Chapitre II

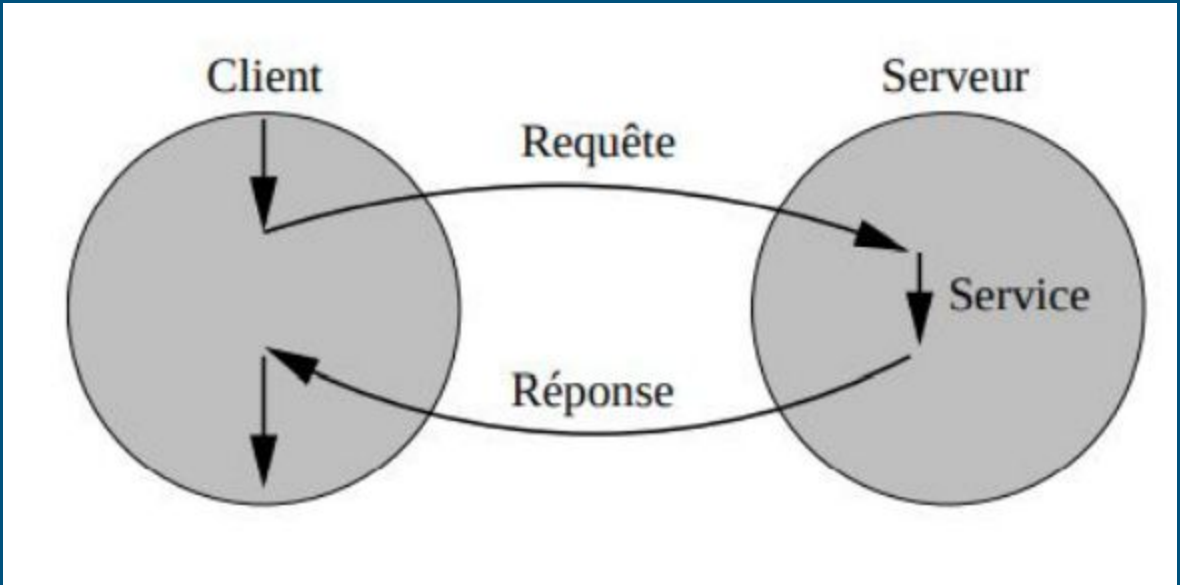
Architectures client / serveur

I. Introduction

Qu’est-ce qu’une architecture client-serveur ?

- Une architecture client-serveur représente l’environnement dans lequel des applications de machines clientes communiquent avec des applications de machines de type serveurs.
- L’exemple classique est le navigateur Web d’un client qui demande (on parle de “requête”) le contenu d’une page Web à un serveur Web qui lui renvoie le résultat (on parle de “réponse”)

I. Introduction



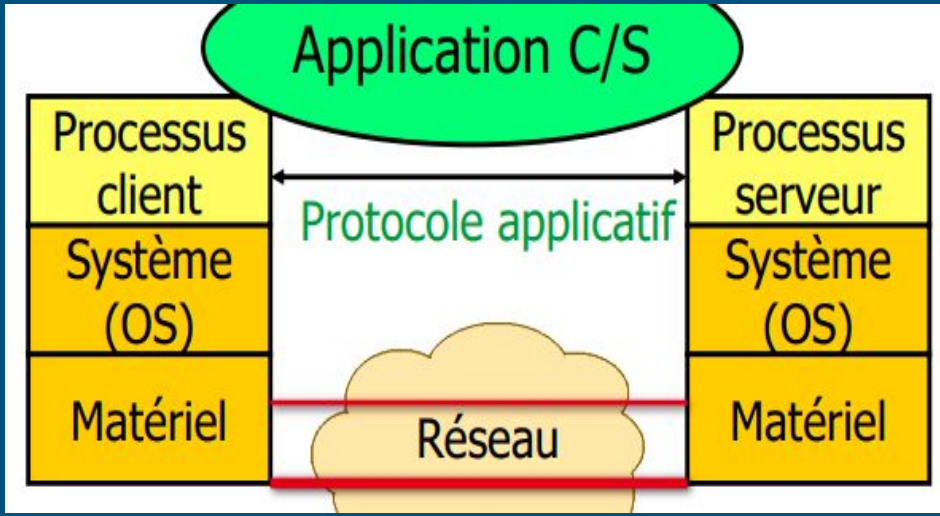
II. Fondement des architectures client/serveur

