Java RMI Programmation Concurrentielle et Réseau

Par : S.Rakrak

Java RMI

Introduction

Des objets aux services : L'objet

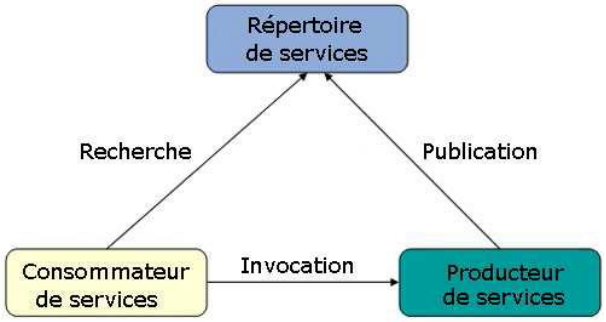
- Une des problématiques fondamentales de l'ingénierie informatique réside dans la capacité à rationaliser le développement des applications.
- Question : Comment réutiliser des briques logicielles déjà développées ?
- Il y a déjà une dizaine d'années, l'approche objet donnait un premier élément de réponse à cette problématique
- Les objets représentent des entités du monde réel
- Mais techniquement, les objets sont dépendants des langages et des environnements d'exécution.

Des objets aux services : Composant

- Un composant
  - est une unité de composition avec des **interfaces** spécifiées à l'aide de **contrats** et de dépendances au contexte seulement.
  - peut être déployé **indépendamment** et
  - sujet à **être composé** par des tiers
- Les composants décrivent des fonctionnalités du monde réel
- Mais Techniquement, les composants se déploient et s'exécutent dans des contextes qui leur sont spécifiques

Des objets aux services : Service

- Un service dans une SOA est un entité étendue de trois fonctionnalités
  1. le contrat d'interface du service est indépendant de plate-forme,
  2. le service peut être dynamiquement localisé et invoqué,
  3. le service est indépendant, c'est-à-dire le service entretient son état propre



Notion d'Intergiciel (Middleware)

- **Intergiciel :** une terminologie pour une classe de logiciels systèmes qui permet d'implanter une approche répartie
  - Fournit une API d'interactions de communication
    - Interactions de haut niveau pour des applications réparties
    - invocation distante de méthode pour des codes objets.
  - Fournit un ensemble de services utiles pour des applications s'exécutant en environnement réparti
    - désignation,
    - cycle de vie,
    - sécurité,
    - transactionnel, etc.
  - Fonctionne en univers ouvert (supporte des applications tournant sur des plate-formes matérielles et logicielles différentes).

Les principales catégories d'Intergiciels

- Intergiciels à **messages** " **MOM: Message Oriented Middleware** (IBM MQSeries**,** Microsofr Message Queues Server,DECmessageQ, JMS)
- Intergiciels de **bases de données** (ODBC)
- Intergiciels à **appel de procédure distante** (DCE)
- Intergiciels à **objets répartis** (CORBA, JAVA RMI)
- Intergiciels à **composants** (EJB, Web services)

Java RMI

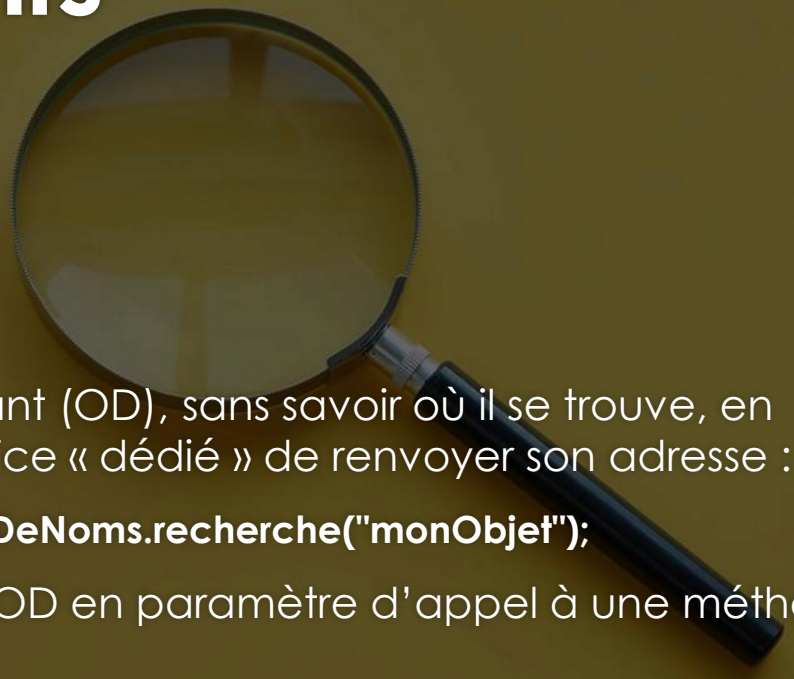
Architecture et Principes

RMI : Objectifs

- L'idéal serait d'avoir un système distribué utilisant la technologie objet et permettant :

