Développement d'applications réparties

3GLSI
A.U 2021/2022

Chapitre 1

Introduction aux applications réparties

Applications ?

- Une application est un ensemble d'entités logicielles, qui permet à un ordinateur ou à un système informatique d'assurer une tâche ou une fonction particulière
- Les énormes capacités des micro-ordinateurs ont permis la réalisation d'applications monolithiques où toutes les tâches sont traitées sur la même machine.
- Capacités de traitement importantes, mais besoins toujours plus importants (en puissance de calcul, en espace de données, en partage de l'information,...).

Réparties 1/2

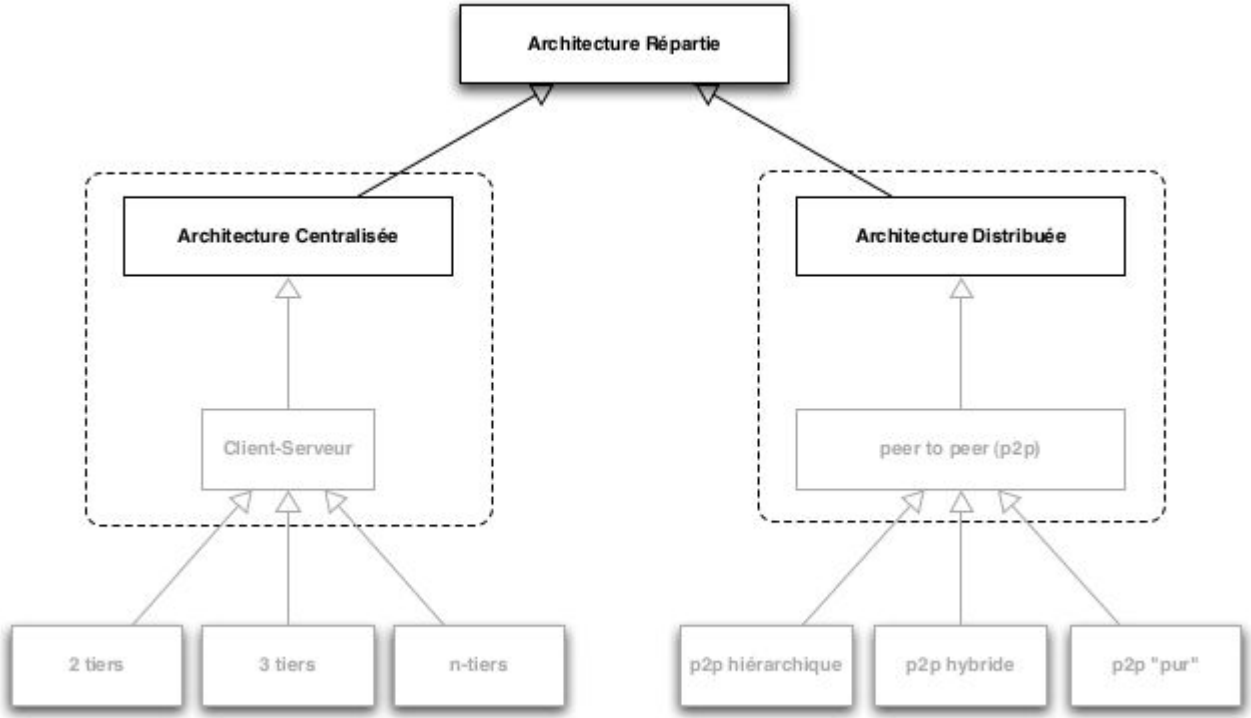
Une application **répartie** est constituée d'entités logicielles pouvant être:

- ➔ Exécutées dans des environnements d'exécution séparés(pas le même espace mémoire).
- ➔ Développées dans différents langages de programmation.
- ➔ Situées sur des plate-formes distinctes connectées via un réseau

On peut regrouper les caractéristiques des applications réparties selon deux grandes catégories architecturales :

