TP Réseau social avec Java RMI

Giuseppe Lipari

27 février 2017

Table des matières

1 TP 4

Il s'agit de reprendre le TP fait en COO, et en faire une application distribué en utilisant le framework Java RMI.

Pour rappel, le sujet de COO est disponible en pièce jointe.

Vous devez transformer l'application déjà développé dans une application client/server. Contrairement au sujet de COO, l'application client/server doit permette d'envoyer et de recevoir les messages en différé **et** en temps réel.

Également, les opérations de gestion des utilisateurs (ajouter/retirer un amis, ajouter/retirer à/de un groupe, etc.) sont effectué en différé **et** en temps réel.

Pour réaliser le programme, il convient de travailler par étapes successives.

1.1 Première étape : la mini-chat

D'abord, on réalisera une application à part, qu'on appelle la *mini-chat*. Il s'agit d'un programme qui permet d'échanger des simple messages entre plusieurs utilisateurs. Ce programme vous permettra de vous entraîner sur les bases de la programmation RMI.

Un serveur avec un adresse bien connu fourni le service de connexion à la *salle de discussion*. Un utilisateur peut :

— S'enregistrer sur la chat en se connectant au serveur, avec "login"/"mot de passe" ;
— Envoyer des messages, qui seront reçus par tous les clients connectés ;
— Recevoir les messages des autres clients ;
— Se déconnecter du serveur.

Quand un utilisateur se connecte ou se déconnecte, tous les clients connectés en sont informés.

On se concentre d'abord sur la structure de base de l'application partagée. Et donc, pour cette petite application, on ignore les requis du projet COO : tous le client envoie tous les messages à tous les autres clients connectés.

1. Définir les interfaces des classes `ChatRoom`, `Client`, `Message`. Quels classes implémentent l'interface `Remote` ? Quels classes implémentent l'interface `Serialize` ?

   Les interfaces sont déclaré dans un *package* Java à part. Ce package sera utilisé par le client et par le serveur.

2. Définir quel objets doivent être enregistrés dans l'annuaire. Est-ce que on doit enregistrer la `ChatRoom` ? Les `Client` ? Les `Messages` ?
3. Est-ce que on a besoin de synchroniser les données ? Lesquels ?
4. Écrire le code de l'application server, puis de l'application client, et donner un simple exemple de fonctionnement. On s'intéresse au mécanisme de communication, donc une interface textuelle pour l'utilisateur sera suffisante.
5. Écrire des tests pour vérifier le bon fonctionnement des cas d'utilisation suivants :
   — un utilisateur ne peux pas se connecter avec un mauvais login/mot de passe ;
   — un utilisateur se connecte correctement avec le bon login/mot de passe ;
   — un utilisateur déjà connecté ne peut pas se connecter un deuxième fois avec un client différent ;
   — un message envoyé par un utilisateur est bien reçu par tous les autres ;
   — après déconnexion, un client ne reçois plus aucun message ;
   — un utilisateur inactif pour plus de 30 seconds est déconnecté automatiquement.

Vous avez **une séance** de TP pour rendre cette application. Le rendu se fait sur Moodle comme d'habitude.

1.2 Deuxième étape : application complète

Maintenant vous devez modifier votre projet de COO pour créer une application repartie. Vous utiliserez toujours une architecture client/serveur, avec un serveur centralisé : tous les client communique que avec le serveur (pas de *peer-to-peer*).

Contrairement au projet COO, l'utilisateur peut recevoir les messages en temps réel. Lors de la réception d'un message, le serveur notifie le message à tous les utilisateurs qui sont connecté. En plus, lors de la connexion d'un utilisateur, le serveur lui envoie tous les messages reçus pendant qu'il était hors ligne.

1. Définir les interfaces complets de l'objet "serveur" et des objets "client".
2. Séparez les classes présent dans votre projet COO :

   - Lesquels font partie du "serveur" et lesquelles font partie du client ?
   - Est-ce que il y a des classes qu'il faut couper en deux ?
   - Est-ce que il faut ajouter des classes supplémentaires ?
3. Contrairement à l'application *mini-chat*, les utilisateurs ne reçois pas tous les messages. Au contraire, un utilisateur reçois que les messages *directs*, et le messages des *groupes* dont il est membre. Concevoir une méthode dans le serveur pour gérer de manière propre les notifications.
4. Coder l'application en réutilisant le maximum possible de code de votre projet COO.
5. Suite de tests :
   - modifier, si nécessaire et si possible, les tests déjà codés pour votre projet COO, pour qu'ils puissent valider aussi votre application partagée.
   - Ajouter les tests suivants :
     - un utilisateur déjà connecté ne peut pas se connecter un deuxième fois avec un client différent ;
     - après déconnexion, un utilisateur ne peut plus recevoir aucun message ;
     - un utilisateur inactif pour plus de 30 seconds est déconnecté automatiquement ;
     - après connexion, le client est notifié de tous les messages reçus pendant que l'utilisateur était hors ligne ;
     - un utilisateur est notifié que de messages directs, ou des messages des groupes dont il est membre ;
     - le client peut toujours récupérer un message reçu dans le passé (sauf les message qui se détruisent automatiquement).

Faites bien attention à structurer votre code en "couches". Il faut toujours séparer le code "application" du code "réseaux" : le premier doit être réutilisable au maximum lors qu'on changera de framework (voir prochain TP sur HTML + REST + Ajax).

Vous avez **2 séances** de TP pour rendre l'application complète.