TURING:

disTribUted collaboRative edItiNG

Laboratorio di reti, Corso A

Progetto di fine corso

A.A. 2018/2019

1 Alessio Bardelli Mat. 544270

Indice

1. Descrizione generale …….. 3
2. Architettura del sistema …….. 3
   - 2.1. Schema dei thread attivati ……...3
   - 2.2. Strutture dati utilizzate ……... 3
   - 2.3. Gestione della concorrenza …… 4
3. Descrizione delle classi …….. 4
4. Scelte implementative …… 6
   - 4.1. Registrazione ……. 6
   - 4.2. Login e logout …… 6
   - 4.3. Creazione di un documento ……... 6
   - 4.4. Gestione inviti e notifiche ……. 6
   - 4.5. Edit, end-edit ……. 6
   - 4.6. Chat …… 6
5. Istruzioni per l'uso di TURING ……. 7
   - 5.1. Compilazione ……..… 7
   - 5.2. Esecuzione del server ……. 7
   - 5.3. Esecuzione del client …….. 7
   - 5.4. Esecuzione degli script di prova ……. 7

1 Descrizione generale

L’architettura utilizzata per l’applicazione TURING è Server-Client. Per massimizzare l’efficienza, TURING effettua il multiplexing dei canali mediante NIO. Il formato delle richieste e delle risposte scambiate tra i vari componenti del server e del client, è implementato rispettivamente dalle classi Request e Result. A differenza di quanto richiesto dalla specifica ho deciso di gestire le notifiche degli inviti tramite un servizio di callback RMI. L’applicazione client si basa su un’interfaccia CLI, dove ho seguito le linee guida dell’appendice A, ad eccezione di una piccola modifica: ho omesso la parola chiave *turing* all’inizio di ogni riga di comando. Quindi quest’ultima ora risponde alla seguente sintassi: “COMMAND [ARGS...]”.

2 Architettura del sistema

2.1 Schema dei thread attivati

L’applicazione TURING si serve di un thread main per il server, un thread main per il client, un thread per la chat e infine un thread cleaner. Di seguito i dettagli:

- **Thread main server;** una volta attivato imposta gli indirizzi di porta TCP, se specificati; crea ed esporta nel registry l’oggetto remoto utilizzato dai client per la fase di registrazione; infine entra in un ciclo infinito in cui accetta nuove connessioni dai client, legge le richieste di questi ultimi processandole, per poi scrivere sul channel del client stesso il risultato della richiesta.
- **Thread main client;** una volta attivato imposta l’indirizzo IP del server e gli indirizzi di porta TCP, se specificati; reperisce l’oggetto remoto per la fase di registrazione; entra in un ciclo, che può essere interrotto dal comando “*--exit*”, dove attende che l’utente inserisca uno dei comandi di TURING.
- **Thread chat;** se una richiesta di edit da parte di un utente va a buon fine, si attiva questo thread che in un ciclo infinito, riceve i messaggi inviati sulla chat del documento editato, e li mette nella lista dei messaggi che l’utente deve ancora leggere.
- **Thread cleaner;** agisce periodicamente, liberando le sezioni dei documenti che sono in stato di EDIT da un tempo maggiore rispetto a quello consentito, rimettendole in stato di STARTED. Questo thread evita che il comportamento inaspettato da parte di un utente non renda più disponibile la sezione che stava editando.

2.2 Strutture dati utilizzate

Il server utilizza una HashMap per memorizzare gli utenti registrati, che ha come chiave una stringa reppresentante il nome dell’utente in quanto unico, e come valore un oggetto istanza della classe User. Il server utilizza un’altra HashMap per la memorizzazione dei documenti creati dagli utenti, che ha come chiave una stringa rappresentante il nome del documento in quanto unico, e come valore, un oggetto istanza della classe Document. La classe remota RegisterMethodImplementation**,** oltre ad avere un riferimento alla HashMap degli utenti registrati, utilizza un’altra HashMap dove per ogni utente che entra in stato di LOGGED memorizza un oggetto remoto necessario per notificare l’invito all’editing di un documento. I dati memorizzati da TURING sono relativi agli utenti e ai documenti creati dagli stessi. Di seguito i dettagli delle informazioni memorizzate per un utente e per un documento:

- **Utente;** password, stato in cui si trova in un dato istante (STARTED, LOGGED, EDIT), lista di notifiche da mandare all'utente quando torna in stato di LOGGED, il timestamp relativo all'ultima volta che è andato online.
- **Documento;** nome dell'utente creatore, numero di sezioni del documento, elenco degli utenti invitati, lista di informazioni della sezione, quindi se la sezione è in stato di EDIT il nome dell'utente che la sta' editando e il timestamp relativo all'ultima volta che la sezione è andata in stato di EDIT.