- ❏ Les applications réparties "*centralisées*". On parle d'architecture centralisée car les nœuds jouant le rôle de serveur centralisent une grande partie de l'information et des communications et sont distincts des nœuds clients.
- ❏ Les applications réparties "*distribuées*". Dans les architectures distribuées, tous les nœuds du réseau partagent théoriquement les mêmes rôles : ils sont simultanément clients et serveurs et disposent de tout ou partie de l'information répartie

Réparties 2/2



Définitions 1/3

- Système réparti ⟺ système distribué
- Un système réparti est un système informatique dans lequel les ressources ne sont pas centralisées". Ces ressources sont notamment :
  - ➔ les moyens de stockage (données, fichiers)
  - ➔ la charge CPU
  - ➔ les utilisateurs
  - ➔ les traitements …

Définitions 2/3

- Un système réparti est un ensemble de machines autonomes connectées par un réseau, et équipées d'un logiciel dédié à la coordination des activités du système ainsi qu'au partage de ses ressources.

- Un système réparti est un système qui s'exécute sur un ensemble de machines sans mémoire partagée, mais que pourtant l'utilisateur voit comme une seule et unique machine

Définitions 3/3

- Un système réparti est un système comprenant un ensemble de processus et un système de communication.
- Ensemble composé d'éléments reliés par un système de communication:
  - ➔ Les éléments ont des fonctions de traitement (processeurs), de stockage (mémoire), de relation avec le monde extérieur (capteurs, actionneurs)
  - ➔ Les différents éléments du système ne fonctionnent pas indépendamment mais collaborent à une ou plusieurs tâches communes.
  - ➔ Une partie au moins de l'état global du système est partagée entre plusieurs éléments.

Système centralisé vs Système distribué

- Système centralisé :
  - ➔ Tout est localisé sur la même machine et accessible par le programme.
  - ➔ Système logiciel s'exécutant sur une seule machine.
  - ➔ Les applications accèdent localement aux ressources nécessaires (données, code, périphériques, mémoire ...).
- Système distribué :
  - ➔ Ensemble d'ordinateurs indépendants connectés en réseau et communiquent via ce réseau.
  - ➔ Cet ensemble apparaît du point de vue de l'utilisateur comme une seule entité.

Vision matérielle et logicielle d’un système distribué

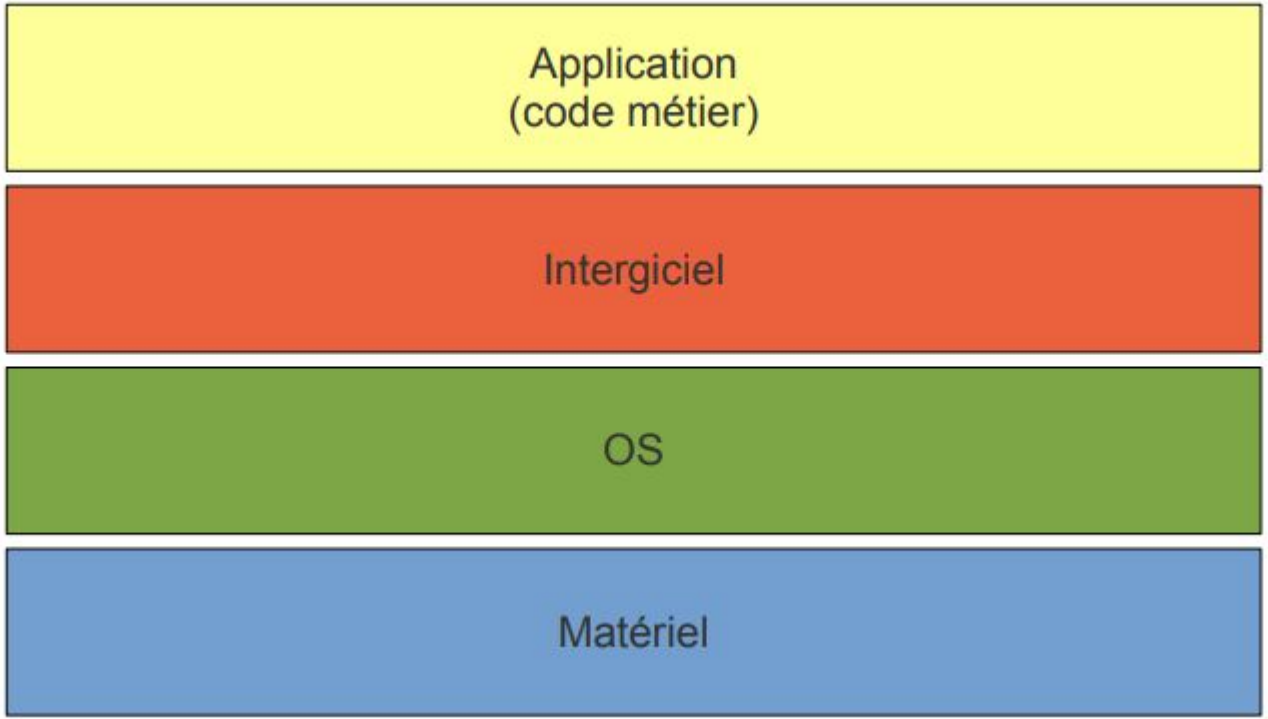
❖ Architecture Matérielle :
- Machine multi-processeurs avec mémoire partagée.
- Cluster d'ordinateurs dédiés au calcul/traitement massif parallèle.
- Ordinateurs standards connectés en réseau

