RMI : Remote Method Invocation

Auteur: Zouheir HAMROUNI

2 RMI : Remote Method Invocation

La technologie RMI, mise en place par Sun MicroSystems, a pour but de permettre l'appel, l'exécution et le renvoi du résultat d'une méthode exécutée sur une machine différente (serveur) de celle de l'appelant (client).

Sans revenir sur les concepts de cette technologie, ce chapitre illustre les différentes étapes nécessaires pour le développement d'applications RMI.

Côté serveur, les principales étapes consistent à :

- définir une interface qui présente les méthodes qui peuvent être appelées à distance
- écrire une classe qui :
  - implémente les méthodes de cette interface
  - en crée une instance et l'enregistre en lui affectant un nom dans le registre de noms RMI (RMI Registry)

Côté client, les étapes d'appel consistent à :

- obtenir une référence sur l'objet distant à partir de son nom
- appeler la méthode souhaitée par le biais de cette référence

Pour illustrer cette technologie, nous allons développer progressivement une petite application qui permet d'exécuter une commande shell sur une machine distante et d'afficher le résultat sur la machine locale (sur le modèle de la commande rsh).

2.1 RMI Côté serveur

Côté serveur, l'objet distant est décrit par une interface, et est implanté dans une classe qui doit l'instancier et l'enregistrer dans le registre RMI.

L'interface

L'interface doit hériter de la classe `java.rmi.Remote`. Cette dernière ne contient aucune méthode mais confère à l'interface qui en hérite la capacité d'être appelée à distance.

L'interface doit présenter toutes les méthodes susceptibles d'être appelées (invoquées) à distance. Ces méthodes doivent déclarer qu'elle sont en mesure de lever l'exception java.rmi.RemoteException, pour couvrir un possible échec de la communication entre le client et le serveur.

L'interface de notre classe distante portera le nom RemoteShellItf (Itf pour interface) et sa méthode sera nommée "execCmd" et prendra comme paramètre la commande à exécuter "cmd".

```
import java.rmi.Remote;
public interface RemoteShellItf extends Remote {
    public void execCmd(String cmd) throws RemoteException;
}
```

L'implémentation

Cette classe implémente l'interface définie dessus, et doit doit hériter de la classe `UnicastRemoteObject` qui contient les différents traitements nécessaires à l'appel d'un objet distant.

La mise en place de l'objet distant passe par différentes actions qui peuvent être réalisées dans une classe dédiée ou dans la méthode main de la classe d'implémentation.

- l'instanciation d'un objet de la classe distante.
- le lancement du registre de noms RMI via la méthode createRgistry() de la classe `java.rmi.registry.LocateRegistry`. Cette méthode attend en paramètre un numéro de port (4000 dans notre exemple).
- l'enregistrement de la classe dans le registre de noms RMI, en lui affectant un nom URL qui permettra de la localiser par le client. Ce nom peut être fourni sous forme d'une chaîne composée par le nom du serveur, le numéro du port, et le nom de la classe distante, par exemple "//localhost:4000/RemoteShell" dans notre exemple.

  L'enregistrement se fait en utilisant la méthode rebind de la classe `Naming`. Elle attend en paramètres le nom URL de la classe et celui de l'objet instancié.

A ce stade, la méthode "execCmd" de notre classe distante se contente d'afficher un message dans son propre terminal. Elle sera complétée progressivement.

```
 1 import java.rmi.* ;
 2 import java.rmi.server.UnicastRemoteObject ;
 3 import java.rmi.registry.* ;
 4 import java.util.* ;
 5
 6 public class RemoteShell extends UnicastRemoteObject implements RemoteShellIt
 7
 8   public RemoteShell() throws RemoteException {
 9   }
10
11   public void execCmd(String cmd) {
12       try {
13          System.out.println("Execute: "+cmd) ;
14          // A compléter
15       } catch (Exception e) {
16          e.printStackTrace() ;
17       }
18   }
19
20   public static void main(String[] args) {
21       try {
22          RemoteShell rsh = new RemoteShell() ;
23          Registry registry = LocateRegistry.createRegistry(4000) ;
24          Naming.rebind("//localhost:4000/RemoteShell", rsh) ;
25          System.out.println("RemoteShell bound in registry") ;
26       } catch (Exception e) {
27          e.printStackTrace() ;
28       }
29   }
30 }
```

2.2 RMI : Côté client

Obtention d'une référence

La méthode lookup() de la classe `Naming` permet d'obtenir une référence sur l'objet distant. Elle prend en paramètre le nom URL de l'objet distant tel qu'il a été défini plus haut.

La méthode lookup() va rechercher l'objet dans le registre du serveur et retourner un objet stub de la classe Remote, classe mère de tous les objets distants. Il faut donc en faire un cast vers l'interface de l'objet distant. Une fois la référence récupérée, on peut appeler la méthode distante.

