WINSOME

A rewarding social media.

Progetto di fine corso A.A. 2021/22

Alberto Baldini

13 October 2022

Indice

Classi

Client

La classe Client gestisce le connessioni con il Server, tiene traccia dello stato corrente e mette a disposizione metodi per interagire con esso.
Per comunicare con il Server, si avvale di un' altra classe "WinsomeServerSender" che si occupa di inviare/ricevere i dati opportunamente formattati.
Il Client, una volta effettuato il Login, si registra ad un servizio di notifiche del Server tramite RMI e mantiene delle informazioni locali dell'utente collegato.

WinMessage

La classe WinMessage viene utilizzata per scambiare messaggi con il server tramite una connessione TCP con SocketChannel. La ricezione e l'invio cambiano leggermente a seconda della natura della SocketChannel (bloccante o meno).
Il messaggio viene inviato con ByteBuffer : al momento dell'invio si calcola la dimensione del messaggio e si invia prima un intero contenente la dimensione, e successivamente il messaggio vero e proprio.
Il messaggio può essere composto da più parti.
Un esempio di scambio di messaggi client-server e server-client:
<***LOGIN***>;<USERNAME>;<PASSWORD>

Il client fa una richiesta di LOGIN con un dato USERNAME e PASSWORD

<***SUCCESS***> oppure <***ERROR***>;<Username <USERNAME> not found>

Il server risponde con SUCCESS se l'operazione e andata a buon fine, altrimenti ERROR, e allega anche un messaggio leggibile dall'utente per comprendere la natura dell'errore

Le stringhe SUCCESS, LOGIN e ERROR sono alcune delle costanti rese disponibili dalla classe per rendere il codice più leggibile e mantenibile.

Server

La classe Server gestisce le connessioni persistenti con i Client, rende disponibile un'interfaccia RMI per servizi specifici e comunica con il Database di Winsome per ogni operazione inerente.
Il server gestisce le connessioni attraverso un selettore che controlla la ServerSocketChannel del server ed ogni SocketChannel creata per comunicare con il Client.

Quando un nuovo Client richiede una connessione dal server, viene inserito nel selettore, in attesa di ricevere un messaggio. Una volta ricevuto, viene creato un nuovo Task addetto all'elaborazione della richiesta. Il Task si occupa di soddisfare una singola richiesta del client. Una volta elaborata, il client viene reinserito nel selettore con la risposta in "allegato". Appena la SocketChannel sarà pronta per la scrittura, il server invierà la risposta al Client.

WinsomeDatabase

Il Database si occupa di mantenere le informazioni persistenti del sistema, quindi tutto ciò che riguarda utenti, post e ogni altro elemento della rete sociale.
Il database salva tutte i dati in due file JSON separati, uno contente gli utenti, ed un altro contenente i post.
Quasi tutte le operazioni messe a disposizione dal server provocano un cambiamento nel database che periodicamente salva sui JSON le collezioni "sporcate" nell'ultimo arco di tempo prestabilito.
Il Database ha il solo controllo sui dati del sistema, tutto ciò che ritorna sono rappresentazioni dei dati contenuti in esso.
Tutte le classi che il DB memorizza nei file JSON implementano l'interfaccia JSON_Serializable.

User

La classe UserDB memorizza tutte le informazioni che riguardano un utente della rete sociale. Contiene:

- Il portafoglio dell'utente.
- La lista degli ID dei post pubblicati (o condivisi) nella bacheca dell'utente.
- Username.
- Password.
- I tag degli interessi.
- Gli username che l'utente segue.
- Gli username che seguono l'utente (questo per rendere più veloci le query).

Questa classe e' utilizzata solamente dal database.

Post

Analogamente alla classe UserDB, la classe PostDB contiene le informazioni riguardanti il post, tra le quali:

- ID
- Username dell'autore del post
- Titolo
- Contenuto del Post
- I voti ricevuti
- Commenti
- Timestamp di creazione
- + altre informazioni utili al calcolo delle ricompense

Questa classe viene utilizzata solamente dal database. Quando il server richiede un post da trasmettere al client, il database restituisce una classe che lo rappresenta esponendo solo i dettagli utili al client.