❖ Vision Logicielle :
- Système logiciel composé de plusieurs entités s'exécutant indépendamment et en parallèle sur un ensemble d'ordinateurs connectés en réseau

Middleware

- Un middleware, appelé aussi logiciel médiateur ou intergiciel, se présente sous la forme d'un logiciel. Il constitue une couche technique supplémentaire entre le système d'exploitation (OS, Operating System) et les applications afin de faciliter leurs interactions. Le middleware permet également la communication de données entre applications hétérogènes.
- Se situe au-dessus de la couche de transport.
- Permet **la gestion des appels** de fonctions de l'application ou la gestion du renvoi des résultats entre les clients et les serveurs.
- Permet **la mise en forme des données** en vue de leur prise en charge par la couche transport

Middleware



Middleware

- ❖ Deux composants :
- ➔ Le protocole d'accès formaté (Format And Protocol, FAP) met en forme les différentes données au niveau du réseau et définit le protocole d'échange des données.
- ➔ L'interface de programmation (Application Programming Interface, API) se charge :
  - des connexions et déconnexions avec le serveur
  - de la définition de l’environnement de la connexion (variables de contexte,...)
  - du transfert des requêtes et de la réception des résultats

La programmation répartie

- La plupart des applications réparties sont de type client/serveur: le client demande des services à un serveur.
- En programmation classique, lorsque un programme a besoin d'un service, il appelle localement une fonction/procédure/méthode d'une librairie, d'un objet, etc.
- En programmation répartie, l'appel de fonction /procédure/méthode peut se faire à distance.

La programmation répartie vs la prog. classique

Programmation classique	Programmation répartie
L'utilisateur du service et le fournisseur de service se trouvent sur la même machine: • Même OS • Même espace mémoire • Même capacité de calcul CPU • Pas de problème de transport • Disponibilité du service assuré (tant que l'on a accès à la librairie)	L'utilisateur et le fournisseur de service ne se trouvent pas sur la même machine: deux machines différentes (sans compter celles traversées) • OS différents • Espace mémoire non unitaire : "passer un pointeur comme argument"? • Problème de transport : pare-feu (firewall),réseau, etc. • Retrouver le service ? où se trouve-t-il ? qui le propose?

