Pràctica 1
RMI

Faysal Badaoui Mahdad
Universitat de Lleida
Computació distribuïda i aplicacions

Curs 2022 - 2023

Índex

1 Introducció

Soc el Faysal Badaoui Mahdad, estudiant de Informàtica i en aquest informe explicaré la meva implementació, validació d'aquesta en local i remot i descripció dels problemes que m'he trobat.

La implementació realitzada consisteix en la creació d'un sistema de cues de missatges suportant el paradigma MOM, tot implementant RMI.
El paradigma MOM, consisteix en una infraestructura de software que permet l'enviament i la recepció de missatges entre aplicacions distribuïdes.
En aquesta pràctica principalment he implementat dos paradigmes dintre del MOM, el punt a punt i el paradigma de publicació/subscripció.

El punt a punt, és un paradigma que no requereix de l'ús de Callbacks, ja que objectes que envien o volen rebre missatges el que fan és la crida a mètodes sobre l'objecte remot (API). En canvi, el paradigma de publicació/subscripció sí que és necessària la implementació de Callbacks perquè l'API haurà de notificar als objectes sobre l'arribada d'algun missatge.

2 Disseny i implementació del sistema

Per a dur a terme la implementació del sistema amb tots els mètodes i funcionalitats demanades, han fet falta dues interfícies i les classes de tots els objectes necessaris.

2.1 Interface MessageAPI

Aquesta, serà la interfície amb els mètodes que els diferents objectes podran invocar per a crear cues, enviar missatges, subscriure's a cues, rebre missatges amb altres mètodes sobre les cues, com a sobre semàfors, els quals han sigut necessaris per a mantenir una sincronització.

2.2 Interface TopicListenerInterface

En aquest cas, la interfície, serà necessària per a permetre l'aplicació de callbacks. És a dir, per a permetre a l'API, notificar als subscriptors de tots els missatges en una cua de publicació/subscripció i també, notificar-los en el moment que es tanca una cua.

2.3 Class EMomError

Aquesta classe té la principal funció de mostrar els errors que ocorren i, per tant, donar a conèixer com s'està executant l'aplicació. Bàsicament, és una classe que té un camp on es guarda el error ocorregut.

2.4 Class EPublishMode

En aquest cas, és una classe amb la funcionalitat de guardar el mode de distribució dels missatges en una Topic Queue i al mateix moment que mètodes per consultar si és un

mètode o un altre.

2.5 Class Message

La classe Message l'he implementat per a poder tenir un objecte que representi cada missatge i d'aquesta forma, tenir llistes dels missatges de forma més correcta i òptima. Conte dos mètodes per a consultar el missatge i el tipus.

2.6 Class TopicQueue

Aquesta és la classe més important per a la gestió de les Topic Queues, ja que conté els clients subscrits i els missatges de cada Topic Queue. Juntament amb el mode de publicació. D'aquesta forma, tinc un control de cada Topic Queue de forma més centralitzada.

2.7 Class MessageAPIImpl

Aquesta classe és la principal. Ja que és la classe contenidora de tots els mètodes que els clients executaran. A part dels mètodes especificats, els elements més importants són, la HashMap que controla les Topic Queues creades i la que controla les Queues normals.
A part dels mètodes especificats en l'enunciat, he creat tres mètodes adicionals. El mètode **getSemaphore(...)** te la funció de agafar el nombre de workers necessitats. Ja que el client master no pot comunicar als workers fins que no estiguin tots subscrits. Per aquest motiu hi han els mètodes **catchSem(...)** i **releaseSem(...)**. Aquest tenen la funció de bloquejar o desbloquejar el Master, per mantenir una execució sincronitzada.

2.8 Class Server

La classe Server, és la classe que s'executa per a poder connectar l'API i així poder usar-la remotament. Tot això nçamitançant el mètode MsgQ Init().

2.9 Class SumatorioMPrimos

En aquest cas, és la classe amb els mètodes que efectuen les operacions de suma en rangs i, per tant, es pot obtenir els resultats.

2.10 Class DisSumMaster

Consisteix en una classe client encarregada de distribuir feina. Bàsicament, ha de connectar-se i obtenir l'objecte RMI. Ja que haurà de dividir la feina en un nombre de workers i penjar la feina en una cua. A part d'això, també haurà de crear una altra cua per on rebrà les solucions dels clients.