- **Idée** : l'application est répartie sur différents sites pour optimiser le traitement, le stockage
- **Le client**
  - ➔ effectue une demande de service auprès du serveur (requête)
  - ➔ initie le contact (parle en premier), ouvre la session
- **Le serveur**
  - ➔ est la partie de l'application qui offre un service
  - ➔ est à l'écoute des requêtes clientes
  - ➔ répond au service demandé par le client (réponse)

II. Fondement des architectures client/serveur

- Le client et le serveur ne sont pas identiques, ils forment un système coopératif
  - ➔ **l**es parties client et serveur de l'application peuvent s'exécuter sur des systèmes différents
  - ➔ une même machine peut implanter les côtés client ET serveur de l'application
  - ➔ un serveur peut répondre à plusieurs clients simultanément

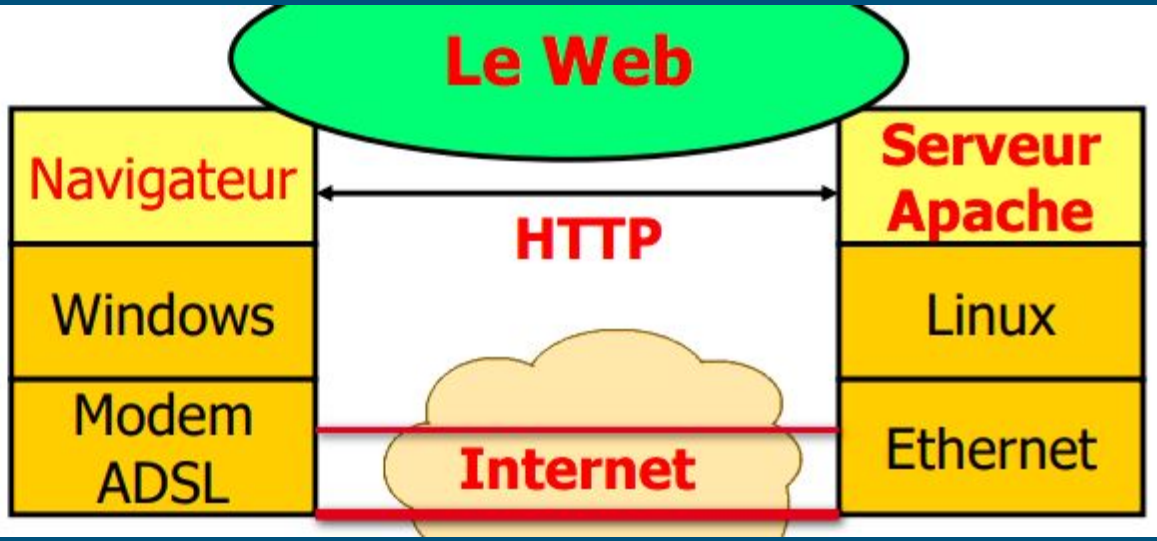
I. Fondement des architectures client/serveur



L'application est répartie sur le client et le serveur qui dialoguent selon un protocole applicatif spécifique

