

Rapport du projet Twitter en Java

Ye SUN (SI4 IMAFA)

Ying JIANG(SI4 Groupe3)

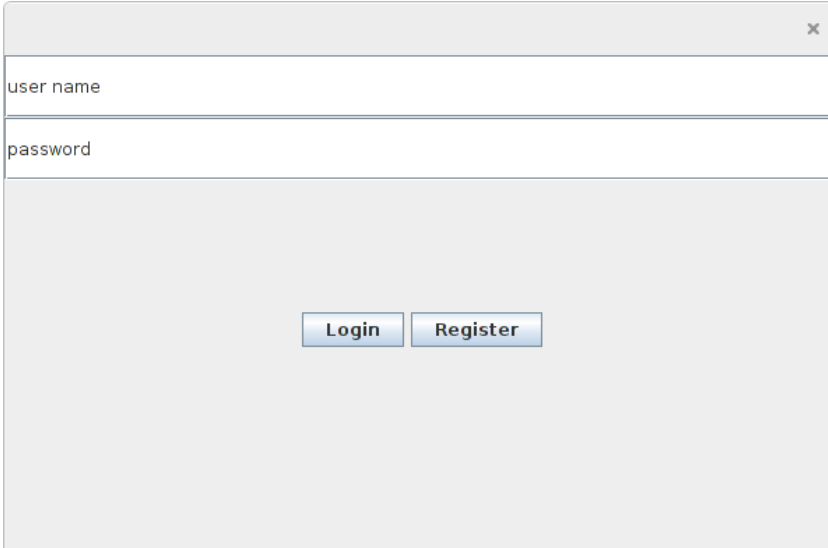
Introduction

Le but de cette application répartie à pour faire un système simplifié ressemblant à twitter. Soit un réseau comme Twitter offre la possibilité à des clients, eux-mêmes remote, et ayant un compte Twitter, d'écrire des petits messages ou simplement d'en relayer (*retweeter*) Nous utilisons RMI pour la connections d'utilisateur, et JMS pour la diffusion des tweets postés.

Interface d'application

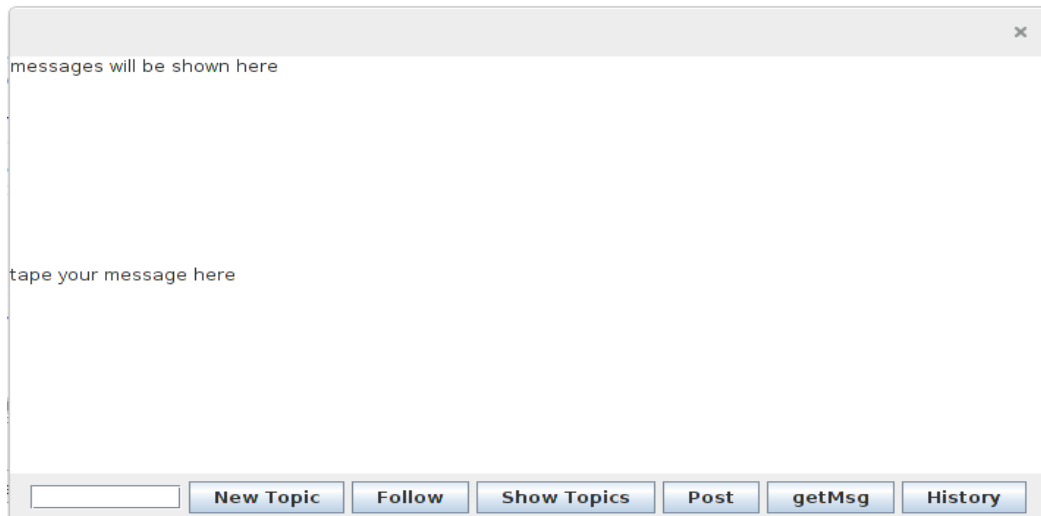
Cette application est composé de deux interfaces:

- loginView:



The screenshot shows a window titled 'loginView' with a close button (X) in the top right corner. Inside the window, there are two text input fields: the first is labeled 'user name' and the second is labeled 'password'. Below these fields, there are two buttons: 'Login' and 'Register'.

- mainView:



Fonctionnalités réalisées

- **créer et connecter un compte** avec pseudo nom et mot de passe.

L'authentification est fait en RMI, puis, côté serveur, on maintien un fichier "userList" en forme .txt. Chaque fois on tente à connecter, nous allons vérifier dans ce fichier, si le nom et le mot de passe existent, l'authentification est réussit. Sinon, on va demander au client de créer un nouveau compte.

- **récupérer les hashtags**(topics) existent déjà.

Pour que le client puisse choisir le topic à suivre.

- **créer un nouvel hashtag**(topic).

- **suivre un hashtag**(topic).

Le client peut suivre plusieurs hashtags.

- **écrire et envoyer des messages** sur un hashtag(topic).

Faut choisir d'abord un topic dans le boîte d'entrée à gauche en bas, puis le message, enfin, clique sur le bouton "post" pour envoyer un message.

- **lire des messages** sur un hashtag(topic) en cas de connections et déconnections.

