Relazione relativa al Progetto di Laboratorio di Reti

Nicola Palomba, mat. 597286, a.a. 2021/22

Compilazione ed esecuzione

Per compilare posizionarsi nella cartella src e usare i seguenti comandi:
```
javac -cp ".\lib\json-java.jar;.\lib\gson-2.8.9.jar;" *.java
```

Per eseguire il server:
```
java -cp ".\lib\json-java.jar;.\lib\gson-2.8.9.jar;"
WinsomeServerMain serverconfig.txt
```

Per eseguire il client:
```
java -cp ".\lib\json-java.jar;.\lib\gson-2.8.9.jar;"
WinsomeClientMain clientconfig.txt
```

Comandi:

- `help`: stampa il messaggio che descrive l'utilizzo dei comandi.
- `quit`: chiude il client ed eventualmente la connessione con il server.
- `register <user> <password> <tag1> [<tag2>][<tag3>][<tag4>] [<tag5>]`: registra un utente con il nome 'user', password 'password' e lista di tag <tagn>. Il primo tag è obbligatorio, gli altri sono opzionali.
- `login <user> <password>`: effettua il login dell'utente 'user' con la password 'password'.
- `logout`: effettua il logout dell'utente corrente.
- `list users`: mostra la lista degli utenti che condividono almeno un tag con l'utente loggato al momento.
- `list following`: mostra la lista degli utenti seguiti dall'utente loggato al momento.
- `list following`: mostra la lista dei followers dell'utente loggato al momento.
- `follow <user>`: permette all'utente correntemente loggato di seguire l'utente 'user', ricevendone gli aggiornamenti sul feed.
- `unfollow <user>`: permette all'utente correntemente loggato di smettere di seguire l'utente 'user', cessando di riceverne gli aggiornamenti sul feed.
- `post "<titolo>" "<contenuto>"`: crea un post avente come autore l'utente loggato al momento. Titolo e contenuto devono essere racchiusi tra doppi apici.
- `blog`: visualizza il blog dell'utente loggato, ovvero la lista di post da esso creati o rewinnati.

- `feed`: visualizza il feed dell'utente loggato, cioè la lista dei post creati o rewinnati dagli utenti seguiti.
- `rate <postId> <val>`: consente di valutare un post presente nel feed dell'utente loggato al momento. 'postId' + l'identificativo del post, mentre 'val' è il valore del voto: se negativo vale -1, altrimenti vale 1.
- `comment <postId> "<contenuto>”`: aggiunge un commento al post con id 'postId', il cui contenuto è 'contenuto'. Tale parametro dev'essere racchiuso tra doppi apici.
- `show post <postId>`: mostra in dettaglio il post identificato dall'id <postId>, se esso è nel feed o nel blog dell’utente.
- `delete <postId>`: elimina il post con id <postId>. Nel caso di rewin si elimina solo il rewin, altrimenti se il post era originale, si eliminano anche eventuali rewin, commenti e voti.
- `rewin <postId>`: effettua il rewin del post con id <postId>. Se si sta cercando di rewinnare un rewin, l'effetto e' quello di rewinnare il post originale.
- `wallet`: visualizza il wallet dell'utente loggato al momento, ovvero il totale di wincoin nel wallet e la lista delle transazioni indirizzate a quell'utente.
- `wallet btc`: converte il totale di wincoin nel wallet dell'utente in bitcoin, generando un tasso di conversione casuale tramite random.org.

Architettura generale

Il progetto si compone di due entità principali, il client e il server. I due comunicano con un protocollo richiesta-risposta inviandosi delle stringhe in formato JSON, contenenti i parametri delle richieste o i dati delle risposte (o informazioni riguardo eventuali errori nel caso la richiesta non fosse stata portata a termine).

Client

WinsomeClientMain

WinsomeClientMain è la classe contenente il main per il client. Il corrispondente eseguibile accetta come parametro il nome del file di configurazione. Il programma permette all’utente di inserire i comandi da linea di comando e di visualizzare il loro risultato sul terminale. Il client attua un processo di validazione preliminare dell’input dell’utente prima di inviare le richieste al server, così da evitare di sovraccaricarlo e da rendere il codice del server più pulito.