II. Fondement des architectures client/serveur

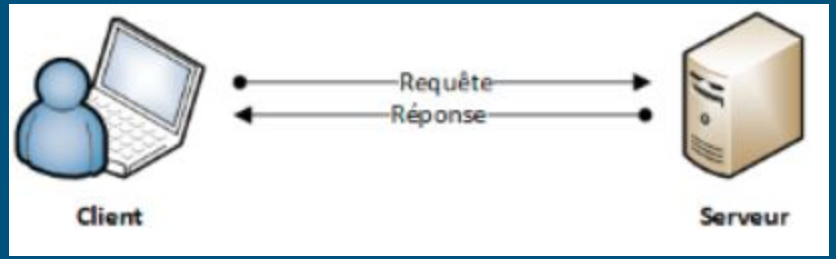
L'exemple du Web



II. Fondement des architectures client/serveur

Les types d'architecture client-serveur

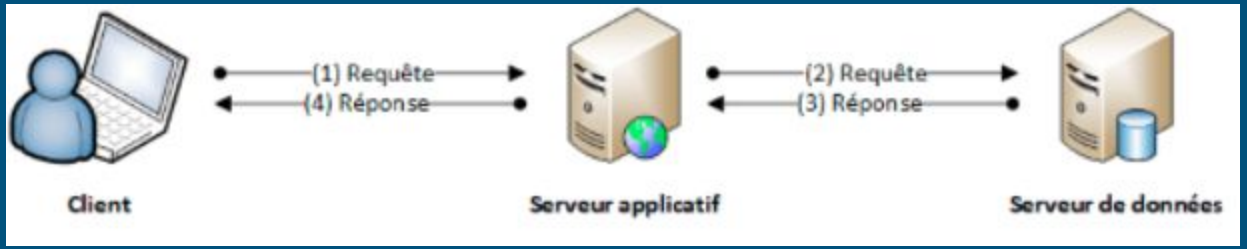
- Si toutes les ressources nécessaires sont présentes sur un seul serveur, on parle d'architecture à deux niveaux ou 2 tiers (1 client + 1 serveur).



II. Fondement des architectures client/serveur

Les types d'architecture client-serveur

- Si certaines ressources sont présentes sur un deuxième serveur (par exemple des bases de données), on parle d'architecture à trois niveaux ou 3 tiers (1 client interroge le premier serveur qui lui-même interroge le deuxième serveur).



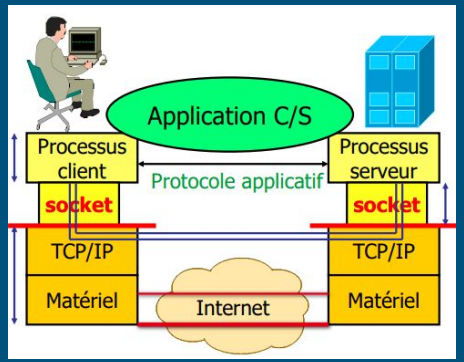
II. Fondement des architectures client/serveur

