SI 4 Polytech'Nice - UNS

UE « Applications Réparties », Partie « Objets » - Epreuve écrite individuelle

25 Mai 2011, durée 1h

Aucun document autorisé

F. Baude

Exercice 1 (barème approximatif : 6 points)

Pour chaque point numéroté ci-dessous, expliquez en quoi l'affirmation est fausse ou vraie. Si l'affirmation est fausse, expliquez en 2 lignes maximum pourquoi, et comment la rendre vraie.

Exemple :
Avec RMI, il est possible d'enregistrer un objet accessible à distance sur un rmiregistry lancé sur une autre machine
Votre réponse : Faux. Il faut que le rmiregistry qui stocke les stub vers les objets distants s'exécute sur la même machine que ces objets.

1. Avec RMI il est possible de créer (instancier) un objet Remote, depuis une JVM distante
2. Avec RMI, on peut récupérer une référence vers un objet Remote sans passer par le rmiregistry
3. RMI permet de contrôler l'ordre d'exécution des requêtes (appels de méthode) sur un objet Remote, jusqu'au point de pouvoir dire « tel appel de cette méthode doit être traité avant tel autre appel de telle autre méthode »
4. Tous les types de Java peuvent être passés en paramètre des appels distants, sauf les types dits « final », et tout est automatique sans même que le programmeur n'ait besoin de spécifier si tel ou tel paramètre est de type sérialisable ou de toute autre nature.
5. Toutes les méthodes d'un objet accessible à distance doivent forcément être rendues accessibles à distance
6. Il est impossible de faire interagir une application décrite en JavaRMI avec une application décrite en CORBA

Exercice 2 (barème approximatif : 6 points)

Pour chaque point numéroté ci-dessous, expliquez en quoi l'affirmation est fausse ou vraie. Si l'affirmation est fausse, expliquez en 2/3 lignes maximum pourquoi, et comment la rendre vraie.

1. Avec JAAS utilisé dans une application Java RMI, on peut sécuriser les sockets utilisés par les objets distants afin qu'ils se basent sur le protocole SSL
2. JAAS se configure via l'adjonction d'un module dit de « Login ». De plus, on peut envisager de modifier dynamiquement la configuration obtenue.
3. Quand l'application Java utilise JAAS pour authentifier un utilisateur, c'est à elle d'invoquer explicitement la méthode login, ainsi que la méthode commit.
4. Pour faire fonctionner une application avec JAAS, il n'est pas nécessaire d'avoir de permissions Java particulières
5. Avec CORBA dans une application simple où l'on a un client et un objet serveur écrit en langages différents (ex en Java d'un côté, en C++ de l'autre), le programmeur n'a pas besoin de connaitre le fonctionnement des deux systèmes de typage de ces deux langages car c'est le langage IDL qui résout l'interopérabilité entre les langages utilisés et donc leur système de typage respectif.
6. En CORBA, il y a beaucoup plus de services standards définis que dans Java RMI, et on ne manipule pas ces services comme des objets CORBA mais avec une API particulière à chaque service.

Exercice 3 (barème approximatif : 4 points)

A propos du téléchargement de code

Un serveur (objet Remote) RMI B tourne sur une machine MB. Il référence un serveur RMI C qui tourne sur la machine MC ; la référence est obtenue par un lookup dans un RMI registry de nom registry1 .
Un client A, s'exécutant sur la machine MA se connecte (utilise) au serveur B. Pour ce faire, il utilise un RMI registry de nom registry2. Il n'y a pas de système de fichier partagé, et l'on souhaite exploiter au maximum le téléchargement dynamique de classes. Pour cette raison, on peut utiliser, sur chaque machine qui le nécessite, un programme permettant de fournir des classes en utilisant le protocole http. Notez que l'on ne préconise pas d'utiliser un unique serveur de classes http, car, on se met dans l'hypothèse où chaque objet (serveur ou client) a été développé indépendamment, par différentes équipes de programmeurs.

Q1/ Le client A appelle la méthode **I1 foo (P1) ;** sur le serveur **B**, mais en lui passant en paramètre effectif un objet de type **P1a**, classe qui hérite de **P1**.
Que se passe-t-il (i.e. qu'est-ce qui est téléchargé, et de quelle machine vers quelle machine) au niveau des téléchargements ?

Q2 / Puis, lors de l'exécution de la méthode **foo**, le serveur **B** appelle la méthode **I1 bar (P1) ;** sur le serveur **C** de la machine MC, en lui passant en paramètre effectif l'objet de type **P1a** reçu précédemment.
Que se passe-t-il au niveau des téléchargements ?

Q3/ A la fin de l'exécution de **bar**, on retourne un objet de type **OI2,** qui implémente l'interface **I2**, qui elle-même dérive de l'interface **I1**.
Que se passe-t-il au niveau des téléchargements ?

Q4/ La méthode **foo** se termine enfin, et retourne ce même objet de type **OI2** reçu précédemment, au client **A**.
Que se passe-t-il au niveau des téléchargements ?

Q5/ Faites un dessin de l'ensemble du système, en indiquant les différentes machines, les différentes JVMs, y compris les registryRMI, les différents serveurs de classe, les objets Remote, etc.

Q6/ Donner machine par machine les commandes qui permettent de lancer chacune des JVMs, les rmiregistry, et les serveurs de classe afin que tout fonctionne, cad, permette les téléchargements identifiés plus haut.

Exercice 4 (barème approximatif : 4 points)

– A propos de l'architecture d'une application RMI

Soit le programme suivant, et en supposant que le main du BrokerImpl est lancé sur une machine de nom « SERVEUR». Donner l'effet de tous les *println*, observés sur la console côté serveur et sur celle côté client et l'ordre exact dans lequel ils se produisent.

```
public interface I {
  public String sayHello() throws java.rmi.RemoteException;
}
public interface J extends I, java.rmi.Remote{ }
```

```
public class A implements I, java.io.Serializable{

  J j;
  String msg;

  public A(J j) throws RemoteException{
    this.j=j;
    msg=j.sayHello();
        System.out.println("TRACE: A() created");
  }

  public String sayHello() throws RemoteException {
    System.out.println("TRACE: A() sayHello:"+msg);
    return msg;
  }
}
public class B extends java.rmi.server.UnicastRemoteObject implements J {

  protected B() throws java.rmi.RemoteException {
    super();
    System.out.println("TRACE: B() created");
  }

  public String sayHello() {
    String msg="unkown";
    try {
      return java.net.InetAddress.getLocalHost().getHostName().toString();
    } catch (Exception e) {}

    System.out.println("TRACE: B() sayHello:"+msg);
    return msg;
  }
}
```

```
public interface Broker extends java.rmi.Remote, java.io.Serializable{
  public J getJ() throws RemoteException;
  public I getI() throws RemoteException;
}
public class BrokerImpl extends java.rmi.server.UnicastRemoteObject implements Broker{

  J j; I i;

  protected BrokerImpl() throws RemoteException {
    super();
    j= new B();
    i= new A(j);
  }

  public I getI() { return i; }

  public J getJ() { return j; }

  public static void main(String args[]) throws Exception{
        java.rmi.Naming.bind("broker", new BrokerImpl());
        System.out.println("TRACE: Broker binded");
  }
}
public class Client {

  public static void main(String args[]) throws Exception{
    Broker broker = (Broker) java.rmi.Naming.lookup("rmi://remotehost/broker");

    I   i = broker.getI();
    J   j = broker.getJ();

    System.out.println("TRACE: I() sayHello:"+i.sayHello());
    System.out.println("TRACE: J() sayHello:"+j.sayHello());
  }
}
```