

Readme pour le mini projet twitter d'applications réparties.

Introduction

La version rendue est basique, c'est à dire l'abonné ne reçoit pas les topics si il n'est pas en ligne. Cependant, ce qui à été implémenté fonctionne bien, et on peut bien envoyer des messages avec des hashtags, recevoir des "twitts" en direct, ...

Architecture globale

Notre projet est divisé, et pour des raisons de partage du travail, et pour des raisons de cohérence objet, en deux modules : un permettant la partie authentification(serveur), un permettant la création de message et la reception(client). Il existe aussi un module de test qui permet le test séparé de ces deux modules.

Ces deux modules sont dans le même projet, mais il sera très aisé de le séparer en deux projets différents.

On supposera que le serveur de login est sur localhost, tout comme le serveur ActiveMQ, qui , lors du lancement des applications, sera déjà lancé.

Partie messagerie

On a organisé notre code sous la forme pub-sub pour ce qui est de la partie messagerie. Une classe s'occupe de la partie envoi de message, une autre s'occupe de la réception. Le pub et sub ont des interfaces afin d'y accéder de manière simplifiée.

Certaines fonctionnalités comme la possibilité de s'abonner ou de se désabonner au beau milieu de l'exécution n'ont pas été exploitées bien que présentes (Il faudrait encore une console supplémentaire, cela deviendrait impossible).

Partie authentification

La partie authentification est un client serveur, le serveur centralisant les logins, et les clients pouvant fournir leur login pour vérification.

On peut aussi ajouter des utilisateurs mais cette fonctionnalité n'est utilisée qu'au début pour générer des utilisateurs "par défaut " (il n'y a pas de console d'administration du serveur).

Difficultés rencontrées.

Il n'est pas évident de voir au premier abord qu'on a besoin de deux applications par machine, une pour publier, une pour recevoir les messages; de même nous avons rencontré un problème de duplication de messages, en effet, si quelqu'un s'abonne à plusieurs sujets, il va recevoir plusieurs fois le même message avec un hashtag différent.

Le seul moyen que nous voyons pour contourner ce problème est de créer une chaîne de caractère puis faire une expression régulière pour éliminer les doublons (mais pas les hashtags). Cependant,

cette solution ne marche pas si les 2 messages identiques arrivent avec un grand intervalle de temps. Nous avons donc juste ajouté aux messages qui peuvent être en doublon un avertissement.

Evidemment(mais ce n'est pas l'exercice), il serait bien plus simple de créer un topic unique avec des StringAttributes sur chaque message représentant les hashtags.

Faire tourner le projet.

Pour ceci , il faut lancer les trois applications (pub.rar et sub.rar serveur.rar) dans 3 terminaux différents, puis se logger sur les deux terminaux clients.

Ensuite, on peut écrire sur le pub(avec hashtags, ..), recevoir des notifications sur le sub, ...

Pour plus de simplicité, un script ./start.sh permet de lancer deux publishers, deux sub et un serveur de login (2 clients et un serveur).

Attention, pour qu'un message soit parsé côté pub il faut qu'il soit de la forme message #hashtag1 #hashtag2 ... #hashtagN

