

Parallélisme

ERIC GOUBAULT
COMMISSARIAT À L'ÉNERGIE ATOMIQUE
SACLAY

1

REMOTE METHOD INVOCATION

- Permet d'invoquer des méthodes d'un objet distant, c'est à dire appartenant à une autre JVM, sur une autre machine
- Architecture de type client/serveur; similaire aux "Remote Procedure Calls" POSIX
- Se rapproche de plus en plus de CORBA (langage indépendant etc., voir cours suivant)

Références: JAVA, Network Programming and Distributed Computing, D. Reilly et M. Reilly, Addison-Wesley.
et <http://java.sun.com/products/jdk/rmi/>

ARCHITECTURE

- Classe qui implémente la méthode distante (serveur):
 - dont les méthodes renvoient un objet **serializable**
 - ou plus généralement une classe **remote**
- Client qui utilise les méthodes distantes
- Registre d'objets distants qui associe aux noms d'objets l'adresse des machines qui les contiennent

3

LES CLASSES IMPLÉMENTANT `Serializable`

- Objets instances peuvent être transcrits en "stream", c'est-à-dire en flots d'octets.
- `writeObject(ObjectOutputStream aOutputStream)`
`readObject(ObjectInputStream aInputStream)`
responsables respectivement de décrire un objet sous forme de flot d'octets et de reconstituer l'état d'un objet à partir d'un flot d'octets.
- La plupart des classes (et de leurs sous-classes) de base **String**, **HashTable**, **Vector**, **HashSet**, **ArrayList** etc. sont **Serializable**

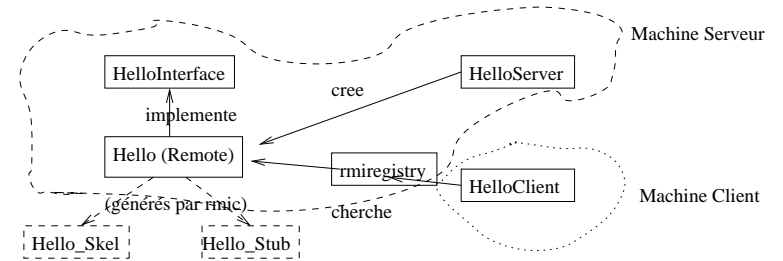
SERIALIZABLE

- dans le cas où on passe une classe **Serializable**, il faut que la définition de cette classe soit connue (\Rightarrow copiée sur les différentes machines) des clients et du serveur
- il peut y avoir à gérer la politique de sécurité (sauf pour les objets “simples”, comme **String** etc. voir prochain cours).

5

EXEMPLE - ON RETOURNE UN Serializable

“Hello World” distribué: on va construire les classes:



7

LES CLASSES remote

- leurs instances sont des objets ordinaires dans l'espace d'adressage de leur JVM
- des “pointeurs” sur ces objets peuvent être envoyés aux autres espaces d'adressage

INTERFACE DE L'OBJET DISTANT

```
import java.rmi.*;

public interface HelloInterface extends Remote {
    public String say() throws RemoteException;
}
```

HELLO WORLD: IMPLÉMENTATION DE L'OBJET DISTANT

```
import java.rmi.*;
import java.rmi.server.*;

public class Hello extends UnicastRemoteObject
    implements HelloInterface {
    private String message;

    public Hello(String msg) throws RemoteException {
        message = msg; }
}
```

9

HELLO WORLD: IMPLÉMENTATION DE L'OBJET DISTANT

```
public String say() throws RemoteException {
    return message;
}
}
```

COMPILATION

```
javac HelloInterface.java
javac Hello.java
(crée HelloInterface.class et Hello.class)
Création des stubs et squelettes:
rmic Hello
(crée Hello_Stub.class et Hello_Skel.class)
```

11

CLIENT

```
import java.rmi.*;
public class HelloClient {
    public static void main(String[] argv) {
        try {
            HelloInterface hello =
                (HelloInterface) Naming.lookup
                    ("//cher.polytechnique.fr/Hello");
            System.out.println(hello.say());
        }
    }
}
```

(le serveur est supposé toujours être sur **cher**, voir plus loin pour d'autres méthodes)

CLIENT

```
    } catch(Exception e) {  
        System.out.println("HelloClient exception: "+e);  
    }  
}  
}
```

13

SERVEUR

```
    } catch(Exception e) {  
        System.out.println("Hello Server failed: "+e);  
    }  
}  
}
```

