TP 4 Introduction à Java Remote Method Invocation

L'objectif du TP est de manipuler le mécanisme d'invocation de méthodes à distance du langage `Java`, c'est-à-dire Java Remote Method Invocation ou Java RMI.

1. Java RMI

Le mécanisme Java Remote Method Invocation (ou Java RMI) permet de rendre accessible à travers le réseau des objets `Java` depuis d'autres programmes écrits en `Java`. La force de cet environnement est de rendre transparente l'utilisation du réseau. Le programmeur conçoit simplement des objets Java et y accède par invocations de méthodes. Il n'a plus à se soucier de la gestion des couches de transport tel que `TCP/IP`, de l'emballage / déballage des messages et des threads. C'est le mécanisme Java RMI qui réalise tout le travail de transport des invocations distantes à travers le réseau.

1.1 Un simple exemple

La suite de ce document illustre sur un simple exemple vu en cours le processus de développement et d'exécution d'une application Java RMI. Le code de cet exemple est accessible sur moodle.

Cette application est composée de :

- la classe `Message` implantant un objet sérialisable par Java RMI,
- l'interface `Hello` listant les m´ethodes invocables à distance,
- la classe `HelloImpl` implantant l'objet Java RMI,
- la classe `Serveur` fournissant un simple serveur pour l'objet Java RMI,
- la classe `Client` fournissant un simple client pour l'objet Java RMI,
- la classe `ClientIHM` implantant un client graphique dialoguant avec des objets `Hello` distants.

1.2 La classe `Message`

Pour qu'un objet Java soit transporté à travers des invocations de méthodes Java RMI, il suffit seulement que sa classe implante directement ou transitivement l'interface `java.io.Serializable`.

```
public class Message implements java.io.Serializable
{
  //L'état interne du message.
  protected java.util.Date date;
  protected String texte;
  // Le constructeur avec un texte.
  public Message(String texte)
  {
    this.date = new java.util.Date();
    this.texte = texte;
  }
```

```
  // Pour afficher le message.
  public String toString()
  {
    return "Message[date=" + date + ",texte=" + texte + "]";
  }
}
```

1.3 L'interface `Hello`

Pour qu'un objet Java soit invocable à distance via Java RMI, c'est-à-dire depuis une machine virtuelle Java différente de celle où s'exécute l'objet, il suffit qu'il implante une interface Java RMI décrivant la liste de ses méthodes invocables à distance. Cette interface doit implanter au moins l'interface `java.rmi.Remote`. Chaque méthode de l'objet Java RMI doit obligatoirement retourner au moins l'exception `java.rmi.RemoteException`.

```
public interface Hello extends java.rmi.Remote
{
  // Afficher une chaine de caracteres.
  public void afficher(String chaine) throws java.rmi.RemoteException;
  // Obtenir le dernier message affiche.
  public Message getDernierMessage() throws java.rmi.RemoteException;
}
```

1.4 La classe `HelloImpl`

La classe d'implantation d'un objet Java RMI (`HelloImpl`) doit implanter l'interface Java RMI décrivant ses méthodes accessibles à distance (`Hello`) et hériter de la classe java.rmi.server.UnicastRemoteObject. Le constructeur de cette classe doit invoquer le constructeur de la classe Java RMI. Ensuite chacune des méthodes de l'interface doit être implantée.

```
public class HelloImpl extends java.rmi.server.UnicastRemoteObject
implements Hello
{
  // Attribut stockant le dernier message affiche.
  protected Message leDernierMessage;
  // Le constructeur.
  public HelloImpl() throws java.rmi.RemoteException {
    // Appel du constructeur java.rmi.server.UnicastRemoteObject.
    // Peut soulever l'exception java.rmi.RemoteException.
    super();
    leDernierMessage = new Message("");
  }
  // Afficher une chaine de caracteres.
  public void afficher(String chaine) throws java.rmi.RemoteException {
    System.out.println("Invocation de HelloImpl.afficher(chaine=" + chaine + ")");
    leDernierMessage = new Message(chaine);
  }
  // Obtenir le dernier message affiche.
  public Message getDernierMessage() throws java.rmi.RemoteException {
    System.out.println("Invocation de HelloImpl.getDernierMessage()");
    return leDernierMessage;
  }
}
```

1.5 Le programme principal d'un serveur Java RMI

Un objet Java RMI doit s'exécuter dans un processus serveur Java RMI. Le serveur doit créer ses objets Java RMI et publier leurs références dans l'annuaire Java RMI. Cet enregistrement permet aux processus clients de retrouver la référence des objets Java RMI contenus dans le serveur.

```
public class Serveur {
public static void main(String args[]) throws Exception {
  // Creer l'objet accessible par Java RMI.
  HelloImpl obj = new HelloImpl();
  // Enregistrer cet objet dans l'annuaire Java RMI.
  java.rmi.Naming.rebind("UnHello", obj);
  System.out.println("Le serveur Java RMI est pret ...");
}
}
```