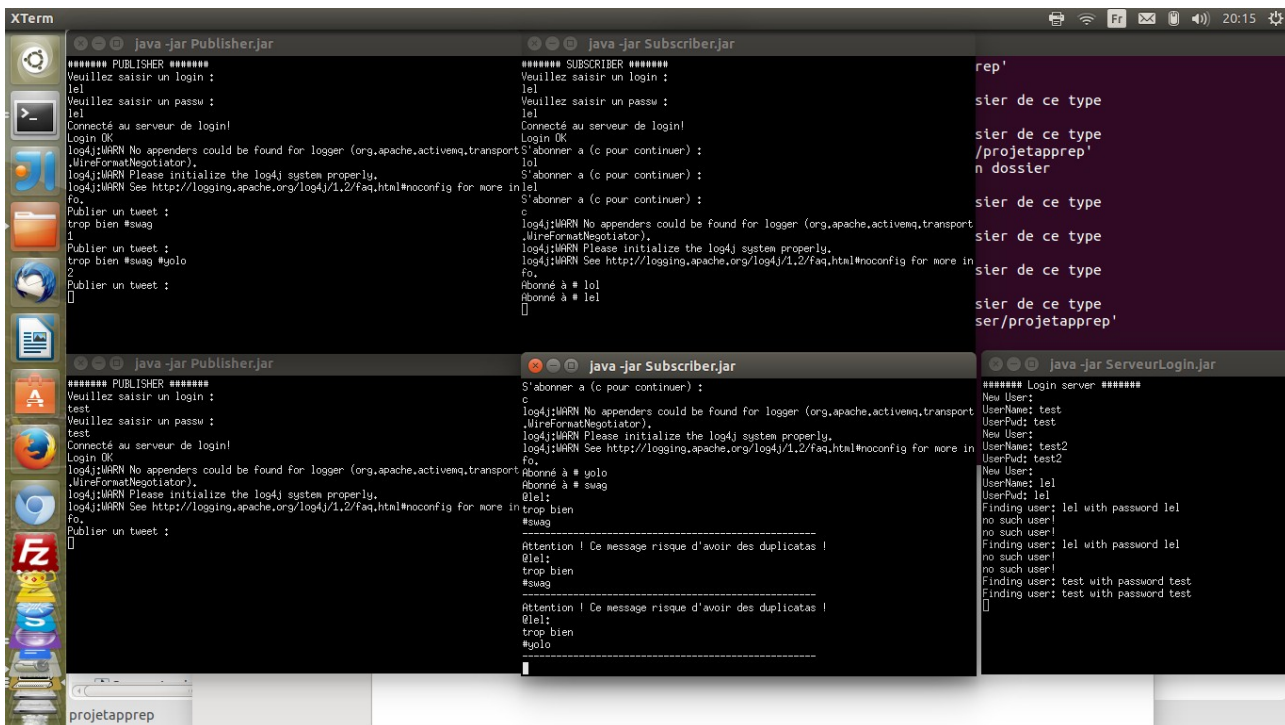
Pour démarrer la réception coté sub, il faut entrer 'c' lors de la boucle qui récupère ce a quoi on veut s'abonner. De ce fait, on ne peut pas avoir de nom de topic "c".

Les login par défaut sur le serveur sont : lel/lel

test1/test1

test/test

Bien sûr on peut directement lancer le projet dans eclipse en lançant PublisherLaunch, SubscriberLaunch, LoginServeurLaunch.



The screenshot shows three terminal windows running Java applications. The top-left window is titled 'java -jar Publisher.jar' and shows the Publisher application's output, including prompts for login and password, and a list of tweets. The top-right window is titled 'java -jar Subscriber.jar' and shows the Subscriber application's output, including prompts for login and password, and a list of tweets. The bottom window is titled 'java -jar LoginServeur.jar' and shows the LoginServer application's output, including prompts for login and password, and a list of users. The bottom window also shows a warning message: 'Attention ! Ce message risque d'avoir des duplicatas !'.

```
##### PUBLISHER #####
Veillez saisir un login :
lel
Veillez saisir un passw :
lel
Connecté au serveur de login!
Login OK
log4j:WARN No appenders could be found for logger (org.apache.activemq.transport
.WireFormatNegotiator).
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more in
fo.
Publier un tweet :
trop bien #swag
1
Publier un tweet :
trop bien #swag #yolo
2
Publier un tweet :
[]

##### SUBSCRIBER #####
Veillez saisir un login :
lel
Veillez saisir un passw :
lel
Connecté au serveur de login!
Login OK
S'abonner a (c pour continuer) :
lel
S'abonner a (c pour continuer) :
c
log4j:WARN No appenders could be found for logger (org.apache.activemq.transport
.WireFormatNegotiator).
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more in
fo.
Abonné à # lol
Abonné à # lel
[]

##### Login server #####
New User:
Username: test
UserPwd: test
New User:
Username: test2
UserPwd: test2
New User:
Username: lel
UserPwd: lel
Finding user: lel with password lel
no such user!
Finding user: lel with password lel
no such user!
Finding user: test with password test
no such user!
Finding user: test with password test
[]

Attention ! Ce message risque d'avoir des duplicatas !
0!el:
trop bien
#swag

Attention ! Ce message risque d'avoir des duplicatas !
0!el:
trop bien
#yolo
```