1. d'invoquer une méthode d'un objet se trouvant sur une autre machine exactement de la même manière que s'il se trouvait au sein de la même machine :
   - **objetDistant.methode()**

RMI : Objectifs



2. d'utiliser un objet distant (OD), sans savoir où il se trouve, en demandant à un service « dédié » de renvoyer son adresse :
   - **objetDistant = ServiceDeNoms.recherche("monObjet");**
3. de pouvoir passer un OD en paramètre d'appel à une méthode locale ou distante :
   - **resultat = objetLocal.methode(objetDistant);**
   - **resultat = objetDistant.methode(autreObjetDistant);**

RMI : Objectifs

4. de pouvoir récupérer le résultat d'un appel distant sous forme d'un nouvel objet qui aurait été créé sur la machine distante :

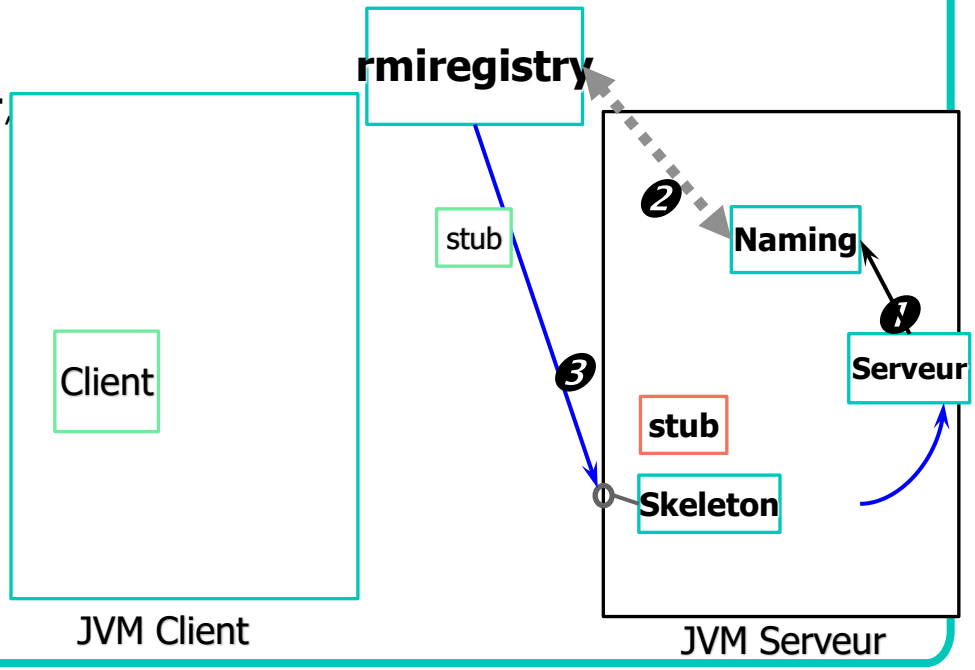
   - **ObjetDistant = ObjetDistant.methode() ;**

Java RMI

Mode opératoire

Mode opératoire coté serveur

- Un **serveur** permet à un **objet** de **recevoir des appels** de méthodes distantes : un client peut utiliser l'objet
- Le **serveur enregistre** cet **objet**, un service de désignation étant exécuté sur le réseau (appelé **RMI registry**)
- Les clients peuvent retrouver l'objet au moyen d'une procédure de **recherche de noms**



Mode opératoire coté serveur : Etapes

A la création de l 'objet, un **stub** et un **skeleton** (avec un port de communication) sont créés du coté du serveur