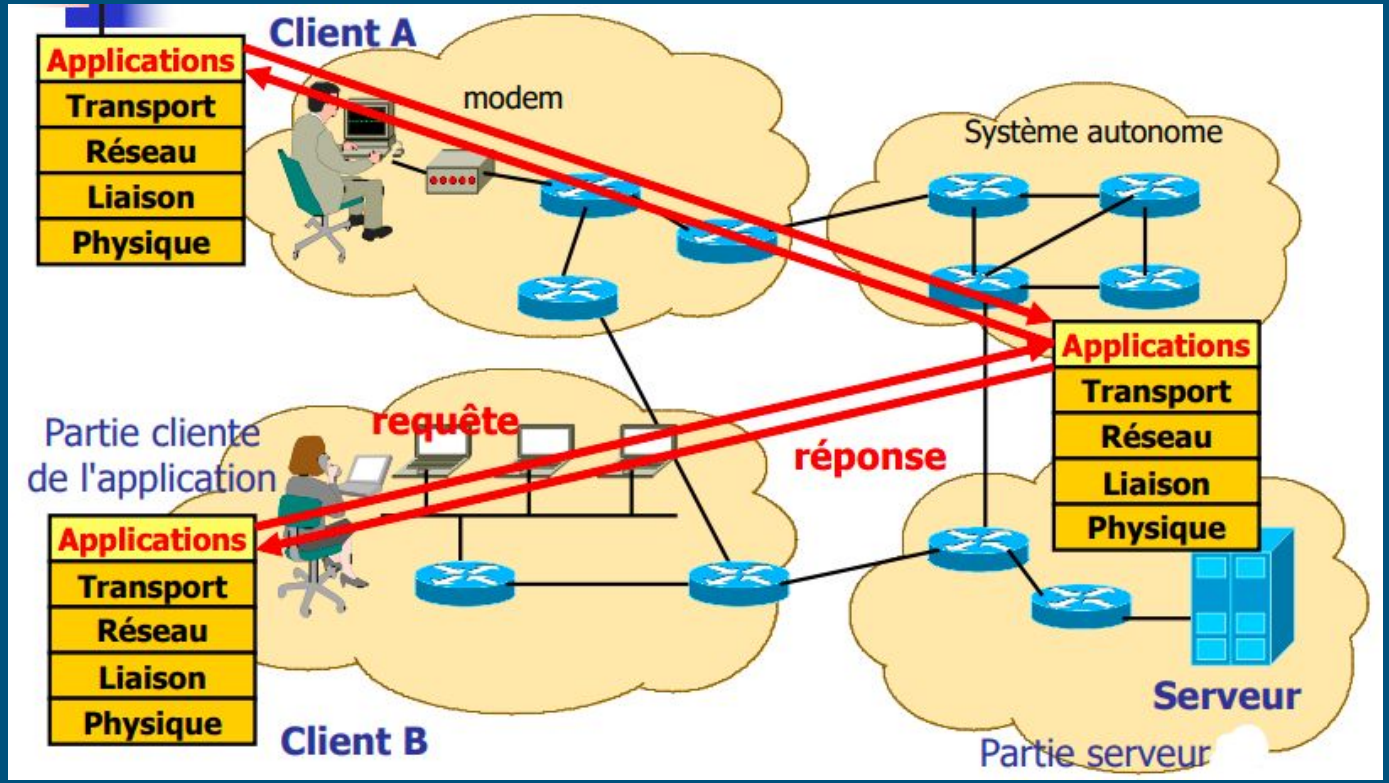
Les types d'architecture client-serveur

- Au delà de 3 acteurs, on parle d'architecture à n tiers.
  - ➔ Architecture physique à 2 tiers mais qui fonctionnera en réalité comme architecture n tiers grâce à la virtualisation.

II. Fondement des architectures client/serveur

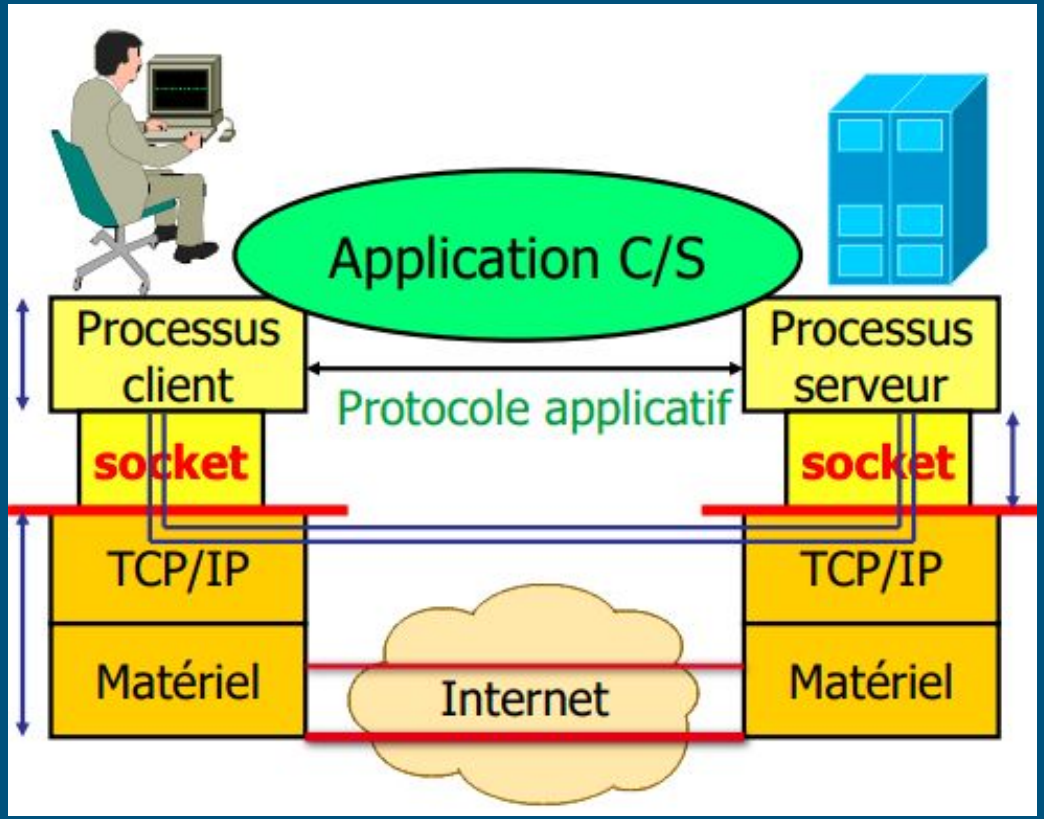
Il faut une interface entre l'application réseau et la couche transport





