UNIVERSITÀ DI PISA

Relazione Progetto
TURING: DISTRIBUTED COLLABORATIVE EDITING

LABORATORIO DI RETI, CORSI A E B
PROGETTO DI FINE CORSO
A.A. 2018/19

Domenico Profumo | Corso B| Matricola 533695

INDICE

1. Descrizione generale architettura

Il progetto consiste nella realizzazione di un programma client/server per la fornitura di un servizio editing di documenti con una chat annessa.

Il server utilizza il multiplexing dei canali tramite NIO per la gestione dei client connessi e con comunicazione tramite diversi protocolli. Per ogni funzione la gestione delle eccezioni viene fatta dalla classe chiamante e in caso di eventuali errori generati dalla stessa avviene la terminazione del server/client con l'eventuale messaggio di errore associato.

Per i documenti si utilizza una cartella di nome "Documenti" contenente tante directory (quanti sono i documenti) con le relative sezioni (file associati). L'accesso a tali file viene effettuato in sola lettura per reperire il contenuto e in sola scrittura per scriverlo dopo l'eventuale operazione di edit.

Durante l'operazione di edit la chat è stata realizzata con la generazione in modo random di un indirizzo ip (compreso nel range degli indirizzi multicast) e un eventuale porta associata, la chat rimane inoltre attiva finché c'è almeno un utente che sta editando altrimenti avviene la chiusura della stessa con una eventuale apertura (indirizzo diverso) quando un utente cercherà di editare la stessa o un'altra parte del documento successivamente.

Il client comunica al server l'eventuale operazione da eseguire con una stringa contenente (il proprio nome), l'operazione associata e gli eventuali parametri legati alla stessa; il server effettuerà per ogni byte inviato dal client il parsing della stessa e comunicherà l'eventuale risposta relativa.

Ogni operazione effettuata all'interno del server turing è legata allo stato in cui l'utente si trova; in tutto abbiamo 3 fasi, nella prima l'utente può scegliere di effettuare la registrazione o il login (ogni altra operazione sarà considerata un errore), nella seconda all'utente è consentito di effettuare la creazione di un documento con le relative sezioni associate ,la modifica di una sezione,la visualizzazione a schermo del contenuto di una sezione o di tutte le sezioni del documento, la lista dei documenti associati all'utente che ne ha fatto richiesta, la condivisione di un documento con un altro utente registrato o il logout, mentre nella terza fase (fase di edit) si può effettuare l'invio di un messaggio a tutti gli utenti che stanno editando la stessa sezione, leggere i messaggi precedentemente inviati nella chat oppure terminare la fase di editing del documento stesso sulla sezione specifica.

Ogni eventuale terminazione brutale dell'utente è gestita mediante il logout dell'utente stesso attraverso la chiave associata ad ogni client subito dopo la fase del login e gestita mediante la funzione attachment.

2. Manuale di istruzioni

Estrarre il contenuto del file "ProgettoTuring_533695_PROFUMO_DOMENICO", aprire il terminale (se si usa windows 10 digitare cmd nella barra di ricerca e cliccare su invio), recarsi nella directory contenente i file appena estratti e compilare con il comando "javac client.java" (per la parte client) e "javac server.java" (per la parte server) come mostrato di seguito:

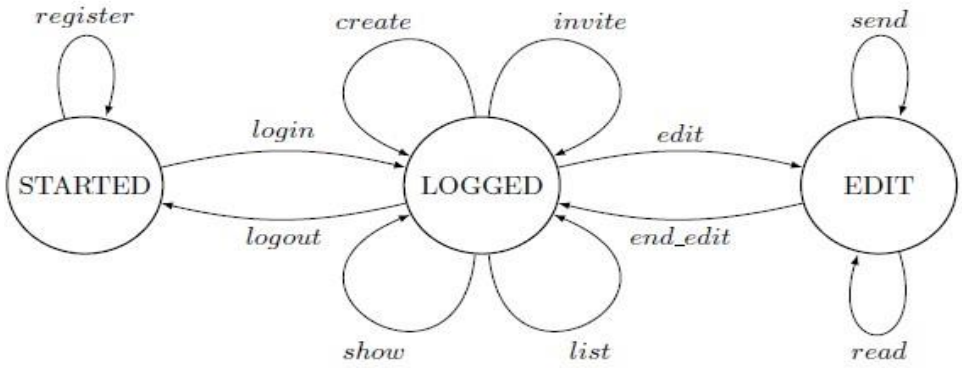
```
C:\Users\domen\eclipse-workspace\Progetto LR\src>javac client.java

C:\Users\domen\eclipse-workspace\Progetto LR\src>javac server.java
```