Clique sur le bouton "getMsg" pour récupérer le message le plus proche. L'intérêt de notre projet est que les messages sont persistants, c'est à dire, un abonné à un topic puisse également être notifié des messages publiés alors qu'il n'était pas connecté. Pour le réaliser, on a ajouté un champs ClientID pour suivre un topic.

- **maintenir les comptes, les message** dans le serveur, sauvegarder dans des fichiers locaux.

- **récupérer des message d'archives.**

Clique sur le bouton "history" pour récupérer tous les messages d'archives.

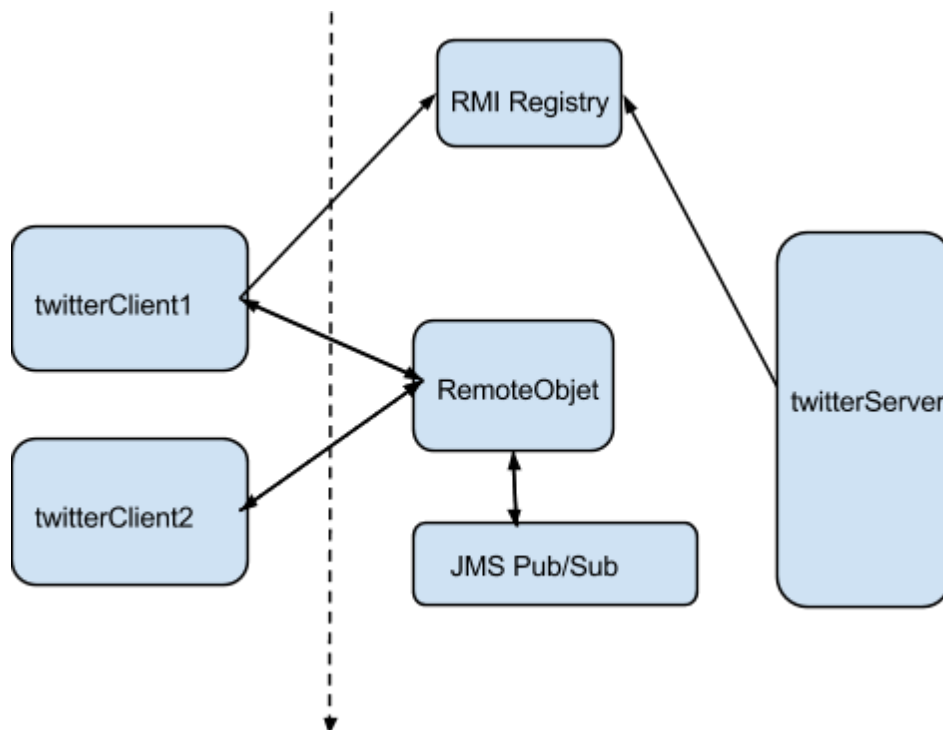
- une **interface simple et facile** à utiliser.

Architecture

Dans ce projet, nous utilisons RMI et JMS.

- Utiliser RMI pour la connection entre le client et le serveur. Le client trouve RemoteObjet par RMI Registry, puis utilise les methodes distantes pour envoyer et recevoir des messages, manipuler hashtags, etc.
- Utiliser JMS pour la diffusion des tweets postés. Nous utilisons la module "Publish-subscribe", pour chaque hashtag est créé un topic JMS correspondant. Les client sont publisher et subscriber dans le même temps.

Nous choisissons de mettre le JMS dans la côte serveur afin que la côte client est claire et simple. Le client n'a pas besoin de comprendre les services nous avons choisis (JMS ou d'autre), il peut tout simplement réaliser l'interface.



Aspects non développés

1. Notifier aux utilisateurs la création d'un nouveau topic.

Explication:

nous avons créé un topic "public", c'est un topic qui est abandonné implicitement par tous les utilisateurs, et qui envoie un message quand un nouvel hashtag est créé, mais les clients ne la reçoivent pas automatiquement.

2. Les messages ne peuvent pas être affichés automatiquement, on doit cliquer sur le bouton "getMsg" pour les récupérer.

Explication:

Vue qu'un client peut être un publisher et un consumer à la même temps. Il ne peut pas utiliser la forme "while(true)" pour écouter l'arrivée des messages tous le temps. Donc, nous avons décidé d'utiliser la fonction setMessageListener(), chaque fois un message arrive, on appelle la fonction onMessage() pour lire le contenu du message. La problème est que l'on a mit la partie JMS sur le côté du RMI serveur. Du coup, on a ajouté une fonction getMessage() retournant le contenu de message au client. Ainsi faut le RMI client faire appeler cette fonction remote pour récupérer un message, soit clique sur le bouton "getMsg".

3. retweet un message

Explication:

La problème se trouve dans comment choisir le message que l'on veut retweet. Ce que l'on a pensé est de cliquer sur un message et puis le retweet, mais c'est compliqué à réaliser.

Difficultés rencontrées

La grande difficulté nous avons rencontrée était la combinaison de RMI et JMS. La configuration et la synchronisation des messages entre le client et le serveur sont compliquées.

Lancement du projet

Les projets Eclipse se trouvent dans le répertoire MyTwitterClient et MyTwitterServer. Pour lancer, nous avons besoin de ajouter activemq-all-5.11.1.jar et javaee.jar dans le build path. Puis nous lançons ActiveMq, twitterServer, twitterClient (peut être pluriel) par l'ordre.