Eventuali errori dipendenti dall’input dell’utente, ma che il client non è in grado di rilevare per mancanza di informazioni (come ad esempio il tentativo di seguire un utente che già si segue), sono rilevati dal server e segnalati al client nella risposta JSON.

WinsomeClientMain implementa l’interfaccia IRemoteClient, che gli permette di essere notificato dell’aggiunta o della rimozione di un follower tramite RMI callback dal server.

Sempre parlando di RMI, WinsomeClientMain reperisce lo stub del server e lo usa per implementare il processo di registrazione (metodo signup).

RewardNotifier

Dal momento che il client è costantemente in ascolto dell'input dell'utente, la notifica di avvenuto calcolo delle ricompense da parte del server viene implementata in un thread separato, RewardNotifier.

Esso si occupa di connettersi al gruppo multicast specificato dal client (che ottiene i parametri nella risposta alla richiesta di login dell'utente) e di stare in ascolto per eventuali notifiche da parte del server. Alla ricezione di un pacchetto, RewardNotifier stampa a schermo un messaggio di notifica.

Server

WinsomeServerMain

WinsomeServerMain è la classe contenente il main per il server. Il corrispondente eseguibile accetta come parametro il nome del file di configurazione. Il programma si occupa di tenere traccia delle operazioni effettuate dagli utenti e dei dati da loro immessi tramite apposite strutture dati.

Il thread principale (WinsomeServerMain implementa Runnable) si occupa di accettare le richieste di connessione da parte dei client, di ottenere i loro input e di inviare loro eventuali risposte. In caso di errore di comunicazione, è anche in grado di chiudere le connessioni per evitare ulteriori problemi lato server.

Una volta ricevuta una richiesta da un client, essa viene inoltrata (insieme alla SelectionKey che identifica il client, entrambe racchiuse in una ClientRequest) a un WinsomeWorker, un oggetto che implementa Runnable e che è volto alla risoluzione di una singola richiesta. Il WinsomeWorker così creato viene inviato a un ThreadPool, che lo esegue secondo i parametri specificati al momento della sua creazione (e specificati nel file di configurazione).

Una volta che il WinsomeWorker ha risolto la richiesta, possono esserci tre tipi di esito: invio di un errore, invio di un acknowledgement nel caso di richieste che comportano solo la restituzione di un codice di errore (per esempio login o follow), invio di una stringa strutturata in formato JSON che rappresenta i dati richiesti dal client. Nei primi due casi ci si avvale della classe ComUtility, che crea e allega una stringa JSON di risposta alla SelectionKey del client, mentre nell'ultimo caso la stringa viene allegata dal worker.

WinsomeServerMain, quando rileva che un client è disponibile alla scrittura e che per tale client esiste un allegato da inviare (quindi la sua richiesta è stata risolta), procede all'invio dell'allegato usando ancora ComUtility.

Al momento della chiusura del server, viene eseguito un semplice shutdown hook, che permette di risolvere le ultime richieste presenti nel ThreadPool prima di chiudere tutte le connessioni rimaste con i client.

WinsomeWorker

I WinsomeWorker sono i thread che si occupano di gestire una singola richiesta, che viene specificata al momento della loro creazione assieme alla SelectionKey che identifica il client a cui deve essere inviata la risposta.

Al momento della creazione di un worker, viene anche passato il server che li gestisce: in questo contesto esso agisce come mero contenitore dei dati necessari ai worker per risolvere le richieste. Essi si occupano di gestire il problema di sincronizzare più operazioni atomicamente thread safe (ma non nel loro insieme) tramite l'utilizzo di monitor (si veda la sezione riguardante la sincronizzazione).

A ogni possibile richiesta del client corrisponde un codice di operazione (uno tra quelli definiti nella classe OpCodes), allegato nella stringa JSON che la rappresenta, che permette al worker di distinguere il metodo corretto da applicare per fornire la risposta al client.