Subito dopo avviare il server digitando "java Server" (per il server) e "java Client" per il client. Dopo aver avviato il client ed aver stabilito la comunicazione basta interagire con il client digitando i seguenti tipi di comando: **$** turing command [ARGS...]

```
commands :
register <username > <password > registra l' utente
login <username > <password > effettua il login
logout effettua il logout

create <doc > <numsezioni > crea un documento
share <doc > <username > condivide il documento
show <doc > <sec > mostra una sezione del documento
show <doc > mostra l' intero documento
list mostra la lista dei documenti

edit <doc > <sec > modifica una sezione del documento
end - edit fine modifica della sezione del doc.

send <msg > invia un msg sulla chat
receive visualizza i msg ricevuti sulla chat
```

Per ricevere la lista dei comandi basta digitare: **$** turing --help

Ogni comando dev'essere associato alla parola "turing" altrimenti verrà dato in output un messaggio di errore. In tutto abbiamo 3 fasi in cui si può trovare lo stato del mio client, per ciascuna fase i comandi possibili sono dati dalla seguente figura:



Ad ogni comando errato verranno stampati a schermo i possibili comandi attesi.

3. Scelte progettuali associate

Il progetto è suddiviso in diversi file:

- Server: classe principale contenente il server turing con lo scopo di soddisfare tutte le richieste provenienti da altri client che vogliono stabilire la connesione.

- Client: classe relativa al client che cerca di connettersi al server per comunicare eventuali richieste

- Documento: classe utilizzata per la creazione dei vari documenti con le relative funzioni di gestione dei file e della chat, essa include le funzioni per l'editing del documento, per end-edit, per la gestione della lista contenente i nomi relativi agli utenti(collaboratori) che sono autorizzati a modificare il documento e una lista contenente i bit di controllo d'uso di ciascuna sezione del documento.

- User: interfaccia utente costituita da varie operazioni sull'oggetto remoto Utente

- Utente: classe utilizzata per gestire i diversi utenti nel server, con i rispettivi documenti associati e le varie funzioni di gestione

- UserImpl: implementazione della classe User definita precedentemente con le relative funzioni sull'oggetto remoto

- ChatMulticastReceiver: classe per la gestione del thread che si mette in ascolto di eventuali messaggi inviati sulla chat

GESTIONE CONCORRENZA

Per gestire l'accesso ad alcune strutture condivise tra client e server come ad esempio "hahsmap" (presente nel file UserImpl ed utilizzata da entrambi attraverso l'RMI per memorizzare la lista degli utenti registrati) si è scelto di utilizzare la classe ConcurrentHashMap in modo da prevenire situazioni di concorrenza e quindi offrire un accesso sicuro alla struttura dati.

4. Funzionamento client server

L'interazione avviene attraverso una stringa con la seguente struttura:

(username):operation parametro1 parametro2

Dove username è user dell'utente che fa richiesta, il parametro operation indica il tipo(nome) dell'operazione richiesta dal client e i parametri sono i rispettivi comandi associati all'operazione da eseguire.

CLIENT

Inizialmente attraverso uno scanner effettua il parsing dei parametri dati durante l'esecuzione, attraverso uno switch effettua la rispettiva funzione da eseguire. Il primo parametro passato da linea di comando dev'essere "turing" altrimenti verrà dato un messaggio di errore. Le varie operazioni eseguite sono:

Register è il comando associato alla fase di registrazione la quale viene implementata mediante l'utilizzo di RMI con l'utilizzo di un registry su porta 9999 nella quale viene inserita la lista degli utenti registrati. Per ogni utente che desidera registrarsi viene fatto un controllo sulla lista di utenti già registrati ed in caso di esistenza dello stesso username nella lista viene mandato un messaggio di errore.

NOTA: In questa fase non avviene una connessione esplicita con il server ma implicita perché entrambi condividono lo stesso oggetto mediante RMI.

Login avviene mediante connessione TCP su porta di default 1919 e indirizzo "localhost" (`127.0.0.1), co`munica al server l'operazione di login passando come parametro1 l'username e come parametro2 la password dell'utente che ha effettuato la richiesta, segue una striga contenente la risposta riguardo l'esito dell'operazione associata.