15

SERVEUR

```
import java.rmi.*;  
  
public class HelloServer {  
    public static void main(String[] argv) {  
        try {  
            Naming.rebind("Hello",new Hello("Hello, world!"));  
            System.out.println("Hello Server is ready.");  
        }  
    }  
}
```

COMPILATION ET DÉMARRAGE DU SERVEUR

Faire attention au **CLASSPATH** (doit au moins contenir `.` et/ou les répertoires contenant les `.class` nécessaires, accessibles de toutes les machines sous NFS)

```
javac HelloClient.java  
javac HelloServer.java
```

Démarrer le serveur de noms:

```
rmiregistry &
```

(attendre un minimum)

COMPILATION ET DÉMARRAGE DU SERVEUR

Démarrer le serveur (Hello):

```
java HelloServer &  
(attendre un peu)
```

17

DÉMARRAGE DES CLIENTS ET EXÉCUTION

(ici en local)

```
> Hello Server is ready.  
> java HelloClient  
Hello, world!
```

INSTALLATION LOCALE AUX SALLES DE TD

- `setenv PATH $PATH:/usr/java/j2sdk1.4.0_01/bin` (pour `rmi` et `rmid`),
- démarrer `rmiregistry` dans le répertoire local où se trouvent les `.class` (ou vérifier le `CLASSPATH`). `nditemize`

19

INSTALLATION LOCALE AUX SALLES DE TD

- `rmiregistry` doit être démarré avec un numéro de port distinct pour plusieurs utilisateurs sur une même machine (numéros à partir de 1099), voir répartition sur fiche TD. Exemple sur machine serveur:

```
rmiregistry 1100 &
```
- Dans ce cas, le serveur devra s'enregistrer par

```
Naming.rebind("rmi://localhost:1100/pi");
```


Et le client devra chercher sur le même port:

```
Naming.lookup("rmi://cher.polytechnique.fr:1100/pi");
```

EXECUTION EN SALLE TD

(récupérer les programmes java dans `~goubaul1/Cours03/RMI/*`)

Exemple: serveur sur cher,

```
[goubaul1@cher HelloNormal]$ java HelloServer
Hello Server is ready.
```

Client sur loire,

```
[goubaul1@loire HelloNormal]$ java HelloClient
Hello, world!
```

CALLBACK

L'idée est la suivante (programmation "événementielle", typique d'interfaces graphique par exemple AWT):

- les "clients" vont s'enregistrer auprès d'un serveur,
- le "serveur" va les "rappeler" uniquement lorsque certains événements se produisent,
- le client n'a pas ainsi à faire de "l'active polling" (c'est à dire à demander des nouvelles continuellement au serveur) pour être mis au fait des événements.

PRINCIPE DU "RAPPEL"

Comment notifier un objet (distant) de l'apparition d'un événement?

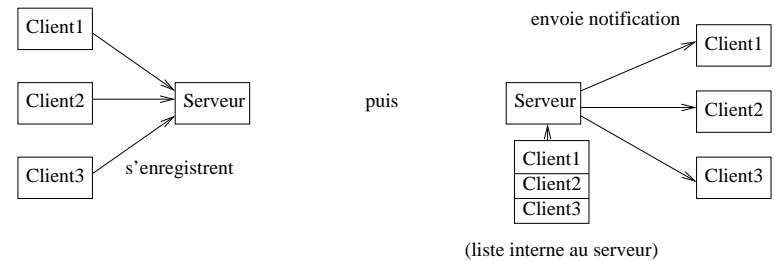
- on passe la référence de l'objet à rappeler, au serveur chargé de suivre (ou source des) les événements,
- à l'apparition de l'événement, le serveur va invoquer la méthode de notification du client.

Ainsi,

- pour chaque type d'événement, on crée une interface spécifique (pour le client qui veut en être notifié),
- les clients potentiels à notifier doivent s'enregistrer auprès d'une implémentation de cette interface.

Cela implique que "clients" et "serveurs" sont tous à leur tour "serveurs" et "clients".

C'EST À DIRE...



EXEMPLE - INTERFACE ASSOCIÉE À UN ÉVÉNEMENT

... ici, changement de température:

```
interface TemperatureListener extends java.rmi.Remote {
    public void temperatureChanged(double temperature)
        throws java.rmi.RemoteException;
}
```

C'est la méthode de notification de tout client intéressé par cet événement. Forcément un objet `Remote`.

