Winsome – progetto laboratorio di reti 2021/22

Alessandro Querci - 578615

Indice

1. Note sulla compilazione ed esecuzione

Il progetto è stato sviluppato e testato principalmente con IntelliJ su windows, ma anche testato e compilato su linux.

Per compilare il progetto è necessario almeno Java 16, le due jar fornite sono state compilate con Java 17 (LTS).

Per compilare il progetto (su linux) si deve eseguire il seguente comando:

javac client common* server\data*.java server*.java -cp "lib*" -d .\build*

Sono anche inclusi i file manifest per creare le jar e due script .sh per compilare, creare jar e avviare il programma corrispondente su linux.

I file di configurazione per i due programmi sono contenuti nella directory config, nel caso in cui siano rimossi essi verranno rigenerati con valori di default all'avvio, gli effetti di ogni opzione di configurazione sono illustrati nel cap. 7.

I dati relativi agli utenti verranno salvati in saved_data/users.

2. Sintesi sull'architettura del progetto

Come da specifica il progetto si divide in due programmi, un Client e un Server.

Il Client, come richiesto “thin”, si occupa principalmente di mettersi in contatto con il Server, leggere gli input dell’utente, fare un controllo preventivo sul numero degli argomenti, tradurla in un codice operazione e incapsulare la richiesta per inviarla al Server e stamparne l’esito.

Il Server si occupa di gestire la permanenza dei dati, serializzati in file json, gestire le richieste da multipli Client e calcolare periodicamente le ricompense in wincoins per gli utenti.

Client e Server hanno in comune solo il package common, che contiene le interfacce Proxy e la struttura dati Triplet utilizzata per le richieste del client e l’esito del login/registrazione.

Come esito all’autenticazione il Client riceve una tripla contenente il suo token utente e le coordinate per mettersi in ascolto sul gruppo multicast (indirizzo e porta).

Il token utente verrà poi utilizzato per le richieste nella tripla < token utente, codice operazione, argomenti> da serializzare ed inviare al Server in modo da riconoscere l’utente ed eseguirla.

Le operazioni possibili sono quelle richieste dalla specifica, con l’aggiunta di “help” per stampare un messaggio con i possibili comandi e di “shutdown” per terminare l’esecuzione del client e del server.

Il client attiva i seguenti thread:

- Main
- RMI
- MulticastListener (SingleThreadExecutor)

Il server attiva:

- Main
- RMI
- Thread Wokers (CachedThreadPool)

3. Strutture dati utilizzate

Il package server.data contiene le classi utilizzate per memorizzare i dati durante l’esecuzione.

In IWinImpl è presente la map userMap, che indicizza gli utenti per il loro username, inizializzata con i dati salvati. Ogni utente contiene una lista dei post appartenti al suo blog, ed ogni post contiene i rispettivi commenti e voti.

In aggiunta, la map userIdLookup traduce gli userToken nell’username dell’utente corrispondente e la map postLookup traduce i postId nel riferimento al post per avere un accesso rapido e diretto.

Dato che queste tre map sono static e utilizzate dai thread worker, sono implementate con ConcurrentHashMap.

Per lo stesso motivo, le liste e i set utilizzati nelle classi sottostanti sono implementate da CopyOnWriteArraySet/List per thread-safety.

3.1 Utente

Il record Utente contiene le informazioni relative a un singolo utente registrato, il cui username è univoco ed utilizzato per la equals.

- String username – username dell'utente, il nome visualizzato agli altri utenti per riconoscerlo e interagire con lui (es. follow/unfollow). Univoco, può essere lungo al massimo 20 caratteri, non vuoto.
- String password – password dell'utente, per eseguire login. Nessun controllo eccetto la lunghezza (non può essere vuota né più lunga di 20 caratteri) viene eseguita sulla password (es. crittografia o sicurezza).
- Set<String> tags – tag relativi all'utente, possono essere al massimo cinque e almeno uno. I tag vengono convertiti in lowercase prima di raggiungere il server. L'utilizzo di un set garantisce che non siano duplicati fra di loro.
- Set<Post> blog – set dei post presenti nel blog dell'utente. Ogni chiamata al getter del set controlla che i post di cui l'utente non è autore siano presenti nella postIdLookup, rimuovendolo se necessario, prima di restituire l'oggetto, in caso un post rewinnato sia stato eliminato dal suo autore originale.
- Set<String> followedUsers – set contenente gli username degli utenti seguiti.
- List<Transaction> wallet – lista di <float, timestamp> che elenca le transazioni wincoins dell'utente. La classe ha metodo per calcolare il totale del saldo partendo dalla lista.

3.2 Post

Il record Post contiene le informazioni relative ad un post ed ha come identificatore univoco il campo idPost, utilizzato per la equals e la compare.

IdPost è rappresentato da un int anziché un UUID per non complicare l'utilizzo della CLI all'utente.

In quanto record non sono necessari getter/setter espliciti. Questo facilita la serializzazione della classe, tuttavia rende anche tutti i suoi campi final portando al compromesso di incapsulare il wallet in un wrapper per incrementarlo.

