TP noté : Java RMI

PAR : Programmation d'Applications Réparties
Troisième année

Sujet TRS (18 Janvier 2006)

Tous documents autorisés. La qualité de rédaction de votre code, des commentaires associés et du compte-rendu est aussi importante que l'exactitude du résultat calculé.

Comment rendre votre copie

Vous devez rendre un compte-rendu avec vos sources. Un simple fichier texte décrivant votre travail et argumentant vos choix suffit, mais vous pouvez faire un .doc si vous préférez.

Quand vous avez fini votre TP, faites une archive de votre répertoire (zip, rar, jar, etc.) et déposez-la sur Moodle après vous être authentifié.

Pour cela, allez sur le moodle de l'université dans le cours intitulé «Informatique/Applications distribuées (ESIAL)» (`http://www.moodle.uhp-nancy.fr/course/view.php?id=22`). Dans la rubrique 7 (TP PAR), cliquez sur TP RMI puis déposez votre fichier.

1 Serveur de calcul simple

L'objectif de ce TP est d'écrire un serveur permettant de réaliser des calculs à distance (pattern de commande). Ce programme exécute les demandes de calcul envoyées par les utilisateurs puis renvoie le résultat après coup. L'un des intérêts de la chose est bien entendu de déporter les calculs trop gros pour être exécutés en local sur une machine plus puissante.

À des fins de simplifications, notre serveur ne pourra calculer que des fonctions mathématiques simples de la forme $U_n = f(U_{n-1})$. Un exemple de telle fonction est la fonction d'Ackerman :

$$A(m,n) = \begin{cases} n+1 & \text{si } m = 0 \\ A(m-1,1) & \text{si } m > 0 \text{ et } n = 0 \\ A(m-1, A(m,n-1)) & \text{sinon} \end{cases}$$

L'objet `serveur` a donc l'interface suivante :

Serveur.java

```
package tpnote;
import java.rmi.*;
public interface Serveur extends Remote  {
     public int ackerman(int m, int n) throws RemoteException;
}
```

▷ **Question 1.** Écrivez les fichiers suivants :
- **ServeurImpl.java :** l'implémentation du serveur de calcul
- **Client.java :** un client demandant le calcul d'$Ack(3,3)$

▷ **Question 2.** Compilez et testez votre programme. Précisez dans votre compte-rendu quelles sont les commandes à taper pour cela, ainsi que si elles doivent avoir lieu coté serveur ou coté client.

2 Serveur générique

Le principal défaut du serveur développé dans l'exercice précédent est qu'il est nécessaire de modifier le serveur chaque fois que l'on souhaite ajouter une fonction. Nous allons maintenant utiliser le polymorphisme de Java pour contourner cette limitation.

Pour cela, nous ajoutons un objet CalculGenerique doté de l'interface suivante :

CalculGenerique.java

```
package tpnote;
public abstract class CalculGenerique {
   public abstract int calcule();
}
```

L'interface du serveur générique est donc :

ServeurGenerique.java
```
package tpnote;
import java.rmi.*;
public interface ServeurGenerique extends Remote  {
     public int execute(CalculGenerique calcul) throws RemoteException;
}
```

▷ **Question 3.** Implémentez les fichiers suivants :
- **ServeurGeneriqueImpl.java :** l'implémentation du serveur
- **CalculAckerman.java :** une classe héritant de `CalculGenerique.java` et dont la méthode `calcule` réalise un calcul de la fonction d'Ackerman.
- **CalculFactorielle.java :** une classe similaire à la précédente calculant la factorielle d'un nombre.
- **ClientGenerique.java :** un client demandant successivement un calcul de $Ack(3,3)$ puis de 5! avec cette nouvelle interface.

▷ **Question 4.** Compilez et testez votre programme. Précisez dans votre compte-rendu quelles sont les commandes à taper pour cela, respectivement coté serveur et coté client.

3 Serveur asynchrone

Un autre problème des serveurs réalisées jusque là est que le client est bloqué pendant le calcul du résultat par le serveur. Nous allons mettre en place un mécanisme d'appel en retour (callback) utilisant des threads pour corriger ce défaut. Voici l'interface du callback utilisé. Il est créé et s'exécute du coté client, mais son exécution est déclenchée coté serveur.

CalculCallbackInterf.java
```
package tpnote;
import java.rmi.*;
public interface CalculCallbackInterf extends Remote {
  void resultat(int res) throws RemoteException; // affiche le résultat coté client
}
```

▷ **Question 5.** Écrivez une classe CalculAsynchrone, qui sera exécuté par le serveur. Elle remplace CalculGenerique et les classes implémentant les calculs devront être modifiées pour hériter de cette nouvelle classe. Elle implémente les interfaces Remote et Runnable. Cette dernière va nous permettre d'exécuter le calcul dans un thread séparé. Pour cela, dotez-la de la méthode `run` suivante. `argument` et `callback` sont des attributs privés de l'objet initialisés à sa création.

CalculAsynchrone.run()
```
public void run() {
  int res = this.calcule(argument);
  callback.resultat(res);
}
```

Dans le serveur, le code lançant le calcul devient :

morceau du serveur lançant le calcul dans execute()
```
  Thread monThread = new Thread(calcul);
  monThread.start(); // cela lance run() dans un nouveau thread
```

▷ **Question 6.** Écrivez les fichiers **ServeurAsynchrone.java**, **CalculAsynchrone.java** et **ClientAsynchrone.java**. Compilez et testez votre programme.

4 Fabriques de serveurs et facturation

Nous allons maintenant utiliser une fabrique d'objets afin de mettre en place un mécanisme de facturation embryonnaire. Nous voulons que le serveur garde une trace du nombre d'invocation du service réalisé par chaque client. Voici l'interface de la fabrique à utiliser et la nouvelle interface du serveur.

ServeurFab.java
```
package tpnote;
import java.rmi.*;
public interface ServeurFab extends Remote {
  ServeurFactureInterf cree() throws RemoteException;
}
```

ServeurFactureInterf.java

```
package tpnote;
import java.rmi.*;
public interface ServeurFactureInterf extends Remote  {
   public int execute(CalculGenerique calcul) throws RemoteException;
   public int getNbInvocation(); //retourne le nombre d'appels à execute() passés
}
```

▷ **Question 7.** Écrivez les fichiers nécessaires à la mise en œuvre de cette solution. Compilez et testez votre travail.