La programmation répartie vs la prog. classique

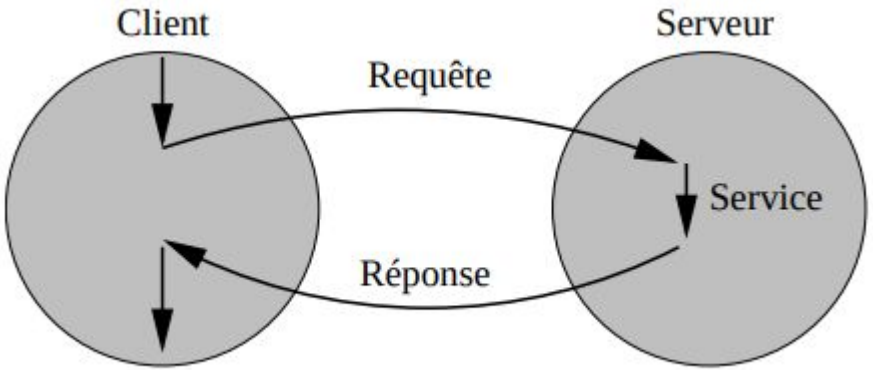
Programmation classique	Programmation répartie
Un même langage de programmation (sinon Utilisation de binding) • Même paradigme de programmation • Même représentation des types de base • Même représentation de L'information composite	Deux langages différents • Représentation de l'information composite différente • Association des paramètres effectifs aux paramètres formels? • Comment gérer les différents types de passage de paramètre ? • Paradigmes de programmation différents : qu'est ce qu'un objet pour un langage procédural ? comment gérer les erreurs ?

Modèles d’interaction dans un système distribué

- Les éléments distribués interagissent, communiquent entre eux selon plusieurs modèles possibles:
  - ➔ Client/Serveur
  - ➔ Diffusion de messages
  - ➔ Mémoire partagée
  - ➔ Pair à pair
- Communication:
  - ➔ Envoi de message d’un élément vers un autre élément.
  - ➔ Construire les protocoles de communication correspondant au modèle d’interaction.

Modèles d'interaction dans un système distribué

Modèle Client/serveur



Paradigme client/serveur

- Modèle Client/serveur : deux rôles distincts
  - ➔ Client: demande que des requêtes ou des services lui soient rendus
  - ➔ Serveur: répond aux requêtes des clients
- Interaction

➔ Message du client vers le serveur pour faire une requête.

➔ Exécution d'un traitement par le serveur pour répondre à la requête.

➔ Message du serveur vers le client avec le résultat de la requête.

Paradigme client/serveur

- ❖ Modèle Client/serveur : modèle le plus répandu
- ➔ Fonctionnement simple
- ➔ Abstraction de l'appel d'un service: proche de l'appel d'une opération sur un élément logiciel (interaction de base en programmation)
- ❖ Particularité du modèle
- ➔ Liens forts entre le client et le serveur
- ➔ Un client peut aussi jouer le rôle de serveur (et vice-versa) dans une autre interaction
- ➔ Nécessité généralement pour le client de connaître précisément e serveur (sa localisation , exemple URL du site web)
- ➔ Interaction de type « 1 vers 1 »

Modèles d'interaction dans un système distribué

- Sockets
- Appel de procédure à distance: RPC
- Objets réparties: Java RMI, CORBA
- Web services
- Enterprise JavaBeans(EJB)

Chapitre 2

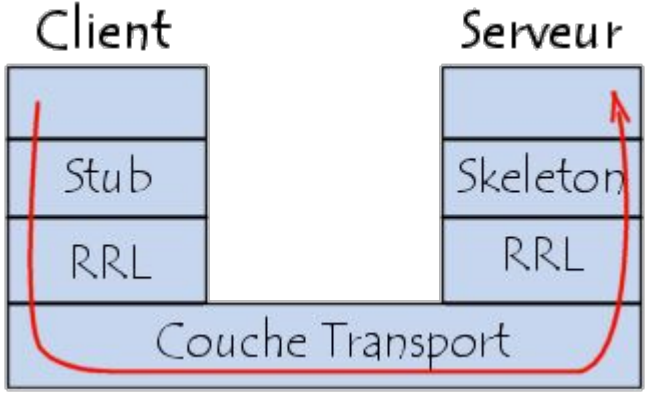
Invocation de méthode à distance en Java

RMI ?

