SCIENCE DU NUMERIQUE

S7 - Systèmes Concurrents et Communicants

Projet Données Réparties

Gestion d'objets duppliqués

Auteurs :
Yvan Charles KIEGAIN DJOKO
Aicha BENNAGHMOUCH

Systèmes Logiciels - Groupe L3
Département Sciences du Numérique - Deuxième année
2022-2023

Table des matières

Table des figures

1 Introduction

Le but de ce projet est d'illustrer les principes de programmation répartie vus en cours. Pour ce faire, nous allons réaliser sur Java un service de partage d'objets par duplication, reposant sur la cohérence à l'entrée (entry consistency). Ceci est un petit rapport présentant l'architecture, les algorithmes des opérations essentielles, une explication claire des points délicats et de leur résolution, et un mot sur les exemples originaux développés pour tester votre système.

2 Première étape

Pour la première étape, il nous a été demandé d'implanter un service de gestion d'objets partagés répartis. Les SharedObject sont utilisés explicitement par les applications. Plusieurs applications peuvent accéder de façon concurrente au même objet, ce qui nécessite de mettre en œuvre un schéma de synchronisation globalement cohérent pour le service que nous implantons. On suppose que chaque application voulant utiliser un objet en récupère une référence (à un SharedObject) en utilisant le serveur de nom. On ne gère pas le stockage de référence (à des objets partagés) dans des objets partagés.

2.1 ServerObject

Nous avons introduit le type énumération **states** pour représenter l'état des verrous.

private enum states NL,RL,WL ;

Les méthodes implantés dans cette classe :
- public Object lock_read(Client_itf c)
- public Object lock_write(Client_itf c)

2.2 SharedObject

Nous avons introduit le type énumération **states** pour représenter l'état des verrous.

private enum states NL,RL,WL ;

Les méthodes implantés dans cette classe :
- public void lock_read()
- public void lock_write()
- public **synchronized** void unlock()
- public **synchronized** Object reduce_lock()
- public **synchronized** void invalidate_reader()
- public **synchronized** Object invalidate_writer()

2.3 Client

Définition d'un dictionnaire dans lesquels chaque objet est stocké avec son identifiant.

private static Map<Integer, SharedObject> annuaire ;

Les méthodes implantés dans cette classe :
- public static void init()
- public static SharedObject lookup(String name)
- public static void register(String name, SharedObject_itf so)
- public static SharedObject create(Object o)
- public static Object lock_read(int id)
- public static Object lock_write (int id)
- public Object reduce_lock(int id)
- public void invalidate_reader(int id)
- public Object invalidate_writer(int id)

2.4 Server

Définition d'un dictionnaire dans lesquels chaque nom d'objet est stocké avec son identifiant.

private Map<String, Integer> annuaire ;

Définition d'un dictionnaire dans lesquels chaque objet est stocké avec son identifiant.

private Map<Integer, ServerObject> annuaire_serverobj_id ;

Les méthodes implantés dans cette classe :
- public int lookup(String name)
- public void register(String name, int id)
- public int create(Object o)
- public Object lock_read(int id, Client_itf client)
- public Object lock_write(int id, Client_itf client)

3 Deuxième étape

Pour la deuxième étape, nous avons modifié les méthodes suivantes de la classe client :
- public static SharedObject lookup(String name) ;
- public static SharedObject create(Object o) ;

Nous avons également créer une classe GenerateStub qui permet de générer un stub à partir d'une interface.

4 Troisième étape

Nous n'avons pas eu le temps de tester l'étape 3.

5 Tests réalisés

5.1 Tests avec Irc et Irc_unlock

Nous avons tout d'abord tester quelques cas de base du programme en utilisant l'interface fournie Irc. Cependant, celle-ci ne contient que deux boutons Read et Write ce qui ne nous permet pas de tester tous les cas décrits dans le sujet et de s'assurer du bon fonctionnement du programme et de la bonne gestion de la concurrence.
Nous avons donc implanté Irc_unlock pour pouvoir explicitement libérer les jetons et de tester plusieurs cas de figures.

5.2 Classes de Tests

Nous avons rédigé deux classes de tests générant des situations entre des clients et un serveur :
- TestSpecific.java : cette classe de test se lance sans paramètre et consiste à deux clients qui essaye l'un d'écrire et l'autre de lire un objet commun stocker sur le serveur en retirant des lignes de code unlock dans un ou dans les deux clients le test ne termine pas car le jeton n'es pas libéré donc un client reste en attente de façon infini.
- TestGeneral.java : cette classe de test génère un système plus complet de clients, serveur et objets. Elle se lance avec deux paramètres le premier le nombre de clients et le deuxième le nombre d'objets. le type des clients(ecrivain ou lecteur) et les objets sont choisis aléatoirement. Ce test a des comportements aléatoire à partir d'un certain de nombre de clients générant des erreurs et montrant que notre projet peut encore doit encore être corriger pour être plus robuste.

6 Conclusion

En conclusion ce projet, aura été très bénéfique pour nous car il nous aura permis de voir en application réelle tous les principes vus pendant nos enseignements de cette UE. Ca aura été un projet très interessant car il nous demandait de réfléchir différemment nous qui avons toujours eu l'habitude d'avoir une réflexion séquentielle de notre code ; nous avons du réfléchir en concurrence. On aura connu de nombreuses difficultés durant ce dernier que ce sois par la compréhension du sujet qui nous a pris pas mal de temps mais également les différentes erreurs auquelles on a été confronté et qui était nouvelle pour nous. Nous avons pu réalisé toutes les étapes demandés sauf l'étape 3 qui n'es pas fonctionnelle . Nous sommes satisfais du travail accomplis mais savons que nous aurions pu faire mieux en s'organisant un peu mieux ce qui nous aurais fais gaganer du temps afin de rendre oplus robuste notre code. Personnellement nous pensons que pour notre propre désir de mieux faire nous allons continuer à travailler dessus pour le rendre meilleur voir le terminer