2.3 Gestione della concorrenza

Nel server, gli unici punti critici per la concorrenza si hanno nell'aggiornare gli elenco degli utenti registrati e dei documenti creati. Per garantire un accesso sincrono a queste strutture dati ho utilizzato dei blocchi di codice "synchronized", passando i riferimenti degli elenchi. Un'altra criticità per la concorrenza si ha nell'accedere e aggiornare l'elenco dei messaggi che un utente deve ancora leggere, in quanto questa struttura dati è aggiornata sia dal thread main del client che dal thread chat di un documento. Anche in questo caso mi sono servito di blocchi di codice "synchronized", passando il riferimento dell'elenco.

3 Descrizione delle classi

Chat

Offre metodi al client per spedire e ricevere messaggi. In oltre contiene il task da far eseguire al thread chat, e un metodo per abbandonare la chat del documento.

Cleaner

Contiene il task per il thread cleaner.

Document

Modella le informazioni di un documento e offre metodi per ottenere e gestire tali informazioni.

MainClassTuringClient

Classe principale del client: entra in un ciclo in cui si mette in attesa di richieste da parte dell'utente, per poi inviarle al server.

MainClassTuringServer

Classe principale del server: inizializza tutte le strutture dati utilizzate per memorizzare le varie informazioni; crea ed esporta l'oggetto remoto; accetta, gestisce e risponde alle varie richieste dei client.

MulticastArray

Classe utilizzata per gestire gli indirizzi di multicast dei documenti. Offre, in oltre, il metodo per ottenere un indirizzo di multicast libero.

Notify

Interfaccia per il metodo remoto utilizzato per notificare l’invito a un utente in stato di LOGGED.

NotifyImplementation

Implementazione dell’interfaccia precedente.

RegisterMethod

Interfaccia remota per la registrazione a TURING. Espone il relativo metodo per la registrazione a quest’ultimo e altri metodi per registrarsi e deregistrarsi al servizio di notifica tramite callback RMI degli inviti.

RegisterMethodImplementation

Implementazione dell’interfaccia precedente.

Request

Implementa il formato delle richieste che il client invia al server di TURING. Il formato delle richieste è il seguente: *request//__//userName//__//[…]* dove la parte opzionale dipende dal tipo di richiesta che il client deve fare al server. Offre inoltre metodi statici per creare richieste predefinite.

Result

Implementa il formato delle risposte che il server manda al client in seguito a una richiesta. Le risposte predefinite sono: *result//__//ok, result//__//fail* utilizzate per indicare rispettivamente una richiesta processata con successo o insuccesso.

State

Contiene due enumerazioni per indicare lo stato di un utente e di un documento, come suggerito dall’automa dell’appendice B.

User

Modella le informazioni di un utente e offre metodi per ottenere e gestire tali informazioni.

UtilsClass

Offre variabili globali per reperire porte e indirizzi predefiniti dei vari servizi offerti da TURING. Contiene inoltre variabili per la gestione delle chat dei documenti, e altre variabili di utilità.

WriteAndRead

Classe che implementa lo scambio vero e proprio delle richieste e delle risposte tra il client e il server di TURING mediante protocollo TCP.

4 Scelte implementative

Di seguito andrò a esporre le principali scelte implementative delle funzionalità di TURING.

4.1 Registrazione

Poiché lo username dell'utente è univoco questo viene controllato in fase di registrazione. Come suggerito dall'automa dell'appendice B, si possono registrare nuovi utenti solo se si è in stato di STARTED.

4.2 Login e logout

Una volta effettuato il login, il server invia le notifiche degli inviti ricevuti se ce ne sono stati, e l'utente si registra al servizio di callback RMI per ricevere le nuove notifiche. Quando si effettua il logout, invece, l'utente si deregistra al suddetto servizio di callback.