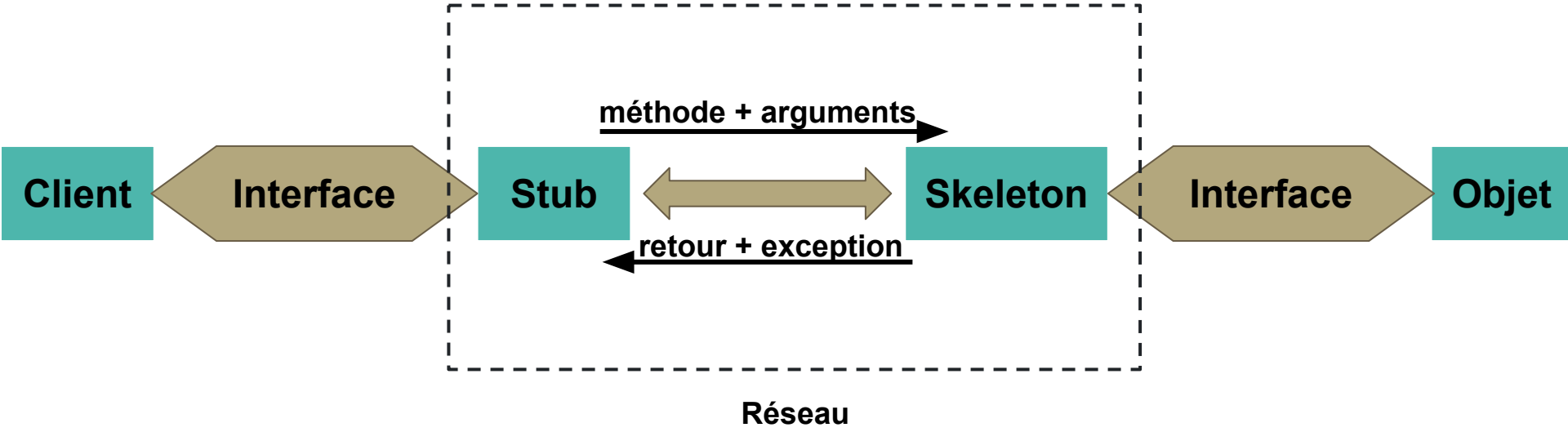
- RMI (*Remote Method Invocation*) est une API Java permettant de manipuler des objets distants (c'est-à-dire un objet instancié sur une autre machine virtuelle, éventuellement sur une autre machine du réseau) de manière transparente pour l'utilisateur, c'est-à-dire de la même façon que si l'objet était sur la machine virtuelle (JVM) de la machine locale.
- Grâce à RMI, un objet exécuté dans une JVM présente sur un ordinateur (côté client) peut invoquer des méthodes sur un objet présent dans une autre JVM (côté serveur). RMI crée un objet serveur distant public qui permet les communications côté client et côté serveur via des appels de méthode simples sur l'objet serveur.

RMI ?

- La transmission de données se fait à travers un système de couches, basées sur le modèle OSI afin de garantir une interopérabilité entre les programmes et les versions de Java.



RMI : Appel distant



RRL ?

- La couche de référence (*RRL*, *remote Reference Layer*) est chargé du système de localisation afin de fournir un moyen aux objets d'obtenir une référence à l'objet distant(permet d’obtenir une référence d’objet distribué à partir de la référence locale au stub).
- Cette fonction est assurée grâce à un service de noms rmiregister (qui possède une table de hachage dont les clés sont des noms et les valeurs sont des objets distants).
- Un unique rmiregister par JVM.
- rmiregister s’exécute sur chaque machine hébergeant des objets distants

Objet stub & skeletoon

- **Le stub** (*souche*) et **le *skeleton*** (*squelette*), respectivement sur le client et le serveur, assurent la conversion des communications avec l'objet distant.
- **Objet stub :** L'objet stub sur la machine cliente crée un bloc d'informations et envoie ces informations au serveur. Le bloc se compose de :
  - ➔ Un identifiant de l'objet distant à utiliser
  - ➔ Nom de la méthode à appeler
  - ➔ Paramètres de la JVM distante

Objet stub & skeletoon

- **Objet *skeleton*** (*squelette*)**:** L'objet squelette réside sur le programme serveur. Il est chargé de transmettre la requête de Stub à l'objet distant.