I. Fondement des architectures client/serveur

Une socket : interface locale à l'hôte, créée par l'application, contrôlée par l'OS.
Porte de communication entre le processus client et le processus serveur



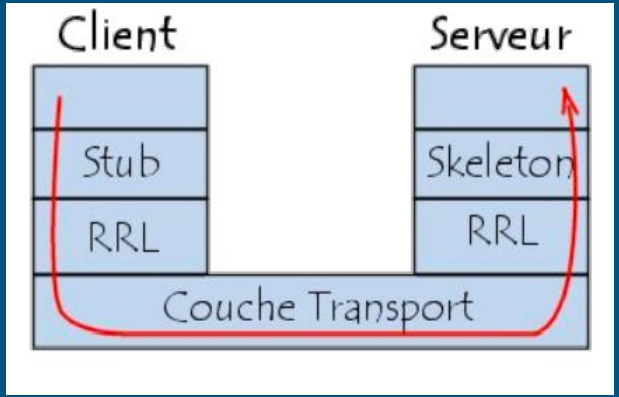
III. Système RMI

- 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.

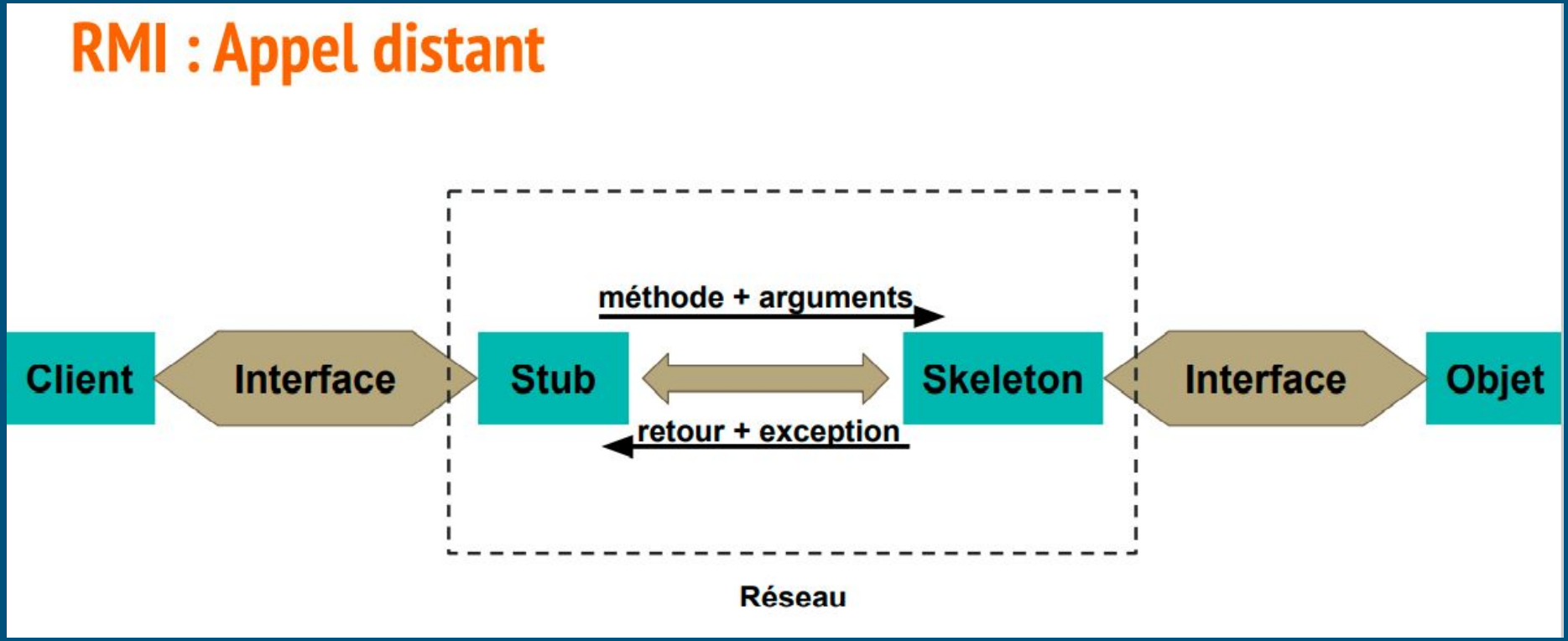
- Grâce à RMI, un objet exécuté dans une JVM présente sur un ordinateur (côté client) peut invoquer des méthodes sur un objet présent dans une autre JVM (côté serveur). RMI crée un objet serveur distant public qui permet les communications côté client et côté serveur via des appels de méthode simples sur l'objet serveur.

