TP 2 – RMI

Kenneth VANHOEY [1]
https ://dpt-info.u-strasbg.fr/~kvanhoey

Le TP est à rendre sur *Moodle* (sous forme d'archive nommée « <nom>_<prenom>_TP2_RMI.zip ») avant vendredi 26 octobre 2012 à 12h00 (fin du quatrième TP) dans la section *rendu TP2* prévue à cet effet. Les retards ne seront pas acceptés. Dans chaque répertoire, un makefile ou un shelscript devra permettre de compiler l'exercice. Dans un fichier *avancement.txt*, précisez pour chaque exercice votre état d'avancement, en indiquant si l'exercice :

1. ne compile pas ;
2. compile mais ne s'exécute pas normalement ;
3. compile et s'exécute normalement mais ne donne pas le bon résultat ;
4. compile, s'exécute normalement et donne le bon résultat.

Vous pouvez vous aider de ce cours/tutoriel sur Java RMI, ainsi que de l'API de Java 6 (version installée sur Turing). Pour les différents exercices, pensez bien à tuer vos processus à chaque fois que vous vous déloggez d'une machine Linux. En ce qui concerne les processus java ou rmiregistry, vous pouvez utiliser les commandes suivantes : **pkill java** ; **pkill rmiregistry**.

1 Observation

Dans le répertoire *1-MessageExercice/*, vous trouverez un exemple simple d'appel java RMI. Complétez-le avant de le compiler et l'exécuter.

Dans un premier temps lancez toutes les commandes sur une même machine. Par la suite, utilisez *Turing* et une autre machine (si vous en avez une à disposition [2]) pour placer le serveur et le client sur deux machines différentes.

1. Effectuez la compilation grâce aux commandes :

```
javac *.java
rmic MessageImpl
```

2. Lancez le serveur de noms sur une machine

```
rmiregistry <No Port> & (Linux)
start rmiregistry <No Port> (Windows)
```

3. Lancez le serveur sur la même machine

```
java Serveur <même No Port> & (Linux)
start java Serveur <même No Port> (Windows)
```

4. Lancez un client

```
java Client <machine serveur>:<même No Port>
```

1. Sujet créé à partir de documents de Guillaume LATU
2. *Codd* devrait à priori être disponible.

Question : Rééditez la procédure ci-dessus tout en remplaçant le commande **rmic MessageImpl** par **rmic -keep MessageImpl** afin de générer le code source du *stub*.

En vous aidant de ce dernier, écrivez le graphe de classes pour l'application RMI générée. Reportez votre réponse dans le fichier *1-MessageExercice/reponse.txt* en donnant la liste des dépendances de type « *hérite* », « *implémente* » ou « *utilise* » entre toutes les classes de l'exercice, tel que dans l'exemple suivant :

- *java.rmi.server.RemoteObject* implémente *Remote*, *Serializable* ;
- *java.rmi.server.RemoteServer* hérite de *java.rmi.server.RemoteObject* ;
- *java.rmi.server.UnicastRemoteObject* hérite de *java.rmi.server.RemoteServer*.

2 Applet et RMI

Rendez-vous dans le répertoire *2-TelephoneExercice/*. Il s'agit d'un exemple simple d'appel java RMI depuis une applet. Compilez-le et faites le fonctionner :

1. Lancez le serveur de noms sur votre propre machine

   ```
   rmiregistry <autre No Port> & (Linux)
   start rmiregistry <autre No Port> (Windows)
   ```

2. Lancez le serveur

   ```
   java AnnuaireImpl <No Port> & (Linux)
   start java AnnuaireImpl <No Port> (Windows)
   ```

3. Lancez le client sur la même machine.

   ```
   appletviewer AppletClient.html
   ```

   Cela ne fonctionne pas ; remarquez les paramètres passés dans le fichier *AppletClient.html* et corrigez en fonction.

4. Ajoutez les fonctionnalités d'ajout et d'effacement d'une entrée dans l'annuaire téléphonique. Vous concevrez ensuite deux petits clients distincts qui ne seront pas des applets (ils ressembleront au client de l'exercice 1). Ils pourront être appelés de la manière suivante :

   ```
   java Ajoute <nom d'une personne> <No Telephone>
   java Supprime <nom d'une personne>
   ```

   Ils contacteront l'annuaire distant pour réaliser les opérations d'ajout et de suppression demandées. **Astuce :** voir la documentation Java, chercher la classe *Hashtable* puis les méthodes **put** et **remove**.

5. Ecrire un nouveau client *Lister* qui affiche à l'écran la liste des entrées de l'annuaire. **Astuce :** vous pouvez utiliser dans la fonction que vous écrirez dans *AnnuaireImpl.java*, la ligne de code : **Iterator it = numeros.keySet().iterator() ;**. À l'aide de cet itérateur, vous pouvez parcourir la liste et en afficher chacun de ses éléments.

6. Lancez le serveur de noms et le serveur d'annuaire sur une autre machine que la votre. Lancer l'applet depuis votre machine. Cela ne doit pas marcher et une exception du type « access denied » est levée. Pour résoudre le problème de droit que cela pose, utilisez la commande :

   ```
   appletviewer -J-Djava.security.policy=java.policy AppletClient.html
   ```

3 Classes générées par rmic

Rendez-vous dans le répertoire *3-MessageClone/*. Regardez puis exécutez le script *construit*.

