Java Remote Method Invocation (RMI)

1

Plan

1. Caractéristiques
2. Principes des RMI
3. Structure des couches RMI
4. Modèle de programmation
5. Services RMI

2

1.Caractéristiques

Solution Sun pour l'invocation à distance de méthodes Java
- inclus par défaut dans le JDK depuis 1.1
- nouveau modèle de souches dans JDK 1.2
- génération dynamique des souches dans JDK 1.5

- implantations alternatives (*open-source*)
  - NinjaRMI (Berkeley)
  - Jeremie (ObjectWeb)

- package java.rmi
- + outils
  - générateur de souches
  - serveur de noms
  - démon d'activation

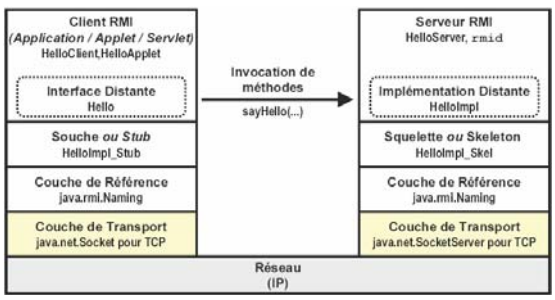
3

2.Principes des RMI

- RPC (Remote Procedure Call) à la Java
  - invoquer de façon simple des méthodes sur des objets distribués.
- Outils
  - pour la génération des stub/skeleton, l'enregistrement par le nom, l'activation
  - Tous les détails (connexion, transfert de données ..)
  sont transparents pour le développeur grâce au stub/skeleton généré
- Mono-langage et Multiplateforme.
  - Java : de JVM à JVM
  (les données et objets ont la même représentation qq soit la JVM)
- Orienté Objet
  - Les RMIs utilisent le mécanisme standard de sérialisation de JAVA pour l'envoi d'objets.
- Dynamique
  - Les classes des Stubs et des paramètres peuvent être chargées dynamiquement via HTTP (http://) ou NFS (file:/)
- Sécurité
  - un SecurityManager vérifie si certaines opérations sont autorisés par le serveur

4

3.Structure des couches RMI



5

3.Structure des couches RMI

- Souche ou Stub (sur le client)
  - représentant local de l' objet distant qui implémente les méthodes "exportées" de l'objet distant
  - "encode" les arguments de la méthode distante et les envoie en un flot de données au serveur
  - "désencode" la valeur ou l'objet retournés par la méthode distante
  - la classe xx_Stub peut être chargée dynamiquement par le client (Applet)

- Squelette ou Skeleton (sur le serveur)
  - "désencode" les paramètres des méthodes
  - fait un appel à la méthode de l'objet local au serveur
  - "encode" la valeur ou l'objet renvoyé par la méthode

6

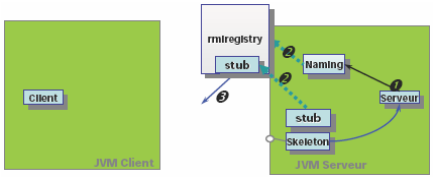
3.Structure des couches RMI

- Couche des références distantes
  - traduit la référence locale au stub en une référence à l'objet distant
  - elle est servie par un processus tiers : `rmiregistry`
- Couche de transport
  - écoute les appels entrants
  - établit et gère les connexions avec les sites distants
  - java.rmi.UnicastRemoteObject utilise les classes Socket et SocketServer (TCP)
  - cependant d ' autres classes peuvent être utilisées par la couche transport (Compression sur TCP, SSL sur TCP, UDP)

7

Mode opératoire

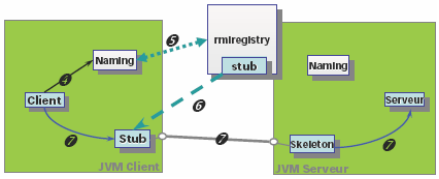
0 - A la création de l'objet-serveur, un stub et un skeleton (avec un port de communication) sont créés sur le host-serveur
1 - L'objet-serveur s'enregistre auprès du Naming de sa JVM (méthode**rebind**)
2 - Le Naming enregistre le stub de l'objet (sérialisé) auprès du serveur de noms (rmiregistry)
3 - Le serveur de noms est prêt à fournir des références sur l'objet-serveur



8

Mode opératoire

4 - L'objet client fait appel à son Naming pour localiser l'objet-serveur sur l'host-serveur (méthode **lookup**)
5 - Le Naming récupère le stub vers l'objet-serveur auprès de la rmiregistry
6 – Le naming installe l'objet Stub sur le poste client et retourne sa référence au client
7 - Le client effectue l'appel à l'objet serveur par appel à l'objet local Stub



9

4. Modèle de programmation

- 5 Packages
- java.rmi : pour accéder à des objets distants
- java.rmi.server : pour créer des objets distants
- java.rmi.registry : lié à la localisation et au nommage d' objets distants
- java.rmi.dgc : ramasse-miettes pour les objets distants
- java.rmi.activation support pour l 'activation d 'objets distants.
- Etapes du développement

1- Spécifier et écrire l'interface de l'objet distant.
2- Ecrire l'implémentation de cette interface.
3- Générer les Stub/Skeleton correspondants. (outil rmic)

- Etapes de l 'exécution

4- Ecrire le serveur qui instancie l'objet implémentant l'interface, exporte son Stub puis attend les requêtes via le Skeleton.
5- Ecrire le client qui réclame l ' objet distant, importe le Stub et invoque une méthode de l'objet distant via le Stub.