III. Système 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



III. Système RMI



III. Système RMI

RRL ?

- La couche de référence (RRL, remote Reference Layer) est chargée 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

III. Système RMI

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

III. Système RMI

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.

III. Système RMI

Création d'applications distribuées à l'aide de RMI, les étapes (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.

III. Système RMI

```
//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;
}
```

III. Système RMI

Création d'applications distribuées à l'aide de RMI, les étapes (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

III. Système RMI

```
//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; }
}
```

III. Système RMI

Création d'applications distribuées à l'aide de RMI, les étapes (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..

III. Système RMI

Création d'applications distribuées à l'aide de RMI, les étapes (côté Client) :

- **Étape 1** : La mise en place d'un security manager. Comme pour le côté serveur, cette opération est facultative.

III. Système RMI

Création d'applications distribuées à l'aide de RMI, les étapes (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.

III. Système RMI

Création d'applications distribuées à l'aide de RMI, les étapes (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 appelée.

  ➔ il suffit simplement d'appeler la méthode comme suit : nom_interface.nom_methode(paramètres)

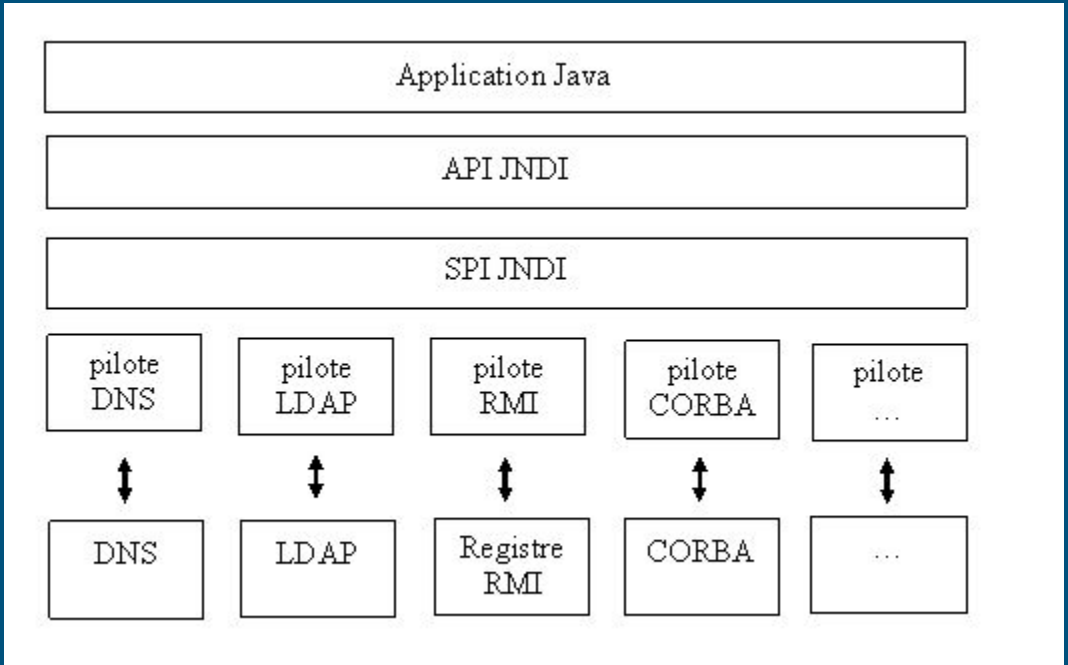
IV. API Java JNDI

JNDI est l'acronyme de Java Naming and Directory Interface. Cette API fournit une interface unique pour utiliser différents services de nommages ou d'annuaires et définit une API standard pour permettre l'accès à ces services.

JNDI est composée de deux parties:

- Une API utilisée pour le développement des applications
- Une SPI utilisée par les fournisseurs d'une implémentation d'un pilote

IV. API Java JNDI



IV. API Java JNDI

https://www.jmdoudoux.fr/java/dej/chap-jndi.htm

Fin

Merci Pour Votre Attention