



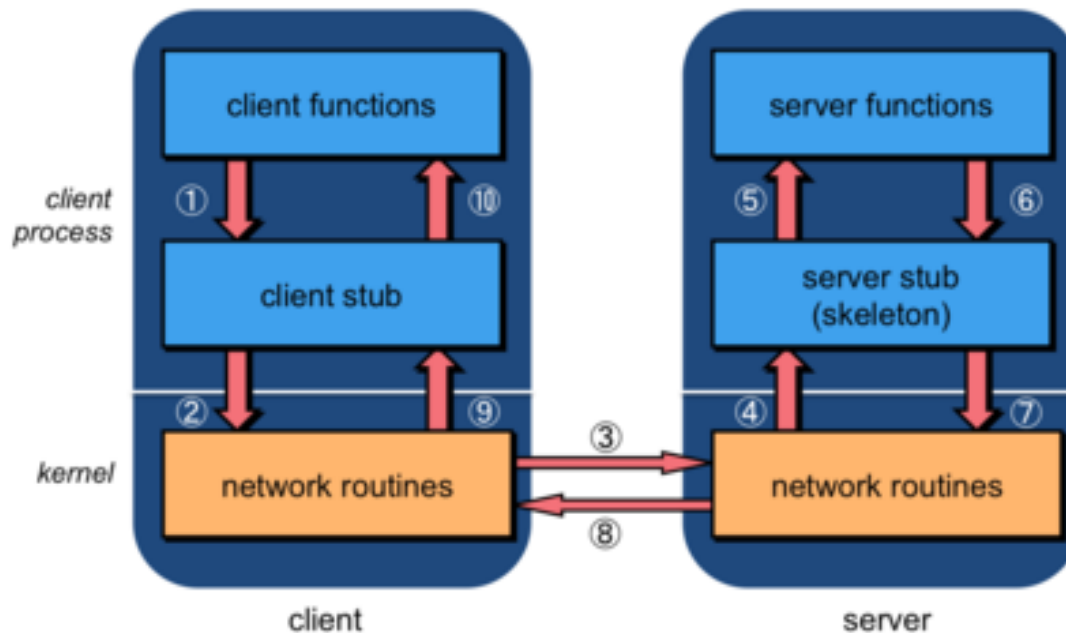
TP Java RMI

Alexandre Denis – Alexandre.Denis@inria.fr

**Inria Bordeaux – Sud-Ouest
France**

Paradigme RMI

- RMI : Remote Method Invocation
 - RPC “orientés objet” (encapsulation, héritage, ...)
 - objet : entité de désignation et de distribution



Java RMI

- Des “**RPC orientés objets**” en Java
 - Passage des **paramètres par copie** lors des appels de méthode à distance
 - **sérialisation**
 - implements Serialized
 - Transport TCP
 - Déploiement via le registre de nom “**rmiregistry**”
- Intégration à l'écosystème Java :
 - ramasse-miette réparti (Distributed Garbage Collector)
 - intégration des exceptions
 - chargement dynamique de “byte code” (codebase)
 - aspect sécurité, applets, persistance, proxy, serveur http

Modèle de threads

- Appel **synchrone** de la méthode distante
 - Client bloqué tout le temps de l'appel distant
 - 1 thread par client côté serveur
- Appel asynchrone
 - En utilisant explicitement un thread côté client...
- Appels concurrents côté serveur
 - Exclusion mutuelle avec le mot clef “synchronized”

Interface d'Objet Distribu 

- Un **Objet Distribu ** (OD)
 - Son interface doit ** tendre java.rmi.Remote**
 - Ses m thodes doivent d clarer qu'elles sont susceptibles de lancer des **exceptions** `java.rmi.RemoteException`
 - Son interface doit manipuler des types :
 - Primitifs, transmis par copie
 - **Serializable** (e.g. `String`, ..., ou d fini par l'utilisateur)
 - **Remote**, r f rences vers d'autres OD
- Ne pas confondre
 - Remote : **r f rence** vers un objet distant
 - Serializable : **objet copi ** via le r seau

Exemple d'interface

-

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

public interface Hello extends Remote
{
    String sayHello() throws RemoteException;
}
```

Implémentation de l'objet

- HelloImpl.java