10

4. Modèle de programmation

Principes

- Chaque classe d'objets serveur doit être associée à une interface
  - seules les méthodes de l'interface pourront être invoquées à distance

1. Ecriture d'une interface
2. Ecriture d'une classe implantant l'interface
3. Ecriture du programme serveur
4. Ecriture du programme client

≡

1. déclaration des services accessibles à distance
2. définition du code des services
3. instanciation et enregistrement de l'objet serveur
4. interactions avec le serveur

11

4. Modèle de programmation

Ecriture d'une interface

- interface Java normale
- doit étendre java.rmi.Remote
- toutes les méthodes doivent lever java.rmi.RemoteException

```
import java.rmi.Remote;
import java.rmi.RemoteException;
interface CompteInterf extends Remote {
public String getTitulaire() throws RemoteException;
public float solde() throws RemoteException;
public void deposer( float montant ) throws RemoteException;
public void retirer( float montant ) throws RemoteException;
public List historique() throws RemoteException;
}
```

12

4. Modèle de programmation

Ecriture d'une classe implantant l'interface

- classe Java normale implantant l'interface
- doit étendre java.rmi.server.UnicastRemoteObject
- constructeurs doivent lever java.rmi.RemoteException
- si pas de constructeur, en déclarer un vide qui lève java.rmi.RemoteException

```
import java.rmi.server.UnicastRemoteObject;
import java.rmi.RemoteException;
public class CompteImpl extends UnicastRemoteObject
implements CompteInterf {
private String nom;
private float solde;
public CompteImpl(String nom) throws RemoteException {
super();
this.nom = nom;
}
public String getTitulaire() { return nom; }
... }
```

13

4. Modèle de programmation

Ecriture d'une classe implantant l'interface

1. Compilation de l'interface et de la classe avec javac
2. Génération des souches clientes et serveurs à partir du bytecode de la classe avec rmic

```
javac CompteInterf.java CompteImpl.java
rmic CompteImpl
```

- Quelques options utiles de rmic

-d path	répertoire pour les fichiers générés
-keep	conserve le code source des souches générées
-v1.1	souches version JDK 1.1
-v1.2	souches version JDK 1.2
-vcompat	**par défaut**, souches pour JDK 1.1 et 1.2

14

4. Modèle de programmation

Ecriture d'une classe implantant l'interface

- Fichiers générés pour chaque classe d'objet serveur RMI

```
rmic CompteImpl
```

- CompteImpl_Stub.java : souche cliente
- CompteImpl_Skel.java : souche serveur

15

4. Modèle de programmation

Ecriture du programme serveur

1. Instanciation de la classe serveur
2. Enregistrement de l'instance dans le serveur de noms RMI

```
import java.rmi.RemoteException
public class Serveur {
public static void main(String[] args) throws Exception {

    CompteInterf compte = new CompteImpl("Bob");
    Naming.bind("Bob",compte);
}
}
```

- compte prêt à recevoir des invocations de méthodes
- programme "tourne" en permanence
  tant que compte reste enregistré dans le runtime RMI
  (attention : ≠ du service de nommage)
- - désenregistrement : UnicastRemoteObject.unexportObject(compte,false)
  false : attente fin les requêtes en cours / true : immédiat

16

4. Modèle de programmation

Ecriture du programme client

1. Recherche de l'instance dans le serveur de noms
2. Invocation des méthodes

```
public class Client {
public static void main(String[] args) throws Exception {
CompteInterf compte = (CompteInterf) Naming.lookup("Bob");
compte.deposer(10);
System.out.println( compte.solde() );
}
}
```

17

4. Modèle de programmation

Exécution des programmes

1. Lancer le serveurs de noms (rmiregistry)
   - une seule fois
   - doit avoir accès au *bytecode* de la souche cliente (→ CLASSPATH)
2. Lancer le programme serveur
3. Lancer le programme client

18

4. Modèle de programmation

Compléments

Passage de paramètres avec RMI

• types de base (int, float, ...)	par copie
• objets implantant java.io.Serializable passés	par copie (sérialisés)
• objets implantant java.rmi.Remote passés	par référence
→ la référence RMI de l'objet est transmise	

• dans les autres cas une exception java.rmi.MarshalException est levée

19

4. Modèle de programmation

Compléments

Invocations concurrentes de méthodes RMI

Un objet serveur RMI est susceptible d'être accédé par plusieurs clients simultanément

- toujours concevoir des méthodes RMI *thread-safe*
  i.e. exécutable concurrement de façon cohérente
- la création de *thread* est faite automatiquement par le *runtime* RMI

20

4. Modèle de programmation

Compléments

- Serveurs de noms
- Problématique : publier la référence du serveur pour que les clients y accède
  machine "connue de tous" sur le réseau (idem DNS)

21

5. Services RMI

Service de nommage

- Permet d'enregistrer des liaisons entre un objet serveur et un nom symbolique
  - par défaut port 1099 (autre port : rmiregistry 12345)
  - noms "plats" (pas de noms composés, pas de hiérarchies)

```
URL RMI        rmi://machine.com:1099/nomSymbolique
rmi:// et :1099 facultatifs
machine.com par défaut localhost
```

- Serveur de noms démarrable
  - de façon autonome dans un *Shell* avec l'outil rmiregistry
  - dans un programme par appel de la méthode static

```
java.rmi.registry.LocateRegistry.createRegistry(int port)
```

22

5. Services RMI

Service de nommage

Classe java.rmi.Naming pour l'accès local
Toutes les méthodes sont static

1. `void bind(String,Remote)`	enregistrement d'un objet
2. `void rebind(String,Remote)`	réenregistrement d'un objet
3. `void unbind(String)`	désenregistrement d'un objet
4. `String[] list(String)`	liste des noms d'objets enregistrés
5. `Remote lookup(String)`	recherche d'un objet (retourne un stub)

Les paramètres String correspondent à une URL d'objet RMI

23