25

EXEMPLE - L'INTERFACE DU SERVEUR D'ÉVÉNEMENTS

... doit au moins pouvoir permettre l'inscription et la désinscription de clients voulant être notifié:

```
interface TemperatureSensor extends java.rmi.Remote {
    public double getTemperature() throws
        java.rmi.RemoteException;
    public void addTemperatureListener
        (TemperatureListener listener)
        throws java.rmi.RemoteException;
    public void removeTemperatureListener
        (TemperatureListener listener)
        throws java.rmi.RemoteException; }
}
```

EXEMPLE - L'IMPLEMENTATION DU SERVEUR

- doit être une sous-classe de `UnicastRemoteObject` (pour être un serveur...).
- doit implémenter l'interface `TemperatureListener` pour pouvoir rappeler les clients en attente,
- implémente également `Runnable` ici pour pouvoir avoir un thread indépendant qui simule les changements de température.

27

EXEMPLE - IMPLEMENTATION SERVEUR

```
import java.util.*;
import java.rmi.*;
import java.rmi.server.*;

public class TemperatureSensorServer
    extends UnicastRemoteObject
    implements TemperatureSensor, Runnable {
    private volatile double temp;
    private Vector list = new Vector();
```

(le vecteur `list` contiendra la liste des clients)

EXEMPLE - IMPLÉMENTATION SERVEUR

Constructeur (température initiale) et méthode de récupération de la température:

```
public TemperatureSensorServer()  
    throws java.rmi.RemoteException {  
    temp = 98.0; }  
  
public double getTemperature()  
    throws java.rmi.RemoteException {  
    return temp; }
```

29

EXEMPLE - IMPLÉMENTATION SERVEUR

Thread responsable du changement aléatoire de la température:

```
public void run()  
{ Random r = new Random();  
  for (;;)   
    { try {  
      int duration = r.nextInt() % 10000 +2000;  
      if (duration < 0) duration = duration*(-1);  
      Thread.sleep(duration); }  
    }
```

31

EXEMPLE - IMPLÉMENTATION SERVEUR

Méthodes d'ajout et de retrait de clients:

```
public void addTemperatureListener  
    (TemperatureListener listener)  
    throws java.rmi.RemoteException {  
    System.out.println("adding listener -"+listener);  
    list.add(listener); }  
  
public void removeTemperatureListener  
    (TemperatureListener listener)  
    throws java.rmi.RemoteException {  
    System.out.println("removing listener -"+listener);  
    list.remove(listener); }
```

EXEMPLE - IMPLÉMENTATION SERVEUR

```
catch(InterruptedException ie) {}  
int num = r.nextInt();  
if (num < 0)  
    temp += .5;  
else  
    temp -= .5;  
notifyListeners(); } }
```

(`notifyListeners()` est la méthode suivante, chargée de broadcaster le changement d'événements à tous les clients enregistrés)

EXEMPLE - IMPLÉMENTATION DU SERVEUR

```
private void notifyListeners() {
    for (Enumeration e = list.elements(); e.hasMoreElements(); )
    { TemperatureListener listener =
        (TemperatureListener) e.nextElement();
        try {
            listener.temperatureChanged(temp);
        } catch (RemoteException re) {
            System.out.println("removing listener -"+listener);
            list.remove(listener); } } }
```

(on fait simplement appel, pour chaque client, à la méthode de notification `temperatureChanged`)

33

EXEMPLE - IMPLÉMENTATION DU SERVEUR

Enregistrement du service auprès du `rmiregistry` (éventuellement fourni à la ligne de commande):

```
public static void main(String args[]) {
    System.out.println("Loading temperature service");
    try {
        TemperatureSensorServer sensor =
            new TemperatureSensorServer();
        String registry = "localhost";
        if (args.length >= 1)
            registry = args[0];
        String registration = "rmi://" + registry +
            "/TemperatureSensor";
        Naming.rebind(registration, sensor);
    }
}
```

EXEMPLE

Démarrage du thread en charge de changer aléatoirement la température, et gestion des exceptions:

```
Thread thread = new Thread(sensor);
thread.start(); }
catch (RemoteException re) {
    System.err.println("Remote Error - "+re); }
catch (Exception e) {
    System.err.println("Error - "+e); } } }
```

35

EXEMPLE - IMPLÉMENTATION CLIENTS

```
import java.rmi.*;
import java.rmi.server.*;
```

```
public class TemperatureMonitor extends UnicastRemoteObject
    implements TemperatureListener {
    public TemperatureMonitor() throws RemoteException {}
}
```

(étend `UnicastRemoteObject` car serveur également! De même implémenter `TemperatureListener`) Rq: constructeur vide (celui d'`Object` en fait).