- Int idPost – id del post, univoco. Il contatore utilizzato per l'univocità assume il valore dell'id massimo tra tutti i post ripristinati dai file e viene incrementato ogni volta che un post viene creato.
- String author – username dell'autore del post.
- List<Comment> comments – lista dei commenti al post.
- Set<Rating> ratings – lista dei voti al post, implementata con un set per evitare votazioni multiple dello stesso utente.
- IntWrapper - timesChecked – wrapper di un int che rappresenta il numero di volte che il post è stato esaminato per le ricompense in wincoins. La varibile del wrapper è volatile e getter e setter sono syncronized per thread-safety.

Le operazioni su comments e ratings vengono eseguite attraverso metodi syncronized della classe Post, quali aggiungere/rimuovere un commento o un voto, oppure ottenere il conteggio dei voti positivi e negativi totali.

La classe post contiene anche un metodo per formattare il post in una stringa, che sarà inviata al client.

3.3 Comment e Rating

Il record Comment contiene le informazioni relative ad un commento, ovvero il suo autore, il suo contenuto e il timestamp di creazione, utilizzato per il calcolo punti.

Il record Rating contiene le informazioni relative alla votazione di un post, ovvero l'username dell'utente, il timestamp di creazione e il voto.
Il metodo equals è sovrascritto per comparare l'username dell'utente, utilizzato dal Set per garantire che in un post non siano presenti più voti dallo stesso utente.
Rating contiene anche un record minore Pair, utilizzato per resituire una coppia <voti positivi totali, voti negativi totali>.

4. Il Client

All'avvio il Client legge il suo file di configurazione e lo utilizza per ottenere il ServerProxy tramite il registro RMI, esportato dal server. Dopodichè viene stabilita una connessione TCP con il server e si mette in attesa di input da parte dell'utente.

Finchè l'utente non è autenticato, può unicamente utilizzare i comandi help, login e register per autenticarsi, o logout e shutdown per chiudere il programma.

La register avviene tramite il proxy ottenuto con RMI, mentre il login avviene attraverso la connessione tcp aperta. Se l'autenticazione ha successo, il server restituisce una tripla <userToken, multicastPort, multicastAddress> contenente il token che autentica il client e le coordinate per mettersi in ascolto sul gruppo multicast di notifica delle ricompense.
Dopodichè il client si registra per le callback di aggiornamento dei follower inviando il proprio ClientProxy al server tramite il ServerProxy, ed inizializza la propria cache dei follower.

Una volta che le coordinate multicast sono state salvate, il client aprirà una MulticastSocket per connettersi al gruppo di multicast. Successivamente un SingleThreadExecutor attiverà un thread che si mette in ascolto sulla socket, restituendo un oggetto Future<String>. Questo oggetto conterrà il messaggio di multicast quando sarà ricevuto. Il thread termina alla ricezione del messaggio e il Client provvederà a riattivarlo dopo aver visualizzato il messaggio.

Gli input dell'utente sono gestiti nella classe CommandParser: viene controllata la prima parola (e le successive se necessario) dell'input e tradotta in un codice operazione. Viene anche controllato che il numero di argomenti sia al minimo quanto richiesto dall'operazione individuata. Se il numero di argomenti è inferiore, oppure non viene trovato un comando corrispondente, il client restituisce un errore senza inviare dati al server. Infine, separa il comando (ora tradotto in numero) dagli argomenti e incapsula il tutto in una tripla <userToken, opCode, args> da restituire al Main.

Con uno switch su opCode, se il codice non corrisponde a un errore o un'operazione locale (ex. help), si serializza e invia la Tripla al Server e si mette in attesa di stampare la risposta

5. Il Server

All'avvio il Server legge il suo file di configurazione e i dati salvati precedentemente, se presenti.

Se il ripristino ha successo, il server esporta la sua instanza di ServerProxy sul Registro RMI specificato nel file di configurazione e apre un Selector per gestire le connessioni TCP in arrivo.

Entra quindi in un loop dove controlla periodicamente se è necessario calcolare ed assegnare punti e poi controlla se ci sono connessioni da accettare. La select ha un timeout configurabile, per evitare di bloccare il timer del calcolo delle ricompense, essendo sul thread principale.

Quando una nuova connessione viene accettata, viene assegnato al SocketChannel un wrapper contenente il buffer e un id. L'id verrà successivamente utilizzato per recuperare l'oggetto Future<String> da restituire al Client richiedente.

Al momento della registrazione o del login di un utente viene generato un UUID per associare le richieste del client autenticato all'username corrispondente, che viene restituito insieme alle coordinate di multicast in una triplet.

Quando il Server riceve una richiesta da un Client, la Tripla viene deserializzata dalla stringa ricevuta e sottomessa ad un threadpool di IWinImpl tramite submit. Quando il Future relativo alla connessione sarà stato elaborato, il Server lo invierà al Client corrispondente.

I casi di logout e shutdown sono un'eccezione: se l'opCode della tripla è 0, la connessione con il client viene interrotta ed il suo token revocato, senza necessità di rispondere al Client.

