Università di Pisa

Università di Pisa, Dipartimento di Informatica

Laboratorio di Reti - A.A. 2021-2022

Docente: Federica Paganelli

Relazione progetto Winsome

Nicola Vetrini
29 dicembre 2021

Indice

1 Introduzione

Lo scopo del progetto è stata la realizzazione di una rete sociale, Winsome, basata sulla condivisione di contenuti tra un insieme di utenti.
Ciascun utente alla registrazione specifica una lista di tag a cui è interessato ed ha la possibilità, in seguito, di seguire altri utenti che hanno almeno un tag in comune. Per ogni utente vi è quindi un blog personale, nel quale egli può pubblicare dei post, ed un feed contenente i post degli utenti che segue. Ciascun utente può anche decidere di votare positivamente o negativamente un post, commentarlo o effettuarne il rewin (analogo del retweet, seppur con limitazioni).
Winsome ha inoltre un meccanismo di reward sia per i creatori di post che hanno delle interazioni, sia per chi commenta o vota un post; i reward sono calcolati periodicamente, nella valuta di Winsome: i wincoin, che possono essere convertiti in altre valute elettroniche (il server supporta solamente la conversione bitcoin).

2 Architettura del sistema

Il social network Winsome consiste di due componenti software: WinsomeClient (client nel seguito) e WinsomeServer (server nel seguito). Il client ed il server comunicano principalmente attraverso delle richieste, contenute nel package WinsomeRequests, di vario tipo. Ogni richiesta è convertita in una stringa JSON tramite la libreria Jackson, e da tale stringa si ottiene un arrray di byte, il quale viene trasmesso su un socket TCP che connette il client ed il server.
Attraverso il client è possibile inoltre registrare un nuovo utente su Winsome: tale operazione viene eseguita utilizzando RMI, come da specifica (l'iterfaccia implementata dal server è Signup, con il metodo register).
Il comando `list followers` del client non genera una richiesta sincrona al server; è presente una struttura dati nel client il cui contenuto viene mostrato all'utente. Tale elenco di followers viene aggiornato periodicamente dal server utilizzando il meccanismo delle RMI callback: il client al login di un utente si registra presso il server utilizzando RMI (**FollowerUpdaterService**, metodo `subscribe()`) e passando il nome dell'utente (loggato) che richiede il servizio ed un oggetto che implementa l'interfaccia **FollowerCallback** (**FollowerCallbackImpl**) al server. Tale oggetto viene utilizzato dal server per eseguire tramite RMI l'aggiornamento dell'insieme dei followers dell'utente loggato in tale client. L'aggiornamento avviene ad intervalli regolari, lo stesso per ogni client, che può essere impostato nel file di configurazione del server (maggiori dettagli in seguito).
Durante l'operazione di logout dell'utente viene mandata al server anche la richiesta di cancellazione dal callback (metodo `unsubscribe()` di **FollowerUpdaterService**), al fine di rimuovere dal server il riferimento all'oggetto del client usato per il callback ed interrompere la task che esegue periodicamente tale aggiornamento (tramite uno ScheduledThreadPool). Dato che il comando `quit` del client richiama al suo interno, se vi è un utente loggato, la procedura di logout, di fatto avviene comunque la deregistrazione dal servizio.
TODO: UDP multicast per wallet

3 Server Winsome

3.1 Argomenti da riga di comando

Il server può ricevere uno o più dei seguenti parametri da riga di comando, che ne influenzano il comportamento all'avvio. Si noti che i valori passati tramite queste opzioni hanno la precedenza su quelli eventualmente letti dai file di configurazione.
Se in entrambi i casi non è stato specificato alcun valore per un parametro, verrà usato il valore di default statico presente nel file `ServerConfig.java`.

