'objet étaient appelées directement (sérialisation-désérialisation automatique). Il faut juste que les différents opérateurs puissent charger les définitions des classes (chargement dynamique de classes) : la solution la plus simple est d'utiliser un simple serveur web (localhost dans notre cas), même si ce n'est pas obligatoire.

1 : Tutorial

Suivre le tutoriel officiel de Sun/Oracle pour la création des classes client et serveur.

http://java.sun.com/docs/books/tutorial/rmi/index.html

Au final, vous devez avoir 5 classes (dont 5 fichiers sources et 5 fichiers de pseudo-codes)

- ComputeEngine, l'objet distant par l'interface Compute mais aussi le serveur dans le sens où cette classe rend un service.
- ComputePi, le client qui demande un service (une tâche à réaliser)
- Pi, la tâche à faire calculer au ComputeEngine
- Compute, le contrat à passer entre le client et le serveur
- Task, la tâche à réaliser.

ComputeEngine est exécuté sur la même machine que le registre RMI mais ce n'est pas une obligation. Il faut que sur la machine qui publie ComputeEngine publie également les interfaces Compute et Task pour que les clients potentiels sachent quoi demander. ComputePi et Pi sont sur la même machine. La classe Pi est publiée pour qu'elle soit connue de ComputeEngine. Evidemment, quand on parle de publication/distribution de code, on ne s'intéresse qu'au pseudo-code.

Le tutoriel vous fait créer un package à partir de Compute et Task car il est bien plus commode de ne distribuer qu'un seul fichier (ces interfaces sont nécessaires pour compiler ComputeEngine et ComputePi)

2 : Remarques

- Pour créer les fichiers jar nécessaires à la mise en œuvre, on pourra procéder comme dans le tutorial ou se servir d'Eclipse. En effet, un clic droit sur un projet ou un package permet d'accéder au menu **Export**. Dans ce menu, se cache la possibilité de créer un fichier jar avec une jolie interface graphique : création automatique d'un fichier MANIFEST, choix de la classe Main, compression …
- Si votre compte dispose d'un répertoire public_html avec les droits corrects, vous avez un site web à votre disposition. Deux sous-répertoires vous permettront de simuler des utilisateurs différents.
- Le paramètre codebase attend un protocole : ftp, http ou file … Si vous ne voulez pas vous embêter avec le serveur web, précisez juste l'emplacement des fichiers (mais le chemin ABSOLU). Les lignes d'exécution du client et du serveur suivant le système d'exploitation ne sont **pas cohérentes** : sous Windows, on donne le chemin des ressources. Sous Linux, on suppose qu'un serveur Web a été installé.
- Pour que le chargement dynamique des classes soit possible, il est impératif que les classes utilisées ne soient pas accessibles lors du lancement du registre RMI …
- Vous n'avez rien compris au tutoriel officiel ? Si vous tombez sur un tutoriel/exemple qui vous demande d'utiliser rmic, vous utlisez un tutoriel pour une plateforme 1.4 ou inférieure.

Pseudo-correction.

Je suppose que vous avez placé les classes dans les répertoires compute, engine et client d'un même sous-répertoire nommé RACINE et que les classes ont été compilées. Le fichier jar compute.jar est placé dans RACINE. RACINE contient également cs.policy. RACINE est le **chemin absolu**.

```
start rmiregistry ou rmiregistry & dans votre home par exemple
```

fichier cs.policy

```
grant codeBase "file:RACINE" {
   permission java.security.AllPermission;
};
```

: sous LINUX
; sous WINDOWS

Lancer le moteur dans un terminal à partir de RACINE

```
java -cp .;./compute.jar -Djava.rmi.server.codebase=file:RACINE/ -Djava.security.policy=cs.policy
 -Djava.rmi.server.hostname=localhost engine.ComputeEngine
> ComputeEngine bound
```

RACINE/compute.jar sous LINUX

Lancer le client dans un autre terminal à partir de RACINE

```
java -cp .;./compute.jar -Djava.rmi.server.codebase=file:/RACINE -Djava.security.policy=cs.policy
client.ComputePi localhost 45
> 3.14159265358979323846264338327950288419716939937
```