IWinImpl è l'implementazione dell'interfaccia callable IWin, contiene i metodi necessari per gestire il social network come da specifica e le tre map per tenere traccia di utenti, post e sessioni attive. Contiene anche il counter per l'id dei post, è una variabile statica che viene inizializzata con l'id massimo dei post al momento del ripristino e viene incrementata con un metodo synchronized quando un post deve essere creato.

Il metodo call di IWinImpl controlla il codice operazione della tripla e converte la stringa con gli argomenti nei parametri da passare al metodo corrispondente al codice. Se il parsing degli argomenti fallisce o il codice operazione non ha un metodo corrispondente, viene restituita una stringa di errore.

5.1 – Calcolo Ricompense

Il calcolo delle ricompense avviene sul thread principale, quando il tempo passato dall'ultimo controllo eccede l'intervallo specificato nel file di configurazione.

Per ogni post presente nella postLookup, vengono calcolate le ricompense per l'autore ed i suoi curatori nella classe RewardsCalculator, il cui metodo getPointsFromPost prende come parametri

il post, il timestamp dell'ultimo controllo e la percentuale da dare all'autore del post, restituendo un set di CoinReward<user, amount> relativo al post. L'ammontare di punti totale viene calcolato utilizzando la formula data nelle specifiche e viene poi separato in punti autore e punti curatori, questi ultimi vengono infine divisi equamente fra i curatori.

Il contenuto del set viene poi inserito in una map di subtotali e quando ogni post è stato controllato viene aggiunta una transazione con il totale punti ottenuti e il timestamp corrente ad ogni utente interessato.

Aggiornati i wallet, il server invia tramite multicast un messaggio "WinCoins Awarded" e aggiorna il timestamp dell'ultimo controllo, per tornare a gestire le operazioni della selector.

La conversione dei punti in Bitcoins viene fatta utilizzando come ratio un numero decimale generato da random.org. Utilizzando le classi HttpRequest e HttpResponse viene inviata una richiesta di GET all'indirizzo www.random.org/decimal-fractions/ specificando i parametri per il generatore direttamente nell'url e richiedendo una risposta in formato plain. Se lo status code della risposta è 200, il body della risposta conterrà il numero decimale da utilizzare per la conversione.

6. Configurazione

All'avvio di entrambi i programmi vengono letti i file di configurazione clientConfig.yaml e serverConfig.yaml se presenti, oppure generati nuovi file con la configurazione di default.

La configurazione del Client consiste in:
ServerAddress: Indirizzo del Server TCP
TCPPort: Porta del Server TCP
RegHost: Host del registro RMI
RegPort: Porta del registro RMI

La configurazione del Server consiste in:

ServerAddress: Indirizzo del server TCP
MulticastAddress: Indirizzo del gruppo di multicast
TCPPort: Porta del server TCP
UDPPort: Porta per il Multicast
RegHost: Host del registro RMI
RegPort: Porta del registro RMI
AuthorReward: Percentuale di ricompensa dell'autore, deve essere compresa fra 0 e 1
PointsAwardInterval: Intervallo in secondi per il conteggio dei punti
selectTimeout: Timeout del selector

7. Proxy

Per gestire le callback e le operazioni di registrazione tramite RMI, Client e Server condividono le rispettive interfacce Proxy del package common, implementate poi nei rispettivi package.

ServerProxy fornisce metodi per registrare nuovi utenti e iscrivere o rimuovere una sessione alle callback di aggiornamento followers.

ClientProxy fornisce il metodo per aggiornare la cache locale dei followers.
ServerProxyIml gestisce l'elenco dei client registrati alle callback di aggiornamento dei followers, utilizzando una mappa <userToken, ClientProxy> per tenere traccia delle sessioni attive con rispettivi proxy.

Quando i comandi follow o unfollow vengono eseguiti per un utente, il ServerProxy controlla se ci sono ClientProxy in ascolto autenticati come l'utente da aggiornare. Se vengono trovati, viene inviata la nuova lista dei follower tramite proxy.

ClientProxyImpl gestisce la cache dei followers e quando riceve la callback aggiorna la propria copia dei followers con quella appena ricevuta.

Dato che la RMI utilizza thread multipli per gestire le richieste, i metodi dei proxy sono synchronized.

8. Salvataggio e ripristino dei dati

I dati degli utenti e relativi post vengono salvati nella directory saved_data/users. Ogni file contiene i dati relativi ad un utente, compresi tutti i post presenti nel suo blog.

Viene inoltre generato in saved_data un file di log relativo all'ultima esecuzione, che viene utilizzato anche per recuperare il tempo dell'ultimo conteggio punti.

I dati vengono ripristinati all'avvio del programma Server e inseriti nelle appropriate map. Il contatore degli id dei post viene aggiornato al valore dell'id maggiore, di modo da poter riprendere il conto da dove era rimasto.

Il salvataggio dei dati avviene durante lo shutdown del server, quando tutte le operazioni sono ormai concluse ma, dato che tutti gli accessi alle strutture dati condivise sono controllati da metodi syncronized o strutture thread-safe, sarebbe teoricamente possibile attivare un thread che avvia il salvataggio dei dati su intervallo regolare.