ServerPersistence

ServerPersistence si occupa di caricare lo stato precedente del server al momento della sua esecuzione e di salvarlo periodicamente.

A intervalli di millisecondi specificati nel file di configurazione del server, il thread ottiene tutti i dati utili conservati nel server, li serializza in formato JSON e li salva su un file il cui percorso viene specificato al momento della creazione del thread. Al momento della creazione del server, esegue il processo contrario, deserializzando gli oggetti dal file (se esiste) e impostandoli nel server, che viene passato come parametro del costruttore del thread.

ServerRewards

ServerRewards è il thread usato per effettuare il calcolo delle ricompense. L'intervallo di tempo che si aspetta tra un calcolo e l'altro è specificato nel file di configurazione del server.

Quando il metodo calculateRewards() viene chiamato, si applica la formula specificata nella traccia del progetto per calcolare la ricompensa totale per un certo post. Dato che la formula comprende una componente legata ai commenti e una legata ai voti, al momento dell'aggiunta di una transazione, è possibile specificare l'origine della ricompensa (creazione di un post, upvote di un post o commento di un post).

La ricompensa totale per un post viene divisa in 2 percentuali: la prima va all'autore del post, la restante viene suddivisa equamente tra i curatori. Se una ricompensa ha valore

minore o uguale di 0, la transazione non viene effettuata. In ogni caso, alla fine dell'esecuzione del metodo, si avvisano eventuali client connessi del termine del calcolo, così da incoraggiarli a controllare il loro wallet.

Componenti di utilità

ComUtility

La classe ComUtility permette di implementare lo scambio di messaggi tra client e server.

- sendSync permette al client di inviare una stringa in formato JSON che rappresenta la sua richiesta
- sendAsync permette al server di inviare al client (tramite la SelectionKey passata come parametro), l'attachment apposto da un WinsomeWorker incaricato di risolvere la richiesta inviata dal client
- receive permette sia al client che al server di ricevere dati (stringhe in formato JSON) dal lato opposto della comunicazione
- attachAck e attachErr si occupano, rispettivamente, di allegare alla SelectionKey passata come argomento una stringa JSON di acknowledgement (codice 0, messaggio di errore "OK") o di allegare una stringa JSON di errore (codice e messaggio specificati come parametri)

Le funzioni sendSync e sendAsync (le uniche che effettivamente spediscono dati), oltre a spedire la stringa specificata, inviano anche la sua lunghezza, in modo tale che al momento della ricezione (in receive) sia possibile essere sicuri di aver ottenuto il contenuto completo.

ClientError

ClientError è la classe che si occupa di stampare correttamente i messaggi di errore eventualmente ricevuti in risposta a una richiesta. Permette di stampare una frase di successo e / o una tabella di riepilogo dell'operazione se non si sono verificati errori. In caso contrario, procede alla stampa del codice di errore e del messaggio corrispondente.

TableList

Componente esterna (di cui non sono l'autore) utilizzata nel progetto per formattare l'output del client in maniera più leggibile sottoforma di tabelle, utile soprattutto quando si mostrano post con tanti commenti o feed con tanti post. La codifica della tabella in unicode (che la rende esteticamente più piacevole) può essere disabilitata dal file di configurazione, se il terminale utilizzato non supporta unicode.

Tecnologie e scelte adottate

NIO e channel multiplexing

Data la natura del progetto e la possibilità da parte di molti client di connettersi contemporaneamente al server, al fine di gestire le sessioni degli utenti, si è optato per l'utilizzo della tecnica del channel multiplexing per gestire le connessioni lato server.

Tale approccio permette l'utilizzo di molti meno thread rispetto a quanti se ne utilizzerebbero se a ogni connessione ne venisse assegnato uno (a lungo andare tale soluzione non è scalabile e l'utilizzo di troppi thread causa un overhead significativo nel context switching) e, assieme all'utilizzo di un thread pool (si veda la sezione successiva), l'utilizzo delle risorse è anche più efficiente.

