TD/TME 9 : Compléments RMI

Ce TD a pour objectif de poursuivre ce qui a été entrepris lors du TD précédent avec l'introduction aux concepts de la programmation distribuée en Java (à l'aide du mécanisme de sérialisation et de RMI). Dans ce TD, l'exemple de la calculette est repris pour aborder 2 nouvelles notions pour RMI : la création d'un service de nommage dynamiquement ainsi que le mécanisme de rappel qui permet des communications asynchrones entre le client, demandeur du résultat d'une méthode sur un objet distant, et le serveur d'objets distants.

Exercice 1 – Lancer un service de nommage par programme

Le mécanisme de rappel, étudié plus après dans ce TD, repose sur des objets enregistrés coté client où ne tourne pas nécessairement de service qui enregistre les objets distants (dans le cas où le client et le serveur sont sur des hotes distincts). Les classes `Registry` (pour le serveur) et `LocateRegistry` (pour un client) permettent de gérer des objets sans lancement *externe* du registre de nommage (le lancement se fait au sein même du programme). On suppose que le service de nommage (registre des noms d'objets distants) n'a pas été lancé.

Question 1

Ecrire `ServAutonomeCalculette` capable de :

- créer un registre de nommage
- créer et d'enregistrer une calculette appelable à distance
- l'enregistrer auprès du service de nommage (registry) sous le nom `calc`.

Question 2

Puis écrire un client `ClientPourServAutonome`, capable de :

- obtenir une référence distante auprès du service de nommage
- de réaliser quelques calculs avant de terminer

Exercice 2 – Le Pattern `RemoteCallBack`

Les appels effectués par le biais de RMI peuvent etre long. Il est souhaitable pour un client de continuer à être réactif même en attendant la réponse d'une `Remote` méthode. Ainsi on souhaite qu'un serveur puisse appeller des `callback` sur le client pour le notifier d'évènements.

Afin d'implanter ce mécanisme de rappel(callback), il est nécessaire de définir un objet de rappel coté client dont une remote reference fournie au serveur lui permttra d'interagir avec le client. En d'autre termes cet objet de rappel est crée par le client comme objet distant et sert d'intermédiaire, *de messager*, portant la réponse du serveur au client. L'intérêt d'un tel mécanisme est de permettre de rendre la communication *asynchrone*. L'objet de rappel dispose typiquement de 5 méthodes :

- Deux pour le client : `boolean is_finished()` qui permet de savoir si le calcul est terminé et `TypeResultat getResult()` qui permet ensuite de récupérer le résultat.
- Trois pour le serveur : `void put(TypeResultat)` qui dépose le résultat à la fin du calcul `void finish()` qui indique que le résultat vient d'être déposé et prévient tous les threads en attente éventuelle du résultat `unreferenced` pour la gestion explicite des références d'objets distants, remplaçant en quelque

sorte le ramasse miette (GC) local L'interface `IRemoteCalculette` et la classe `RemoteCalculette` demeurent inchangées.

Question 1
La calculette est désormais considérée comme une sorte de caisse enregistreuse. A ce titre, tout nombre fourni à la méthode `cumuler` est stocké dans un vecteur (`class java.util.Vector`). Modifier la calculette pour que l'opération `getCumul` devienne *longue* (au sens de prendre beaucoup de temps).

A partir de maintenant nous allons définir un `Remote` objet sur lequel une `Remote` référence pourra être donnée à une autre jvm. Ca sera lui notre objet de callback.

Question 2
Ecrire la classe `RappelResRMI`, décrite plus haut, destinée à porter le résultat de la méthode `getCumul`, cette classe sera instanciée coté client et aura deux membres prives :

— `result`, un entier qui sera le resultat.
— `f` , un booleen qui sera positionné a vrai lorsque le calcul cote serveur sera terminé.

Ecrire Maintenant l'interface `IRappelResRMI` destinée à être transmise au serveur par le client (On prendra bien soin de n'exporter que les méthodes *vraiment nécessaires*)

Question 3
Créer une nouvelle interface `IRappelCalculette` ajoutant la methode `getCumulR (IRappelResRMI)` - pour passer l'objet temporaire de rappel- et créer aussi une nouvelle classe `RappelCalculette` implantant cette interface. Les clients distants de RappelCalculette appellent maintenant méthode `getCumulR (IRappelResRMI)` pour obtenir le résultat du cumul (et non plus la méthode `getCumul()`).

Question 4
Pour que le client puisse, en théorie, réaliser d'autres tâches sans attendre la réponse calculée par le serveur d'objet, la méthode `getCumulR` va démarrer un *thread* `TcalculCumul` qui prend en charge le travail. Pour ce faire, il devra nécessairement avoir accès à la `calculette` ainsi qu'au messager `IRappelResRMI` (qui lui seront donc passés en paramètre). Ecrire la classe de `Thread TcalculCumul` dont la méthode `run` :

— s'endort un peu (pour rallonger artificiellement l'opération)
— appelle `getCumul()` sur l'objet réel du serveur
— définit le résultat et marque l'opération comme terminée dans le messager coté client (par l'intermédiaire de son représentant coté serveur : méthode `RappelResRMI.finish()`)

Question 5
Quelle(s) modification(s) apporter dans le serveur ? Ecrire le serveur nommé `ServRappelCalculette.java`.

Question 6

Modifier le client pour appeler `getCumulR()` et attendre que le résultat soit disponible, c'est-à-dire que `isFinished()` du messager renvoie vrai. On portera une attention particulière au fait que l'attente active doit être évitée (sinon le mécanisme de rappel n'est pas utile).

Exercice 3 – en TME

Question 1
Reprendre l'exercice 2 en rajoutant un autre objet `echo` (de type `Remote`) qui possède une méthode `String echoChaine (String chaine)`. Cette dernière renvoie la chaîne concatenée avec elle même. On souhaite que cet objet est accessible sur le même port (avec un nom distinct de celui de la calculette).

On suppose que la méthode `String echoChaine (String chaine)` du serveur ci-dessus a besoin d'un autre objet `echoSuite` (de type `Remote`) appartenant à un 2ème serveur et qui possède une méthode `String echoChaineSuite (String chaine)`.

Question 2

Réecrire la méthode `String echoChaine (String chaine)` ci-dessus.

Question 3

Définir cet nouvelle objet `echoSuite` et lancer un 2ème serveur sur un autre port (sur la même machine) que le premier pour servir cet objet.