2.11 Class DisSumWorker

És l'altra classe client, en aquest cas, només haurà de subscriure's en una Topic queue per a rebre la feina del Master i fer els càlculs necessaris fins a poder penjar el resultat a una altra cua. D'aquesta forma, podrà enviar el resultat al Master.

3 Validació

Per a realitzar la validació, fan falta com a mínim 4 terminals. És degut al fet que caldrà inicialitzar el rmiregistry, servidor, el master i com a mínim 1 client. Per tant, per validar el funcionament:

3.1 Local

Figura 1: Terminal rmiregistry

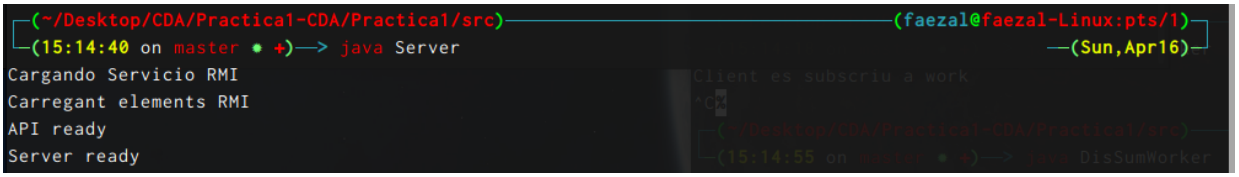

Figura 2: Terminal Server

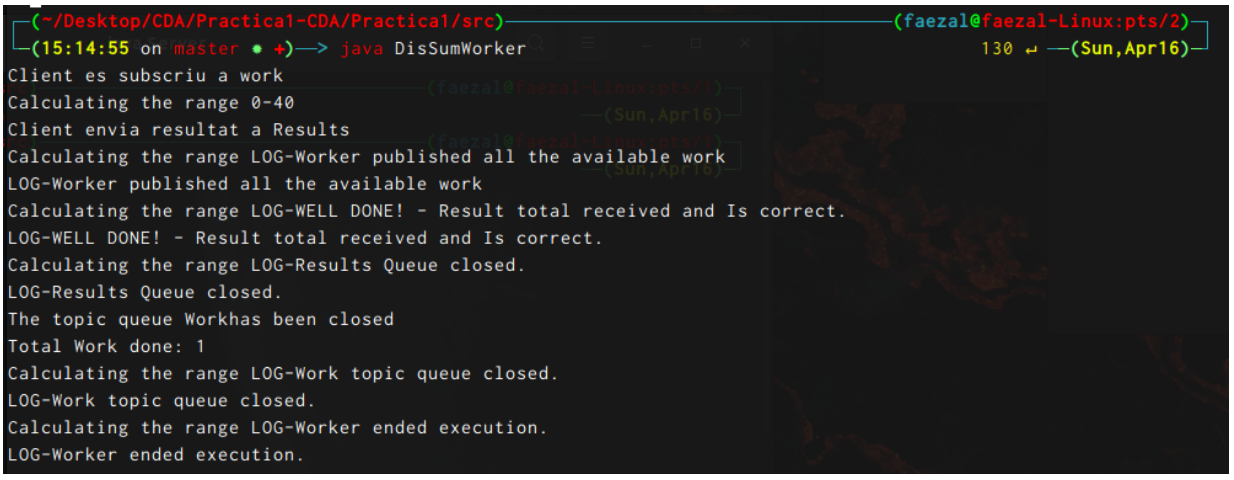

Figura 3: Terminal Worker

Figura 4: Terminal Master

Per a validar la distribució correcta en round robin i la recepció correcta de resultats. Els processos mostren per pantalla la informació que reben.

3.2 Remota

Per a la validació remota, es realitza un procés similar. La diferència és que el servidor i el rmiregistry es trobaran en el cluster de la Udl i, per tant, els clients es connectaran usant la ip del cluster. La qual és passarà per paràmetre.

4 Problemes trobats

Els problemes trobats han consistit bàsicament en el plantejament dels programes i com realitzar totes les operacions. Un altre dels problemes consistia en el treball amb objectes externs. És a dir, problemes amb el retorn de classes en els mètodes remots.

5 Conclusió

Finalment, vull comentar que aquesta pràctica ha sigut molt interessant i m'ha ajudat a aprendre molt millor com funcionen els objectes RMI. Al mateix moment he gaudit molt programant i veient el potencial que tenen els objectes remots.