4.3 Creazione di un documento

Se l'utente crea un documento con un numero di sezioni minore o uguale a zero, TURING crea il nuovo documento utilizzando il numero di sezioni di default, specificato nella variabile *UtilsClass.*`DEFAULT_NUM_SECTIONS`. Nel server il nome del documento diventerà il nome di una directory, che al suo interno avrà tanti file quanti il numero di sezioni specificate. I nomi dei file saranno dati dal numero di sezione.

Es. Se si esegue il comando "*create Prova 3"* all'interno di TURING, come risultato si otterrà che all'interno del server verrà creata una cartella chiamata "Prova" contenente i file denominati 0, 1, 2;

Prova\0.txt

Prova\1.txt

Prova\2.txt.

4.4 Gestione inviti e notifiche

Quando un utente esegue una login o una end-edit riceve l'elenco dei documenti a cui è stato invitato alla modifica. Se un utente riceve un invito quando è in stato di LOGGED, l'invito gli viene notificato subito mediante callback RMI. Mentre se un utente non è in stato di LOGGED, e riceve un invito, la notifica sarà aggiunta all'elenco delle notifiche non ancora ricevute.

4.5 Edit, end-edit

La end-edit rende permanente la modifica da parte di un utente su una particolare sezione. Nel caso in cui si verificasse un qualsiasi evento che porti all'inaspettata interruzione del processo su cui è in esecuzione il client, mentre l'utente sta editando una sezione di un documento, quest'ultima ritornerà disponibile, in quanto il server implementa un sistema di cleaning: un thread in un ciclo sempre attivo dopo aver dormito per UtilsClass.*SLEEPING_TIME* si sveglia e iterando sui documenti individua le sezioni che sono in stato di EDIT da più tempo di *UtilsClass.MAX_EDIT_TIME*, per poi liberare la stessa e mettere l'utente in stato di STARTED.

4.6 Chat

Gli indirizzi di multicast, ai quali si appogga la chat dei documenti, sono stati scelti staticamente. Sono 256 indirizzi che vanno dal 230.0.0.0 al 239.0.0.255. Questi sono nel blocco degli indirizzi destinati all'uso locale e quindi non assegnati ad altri servizi.

5 Istruzioni per l'uso di TURING

L'applicazione TURING è stata sviluppata in ambiente Windows, per tanto non è garantito il corretto funzionamento della stessa applicazione in ambienti differenti. È possibile personalizzare alcune variabili di utilità nella classe *UtilsClass,* per maggiori dettagli consultare il codice della suddetta classe.

5.1 Compilazione

Per compilare il modulo relativo al server, spostarsi da terminale Windows nella directory TURING\Server e lanciare da riga di comando: *javac *.java*. Per compilare il modulo relativo al client, ripetere i passi precedenti nella directory TURING\Client.

5.2 Esecuzione del server

Assicurarsi prima di aver eseguito correttamente i passi del capitolo 5 paragrafo 1. Spostarsi da terminale Windows nella directory TURING\Server e lanciare da riga di comando:

java MainClassTuringServer [-sp server_port] [-srp service_register_port].

La parte opzionale è la seguente:

- *server_port* : int ; imposta la porta TCP su cui il server accetta connessioni dai client.
- *service_register_port* : int ; imposta la porta TCP su cui il server esporta l'oggetto remoto per la registrazione.

Se non si specifica la parte opzionale, verranno utilizzate le impostazione di default definite dalle variabili globali nella classe *UtilsClass.*

5.3 Esecuzione del client

Assicurarsi prima di aver eseguito correttamente i passi del capitolo 5 paragrafo 1 e che il server sia correttamente in esecuzione. Spostarsi da terminale Windows nella directory TURING\Client e lanciare da riga di comando:

java MainClassTuringClient [-sa server_address] [-sp server_port] [-srp service_register_port].

La parte opzionale è la seguente:

- *server_address* : stringa ; imposta l'indirizzo IP dove è in esecuzione il server.
- *server_port* : int ; imposta la porta TCP dove è in esecuzione il server.
- *service_register_port* : int ; imposta la porta TCP su cui il server ha esportato l'oggetto remoto per la registrazione.

Se non si specifica la parte opzionale, verranno utilizzate le impostazione di default definite dalle variabili globali nella classe *UtilsClass.*

5.4 Esecuzione degli script di test