Dal momento che letture e scritture sui canali sono non bloccanti, è stato necessario includere, per ogni stringa inviata, la sua dimensione, in modo tale che sia in lettura che in scrittura, le funzioni di utilità in ComUtility rimanessero in ascolto sul canale finché non fossero certe di aver terminato.

ThreadPool

WinsomeServerMain utilizza NIO e channel multiplexing per gestire le connessioni con i client, accettandole, ricevendo e inviando dati. Le richieste vere e proprie sono invece delegate ai WinsomeWorker, che vengono inviate a un ThreadPool personalizzato non appena l'input del client è stato ricevuto completamente.

L'utilizzo di un ThreadPool fa sì che il numero di thread contemporaneamente attivi sia limitato (e configurabile tramite file di configurazione), limitando l'overhead del context switching. Dal momento che la risoluzione di una richiesta è un processo che, mediamente, richiede pochi millisecondi (fatta eccezione per la visualizzazione del wallet in bitcoin, che richiede di reperire dati tramite URL), è ragionevole supporre che i thread attivi in un certo istante siano pochi.

Il ThreadPool utilizzato è customizzato e l'utente può specificare tramite file di configurazione il numero di core threads, il numero massimo di thread nel pool, il tempo di inattività dopo il quale un thread viene disattivato e la dimensione della BlockingQueue.

Il pool utilizza una ArrayBlockingQueue con dimensione specificata nel file di configurazione in alternativa a una LinkedBlockingQueue poiché si è ritenuto necessario fornire un limite al numero di richieste in coda. Infatti, se tale numero diventasse troppo grande, i client rimarrebbero in attesa per molto tempo e si dovrebbe avere un modo per essere a conoscenza del problema al fine di cambiare i parametri del ThreadPool e migliorarne il funzionamento.

E’ stata inoltre implementato un RejectionHandler customizzato (RepeatPolicy), che fa del suo meglio per reintrodurre il task all’interno della coda. Esso tenta un certo numero di volte di reinserire il task, alternando ogni tentativo con un attesa di un certo numero di millisecondi (entrambi i parametri sono specificati al momento della creazione della RepeatPolicy e configurabili dall’utente).

Sincronizzazione

I processi di sincronizzazione sono implementati in due forme. La prima consiste nell’utilizzo di strutture concorrenti (ConcurrentHashMap e Vector) da parte di WinsomeServerMain per tenere traccia dei dati necessari: in questo modo si rendono thread-safe le operazioni atomiche messe a disposizione.

Spesso però, i WinsomeWorker hanno bisogno di svolgere più operazioni sulla stessa struttura, pertanto utilizzano dei blocchi synchronized (monitor) sulle strutture necessarie. I blocchi iniziano solo quando necessario e terminano il prima possibile in modo da sbloccare il prima possibile eventuali thread in attesa.

Multicast

Il client riceve dal server le notifiche di avvenuto calcolo delle ricompense: per far ciò, il thread RewardNotifier lanciato dal client, si iscrive a un gruppo multicast i cui parametri sono specificati dal client al momento della creazione del thread. I suddetti parametri sono inviati in allegato alla risposta del server al momento del login.

Al momento del logout, il RewardNotifier viene fermato e ne viene creato uno nuovo al momento del login di un altro utente.

RMI

Sia WinsomeClientMain che WinsomeServerMain implementano un’interfaccia che permette loro di utilizzare la funzionalità RMI. Rispettivamente, WinsomeClientMain e WinsomeServerMain implementano IRemoteClient e IRemoteServer.

IRemoteClient mette a disposizione del server due metodi (newFollower e unfollowed) che permettono al client di essere notificato riguardo all’aggiunta o alla rimozione di un follower. Lo stub del client viene creato al momento dell’esecuzione di WinsomeClientMain e resta lo stesso anche nel caso di sessioni diverse sullo stesso client: semplicemente, al momento del logout, la lista contenente i followers viene resettata. Al momento del login, il server invia la lista dei follower al client in modo da sincronizzarlo con il suo stato attuale.

