Relazione Turing

Raffaele Ariano 530519

Gennaio 2019

1 Introduzione

Il progetto è costituito da un server che esegue multiplexing dei canali tramite NIO; i client vi si connettono e chiedono l'esecuzione delle operazioni prendendo gli input attraverso uno scanner che legge i dati passati da tastiera. Le comunicazioni tra client e server (tranne per la registrazione) avvengono esclusivamente attraverso canali con bytebuffer. I file di testo vengono salvati in Cartella_Clients e Cartella_Server. Cartella_Clients è organizzata in sottocartelle, ognuna per ogni utente registrato.

2 Interazione client-server

Una volta eseguito il client (non sono necessari argomenti da passare prima dell'esecuzione) si attiva uno scanner ed è possibile iniziare ad interagire con Turing semplicemente scrivendo da tastiera. Il client parsa il primo argomento scritto e lo confronta, con un case-switch, con tutti i comandi implementati. Per identificare un operazione tra client e server si sfrutta un intero (sarà il primo intero inserito inserito all'interno del bytebuffer), quest'intero sarà generato grazie la classe "Operazioni": questa classe contiene un enum con tutte le possibili operazioni e utilizzando il metodo ordinal() di java.lang.Enum creo l'intero menzionato. I restanti valori inseriti nel bytebuffer sono il nome dell'utente loggato seguito dagli altri argomenti a seconda dell'operazione richiesta. Dopo aver passato i dati al server mi aspetto un intero di risposta: questo può indicare un possibile successo, un successo con notifica o un fallimento. Questo intero (in caso di successo) potrà essere seguito da altri dati necessari per il completamento dell'operazione. Il Server prende dati dal canale attraverso un bytebuffer: il primo dato letto sarà l'intero che indica l'operazione da eseguire seguito dal nome dell'utente che la esegue. L'intero, attraverso uno switch-case, permetterà l'esecuzione del codice relativo al comando richiesto. Al client verrà successivamente inviata una risposta contenente un intero più, opzionalmente, altri dati.

3 Informazioni Client

Il client mantiene alcune informazioni sullo stato dell'utente attualmente loggato: il path della sua cartella, il suo stato (loggato o in editing) , il suo nome, il file eventualmente in editing, il numero di sezioni che sta editando e l'eventuale indirizzo ip per la chatroom. Queste informazioni mi servono per eseguire controlli prima di fare un eventuale richiesta al server (esempio: non posso fare una richiesta se non c'è nessuno loggato, non posso creare un file se sono in editing, non posso inviare un messaggio nella chat se non sto editando nulla, etc.) Per ogni client posso avere al massimo un utente loggato. Se un client non è nello stato loggato può registrare infiniti utenti. Posso chiudere il client con il comando exit se non sono loggato con nessun utente.

4 Informazioni Server

Il server, come già accennato, è un multiplexed server e utilizza una select per selezionare i channel con cui comunicare. All'interno del server mantengo le seguenti informazioni: tutti i documenti creati da quando il server è stato attivato, la lista degli utenti che devono ricevere una notifica, un oggetto che rappresenta un controllore per gli utenti registrati e/o online, un indirizzo ip da assegnare alla chatroom del prossimo documento creato. Grazie alla select scorro tutto il Selected-key Set e per ognuna di queste chiavi eseguo un operazione di lettura o di scrittura. Lettura: leggo l'operazione richiesta e gli eventuali dati aggiuntivi e in base a questi associo alla key un attachment e passo la key nel set delle chiavi pronte in scrittura. Scrittura: in base all'attachment della key il server decide che risposta inviare al client scrivendo sul canale associato alla chiave.

5 Altre classi

- Documento: viene creata un istanza di questa classe in concomitanza alla creazione di un nuovo documento; serve per mantenere informazioni relative al proprietario, collaboratori, indirizzo ip per il multicast e sezioni in editing. I metodi sono utili per fare controlli sulle proprietà del documento.
- IPAddress: classe, rinvenuta su stackoverflow, utile per convertire stringhe che rappresentano ip in interi e viceversa. Utilizzata per generare gli indirizzi ip per le chat.
- Utente: classe utile per tener traccia delle associazioni tra utenti e documenti. Per ogni utente all'interno c'è un ArrayList con tutti i documenti che quell'utente può modificare.
- UtentiRegistrati: classe che implementa l'interfaccia remota per la registrazione con anche, oltre il metodo per registrare, altri metodi di controllo,

visibili solo dal lato server, come quelli per il login e per il logout.