```
import java.rmi.RemoteException;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import java.rmi.server.UnicastRemoteObject;

class HelloImpl implements Hello
{
    public String sayHello() throws RemoteException
    {
        System.out.println("Remote Invokation of method sayHello()");
        return "Hello World!";
    }
}
```

Implémentation du serveur

- HelloServer.java

```
public class HelloServer
{
    public static void main(String args[])
    {
        try
        {
            // activation de l'objet distribué
            Hello hello = new HelloImpl();
            Hello stub = (Hello)UnicastRemoteObject.exportObject(hello, 0);

            // binding de l'objet distribué dans le registre de nom
            Registry registry = LocateRegistry.getRegistry(args[0]);
            registry.rebind("hello", stub);
        }
        catch (Exception e)
        {
            System.out.println("HelloServer Exception: " + e.getMessage());
            e.printStackTrace();
        }
    }
}
```


HelloClient.java

```
import java.rmi.RemoteException;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;

public class HelloClient
{
    public static void main(String args[])
    {
        try
        {
            // recherche de l'objet distribué dans le registre de nom
            Registry registry = LocateRegistry.getRegistry(args[0]); // ← machine serveur
            Hello hello = (Hello) registry.lookup("hello");

            // appel de méthode à distance
            String s = hello.sayHello();
            System.out.println(s);
        }
        catch (Exception e)
        {
            System.out.println("HelloClient exception: " + e.getMessage());
            e.printStackTrace();
        }
    }
}
```

Registre de nom (= annuaire)

- `rmiregistry [port]`
 - démarre le registre de nom des objets distants sur la machine en cours avec un certain numéro de port (par défaut, port 1099)

- Exemples

```
rmiregistry &
```

```
rmiregistry 2011 &
```

Déploiement

1) Compilation du client et du serveur RMI

```
HostA> javac *.java
```

2) Démarrage du registre de nom RMI (par défaut port 1099)

```
HostA> rmiregistry &
```

3) Démarrage du serveur (sur la même machine que le registre de nom)

```
HostA> java HelloServer localhost
```

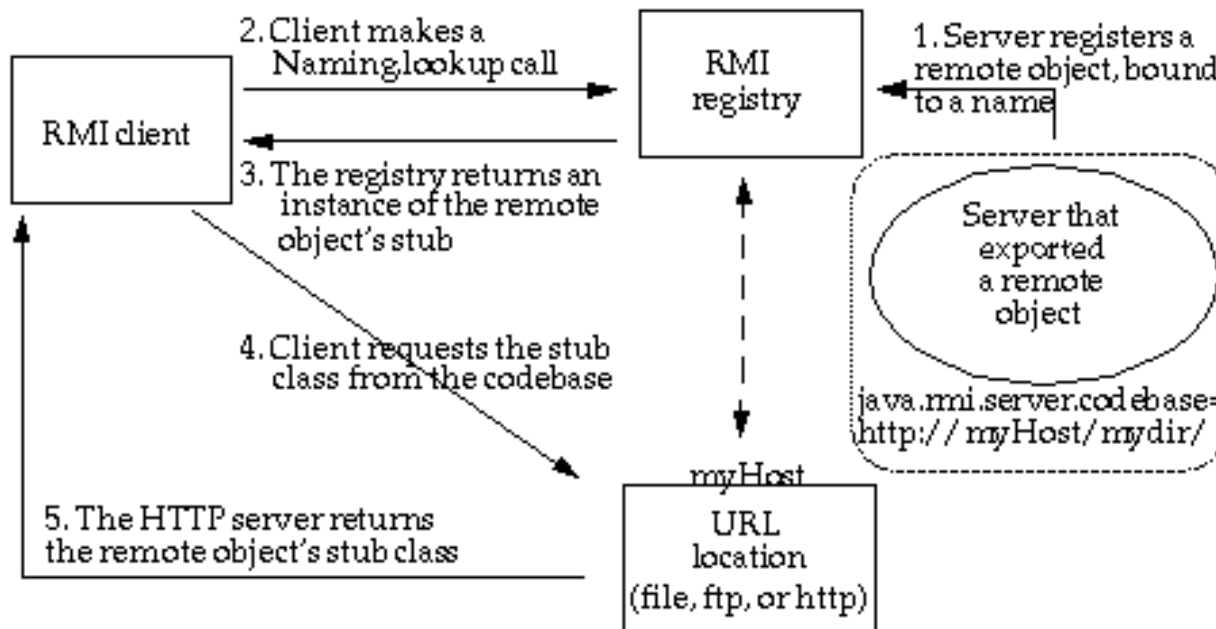
4) Démarrage du client RMI (en local et à distance)