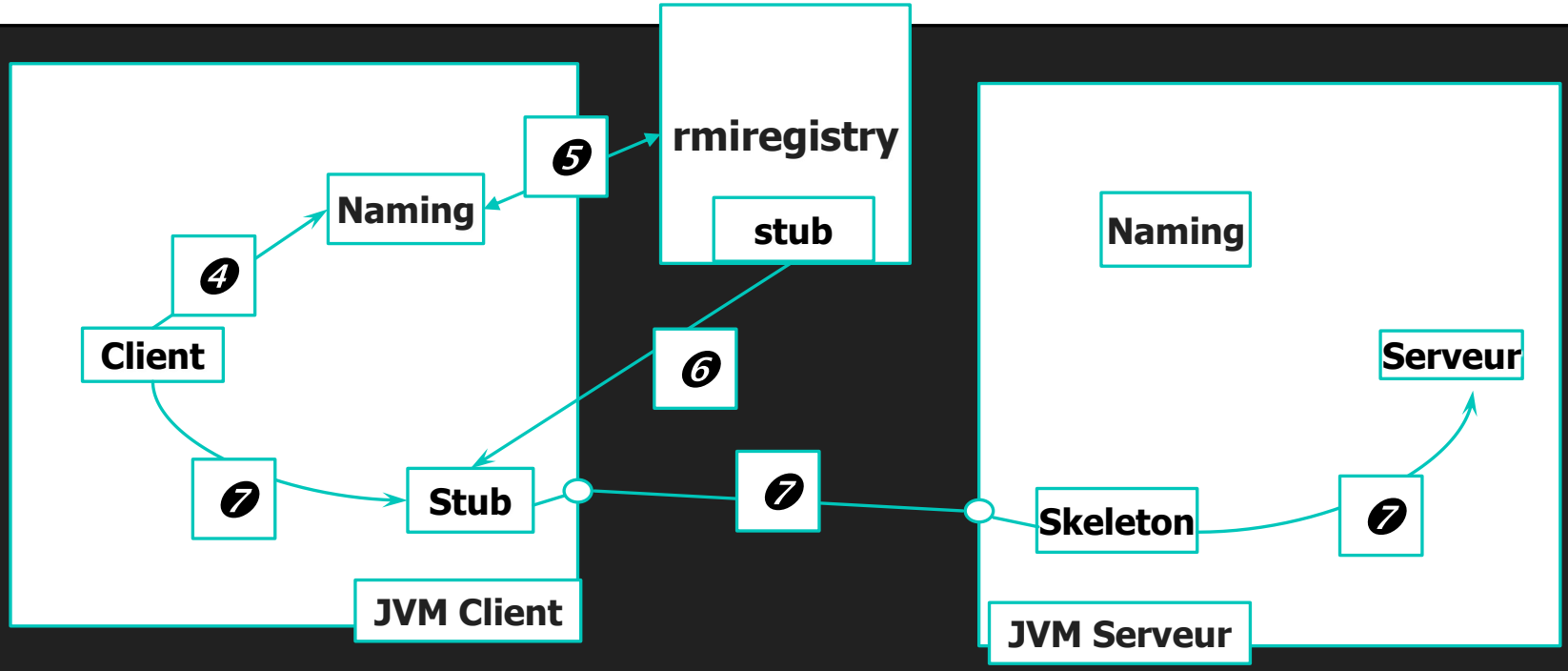
1. L'objet serveur s'enregistre auprès du **Naming** de sa JVM (méthode ***rebind***)

2. Le **Naming** enregistre le **stub** de l'objet (**sérialisé**) auprès du serveur de noms (***rmiregistry***)

3. Le serveur de noms est prêt à donner des références à l'objet serveur

Mode opératoire coté client

- Appel de méthodes distantes (**Stub** et **squelette**)
  - Le client fait appel à un proxy local, appelé (Souche) **Stub** (interface de l'objet distant)
  - Le serveur converti les appels de méthodes sur le stub en appels locaux grâce à un objet spécial appelé **Squelette**
  - Les résultats sont envoyés au squelette, qui les **empaquette** et les **transmet** au **Stub**



Mode opératoire coté client : Etapes

4. L'objet client fait appel au **Naming** pour localiser l'objet serveur (méthode ***lookup***)

5. Le **Naming** récupère le **Stub** vers l'objet serveur,

6. Le **Naming** installe l'objet **Stub** et retourne sa référence au client

7. Le client effectue l'appel à l'objet serveur par appel à l'objet Stub

Java RMI

Mise en oeuvre de RMI

Mise en oeuvre de RMI : Etapes

1. Créer l'interface distante
   1. Elle doit étendre **`java.rmi.Remote`** et déclarer les méthodes publiques globales accessibles.
   2. Chaque méthode doit déclarer qu'elle peut lancer une exception **`java.rmi.RemoteException.`**
2. Mettre en œuvre l'interface distante
   1. Créer **l'objet distant en implémentant** l'interface distante et en étendant **UnicastRemoteObject**.
   2. Dans main(), installer un gestionnaire de sécurité, créer une instance de l'objet et la lier au registre.