Create viene effettuata solo dopo che l'utente ha superato con successo la fase di login, effettua il parsing dalla linea di comando e comunico la richiesta al server fino a ricevere una stringa contenente il relativo esito dell'operazione

Show consiste in due operazioni separate legate al numero di parametri passati da linea di comando, se passiamo solamente il parametro1 significa che vogliamo eseguire la funzione di visualizzazione di tutto il documento (con le relative sezioni), in questo caso comunica la richiesta al server ed in caso esito positivo, mi invia un primo valore relativo alla dimensione del numero di sezioni che devo ricevere e per ogni sezione presente all'interno del documento invia il relativo contenuto.

Se invece passiamo anche il secondo parametro significa che stiamo facendo l'operazione di visualizzazione di una sezione all'interno del documento, quindi in questo caso viene inviata solo quest'ultima.

List è l'operazione che mi permette di ricevere tutte le informazioni associate alla lista dei documenti che l'utente possiede da autore o collaboratore, in questo caso non si passa nessun parametro in input (solo l'operazione "list") perché il client possiede già il nome dell'utente che sta facendo richiesta (dato che siamo nella fase di login) ed invia la richiesta al server. La risposta può essere una stringa vuota (mi indica che non ci sono documenti associati) oppure una stringa contenente le info relative a tutti i documenti associati all'utente.

Share è l'operazione di condivisione, in questo caso si prende in input il paametro1 legato al documento che si vuole condividere e l'utente designato alla condivisione del file (parametro2). Comunica al server la richiesta e se tutto è andato a buon fine il server riceve l'esito relativo, altrimenti un messaggio di errore.

Edit permette l'editing di una sezione relativa al documento, inizialmente si effettua il parsing da linea di comando, controlla che i parametri passati siano corretti, comunica la richiesta al server e successivamente viene effettuato il download del file. Subito dopo il download si passa alla fase di editing attraverso un programma editor associato al file aperto una volta che viene scaricato. In questa fase si entra nella **chat** ovvero, con le relative opzioni di **send** per inviare un "DatagramPacket" (contenente il messaggio dato a linea di comando) attraverso l'indirizzo ip multicast associato al documento da cui si sta editando e l'operazione di **receive** per ricevere tutti i messaggi inviati dal client stesso e da altri utenti a partire da quando ha iniziato la fase di editing.

End-Edit utilizzata per completare l'editing della sezione associata, avviene inviando l'intero contenuto presente nel file temporaneo (che utilizzavamo per l'editing) nella sezione originale del documento.

NOTA: questa sezione è stata modificata solo da client stesso e non da altri perché per ogni sezione modificata setto un bit che mi indica che la sezione è occupata.

Log-out ultima operazione nella fase di login e consiste nel disconnettere l'utente dalla lista utenti connessi, anche qui abbiamo la comunicazione della richiesta da parte del client al server e la relativa risposta dello stesso.

NOTA: lo stesso client dopo il logout può continuare con le fasi di registrazione di nuovo utente o effettuate nuovamente il login.

SERVER

Inizialmente viene creato il registro nella porta 9999 e si inserisce la lista utenti che sarà poi utilizzata dal client per reperire eventuali info legati agli utenti stessi. La comunicazione con il client avviene mediante connessione TCP dove per ogni operazione richiesta dal client (vedi pagina precedente) è associata una stringa di risposta con l'eventuale esito dell'operazione.

La gestione dei client connessi avviene tramite una select la quale seleziona i relativi channel e permette di capire lo stato in lettura/scrittura in cui si trova il relativo client. Il server rimane quindi in attesa di eventuali richieste e non appena ne trova una messa in ascolto di una nuova connessione la inserisce alla lista delle chiavi associate, mentre se la chiave è in stato di ascolto (ready) si esegue l'operazione richiesta. Per la comunicazione si utilizzano i bytebuffer dove si memorizza al loro interno la richiesta fatta dal client, subito dopo tramite il parsing si esegue la funzione richiesta.

Login si effettua il parsing della stringa ricevuta per salvare il nome utente e la password del client che ne ha fatto richiesta, avviene il controllo relativo alla registrazione dell'utente e si setta il bit di login (per evitare un accesso duplicato), infine si comunica l'esito al client.

Create si effettua il parsing della stringa ricevuta dal client, si controlla se esisteva già il documento associato, altrimenti creo il nuovo documento aggiungendolo alla lista dell'utente che ne ha fatto richiesta comunicando l'esito dell'operazione al client.

Show si effettua il parsing della stringa ricevuta dal client per reperire il documento con la relativa sezione da mostrare, recupera il documento (se presente nella lista dei documenti associata all'username che ha fatto richiesta) e invia il relativo contenuto al client

ShowD fa le stesse funzioni della funzione Show ma utilizza tutte le sezioni relative al documento richiesto anziché una sola, segue infine il relativo esito dell'operazione annessa.

