SCIENCES SORBONNE UNIVERSITÉ

SRCS : Systèmes Répartis Client/Serveur

TME 8 : Déploiement et migration de services RMI

Objectifs pédagogiques

- Serveur de service RMI
- Déploiement distribué

Dans ce tme nous souhaitons pouvoir déployer sur un ensemble d'**hôtes** des **services** invocables à distance. Un service se résumera par l'appel à une fonction.

Un **service** est un objet distant qui doit offrir les fonctionnalités suivantes :

- `getName` : renvoie une chaîne de caractères égale au nom du service. On considérera que ce nom identifie de manière unique le service dans le système.
- `invoke` : la méthode permettant d'invoquer le service. Elle prendra comme seul argument, un objet quelconque transmis par copie qui permettra de paramétrer l'invocation. Le résultat de l'invocation devra être également un objet quelconque transmis par copie à l'appelant.
- `migrateTo` : permet de migrer le service vers un hôte passé en paramètre. La migration doit déployer sur l'hôte une copie de l'instance courante au moment de l'appel à `migrateTo`. Durant le processus de migration, les requêtes sur les autres méthodes de l'instance courante doivent être mises en attente. Une fois la migration terminée, la copie devient l'objet sur lequel les traitement opèrent et l'instance courante se comporte comme un mandataire vis vis de la copie. Les futurs appels à `invoke`, sur l'instance courante doivent donc être relayé vers la copie. Si on invoque la méthode `migrateTo` sur une instance qui a déjà été migrée, alors une exception est jetée.

Un **hôte** est un objet distant qui gère et héberge plusieurs services. Il doit offrir les fonctionnalités suivantes :

- `deployNewService` qui prend un nom de service et une classe qui étend service en argument. Elle permet d'instancier une nouvelle instance d'un service grâce à la reflexivité java et de renvoyer une référence distante sur le service instancié. Si le nom de service existe déjà sur l'hôte, une exception devra être jetée.
- `deployExistingService` qui permet de déployer sur l'hôte un service à partir d'une instance de service déjà existante qui sera passé par copie. La méthode transforme le service en objet distant et renvoie la référence distante ainsi créé.
- `undeployService` qui pour un nom de service donné permet de désactiver le service associé afin qu'il ne soit plus accessible à distance. Si le nom passé en paramètre désigne un service inexistant, la méthode renverra faux, vrai sinon.
- `getServices` qui renvoie la liste des noms des services existants (déployés ou pas) sur l'hôte.

Question 1

Dans le package `srcs.rmi.service`, écrire l'interface `Host` décrivant les méthodes que doit offrir un hôte.

Question 2

Dans le package `srcs.rmi.service`, écrire l'interface `FunctionService` décrivant les méthodes que doit offrir un service. Cette interface possédera deux variables de type P et R qui correspondront respectivement au type de l'objet paramètre de l'invocation et au type résultat de l'invocation. Assurez vous que ces types soient compatibles avec des appels de méthode RMI.

Question 3

Dans le package `srcs.rmi.service`, écrire une classe `HostImpl` implémentant l'ensemble des méthodes de l'interface `Host`.

Question 4

Dans le package `srcs.rmi.service`, écrire une classe abstraite `AbstractFucntionService<P,R>` qui implante l'ensemble des méthodes de l'interface `FunctionService` y compris la méthode `invoke`. Cette méthode appellera la méthode abstraite `R perform(P param)` proposée par la classe et qui sera implantée ultérieurement par la classe qui implante le service. L'avantage de définir une méthode `invoke` qui appelle une méthode `perform` est de pouvoir effectuer traitement avant et après l'invocation de la méthode du service.

Question 5

Afin de pouvoir utiliser les fichiers de tests unitaires JUnit4 qui vous sont fournis, il est nécessaire de programmer une classe mère `SystemDeployer` dans le package `srcs.rmi.service.SystemDeployer` sur le même principe que le TME précédent. Écrire cette classe.

Question 6

Faire en sorte que les classes `ServiceSansEtatTest` et `ServiceAvecEtatTest` compilent s'assurer que le test est correct.