➔ Il appelle la méthode souhaitée sur l'objet réel présent sur le serveur.

➔ Il transmet les paramètres reçus de l'objet stub à la méthode.

Création d'applications distribuées à l'aide de RMI

- L'utilisation de RMI pour développer une application distribuée implique les étapes générales suivantes :

- ➔ Concevoir et implémenter les composants de votre application distribuée.
- ➔ Compilation des sources.
- ➔ Rendre le réseau de classes accessible.
- ➔ Démarrage de l'application.

Conception et mise en œuvre des composants de l'application (1/4)

- ❖ Tout d'abord, déterminez l'architecture de votre application, notamment quels composants sont des objets locaux et quels composants sont accessibles à distance.Cette étape comprend :
- ➔ Définition des interfaces distantes.
- ➔ Implémentation des objets distants.
- ➔ Implémentation des clients.

Compilation des sources (2/4)

- Comme pour tout programme Java, vous utilisez le compilateur javac pour compiler les fichiers source. Les fichiers source contiennent les déclarations des interfaces distantes, leurs implémentations, toute autre classe serveur et les classes client.

Rendre le réseau de classes accessible(3 /4)

- Dans cette étape, vous rendez accessibles au réseau certaines définitions de classes, telles que les définitions des interfaces distantes et leurs types associés, et les définitions des classes qui doivent être téléchargées sur les clients ou les serveurs. Les définitions de classes sont généralement rendues accessibles sur le réseau via un serveur Web.

Démarrage de l'application(4/4)

- Le démarrage de l'application inclut l'exécution du registre d'objets distants RMI, du serveur et du client.

Création d'applications distribuées à l'aide de RMI

- L'utilisation de RMI pour développer une application distribuée implique les étapes générales suivantes :

- ➔ **Concevoir et implémenter les composants de votre application distribuée.**
- ➔ Compilation des sources.
- ➔ Rendre le réseau de classes accessible.
- ➔ Démarrage de l'application.

Etapes (côté serveur) :

Étape 1 : Définir l'interface distante

La première chose à faire est de créer une interface qui fournira la description des méthodes pouvant être invoquées par les clients distants. Cette interface doit **extends** l'interface **Remote**,le prototype de la méthode dans l'interface doit déclencher **throws** l'exception **RemoteException**.

Exemple

```
//Définir l'interface distante de l’objet distribué
import java.rmi.*;
// Cette interface étend l’interface Remote définie dans java.rmi.
public interface SommeInterface extends Remote
{
    // Déclarer la méthode(prototype)
public int somme(int x, int y) throws RemoteException;
}
```

Etapes (côté serveur) :

Étape 2 : Implémentation de l'interface distante

la deuxième étape consiste à implémenter l'interface distante. Pour implémenter l'interface distante, la classe doit s'étendre à la classe UnicastRemoteObject du package java.rmi. En outre, un constructeur par défaut doit être créé pour lancer l'exception java.rmi.RemoteException à partir de son constructeur parent dans la classe.