1. Dans le répertoire *3-MessageClone/* lancez les commandes suivantes :

   ```
   pkill rmiregistry
   rmiregistry &
   ```

2. Rendez-vous dans **3-MessageClone/tServeur** puis exécutez le serveur. Cela ne fonctionne pas car *rmiregistry* n'arrive pas à faire le lien avec l'objet en question. Expliquez ce qu'il manque dans *3-MessageClone/reponse.txt* et modifiez le script *construit* de façon à pouvoir faire fonctionner la séquence d'instructions que vous venez d'exécuter. Lancez le serveur correctement avant de passer au point suivant.
3. Sur une autre machine (qui va héberger le client), déplacez-vous dans le répertoire *3-MessageClone/tClient* et exécutez le client. Cela devrait fonctionner sans problèmes, il y a même des fichiers inutiles dans ce répertoire. Déterminez la/les classe(s) superflues et modifiez le script *construit* afin d'obtenir qu'un ensemble minimal de fichiers dans le répertoire *tClient*.

4 Exercice complet

On désire implanter un serveur de calcul sur des matrices $N \times N$ à coëfficients réels qui offre les services suivants :

- la somme de deux matrices ;
- la multiplication de deux matrices.

Vous utiliserez pour ce service distant l'interface fournie dans le fichier *4-MatricesNN/OpMatrice.java*

Mettre en place un tel serveur. Il devra pouvoir être appelé par un client depuis une autre machine (vous écrirez un client qui effectuera un appel à la méthode de multiplication de matrice et qui affichera le résultat).

5 Chargement dynamique de classes

Dans l'exercice 3, nous avons vu que le serveur demandait à *rmiregistry* de charger les classes dont il avait besoin. Un autre moyen pour effectuer ce chargement à la demande est de passer par un serveur web qui propose le téléchargement des classes nécessaires. Ce mécanisme (de chargement dynamique de classes) est extrêmement puissant.

Dans cet exercice, vous allez faire fonctionner un mini-serveur web (qui se trouve dans le répertoire *5-TelephoneTelecharge/ServeurDeClasse/*). Puis vous allez demander à un tout petit code java (*Lance.java*) de démarrer telle ou telle classe accessible sur ce mini-serveur web.

Cela veut dire que sur un certain site où l'on souhaite exécuter, soit le serveur, soit le client, on n'a pas besoin d'avoir le bytecode (le fichier *.class*) de leur classe ! On lance simplement le programme *Lance* en précisant sur quelle machine aller chercher le bytecode du client ou du serveur.

Cette approche permet de faciliter grandement la maintenance d'une application. Lorsque celle-ci évolue, les utilisateurs n'ont pas à se soucier de télécharger une nouvelle version car elle est téléchargée automatiquement.

Pour cet exercice vous aurez besoin d'au moins trois interfaces de ligne de commande et vous exécuterez une application dans chacune. Rendez-vous dans le répertoire *5-TelephoneTelecharge*.

1. Compilez le répertoire contenant les sources à mettre à disposition sur le serveur ainsi que le code du serveur :

```
cd Sources
javac *.java
cd ../ServeurDeClasse
javac *.java
cd ..
javac Lance.java
```

2. Lancez le serveur de classes sur machine1 (création du mini serveur web). Par précaution, tuer au préalable tous les programmes java et *rmiregistry* qui tournent :

```
pkill java ; pkill rmir

cd ServeurDeClasse
java -Djava.security.policy=java.policy ClassFileServer <NumPort> ../Sources &
(cf. StartClassServer.sh)
```

3. Lancez le serveur de l'application RMI sur machine2 (chargement de la classe AnnuaireImpl depuis le mini serveur web et démarrage du serveur d'annuaire) :

```
rmiregistry <NumeroPort RMIregistry>
java Lance <machine1> <No port serveur de classes> AnnuaireImpl <NumeroPort RMIregistry>
(cf. LanceAnnuaireImpl.sh)
```

4. Lancez le client sur une machine3 (lancement du client avec téléchargement des classes nécessaires) :

```
java Lance <machine1> <No port serveur de classes> Client <machine2>
(cf. LanceClient.sh)
```

   Si vous n'avez que Turing et Codd à disposition, prenez par exemple :
   – machine1 : Turing
   – machine2 : Codd
   – machine3 : Codd
   Ainsi, l'application cliente (sur m3 : Codd) demandera au serveur web (m2 : Turing) de télécharger la classe *Client* (vers m3) qui se situe sur m1 (Codd) afin de l'exécuter.

Exercice : regardez le code de l'application que vous venez de faire fonctionner et expliquez clairement comment il fonctionne :

1. en fournissant un schéma (éventuellement commenté) représentant les suites d'appels entre différentes machines ;
2. en commentant de façon claire et explicite le code du fichier *Lance.java*.

Ensuite, écrivez deux nouveaux clients permettant respectivement d'ajouter ou d'enlever un nom de l'annuaire. Ajoutez également les shellscripts qui, de façon analogue au script *LanceClient.sh*, exécutent ces clients.