Projet SocialNetwork

Matthieu Jan - Info2

River

Présentation générale

L'application de River Javaspaces au projet Socialnetwork pour gérer les recommendations se compose de : - La réalisation d'un RiverTalker (à l'image de OpenJmsTalker) pour faciliter l'ecriture de certaines transactions - La réalisation d'un JSRecommandationManager répondant à l'interface RecommandationManager - Une classe ListEntry symbolisant nos entrées

De plus, quelques adapations ont été faites pour intégrer River : - Rajout d'une méthode startRiver dans ServerMain - Ajout de l'objet JSRecommandationManager dans le CommunicationFactory

Fonctionnement

Bien le système se porte plus à de petite entrée, j'ai fait le choix de stocker des listes, toutes aux même format, pour gérer les données. - Une liste d'ami d'un utilisateur, pour stocker ses amis, et retrouver quels sont les nouveaux ajouts. - Une liste d'interet d'un utilisateur, pour stocker ses interets, et retrouver quels sont les nouveaux ajouts. - Une liste d'utilisateur interessé à un sujet, afin de créer une sorte de "liste d'ami de l'interet"

Ensuite, on guette les mises à jours sur les listes qui nous interessent. En effet, lorsqu'un on met une liste à jour, on l'enlève puis on la remet. C'est cette remise qui va activé les listeners. On compare simplement les nouvelles listes avec notre liste d'ami, puis on emet les recommendations.

Ce système est plus lourd qu'avec des simples couples, mais est le seul à pouvoir être vraiment exhaustif sur la version de River que nous avons. L'absence de transaction ne permet pas de faire des recherches correctement (il faut enlever puis remettre les données, activant tout les listeners à chaque recherche).

Axis2

Présentation générale

L'application d'Axis2 au projet Socialnetwork pour gérer le stockage et la récuperation se compose de : - La réalisation d'un ImageManager côté client - La réalisation d'un ImageService côté serveur.

De plus quelques adaptations légères ont été faites pour intégrer Axis2 : - Rajout d'une

méthode startAxis() dans ServerMain - Ajout de l'objet ImageManager dans le CommunicationFactory

Fonctionnement

- Lorsqu'un utilisateur ajoute une photo dans une publication, celle ci est transformé en bytes (via ImageIO et ByteArrayOutputStream), puis encodé en base 64 et finalement transmise à l'ImageService
- Lorsque le service reçoit une image, il la stocke dans une table lié à un token, qu'il renvoit à l'utilisateur.
- Lorsqu'un utilisateur reçoit une publication avec photo, il fait appel au service avec le token fourni, et le service lui renvoi la chaine en base 64, qu'il décode et affiche.

OpenJMS

Présentation générale

L'application d'OpenJMS au projet Socialnetwork pour gérer les demandes d'amis et l'envoi / publication de message se compose de :

- La réalisation d'un OpenJmsTalker se chargeant du maintient des sessions et de la communication.
- Les implémentations de FriendManager et MessageManager
- La réalisation de deux lanceurs d'évenements, FriendEventCaller et MessageEventCaller pour fire des events
- Une classe JmsType dans le pacckage common pour faciliter l'écriture des messages.

De plus, quelques adaptations on été réalisé auprès du projet pour intégrer OpensJms :

- Les méthodes logins et logout de RmiProfileManager (côté client) appellent les méhtodes liés d'OpenJmsTalker
- La méthode register de RMIProfileManagerRemoteImpl (côté server) créé les Destinations associés à l'utilisateur.
- La classe ServerMain démarre le serveur openjms

Fonctionnement

- Lorsqu'un utilisateur s'enregistre auprès du serveur, celui ci lui créé une compte sur le serveur openjms, une queue privée (format email+priv) et un topic public (format email+pub) qui seront les destinations qu'il utilise pour communiquer avec ses petits camarades.
- Lorsqu'un utilisateur se connecte, OpenJmsTalker créé une session qu'il va conserver jusqu'à la déconnexion.
- Lorsqu'un utilisateur se connecte, il se met à écouter sur sa propre queue privée.
- Lorsqu un utilisateur souhaite communiquer avec un autre, il dépose un message sur la queue privée du destinataire
  - Le destinataire récupere le message, puis le traite.
- Lorque deux utilisateurs sont amis, ceux ci écoutent les files publiques de l'autre.
  - Lorsque l'un publie un message, il le dépose sur sa file publiques, écouté par ses amis.
- Lorsqu'un utilisateur se déconnecte, OpenJmsTalker ferme la session.

RMI

Présentation générale

L'application de java RMI au projet Socialnetwork se compose de : - Une implémentation de ProfileManager avec : - Une interface commune - Une implémentation sur le serveur sous forme de RemoteServer - Une implémentation sur le client qui va faire des appels au serveur (via l'interface commune) - Une implémentation de RMICallback avec : - Une interface client, associé à une implémentation client - Une interface serveur, associé à une implémentation serveur - Un main pour le client qui lance le GUI - UN main pour le serveur qui crée les deux RemoteServer utilisé

Termes

Une rapide liste des termes utilisés - Une classe XXXImpl est une implémentation de l'interface XXX - Une classe RMIXXX est une classe réalisé pour la partie 1 (qui nous concerne ici)

Main(s)

Présentation des lanceurs des applications

ServerMain

Initialise le Registre de RMI, en lui associant un objet ProfileManagerRemoteImpl et un objet RMICallbackServerImpl

ClientMain

Initialise le Socialnetwork object et la frame

ProfileManager

RMIProfileManagerRemote (common)

L'interface reprend les méthodes du packages logic, permettant ainsi d'avoir une communication qui ressemble au fonctionnement et soit facile à utiliser.

RMIProfileManager(client)

Implémente directement le ProfileManager du package logic. Ces méthodes adaptent quelques éléments (notement les callback) mais font essentiellement des appels aux méthodes de l'objet RMIProfileManagerRemote instancié sur le serveur

RMIProfileManagerRemoteImpl(server)

Implémente l'interface de common, et est utilisé par le client qui l'appelle pour réalisé ses actions (signup, login ...).

Il stocke plusieurs Hashmap gérer : - les enregistrements - les connexions - les mots de

passes.

Il fait ensuite certaines opérations pour assurer le bon comportement du système (vérifié l'unicité, la validité des tokens ...). Il utilise l'objet callback pour communiquer les modifications de profile aux utilisateurs.

RMICallback

RMICallback fonctionne en deux parties et en trois temps : Deux parties : - Un RMICallbackClient, côté client - Un RMICallbackServer, côté serveur

Trois temps : - Le serveur met a disposition un objet RMICallbackServerImpl qui va permettre d'enregistrer des RMICallbackClient - Le client enregistre un objet RMICallbackClientImpl auprès du serveur - Lorsqu'un update est fait, le serveur invoque les objets enregistrés pour faire les mise à jour.