```
HostA> java HelloClient localhost
```

```
HostB> java HelloClient HostA
```

Téléchargement des stubs

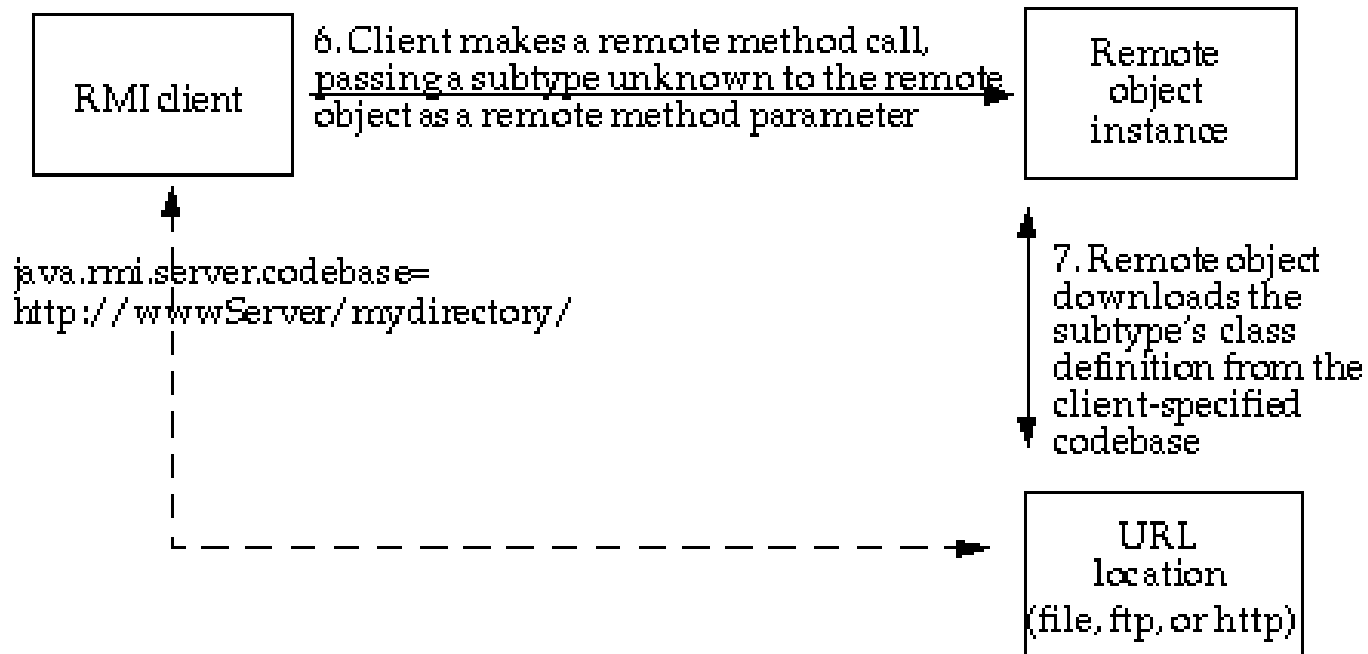
- Cas où le client ne connaît que l'interface Remote et ne dispose pas par avance des stubs...



Source : Sun

Téléchargement d'un sous-type

- Le serveur manipule un sous-type défini uniquement par le serveur...



Source : Sun

Security Manager et Codebase

- Utilisation d'un Security Manager

```
if (System.getSecurityManager() == null) {  
    System.setSecurityManager(new SecurityManager());  
}
```

- Politique de sécurité (fichier all.policy)