- `-c --configure <FILE>`: path del file di configurazione che il server deve caricare
- `-h --help`: messaggio di uso del server
- `-p --socket-port <PORT>`: porta sulla quale, se possibile, viene creata la ServerSocket sulla quale il server si mette in ascolto per accettare le connessioni dai client
- `-r --registry <PORT>`: porta sulla quale, se possibile, viene creato il registry RMI utilizzato sia per la registrazione, che per l'iscrizione/disiscrizione dal servizio di aggiornamento dei followers

3.2 Configurazione

Il server è configurato attraverso un file JSON che viene cercato, nell'ordine, ai seguenti path:

1. il path passato attraverso l'opzione `-c <path>`
2. `config.json` nella directory corrente
3. `./data/WinsomeServer/config.json` il file di configurazione di default

Se tutte le opzioni precedenti non contengono un file di configurazione valido allora il server termina immediatamente.
I parametri configurabili sono i seguenti:

dataDir : Il path alla directory contenente i dati del server, ovvero la directory che contiene il file `users.json`, la directory `blogs`, etc ...

registryPort : La porta del registry RMI nel quale inserire lo stub per la registrazione. Il valore deve essere un intero nel range $[0, 65535]$

serverSocketAddress : Nome host o indirizzo IP del ServerSocket creato dal server per accettare le connessioni dai client (IP pubblico del server nel caso generale, ma in questo caso localhost)

serverSocketPort : Porta alla quale deve essere legato il socket sopracitato, nel range $[0, 65535]$

minPoolSize : Intero (> 0) che rappresenta il numero di core thread nella threadpool per la gestione delle richieste del server

maxPoolSize : Intero ($x : INT\_MIN \leq minPoolSize \leq x < INT\_MAX$) che rappresenta il numero massimo di thread che possono essere gestiti contemporaneamente dalla threadpool

workQueueSize : Dimensione della coda di task che la threadpool può accumulare in attesa che un thread del pool sia libero, in accordo con la politica di gestione delineata dalla documentazione di ThreadPoolExecutor

retryTimeout : long (> 0) che rappresenta il numero di millisecondi che l'handler per la gestione delle richieste rifiutate dal threadpool attende prima di provare a sottomettere nuovamente la richesta

callbackInterval : long (> 0) che rappresenta l'intervallo tra due consecutivi aggiornamenti alla lista dei follower del client registrato al servizio. L'unità di misura che quantifica il valore è specificata dal parametro indicato di seguito

callbackIntervalUnit : Unità di misura di callbackInterval. Essendo rappresentato con una TimeUnit i valori possibili sono quelli dell'enumerazione indicati nel link. Di default l'unità di tempo sono i secondi. Nel file JSON una TimeUnit è serializzata come stringa, quindi per indicare di misurare callbackInterval in secondi è sufficiente scrivere "callbackIntervalUnit": "SECONDS" nel file JSON.

Non è necessario specificare tutti i parametri nel file, poiché quelli assenti assumeranno i valori di default definiti nella classe `ServerConfig.java`. Il file JSON viene deserializzato utilizzando Jackson con ObjectMappper in un'istanza della classe menzionata, ed un riferimento ad essa viene passato alla creazione del server, nel main.
Nella classe è in realtà presente anche il campo (non serializzato) configFile, che serve soltanto per determinare, nella funzione `getServerConfiguration()` in `main()`, se era stato settato con l'opzione -c il path per un file di configurazione, che ha la precedenza rispetto a quello di default.

3.3 Esecuzione

Il codice del server Winsome è contenuto all'interno del package WinsomeServer; la classe ServerMain in tale package contiene il metodo main, quindi è quella da eseguire per far partire il server.

