TP 1-HAI704I : Architectures Distribuées

Java RMI
Références distantes, sérialisation, téléchargement de code

Objectifs du TP :
- comprendre la notion d'objet distribué,
- utiliser le passage de stub ou d'objet sérialisé,
- utiliser le téléchargement de code via le codebase

On se place dans la cadre d'un cabinet vétérinaire. Chaque patient du cabinet (les animaux) possède une fiche avec un dossier de suivi. Chacun des vétérinaires du cabinet peut accéder aux fiches. On a donc un serveur de fiches et des programmes client pour les vétérinaires.

Il ne s'agit bien évidemment pas de réaliser de manière réaliste une telle application, mais juste de s'en servir comme support pour atteindre les objectifs du TP.

1 Une première version simple

1. On commence par mettre en place une version de base sur laquelle on va pouvoir travailler. On écrira une classe animal qui sera distribuée, et qui ne contiendra que des types simples (avec par exemple un nom, le nom du maître, l'espèce et la race sous forme de chaînes, ...), et des méthodes simples permettant de tester l'application (affichage et obtention du nom complet de l'animal – nom + nom du maître –, ...). On écrira ensuite une classe serveur qui créera un animal, et le rendra distribué, ainsi qu'un client qui récupérera l'animal en question. Dans un premier temps, on travaillera en local, sur le port par défaut, et sans gestionnaire de sécurité.
2. Ajouter un gestionnaire de sécurité.
3. Rajouter à l'animal son dossier de suivi, et donc une classe représentant le dossier de suivi (qui pourra être très simple : une simple chaîne de texte suffira pour les besoins de l'exercice). Modifier l'application de manière à ce que le client puisse récupérer et modifier le dossier de suivi.
4. L'espèce d'un animal devient une classe (contenant par exemple le nom de l'espèce et la durée de vie moyenne). On veut pouvoir consulter l'espèce d'un animal, mais pas la modifier. Modifier l'application en conséquence (**on transmettra des copies d'espèces et pas des références distantes**).

2 Classe CabinetVeterinaire

Plutôt que notre serveur distribue les animaux, modifier l'application pour avoir une classe distribuée représentant le cabinet vétérinaire, qui connaît sa collection de patients. Le serveur (*main* du cabinet vétérinaire ou *main* d'une classe à part) se contente donc de distribuer le cabinet vétérinaire. On écrira une méthode de recherche d'animal par nom (par exemple) pour tester cette version.

3 Création de patient

Il est nécessaire que depuis la partie cliente de l'application, on puisse ajouter de nouveaux patients au cabinet vétérinaire. Écrivez le code nécessaire et testez-le.

4 Téléchargement de code

Il est assez raisonnable de penser que le client et le serveur ne s'exécutent pas sur la même machine : ailleurs sur un réseau local, voir sur un site distant. Au cours de ce TP, on ne testera pas le cas où l'on doit passer par http, on restera donc sur le même NFS.

On va ici faire en sorte que le serveur ait à télécharger des classes du client.

Imaginez un scénario qui impose un tel téléchargement. Mettez en place ce scénario sans utiliser le mécanisme de codebase, et constatez l'erreur obtenue (vous devez obtenir une erreur de chargement de classe). Mettez ensuite en place le codebase pour résoudre le problème. Attention, pour utiliser ce mécanisme, il est indispensable de mettre en place un gestionnaire de sécurité.

5 Alertes

On souhaite intégrer à l'application un mécanisme qui alerte automatiquement tous les vétérinaires (les clients) d'un même cabinet vétérinaire quand on franchit à la hausse ou à la baisse les seuils de 100, 500 et 1000 patients. Ici, cette alerte se traduira chez les clients par une écriture sur leur sortie standard.

Effectuez les modifications nécessaires pour mettre en place ces alertes.