IRemoteServer fornisce invece al client tre metodi: uno (signup) viene utilizzato per la funzione di registrazione, che segue lo stesso protocollo richiesta-risposta in formato di stringa JSON delle normali interazioni, senza però passare dall’utilizzo di un canale. Gli altri

due metodi (registerNotifications e unregisterNotifications) permettono invece al client di iscriversi o di disiscriversi dal servizio di notifica dei follower. Il client si iscrive al momento del login e si disiscrive al momento del logout.

URL

Il tasso di conversione da wincoin a bitcoin viene reperito usando il servizio di generazione di frazioni di random.org. L'URL della richiesta si riferisce alla generazione di un singolo numero compreso tra 0 e 1, stampato in formato plain text: il server non fa altro che leggere il contenuto restituito e convertire la stringa ottenuta in numero decimale, per poi moltiplicarlo per la quantità di wincoin nel portafoglio dell'utente.

Classi di supporto

- **Post:** classe usata per rappresentare un post. Ogni Post è identificato da un valore, il suo id, che viene incrementato ogni volta che si crea un nuovo post. All'avvio del server, se il caricamento dei dati precedenti è andato a buon fine, il prossimo id assegnabile (postId) viene assegnato al massimo id salvato precedentemente + 1.

  Il post salva anche alcuni dati riguardanti i rewin. Per uniformare l'implementazione dei metodi riguardanti i Post, i rewin vengono considerati come Post: l'attributo rewin indica quindi se un Post è un rewin o meno e l'attributo rewinner memorizza il nome dell'utente che ha effettuato il rewin.

  Ogni Post si occupa anche di salvare il numero di volte che ha "subito" l'esecuzione dell'algoritmo di calcolo delle ricompense.

- **Comment:** classe usata per rappresentare un commento. Dato che l'id del post a cui un commento si riferisce è la chiave nell'hashamp che collega un Post ai suoi commenti, non è necessario creare ridondanza e salvarlo anche nel commento. E' invece necessario salvare il nome dell'utente che ha inviato il commento.

- **Vote:** classe usata per rappresentare un voto. Similarmente a Comment, non è necessario salvare l'id del Post, ma serve invece salvare l'autore del voto.

- **Transaction:** classe usata per rappresentare una transazione, cioè un'assegnazione di wincoin a un utente. Una Transaction permette anche di memorizzare la causale, cioè la fonte della ricompensa (creazione, voto o commento di un post di successo).

- **User:** classe usata per rappresentare un utente. Ogni User è identificato dal suo username, dato che esso dev'essere univoco all'interno del social.

  Ogni User si occupa inoltre di salvare la lista delle transazioni ad esso relative, aggiornando di conseguenza la quantità di wincoin nel suo wallet.

Librerie esterne

- **Gson:** usato per la serializzazione e la deserializzazione degli oggetti in formato JSON. Permette di implementare le funzioni di salvataggio e caricamento del server, oltre a permettere l'invio e la ricezione di tipi di dato non primitivi (come ad esempio la lista dei post in un feed).
- **org.json:** utilizzata principalmente per la classe JSONObject, che permette di manipolare facilmente gli attributi di un oggetto in formato JSON, impostando chiavi e valori (anche non primitivi se si usa prima Gson per serializzarli).

Altre note

- Dal momento che il rewin è un metodo per dare più visibilità a un post, commenti e voti di un rewin vengono assegnati al post originale. Il vantaggio di rewinnare un post consiste nella possibilità di aumentare le interazioni (e quindi le ricompense) per quel post e quindi di ottenerne di maggiori per sé stessi.
- Effettuare il rewin di un rewin ha come risultato di rewinnare il post originale. Al momento del rewin infatti, dato l'id del rewin, si ottiene l'id del post originale e lo si usa per costruire il nuovo rewin.
- Al momento dell'eliminazione di un post, si distinguono due casi:
    - Se il post è un rewin, si elimina solamente il post di rewin, lasciando quindi inalterato il post originale e altri rewin (dato che il rewin di un rewin è semplicemente il rewin del post originale).
    - Se il post non è un rewin invece, oltre a eliminare tutti i rewin del post, si eliminano anche voti e commenti relativi a quel post.