6 Inviti

L'eventuale invito viene inserito in fase di scrittura dal server, sotto forma di stringa, se l'utente a cui devo inviare la risposta si trova nella lista degli utenti con notifiche pendenti. La risposta inoltre conterrà un intero necessario a comunicare al client di stampare la stringa con l'invito. La notifica dell'invito sarà quindi disponibile all'utente dopo un operazione eseguita con successo.

7 Registrazione

La registrazione è fornita tramite RMI, il client può accedere a un solo metodo della classe UtentiRegistrati per registrare un utente. Con quel metodo varrà salvata in una ConcurrentHashMap l'associazione tra utente e password e in un'altra quella tra l'utente un oggetto della classe Utente.

8 Comandi

- register "utente" "password": viene chiamato il metodo "register" di UtentiRegistrati e viene creato un nuovo oggetto Utente e viene salvata l'associazione tra utente e password.
- login "utente" "password": se con quel client non c'è ancora nessun utente loggato, quest'operazione permette di fare il login; il server esegue dei controlli sull'esistenza dell'utente e sulla correttezza della password in UtentiRegistrati e se questi hanno successo lo comunica al client e salva l'utente nella lista degli utenti online.
- logout: il client invia la richiesta di fare il logout al server, il quale eliminerà l'utente dalla lista degli utenti online.
- create "nomefile" "numsezioni": chiede al server di creare un nuovo file con nome "nomefile" con "numsezioni" sezioni, in caso di esito positivo il server crea un numero di file pari a "numsezioni" in CartellaServer, un nuovo oggetto Documento e salverà l'associazione tra il documento e l'utente nell'oggetto Utente.
- list: viene generata e spedita una stringa dove sono indicati tutti i file che l'utente può modificare più informazioni sul proprietario e i collaboratori.
- showd "nomedoc": l'utente chiede la spedizione di tutte le sezioni del documento "nomedoc" più una stringa finale contente il numero delle sezioni attualmente in modifica.

- shows "nomedoc" "numsezione": come sopra ma viene inviata solo la sezione "numsezione" del file "nomedoc" più un intero per indicare se la sezione è in modifica o meno.
- invite "nomedoc" "utente": aggiungo l'utente ai collaboratori del mio file "nomedoc", il server crea un'associazione tra l'oggetto Documento associato a "nomedoc" e l'utente invitato.
- edit "nomedoc" "sezione": l'utente chiede di poter editare una sezione di "nomedoc", l'utente può editare quella sezione se è un file per cui è collaboratore o proprietario e se non sta già editando documenti diversi da "nomedoc". Il server all'interno dell'oggetto Documento setta il booleano associato a quella sezione a true segnando anche l'utente che lo sta modificando e, subito dopo, invia il file al client, il quale, penserà a salvarlo all'interno della cartella corrispondente a quella all'utente richiedente; a questo punto l'utente potrà modificare il file con lo strumento di editing che più l'aggrada. Il nome del file di editing è uguale al nome del file normale con "editVersion" alla fine.
- end_edit "nomedoc" "sezione": il client invia la "editVersion" del file al server il quale lo sovrascriverà all'interno della propria cartella "CartellaServer" e 'sbloccherà' la sezione risettando il booleano a false.
- send "mess": invia la stringa "utente: mess" al server, il quale la invierà a sua volta alla chat associata al documento che l'utente sta modificando.
- receive: leggi tutti i messaggi ancora non visualizzati.
- exit: chiude il client; funziona solo se non c'è nessun utente attualmente loggato.

9 Conclusione e considerazioni finali

Il progetto è stato sviluppato interamente con Eclipse su Windows 10. Le slide e le soluzioni degli esercizi del corso sono stati sufficienti per completare la gran parte di Turing. Tutto il progetto è commentato e sono stati utilizzati nomi di variabili esplicativi rispetto la funzione che compiono.