UserInteraction

Questa e' una classe astratta utilizzata come base di ogni classe che identifica una interazione di un utente. Le implementazioni disponibili sono Post, Rate, Comment. Al momento della creazione di un nuovo oggetto UserInteraction viene registrato il timestamp del tempo corrente, e l'autore dell'interazione.

WinSerializable

Altra classe astratta che implementa i metodi serialize e deserialize.
Serve principalmente per ottenere una stringa rappresentante l'oggetto che si vuole trasmettere ai client e per poter ricostruirlo dallo stesso al momento della ricezione.
Per la serializzazione viene usata la libreria Jackson.

WinsomeException

Questa classe astratta viene implementata da ogni eccezione gestita dal server e rende disponibile un metodo niceMessage che restituisce un messaggio leggibile dall'utente del client che il server può restituire in caso di errore.

Connessioni

TCP

Client e Server comunicano principalmente tramite un sistema di connessioni TCP gestite con SocketChannel NIO.
Il Server adotta canali non bloccanti, mentre il Client ne usa uno bloccante.
In questo modo il Server può rispondere ai client quando non e' impegnato ad accettare nuove connessioni, se non ci sono risposte da inviare controlla se tra i Client connessi ci sono nuove richieste disponibili da inviare ai workers.
Il Server legge la porta TCP dal file di configurazione, mentre il Client legge sia la porta che l'indirizzo.

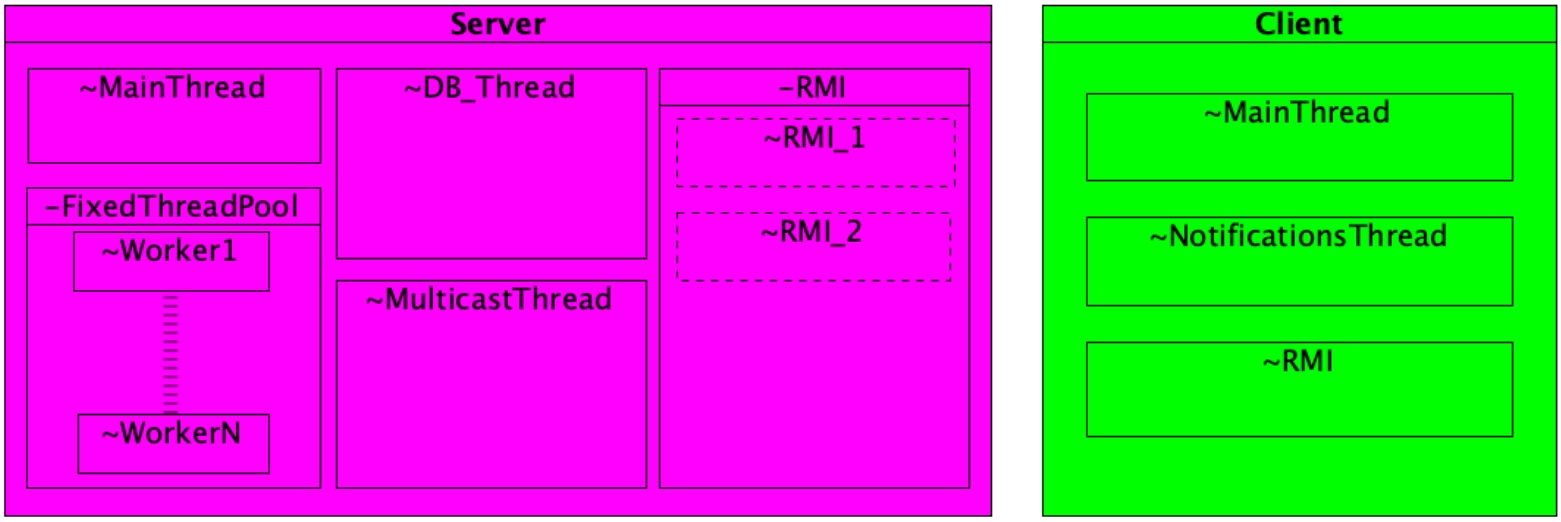
RMI

Il Server mette a disposizione dei Client una interfaccia RMI (i dettagli sono disponibili al Client tramite file di configurazione) che espone metodi specifici per la registrazione di nuovi utenti , per l'iscrizione al servizio di notifiche dei follower e per ricevere dettagli sulla connessione UDP.
Il Client a sua volta espone un'interfaccia RMI tramite l'invio di un oggetto RMI Callback, inviato al Server al momento del login.
Il Server memorizza indirizzo, RMI Callback e username di ogni utente attualmente connesso alla rete in una collezione dedicata.