```
grant {  
    permission java.security.AllPermission;  
};
```

- Prise en compte de la politique de sécurité

```
java -Djava.security.policy=all.policy ...
```

- Utilisation du codebase avec HTTP (extension du classpath)

```
java ... -Djava.rmi.server.codebase=http://webserver/classes/
```

Récapitulatif

- Définition de l'**interface**, partagée entre client et serveur
- Implémentation de l'**objet**, dans le serveur
- Implémentation du **processus serveur**
 - Activation de l'objet
 - Publication de la référence dans le registre
- Implémentation du **client**
 - Recherche de la référence dans le registre
 - Appel de méthode à distance

Travail à faire

Echauffement

- Compiler et faire tourner les exemples Hello World
 - En local
 - A distance
 - Avec le serveur d'un voisin
- Modifier l'exemple pour que le message soit passé en argument de la méthode
- Modifier le serveur pour avoir plusieurs instances d'objets distribués

Calcul distribué

- Approximation de Pi par la formule de Bailey–Borwein–Plouffe (1995)

$$\pi = \sum_{k=0}^{\infty} \left[\frac{1}{16^k} \left(\frac{4}{8k+1} - \frac{2}{8k+4} - \frac{1}{8k+5} - \frac{1}{8k+6} \right) \right]$$

- Permet de calculer directement le N^{ème} chiffre de Pi
 - En hexadécimal...
- Notes :
 - Calculer chiffre par chiffre n'est pas le meilleur algo pour calculer Pi
 - BBP pas le meilleur algo pour calculer les chiffres de Pi
 - Bellard, puis Gourdon ont proposé un algo plus efficace
 - Mais : algo **simple** pour calculer en distribué

Calcul distribué – étape 1, version séquentielle

- Télécharger pi.java
- Compiler, faire tourner localement
- Faire un service de calcul distribué maître-esclave
 - Un objet esclave fournissant la méthode `piDigit`
 - Un maître appelle la méthode sur un esclave
- Ça sert juste à se faire la main sur les RMI
 - Sinon, aucun intérêt...
- **Attention**, le code donné est entièrement `static`
 - À modifier pour pouvoir instancier des objets

Calcul distribué – étape 2, version parallèle naïve

- On veut en faire une version parallèle
 - Plusieurs noeuds de calcul
 - -> plusieurs serveurs, un seul client
- Modifier le client pour en faire une version parallèle
 - Le client envoie une requête à chaque serveur à tour de rôle
 - Au fur et à mesure, il reconstitue le résultat global
 - Faire tourner avec plusieurs esclaves simultanés
- Les requêtes RMI sont synchrones, donc ne s'exécutent pas en même temps
 - Comment les rendre **asynchrones** pour avoir un vrai parallélisme ?

Calcul distribué – étape 3, paquets

- La première version n'est pas très efficace avec un RMI pour chaque chiffre
 - Nous allons désormais calculer par paquet de chiffres, et non chiffre par chiffre
 - Quel type d'objet circule alors sur le réseau avec les RMIs ?
- Proposer une nouvelle version où chaque appel calcule un ensemble de chiffres
 - Le maître doit réassembler les paquets/chiffres dans le bon ordre !
 - Vérifier que le résultat est identique à celui donné par le `pi.java` original

Calcul distribué – étape 4, callback

- Pour gérer l'asynchronisme entre maître et esclaves, le **callback** est une alternative fréquente
 - Le maître est serveur, propose du travail
 - Les esclaves demandent du travail et envoient leur résultat
 - 2 méthodes séparées
 - Attention, modèle renversé par rapport à la version précédente
 - Maître serveur, esclave client
- Proposer une nouvelle version basée sur ce principe
 - Vérifier encore une fois que le résultat est identique à celui donné par le `pi.java` original
- Quelle version est la plus simple ?

FAQ

- `ClassNotFoundException` à l'enregistrement d'un objet dans le registry
 - `rmiregistry` doit avoir la définition de l'interface dans son CLASSPATH
- `Address already in use` au lancement de `rmiregistry`
 - Utilisez un autre port que le port par défaut 1099

À vous de jouer !

inria
informatics mathematics

<http://dept-info.labri.fr/~denis/>