EXEMPLE - IMPLÉMENTATION CLIENTS

Recherche du service serveur d'événements:

```
public static void main(String args[]) {
    System.out.println("Looking for temperature sensor");
    try {
        String registry = "localhost";
        if (args.length >= 1)
            registry = args[0];
        String registration = "rmi://" + registry +
                               "/TemperatureSensor";
        Remote remoteService = Naming.lookup(registration);
        TemperatureSensor sensor = (TemperatureSensor)
                                    remoteService;
```

37

EXEMPLE - IMPLÉMENTATION CLIENTS

Création d'un moniteur et enregistrement auprès du serveur d'événements:

```
double reading = sensor.getTemperature();
System.out.println("Original temp : "+reading);
TemperatureMonitor monitor = new TemperatureMonitor();
sensor.addTemperatureListener(monitor);
```

EXEMPLE - IMPLÉMENTATION CLIENTS

Gestion des exceptions:

```
} catch (NotBoundException nbe) {
    System.out.println("No sensors available"); }
catch (RemoteException re) {
    System.out.println("RMI Error - "+re); }
catch (Exception e) {
    System.out.println("Error - "+e); } }
```

39

EXEMPLE - IMPLÉMENTATION CLIENTS

Implémentation de la méthode de rappel:

```
public void temperatureChanged(double temperature)
    throws java.rmi.RemoteException {
    System.out.println("Temperature change event : "
        +temperature);
}
```

COMPILATION

```
[goubaul1@cher Ex3]$ javac *.java
[goubaul1@cher Ex3]$ rmic TemperatureMonitor
[goubaul1@cher Ex3]$ rmic TemperatureSensorServer
```

41

EXÉCUTION

```
Temperature change event : 100.0
Temperature change event : 100.5
Temperature change event : 101.0
Temperature change event : 100.5
Temperature change event : 100.0
Temperature change event : 100.5
Temperature change event : 101.0
Temperature change event : 101.5
```

43

EXÉCUTION

```
[goubaul1@cher Ex3]$ rmiregistry &
[goubaul1@cher Ex3]$ java TemperatureSensorServer
Loading temperature service
Premier client (sur loire):
[goubaul1@loire Ex3]$ rmiregistry &
[goubaul1@loire Ex3]$ java TemperatureMonitor cher
Looking for temperature sensor
Original temp : 100.0
Temperature change event : 99.5
Temperature change event : 100.0
Temperature change event : 100.5
```

EXÉCUTION

On voit alors sur la console de cher:

```
adding listener -TemperatureMonitor_Stub[RemoteStub
[ref: [endpoint:[129.104.254.64:3224](remote),
objID:[6e1408:f29e197d47:-8000, 0]]]]
```

Rajoutons un moniteur sur doubs:

```
[goubaul1@doubs Ex3]$ rmiregistry &
[goubaul1@doubs Ex3]$ java TemperatureMonitor cher
Looking for temperature sensor
Original temp : 101.5
Temperature change event : 102.0
```

EXÉCUTION

```
Temperature change event : 102.5
Temperature change event : 103.0
Temperature change event : 102.5
Temperature change event : 103.0
Temperature change event : 103.5
Temperature change event : 102.5
Temperature change event : 102.0
```

45

EXÉCUTION

```
Temperature change event : 102.5
Temperature change event : 103.0
Temperature change event : 102.5
Temperature change event : 103.0
Temperature change event : 103.5
^C
[goubaul1@loire Ex3]$
```

47

EXÉCUTION

Ce qui produit sur cher:

```
adding listener -TemperatureMonitor_Stub[RemoteStub
[ref: [endpoint:[129.104.254.57:3648] (remote),
objID:[6e1408:f29de7882e:-8000, 0]]]]
```

On voit bien que les températures et événements sont synchronisés avec l'autre client sur loire:

```
Temperature change event : 102.0
```

EXÉCUTION

On a interrompu sur loire, du coup sur cher:

```
removing listener -TemperatureMonitor_Stub[RemoteStub
[ref: [endpoint:[129.104.254.64:3224] (remote),
objID:[6e1408:f29e197d47:-8000, 0]]]]
```

On interrompt par Control-C sur doubs, du coup sur cher:

```
removing listener -TemperatureMonitor_Stub[RemoteStub
[ref: [endpoint:[129.104.254.57:3648] (remote),
objID:[6e1408:f29de7882e:-8000, 0]]]]
```