List si effettua il parsing per reperire l'username dell'utente che ha fatto richiesta, recupera tutti i documenti posseduti dall'utente e per ciascuno di essi salva in una stringa (unica) tutte le info associate (autore, lista collaboratori, nome documento), infine invia la stringa ottenuta.

Share dopo aver effettuato il parsing dei comandi si effettua un controllo relativo alla presenza del documento nella lista dell'utente(client) e all'effettiva registrazione dell'utente designato alla condivisone del file, se tutto è andato a buon fine comunica al client l'esito della richiesta, altrimenti mando un messaggio di errore.

Edit riceve dal client il documento e la sezione relativa all'editing, recupera il documento associato all'username del client, controlla se l'utente possedeva nella sua lista il documento e infine se quest'ultimo ha diritto di editare quel documento prende l'accesso. Invia al client il file richiesto con il rispettivo indirizzo e il numero di porta associati al multicast(chat) relativi al documento stesso ed incremento il numero di utenti connessi. Se una delle operazioni scritte precedentemente non è andata a buon fine manda un messaggio di errore.

End-Edit utilizzata per completare l'editing della sezione associata, recupera il documento dell'utente user, controlla se l'utente possedeva nella sua lista il documento e infine se quest'ultimo ha diritto di editare quel documento prende l'accesso, riceve dal client il contenuto del file da aggiornare e lo copia nella sezione originaria associata.

Log-out si riceve il nome dell'utente da disconnettere, se è già registrato (implicitamente vero) setta il bit di login a false e comunica il rispettivo esito al client.

5. Scelte implementative

Per prevenire eventuali anomalie dovute alla disconnessione di un utente, alla fine di ciascun'operazione si associa alla chiave del channel un oggetto appartenente alla classe "Attachments", contenente il nome utente che fa richiesta, password, tipo di operazione, ed una serie di stringhe contenenti vari messaggi utilizzati poi nella gestione dell'operazione richiesta.

Per ogni operazione la chiave del channel viene registrata in modalità lettura o scrittura memorizzandone al suo interno l'oggetto attachment contenente le info associate allo stesso. Ogni lettura da parte del server viene eseguita quando la chiave è in isReadable la scrittura viene invece eseguita attraverso il reperimento delle info memorizzate nell'attachment associato alla chiave quando quest'ultimo si trova nello stato isWritable.

Le operazioni di lettura e scrittura vengono effettuate da parte di entrambi (abbiamo una comunicazione bidirezionale) sempre attraverso l'uso dei bytebuffer e l'utilizzo di una "read" associata al channel. Per la gestione dei buffer si sono utilizzate le solite operazioni di clear (cancellazione dell'eventuale contenuto associato) e flip (per settare il limite alla posizione corrente e la posizione corrente a 0).

Il download e l'upload di ciascun file avvengono inviando l'intero contenuto del file stesso con il relativo controllo del corretto invio di tutti i byte associati allo stesso.

6. Ambiente di sviluppo e Test relativi

Il codice è stato realizzato mediante l'ambiente di sviluppo "eclipse" su macchina contenente OS Windows 10.

I test sono stati effettuati senza utilizzare programmi esterni, ma da linea di comando, utilizzando la shell del terminale; sono stati effettuati test mirati ad evidenziare le funzionalità di base del progetto e ai problemi riguardante alcune situazioni particolari di gestione degli errori da parte del client e del server stesso. Per ogni errore generato da parametri incorretti dati da linea di comando, viene stampato a schermo l'errore relativo e la gestione dello stesso con l'eventuale terminazione in caso di errori gravi.

Per ogni errore da parte del client, il server esegue la disconnessione/il rilascio della sezione, rispettivamente nella fase di login/editing in modo da consentire nuovamente l'accesso al client che ha terminato.

Un esempio di test che sono stati realizzati da linea di comando sono questi:

Test Fase di registrazione: C1: tenta di fare il login con un utente che non esiste C1: registra un utente C1: esegue il login con l'utente U1 C1: esegue il logout C2: registra U2, esegue il login con U1 C1: logga con U2
Test nella fase di login e della chat: - Visualizza i tuoi documenti (nessuno, quindi le prossime 3 falliscono) - Cerca di leggere un documento (intero) - Cerca di leggere un documento (parte) - Cerca di modificare un documento (parte) - U1 Crea un documento (success) - U2 cerca di leggere e modificare il documento (fallisce, non è invitato) - U1 invita U2 al documento (invito live) - Mostra i documenti di nuovo (sia U1 che U2) - U1 modifica un doc (es. parte 1) - U2 cerca di modificare la stessa parte (fail) - U2 modifica un'altra parte (success) - Controlla la sezione che stai editando se è nel range dei valori - Test che la chat funzioni - Esci e rientra nella fase di editing

NOTA: C1 e C2 sono i client e U1 e U2 sono gli utenti

7. Gestione della memoria

I file relativi a tutti i documenti sono memorizzati nella directory principale in una cartella “Documenti” contenente tutte le directory relative a ciascun documento con all’interno i file che rappresentano le varie sezioni associate.