Exemple

```
//Programme Java pour implémenter l'interface SommeInterface
import java.rmi.*;
import java.rmi.server.*;
public class ...
{
    // ?
}
```

```
//Programme Java pour implémenter l'interface SommeInterface
import java.rmi.*;
import java.rmi.server.*;
public class Somme extends UnicastRemoteObject implements SommeInterface
{    // Le mot clé super représente l'objet parent,
   // permet d'accéder à une super-méthode, donc à un super-constructeur (le constructeur par défaut)
    Somme() throws RemoteException {super();}
  // Implémentation de l'interface SommeInterface
  public int somme(int x, int y) { return x + y; }
}
```

Etapes (côté serveur) :

Étape 3 : L'écriture d'une classe pour instancier l'objet et l'enregistrer dans le registre

Cette étape peut être effectuée dans la méthode main d'une classe dédiée ou dans la méthode main de la classe de l'objet distant. L'intérêt d'une classe dédiée et qu'elle permet de regrouper toutes ces opérations pour un ensemble d'objets distants..

Etapes (côté serveur) :

Le développement coté serveur se compose de :

❖ La définition d'une interface qui contient les méthodes qui peuvent être appelées à distance

❖ L'écriture d'une classe qui implémente cette interface

❖ L'écriture d'une classe qui instancie l'objet et l'enregistrer en lui affectant un nom dans le registre de noms RMI.

Etapes (côté client) :

L'appel d'une méthode distante peut se faire dans une application

Étape 1 : La mise en place d'un security manager

Comme pour le côté serveur, cette opération est facultative.

Etapes (côté client) :

Étape 2: L'obtention d'une référence sur l'objet distant

Pour obtenir une référence sur l'objet distant à partir de son nom, il faut utiliser la méthode statique **lookup()** de la classe **Naming.**

Cette méthode attend en paramètre une URL indiquant le nom qui référence l'objet distant. Cette URL est composée de plusieurs éléments : le préfix **rmi://**, le nom du serveur (hostname) et le nom de l'objet **tel qu'il a été enregistré dans le registre** précédé d'un slash.

Etapes (côté client) :

Étape 3: Appel de(s) méthode()s distante(s)

Le client détient une référence à un objet distant qui est une instance de la classe de stub. L'étape suivante consiste à appeler la méthode sur la référence. Le stub implémente l'interface qui contient donc la méthode que le client a appelé.

➔ il suffit simplement d'appeler la méthode comme suit :

nom_interface.nom_methode(paramètres)

Ecriture d'une application RMI (récapitulatif) :

1. **Déclaration des services accessibles à distance**

➔ Ecriture d'une interface distante.

2. **Définition du code des services**

➔ Ecriture d'une classe d'objet serveur implantant l'interface.

3. **Instanciation et enregistrement de l'objet serveur**

➔ Ecriture du programme serveur

4. **Interaction avec l'objet serveur**

➔ Ecriture du programme client

Ecriture d'une application RMI (récapitulatif) :

- Dans l'architecture **RMI**, un serveur accessible à distance **est un objet** qui expose différents services via ses méthodes.
- Chaque classe d'objet serveur doit donc implémenter une interface visible depuis l'extérieur.

➔ On parle d'interface distante: seules les méthodes de cette interface pourront être invoquées à distance. Les autres, uniquement depuis la JVM locale au serveur.

- Il s'agit d'une interface Java:

➔ qui doit étendre **java.rmi.Remote**

➔ dont toutes les méthodes doivent lever une exception du type **java.rmi.RemoteException**

Ecriture d'une application RMI (récapitulatif) :

- Afin de spécifier le fonctionnement de l'objet serveur, on écrit une classe **implémentant l'interface distante** définie précédemment.
- Les points suivants doivent être respectés:

➔ Les constructeurs doivent **lever une exception** java.rmi.RemoteException.

➔ Si elle ne possède pas de constructeurs alors **il faut** quand même en déclarer **un vide qui lève une exception : java.rmi.RemoteException.**

➔ De préférence la classe doit étendre **java.rmi.server.UnicastRemoteObject**, ce qui facilitera le déploiement de l'objet serveur dans le runtime RMI.

Le code du serveur (récapitulatif) :

Le code du serveur, est une classe qui dispose d'une méthode **main(...)** qui se charge:

- D'instancier la classe définie précédemment de manière à obtenir l'objet serveur qui fournis les services.
- D'exporter l'instance dans le runtime RMI, le stub est créé lors de cette étape.
- D'enregistrer l'instance (en fait son stub) dans le serveur de nom RMI.

