CAR 5
Séance questions ouvertes

Ce support fourni un condensé des concepts émanants des questions posées par les étudiants par mail pour la séance de questions ouvertes du groupe 5 de CAR.

Il fournit une synthèse de ces concepts pour aider les étudiants dans leur révision pour l'examen, mais ne remplace aucunement le cours associé à ces concepts.

RMI

But

Etre capable d'appeler des méthodes sur des objets Java distants. Implémentation de la notion de RPC (*Remote Procedure Call*) dans l'univers Java.

Pourquoi

Pour communiquer avec des machines distantes en utilisant le paradigme orienté objet.

Comment

Modèle RMI

- On associe chaque classe d'objet présent sur le serveur à une interface qui sera commune entre l'appelant (client) et l'appelé (l'objet)
- Le serveur enregistre l'objet au sein d'un annuaire (`rmiregistry`) qui permettra d'être requêté par le client pour obtenir la référence de l'objet
- Le client communique avec le serveur par le biais d'une souche (`stub`) implémentant l'interface et réalisant les appels réseaux. Cette souche est généré par l'outil `rmic`
- Le serveur implémente l'interface en définissant le code métier. La communication avec le client est assuré par un squelette (`skeleton`). Ce squelette est généré par l'outil `rmic`.

Conditions structurelle d'une interface RMI :
- Etendre `java.rmi.Remote`
- Chaque méthode doit lever une `java.rmi.RemoteException`

Conditions structurelles d'une classe implémentant une interface RMI :
- Étendre `java.rmi.server.UnicastRemoteObject` ou être manipulée via méthode statique `java.rmi.server.UnicastRemoteObject#exportObject()`
- Définissant un constructeur levant une `java.rmi.RemoteException`

En plus

- RMI présent par défaut à partir du JDK 1.1
- RMI utilise TCP comme couche de transport (=> communication sans perte théorique)
- Le transfert des données s'effectue :
  - Par copie pour les types primitifs (`int`, `float`…) et les objets implémentant `java.io.Serializable`
  - Par référence pour les objets implémentant `java.rmi.Remote`
- L'accès à un objet peut être concurrent. Il faut donc le prévoir dans l'implémentation
- Les objets serveur peuvent être activés/désactivés à la demande grâce au démon `rmid` (à partir du JDK 1.2)
- RMI défini son propre *garbage collector* pour prendre en compte le contexte réparti de l'utilisation des références d'objets.
- La génération et l'utilisation des souches clientes et squelettes serveurs sont faites automatiquement à partir du JDK 1.5. Plus de nécessité d'utiliser `rmic`. (d'ailleurs déprécié à partir du JDK 8)

Références TDs

- TD 5 - RMI
- TD 6 - Annuaire (concept du registre RMI et implémentation de la solution en RMI)
- TD 7 - Election sur un anneau (implémentation de la solution en RMI)
- ~~TD 8 - Election contrarotative sur un anneau (implémentation de la solution en RMI)~~ (annulé)
- TD 9 - Répartiteur de charge (implémentation de la solution en RMI)
- TD 11 - Vidéo-club en ligne (implémentation de la solution en RMI)

Annuaire

But

Mettre à disposition un service permettant de rechercher une adresse physique (pointant donc sur une ressource existante) à partir d'une adresse logique. Analogie avec DNS.

Pourquoi

Par exemple dans un contexte RMI, être capable de récupérer l'adresse physique des objets distants à partir d'un nom plus simple à utiliser (`rmiregistry`).

Comment

C'est tout le sujet du TD 6 : développer son propre serveur de nom.

L'idée principale résulte dans le fait que :
- Chaque ressource possède une adresse physique qui lui est propre
- Chaque ressource est associée à une adresse logique (le nom)
- Un registre de nom permet d'ajouter, retirer ou récupérer une adresse physique associée à son nom (adresse logique)

Références TDs

- TD 6 - Annuaire

Election

But

Choisir, au sein d'une application répartie sur plusieurs noeuds, le noeud qui devra exécuter une action.

Pourquoi

Par exemple :
Soit un ensemble de machines (noeuds) reliées entres elles pour former une application répartie. Nous souhaitons sélectionner la machine qui est la moins chargée pour lui exécuter une tâche lourde. Nous souhaitons donc élire la machine qui est la moins chargée parmi toutes celles qui forment notre application répartie.

Comment

Application directe des TD 7 et 8 : exemple où les machines sont réparties selon une topologie en anneau.
On souhaite répondre à la question : comment pourrions-nous réaliser une élection en se basant sur cette topologie ?
On se rend compte que l'algorithme d'élection se base sur deux critères :
- Le test qui nous permettra de choisir la machine (ici l'identifiant le plus élevé)
- La manière dont les machines sont reliées entres elles (ici en anneaux unidirectionnels ou bidirectionnels)

Tout l'enjeu des TD 7 et 8 est de concevoir cet algorithme

Références TDs

- TD 7 - Election sur un anneau
- ~~TD 8 - Election contrarotative sur un anneau~~ (annulé)

Serialisation/Deserialisation

But

Importer ou exporter des données structurées selon un format défini à l'avance.

Pourquoi

Utilisé principalement pour importer ou exporter des données exprimées selon un langage de programmation choisi. On évite ainsi le traitement technique de l'importation ou l'exportation des données mais on utilise plutôt les services offerts par notre langage de programmation.

Par exemple en Java : sauvegarder en base de données un objet (i.e., son état actuel) Java. Ou transmettre un objet Java entres deux machines.

Comment

Utilisation directe du TD 4 : étude du format de serialisation et de déserialisation de données Java (*Object Serialization Stream Protocol*)

- Le format est défini au sein d'une grammaire
- La grammaire définie une structure différente pour les données primitives et les données objets
- La grammaire définie une entête commune pour toute donnée Java (`magic`, `version`)
- Une attention particulière doit être portée au type architectural physique de stockage des données de la machine (petit boutiste / grand boutiste)

Références TDs

- TD 4 - Représentation des données