Mise en oeuvre de RMI : Etapes

3. Compiler les sources et générer les stubs et les squelettes
    1. Compiler l'interface et l'objet distant
    2. Générer le stub et le squelette, à l'aide de l'outil **`rmic`**

4. Lancer le registre et activer l'objet distant
    1. Dans l'application client, appeler **`Naming.lookup()`** pour obtenir une référence à l'objet distant.
    2. Convertir le type avec l'interface distante. Les méthodes deviennent accessibles localement

5. Compiler les sources clients et exécuter l'application

Interface distante

- Tout objet Java mettant en œuvre une interface distante peut être considéré comme distant
- L'interface distante doit étendre **java.rmi.Remote**
- Elle définit les méthodes qui peuvent être accessibles aux objets distants qui implémentent cette interface
- Chaque méthode doit pouvoir lancer une exception de type **`java.rmi.RemoteException`** (dans la clause `throws`)
- Les paramètres ou les valeurs de retour de ces méthodes doivent être des objets de type **`Serializable`**

```
import java.rmi.Remote;
import java.rmi.RemoteException;

public interface Echo extends Remote {
public String echo(String str) throws RemoteException;
}
```

Java RMI : écriture du serveur

Serveur = la classe qui implémente l'interface

spécifier les interfaces distantes qui doivent être implémentées

- objets passés par copie (il doivent implémenter l'interface java.io.serialisable)

Serveur = c'est un objet java standard

définir le constructeur de l'objet

fournir la mise en œuvre des méthodes pouvant être appelée à distance

ainsi que celle des méthodes n'apparaissant dans aucune interface implémentée

créer au moins une instance du serveur

enregistrer au moins une instance dans le serveur de nom (rmiregistry)

Objet distant : implémentation de l'interface

- Création de l'objet côté serveur
- implémentation de l'interface par extension de la classe `java.rmi.UnicastRemoteObject`

```
import java.rmi.*;
import java.rmi.server.*;
public class EchoImpl extends UnicastRemoteObject implements Echo {
private static final long serialVersionUID = 1L;

public EchoImpl() throws RemoteException {
super();
}

public String echo(String str) throws RemoteException {
return "Le Serveur Repond Au : " + str;
}
}
```

Objet distant : implémentation de l'interface

- Création d'une instance de l'objet **`EchoImpl`** et l'enregistre comme "Server" sur "localhost". Tout client peut alors retrouver EchoImpl

```
import java.rmi.*;
import java.rmi.registry.LocateRegistry;
public class EchoAppliServer {
  public static void main(String args[]) {
    try {// Cration de l'OD
      EchoImpl od = new EchoImpl();
      LocateRegistry.createRegistry(1099);
      // Enregistrement de l'OD dans RMI
      Naming.rebind("IRISI", od);
      System.out.println("L'Objet Distant (OD) est Enregistré dans RMI... Serveur Prêt");
    } catch (Exception e) {
      System.out.println("Serveur Non Lancé!!!");
    }
}
}
```

Client Java

- L'application cliente doit obtenir une référence à l'objet distant.
  - Importer le package **`java.rmi`** et **`java.rmi.registry`**
  - La méthode **`Naming.lookup(String server)`** permet d'obtenir l'interface distante
- L'objet distant peut ensuite être utilisé comme s'il était local
- Le client ne manipule pas explicitement le stub, ni le marshalling des appels à l'objet distant

```
import java.rmi.*;
public class EchoClient {
public static void main(String args[]) {
        // Recherche de IOD
        String url = "rmi://localhost/IRISI";
        try {
                Echo od = (Echo) Naming.lookup(url);
                System.out.println(od.echo("Message du Client"));
        } catch (Exception e) {
                System.out.println("Serveur introuvable!!!");
        }
}
}
```