Durante l’esecuzione l’accesso a tali file avviene in sola lettura (se stiamo facendo un editing) altrimenti in scrittura se abbiamo finito di editare. Nella prima (editing) abbiamo la copia dell’intero contenuto della sezione in un file temporaneo con la seguente semantica: tempEdit-username-numerosezione dove username è legato al nome dell’utente che sta editando e il numerosezione è la sezione del documento stessa. Nella seconda fase avviene la copia dell’intero contenuto del file temporaneo nella sezione designata.

In caso di terminazione forzata del server i file rimangono nel disco e non vengono eliminati perdendo quindi l’associazione tra questi e le varie strutture dati utilizzate.

Se un client vuole creare un documento con lo stesso nome di un altro documento esistente viene mandato un messaggio di errore (mi consente di non avere documenti duplicati), stessa cosa per l’eventuale accesso ad una sezione non presente in un documento.

8. Chat

Viene utilizzata solo durante la fase di editing di un documento e rimane aperta finché non si esegue end-edit. Ad ogni client è comunicato l’indirizzo multicast associato al documento, esso viene generato casualmente tra il range di indirizzi 224.0.0.0-238.255.255.255 escludendo gli indirizzi speciali 224.0.0.1, 224.0.0.2 e 224.0.1.1.. Ad ogni documento viene associato un indirizzo multicast diverso tra loro garantito attraverso il controllo da parte dei una struttura dati contenente tutti gli indirizzi attualmente utilizzati da altre chat e il numero di utenti connessi per ogni indirizzo attualmente in uso. Ad ogni client è associato un thread che rimane in attesa di ricevere nuovi messaggi nella chat e salvarli in una struttura dati condivisa per permetterne l’eventuale richiesta di lettura.

Ogni messaggio ricevuto nella chat viene memorizzato con diversi parametri relativi all’ora e ai minuti di invio, la stringa contenente il messaggio stesso e l’indirizzo del client che ha inviato il messaggio. I messaggi inviati da ciascun client vengono salvati nella chat stessa.
Durante la fase di end-edit si interrompe l’esecuzione del thread in ascolto dei messaggi e si chiude il socket associato al documento se non c’è nessun utente che sta editando lo stesso, rimane aperto altrimenti.

9. Strutture dati utilizzate

Server

- “userlist”: Struttura dati di tipo User, utilizzata per memorizzare al suo interno oggetti di tipo UserlistImp contenenti le coppie (nome utente, oggetto Utente associato)
- “address”: ArrayList<String> contenente tutti gli indirizzi ip utilizzati da altre chat più quelli speciali
- “connecteduser”: ArrayList< Integer> contiene il numero di utenti connessi associato allo stesso indice di address

Client

- “memory”: Struttura dati di tipo ArrayList<String> contenente la lista dei messaggi ricevuti associati alla relativa chat del documento

Documento

- “collaboratori”: Struttura dati di tipo ArrayList<String> contenente la lista degli username che sono stati invitati a collaborare su quel determinato documento.
- “inuso”: Struttura dati di tipo ArrayList<Boolean> contenente valori di verità associati alle rispettive sezioni di editing. Un valore true attribuito a quel determinato indice indica che la sezione è in stato di editing da parte di un utente, false altrimenti. La dimensione dell’array è relativa al numero di sezioni in uso nel documento.

UserImpl

- “hashmap”: Struttura HashMap<String, Utente> contenente come chiave il nome dell’utente(user) e come valore la struttura Utente associata con le rispettive info di gestione

Utente

- “listadoc”: Struttura ArrayList<Documento> contenente la lista di tutti gli oggetti di tipo Documento associati all’utente che ha creato o per i quali è stato invitato a collaborare

ChatMulticastReceiver

- “memory”: Struttura di tipo ArrayBlockingQueue<String> utilizzata dal thread per salvare eventuali messaggi da comunicare eventualmente al client che ne fa richiesta.