Si le nom fourni n'est pas référencé dans le registre, la méthode lève l'exception NotBoundException.

```
 1 import java.rmi.* ;
 2 import java.rmi.registry.* ;
 3 import java.util.* ;
 4
 5 public class Rsh {
 6
 7   public static void main(String[] args) {
 8      try {
 9         if (args.length < 3) {
10            System.out.println("Usage : Rsh hostname port command") ;
11         }
12         else {
13            RemoteShellItf rsh = (RemoteShellItf)Naming.lookup("//" + args[0]
14            rsh.execCmd(args[2]) ;
15         }
16      } catch (Exception e) {
17         e.printStackTrace() ;
18      }
19   }
20 }
21 }
```

2.3 Méthode client appelée par le serveur

A ce stade, notre méthode distante ne fait qu'afficher un message sur son propre terminal. L'étape suivante consistera à afficher ce message sur le terminal de l'appelant (le client). Pour cela, on doit :

- fournir une interface de ce terminal à l'objet distant dans le fichier ConsoleItf.java.

  ```
  1 import java.rmi.* ;
  2
  3 public interface ConsoleItf extends Remote {
  4    public void println(String s) throws RemoteException ;
  5 }
  ```

- implémenter la classe avec la méthode qui affiche un message sur ce terminal

  ```
   1 import java.rmi.* ;
   2 import java.rmi.server.UnicastRemoteObject ;
   3 import java.rmi.registry.* ;
   4 import java.util.* ;
   5
   6 public class Console extends UnicastRemoteObject implements ConsoleItf {
   7
   8   public Console() throws RemoteException {
   9   }
  10
  11   public void println(String s) throws RemoteException {
  12      System.out.println(s) ;
  13   }
  14 }
  ```

- ajouter l'interface de ce terminal comme second paramètre de la méthode distante execCmd (execCmd (String cmd, ConsoleItf cons)) dans les fichiers RemoteShellItf.java et RemoteShell.java

- et remplacer dans la méthode execCmd de la classe RemoteShell l'appel System.out.println(...) par cons.println(...)

Le message affiché par la méthode execCmd apparaît désormais dans le terminal du client (terminal de lancement de l'appel distant Rsh).

2.4 Rediriger les sorties

La dernière étape consistera à exécuter la commande passée en paramètre de l'appel distant sur le serveur et à afficher son résultat sur le terminal client. Pour cela, il faut :

- lancer l'exécution de la commande passée en paramètre avec :

  ```
  Process p = Runtime.getRuntime().exec(cmd);
  // ou
  ProcessBuilder processBuilder = new ProcessBuilder(cmd);
  Process p = processBuilder.start();
  ```

- rediriger les sorties (standard et erreur) du processus vers un BufferedReader avec :

  ```
  BufferedReader output = new BufferedReader(new InputStreamReader(p.getInputS
  BufferedReader error = new BufferedReader(new InputStreamReader(p.getErrorSt
  ```

- et afficher le contenu de ce buffer dans le terminal du client :

  ```
  String ligne = "";
  while ((ligne = output.readLine()) != null) {
          cons.println(ligne);
  }
  ```

Ce qui donne le code complet de la classe distante :

```
 1 import java.rmi.* ;
 2 import java.rmi.server.UnicastRemoteObject ;
 3 import java.rmi.registry.* ;
 4 import java.util.* ;
 5 import java.io.BufferedReader ;
 6 import java.io.IOException ;
 7 import java.io.InputStreamReader ;
 8
 9 public class RemoteShell extends UnicastRemoteObject implements RemoteShellIt
10
11   public RemoteShell() throws RemoteException {
12   }
13
14   public void execCmd(String cmd, ConsoleItf cons) {
15       try {
16          cons.println("Execute: "+cmd) ;
17          Process p = Runtime.getRuntime().exec(cmd) ;
18          BufferedReader output =
19              new BufferedReader(new InputStreamReader(p.getInputStream())) ;
20          BufferedReader error =
21              new BufferedReader(new InputStreamReader(p.getErrorStream())) ;
22          String ligne = "" ;
23          while ((ligne = output.readLine()) != null) {
24             cons.println(ligne) ;
25          }
26          while ((ligne = error.readLine()) != null) {
27             cons.println(ligne) ;
28          }
29          int exitValue = p.waitFor() ;
30       } catch (Exception e) {
31          e.printStackTrace() ;
32       }
33   }
34
35   public static void main(String[] args) {
36       try {
37          RemoteShell rsh = new RemoteShell() ;
38          Registry registry = LocateRegistry.createRegistry(4000) ;
39          Naming.rebind("//localhost:4000/RemoteShell", rsh) ;
40          System.out.println("RemoteShell bound in registry") ;
41       } catch (Exception e) {
42          e.printStackTrace() ;
43       }
44   }
45 }
```

2.5 Passage en paramètre d'un objet sérialisable

Dans l'exemple dessus, la commande à exécuter est passée sous la forme d'un String. Mais dans certaines situations, le paramètre passé à la méthode distante peut être constitué de plusieurs attributs, et on peut être amené à le passer sous la forme d'un objet sérialisable.

Dans cet exemple, on peut imaginer la commande composée d'un tableau d'arguments, et d'un tableau de variables d'environnement. Ces informations peuvent être encapsulées dans un objet sérialisable qui sera passé en paramètre à la méthode execCmd.

Pour cela, le client doit fournir une interface pour la commande et les méthodes qui permettent au serveur d'en extraire toutes les informations.

```
import java.io.Serializable;
public interface CommandeItf extends Serializable {
        public String[] getCmdarray() ;
        public String[] getEnvp() ;
}
```

```
import java.io.Serializable;
public class Commande implements CommandeItf {

        String[] cmdarray;
        String[] envp;

        public Commande (String[] cmda, String[] env, File d) { … }

        public String[] getCmdarray() { … }
        public String[] getEnvp() { … }
}
```

```
public class RemoteShell ...
   ...
   public void execCmd (Commande cmd, ConsoleItf cons) throws
            RemoteException {
            ...
            p = Runtime.getRuntime().exec (cmd.getCmdarray(), cmd.getEnvp()) ;
            ...
   }
   ...
}
```