Le code du serveur (récapitulatif) :

```java
package service;

import java.rmi.Naming;
import java.rmi.registry.LocateRegistry;

public class RMIserveur {

    public static void main(String[] args) { //<-- Méthode main()

        try {

            LocateRegistry.createRegistry(1099);
            CalculatriceImplt objetDistant = new CalculatriceImplt();//<-- Instanciation
            Naming.rebind("rmi://localhost:1099/calculatrice", objetDistant);//<-- Enregistrement
            System.out.println(objetDistant.toString());
            System.out.println("Le serveur est prêt !");

        } catch (Exception e) {
            System.out.println("Échec de l'exécution du serveur :" +e);
        }
    }

}
```

Le code du client (récapitulatif) :

Tous les clients RMI vont suivre ce modèle de programmation:

1. Recherche de l’objet serveur dans le serveur de noms RMI.
2. Invocation des méthodes de l’objet serveur.

Appels simultanés

- Un objet serveur RMI est susceptible d'être accédé par plusieurs clients simultanément.
- A plusieurs appels simultanés d'une même méthode peuvent correspondre plusieurs threads concurrentes sur le même objet serveur.
- Dans ce cas il faut concevoir des méthodes RMI "thread-safe", c'est à dire exécutables concurremment de façon cohérente.

Services complémentaires de RMI

Outre sa fonctionnalité principale qui est de permettre la répartition transparente sur un réseau des objets java (mécanisme de communication), On détaille ici 3 services complémentaires de RMI:

- Service de nommage (rmiregistry).
- Service d'activation d'objets à la demande .
- Service de Garbage Collector (gestion de la mémoire).

RMI-Nommage

- Problématique: comment faire en sorte que les objets serveurs soient connus de tous les clients ?
- Solution: utilisation du service de nommage de RMI: le “RMI Registry”.

➔ doit être lancé, une seule fois, avant les programmes clients et serveurs .

➔ disponible par défaut sur le port 1099

➔ Il permet d’enregistrer les liaisons entre un objet serveur et un nom symbolique.

RMI-Nommage

Le registry peut être lancé (serveur):

- ➔ de façon autonome dans un shell avec l'outil Java rmiregistry
- ➔ à partir d'un programme par l'appel de le méthode statique:
  java.rmi.registry.LocateRegistry.createRegistry(int port)

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

LocateRegistry.createRegistry(1099);
```

RMI-Nommage

Le registry peut être accédé (client):

➔ directement via les méthodes statiques de la classe j**ava.RMI.Naming** qui prennent en argument l'url complète du registry (url+port):

[rmi://]machine[:1099]/nomSymbolique

```
CalculatriceInerface stub = (CalculatriceInerface) Naming.lookup("rmi://localhost:1099/calculatrice");
```

RMI-Activation d’objets

…

RMI-Service de Garbage Collector

...

Création d'applications distribuées à l'aide de RMI

- L'utilisation de RMI pour développer une application distribuée implique les étapes générales suivantes :

- **Concevoir et implémenter les composants de votre application distribuée.**
- Compilation des sources.
- Rendre le réseau de classes accessible.
- Démarrage de l'application.

Création d'applications distribuées à l'aide de RMI

- L'utilisation de RMI pour développer une application distribuée implique les étapes générales suivantes :

- **Concevoir et implémenter les composants de votre application distribuée.**
- Compilation des sources.
- Rendre le réseau de classes accessible.
- Démarrage de l'application.

Création d'applications distribuées à l'aide de RMI

- L'utilisation de RMI pour développer une application distribuée implique les étapes générales suivantes :

- **Concevoir et implémenter les composants de votre application distribuée.**
- Compilation des sources.
- Rendre le réseau de classes accessible.
- Démarrage de l'application.

FIN DU COURS