MULTICAST UDP

Il Server crea un gruppo multicast per l'invio di notifiche ai Client in caso di aggiornamento dei portafogli degli utenti.
L'intervallo del calcolo delle ricompense e' letto dal file di configurazione.
Il Client, una volta che un utente ha effettuato il login, si mette in ascolto sul gruppo multicast e notifica l'utente una volta ricevuto il messaggio di aggiornamento.
Se l'utente effettua il logout, il Client smette di ascoltare e rimuove eventuali notifiche registrate.

Thread e Concorrenza



Organizzazione dei Threads

Il Server utilizza un FixedThreadPool di dimensione N fissata dal file di configurazione che permette l'esecuzione di N o meno worker tasks.
Il MulticastThread calcola periodicamente le ricompense e fa la broadcast di un messaggio di notifica ai client.
Il Server espone un'interfaccia RMI con metodi che possono essere invocati da diversi client, ciò provoca l'attivazione di più thread per gestire le richieste.
Il DB_Thread serve per salvare periodicamente il database sui file JSON permanenti.
Il Client espone a sua volta un'interfaccia RMI e usa un thread (NotificationsThread) per controllare se ci sono nuove notifiche dal Server.

Gestione Concorrenza

Per il Server le sezioni critiche coinvolgono il Database, il quale si avvale di 2 coppie di RW_Lock per sincronizzare le operazioni (una per il database di utenti, una per i post).
Lato Client troviamo delle sezioni critiche solo sui metodi esposti da RMI che potrebbero essere chiamati da più worker contemporaneamente.

Utilizzo

File di Configurazione

Un esempio di file di configurazione per il server (default “server_config.txt”):

```
POSTS_DATABASE=posts_backup.json          [file di backup per I post]
USERS_DATABASE=users_backup.json          [file di backup per gli utenti]
WORKERS=10                                [N worker in ThreadPool]
MULTICAST_ADDRESS=224.0.1.1
TCP_PORT=8080
MULTICAST_PORT=8000
REGISTRY_PORT=1099
REWARD_TIME=1   [intervallo di aggiornamento delle ricompense (in minuti)]
RMI_NAME=Server_registration_RMI
```

Nell’esempio sono specificati i valori di default del file di configurazione. Se il server al momento dell’avvio non legge valori per questi campi, li imposta con i valori di default. Il Client legge da un file di configurazione (default “client_config.txt”), per il progetto quest’ultimo viene creato dal server al momento dell’inizializzazione per poter scrivere dati coerenti alla corrente esecuzione.

Comandi

Tutti i comandi richiesti nel progetto sono implementati come da specifica, in aggiunta i comandi “help” e “exit” stampano un messaggio con la legenda dei comandi disponibili e chiudono il programma client.
Per compilare il progetto basta recarsi nella cartella “compile_and_run” ed utilizzare i target makefile:

- *all* : per compilare il progetto
- *run_client* : per avviare il client nella Shell corrente
- *run_server* : per avviare il server in background nella Shell corrente

Altrimenti possiamo compilare il progetto (sempre dalla cartella “compile_and_run”) tramite comando javac:
javac -d ../bin -cp @compiler_classpaths @compiler_classes

Per avviare il Server utilizzando il file Server.jar disponibile nella cartella “bin”:
java -cp @classpaths_server winsome_server.ServerMain <server_conf_filename> <client_conf_filename> (*&> server.log &)**
** per l’avvio in background del processo server*
Se non vengono specificati argomenti per il programma Server, i valori di default per i file di configurazione (server_config.txt & client_config.txt) vengono utilizzati.

Per il Client il comando e’ analogo al Server
java -cp @classpaths_client winsome_client.ClientMain <client_conf_filename>
Se non viene specificato il nome del file di configurazione tramite argomento, il valore di default (client_config.txt) viene utilizzato.