1.6 Le programme principal d'un client Java RMI

Un client Java RMI doit retrouver les objets Java RMI qu'il désire invoquer. Pour cela, il s'adresse à l'annuaire Java RMI (classe java.rmi.Naming) afin d'obtenir les souches de ces objets (méthode lookup). La chaîne utilisée par le client pour retrouver l'objet Java RMI est une URL composée de deux parties : le nom de la machine internet où se trouve l'annuaire du serveur et le nom utilisé par le serveur pour enregistrer cet objet. Ensuite le client peut invoquer de manière transparente les méthodes de l'objet distant.

```
public class Client {
  public static void main (String[] args) throws Exception {
  // Obtenir la souche sur l'objet distant via l'annuaire Java RMI.
  Hello helloDistant = (Hello) java.rmi.Naming.lookup("//localhost/UnHello");
  // Invoquer des methodes distantes comme si l'objet etait local.
  helloDistant.afficher("Hello world !");
  Message message = helloDistant.getDernierMessage();
  System.out.println("Le dernier message affiche est " + message);
  }
}
```

Vous trouverez dans l'archive précédemment citée un programme client fournissant une interface graphique (`ClientIHM`).

1.7 La compilation

Pour compiler cette application, il suffit de compiler la classe Message, l'interface `Hello`, l'implantation `HelloImpl` et les programmes `Serveur`, `Client` et `ClientIHM`.

```
unix>javac Message.java Hello.java HelloImpl.java Serveur.java Client.java ClientIHM.java
```

Ensuite, il était nécessaire de passer le bytecode de la classe d'implantation de l'objet Java RMI (`HelloImpl.class`) dans le compilateur Java RMI afin de produire la souche cliente (`HelloImplStub.class`) et la souche serveur (`HelloImplSkel.java`).

```
unix> rmic HelloImpl
```

1.8 L'exécution

Pour exécuter cette application, utiliser si possible deux machines ou au moins deux fenêtres shells Unix différentes. Sur la première, lancer l'annuaire Java RMI et le serveur :

```
unix> rmiregistry &
unix> java Serveur MonHelloDistant
```

Sur la seconde, lancer un client Java RMI et indiquer lui l'URL de l'objet Java RMI :

```
unix> java Client //machineServeur/MonHelloDistant
unix> java ClientIHM //machineServeur/MonHelloDistant
```

Attention, `machineServeur` doit être remplacé par le nom de la machine où s'exécute le serveur Java RMI. Le client graphique propose deux champs de saisie à valider : le premier permet de fixer l'URL de l'objet Java RMI invoqué tandis que le second permet de saisir le texte à envoyer à l'objet Java RMI.

2. Exercices

Après avoir étudié, compilé et exécuté l'exemple précédent, vous devez maintenant reprendre les applications précédemment réalisées avec des sockets TCP/IP afin de les transformer en applications Java RMI. Attention, avant de lancer les nouveaux serveurs Java RMI à développer, vous devez arrêter le processus `rmiregistry` puis le relancer dans le répertoire contenant les classes associées au nouveau serveur à tester. Sinon, vous aurez des problèmes d'exécution car l'annuaire Java RMI ne trouvera pas les classes Stub associées aux nouveaux serveurs.

2.1 Le panneau d'affichage électronique

Vous devez :

- Définir l'interface Java RMI `Panneau` proposant les méthodes afficher et effacer.
- Implanter la classe Java RMI `PanneauImpl` implantant l'interface `Panneau` en utilisant la classe `IHM`.
- Réaliser la classe `Serveur` instanciant un `PanneauImpl` et publiant sa référence dans l'annuaire Java RMI.
- Réaliser un client graphique permettant de piloter à distance des panneaux d'affichage. Pour cela, inspirez vous de la classe `ClientIHM` précédente.

Comparer l'implantation de l'application réalisée avec les sockets Java à celle réalisée avec Java RMI. Quelles leçons en tirez-vous ?

2.2 Le répertoire d'adresses Internet

Vous devez :

- Modifier l'interface `Repertoire` afin de la rendre accessible à distance via Java RMI.
- Modifier la classe `Carnet` afin d'implanter l'interface Java RMI.
- Réaliser la classe `Serveur` instanciant un Carnet et publiant sa référence dans l'annuaire Java RMI.
- Modifier le client graphique permettant d'accéder à distance à des répertoires d'adresses Internet.

Comparer l'implantation de l'application réalisée avec les sockets Java à celle réalisée avec Java RMI. Quelles leçons en tirez-vous ?