3.3.1 Thread e gestione della concorrenza

Il server Winsome è multithreaded: vi è il thread main, il cui compito è l'inizializzazione del server e del registry; dal thread principale è fatta partire un istanza di `WinsomeServer`, sottoclasse di `Thread`, che si occupa di gestire lo smistamento delle richieste attraverso un selector. Una volta lanciato con successo il WinsomeServer il thread main termina.
Nel thread del WinsomeServer vi è un selector che permette di leggere le richieste provenienti dai client e sottometterle ad una threadpool le cui dimensioni minime e massime sono fissate dal file di configurazione.
Nel costruttore di WinsomeServer, inoltre, viene attivato un thread per leggere gli utenti di winsome dal file `users.json` e caricarli in memoria. Una volta letti gli utenti viene attivato un `BlogLoaderThread` per ogni utente letto dal file, il cui compito è deserializzare dal file `<datadir>/blogs/<user>.json` tutti i post presenti in tale blog e caricarli nelle strutture dati del server (mappa post globale e lista dei post di ogni blog). Tali thread terminano una volta caricato il blog (il thread che esegue il costruttore di WinsomeServer si blocca finché la join su ciascuno di essi non ritorna).
Alla terminazione del server, poiché lo stato deve persistere, ogni utente ed ogni post dei loro blog devono essere scritti su file. Per fare questo è stato utilizzato il meccanismo degli shutdown hook, settati come prima istruzione del metodo `run()` del WinsomeServer.
Vi è un hook (thread su cui non è stato invocato `start()`) per la sincronizzazione del file degli utenti: **SyncUsersThread.java**. L'altro hook è **SyncBlogsThread.java**, il cui unico compito è quello di creare e far partire un **SyncPostsThread.java** per ogni utente Winsome, che sincronizza i post di un solo blog. Tali thread operano su dati totalmente indipendenti, per cui non è richiesta alcuna sincronizzazione delle loro operazioni, il che consente di avere il massimo grado di parallelizzazione del processo consentito dalla macchina. L'unica accortezza è che il thread che li ha creati non termini fino a che tutti questi thread non sono terminati, altrimenti la JVM potrebbe terminare lasciando uno o più blog in uno stato inconsistente, che provovcherebbe degli errori al riavvio del server.

4 Client Winsome

client

5 Compilazione ed esecuzione

I comandi elencati di seguito assumono che la directory corrente sia quella base del progetto (quella con la subdirectory src). Vi sono inoltre un Makefile e degli script per la compilazione ed esecuzione (serv_run.sh e client_run.sh), il cui scopo è semplicemente di automatizzare l'esecuzione dei comandi riportati di seguito.
La compilazione ed esecuzione è stata testata su Java 11, ma dovrebbe compilare ed eseguire senza modifiche su Java >= 8

5.1 Compilazione

Il comando per la compilazione del server è il seguente:
`javac -d bin -cp "libs/*" -sourcepath src/ src/Winsome/WinsomeServer/*.java`
Per compilare il client è sufficiente sostituire WinsomeClient al posto di WinsomeServer.

5.2 Esecuzione

Per eseguire il server è sufficiente il comando
`java -cp ".:bin/:libs/*" Winsome.WinsomeServer.ServerMain`
Sostituendo a WinsomeServer.ServerMain WinsomeClient.ClientMain si può eseguire il client.
Eventuali argomenti da riga di comando possono essere aggiunti in fondo alla stringa.

5.3 Documentazione

È disponibile nel makefile un target (make doc) per generare con javadoc la documentazione delle classi realizzate, visualizzabile tramite browser.
Con l'opzione -link di javadoc si ottengono link cliccabili alla documentazione delle classi della libreria standard Java e delle librerie utilizzate, anche se ciò potrebbe introdurre un leggero ritardo nella generazione, per cui le righe contenenti -link possono essere tolte, se necessario.

6 Librerie utilizzate

Le librerie esterne utilizzate dal software sono state incluse, sotto forma di file .jar, nella cartella *libs*

6.1 Apache Commons CLI

È stata utilizzata la libreria Apache Commons CLI versione 1.5.0 per il parsing degli argomenti da riga di comando sia in WinsomeServer che in WinsomeClient.

6.2 Jackson

All'interno del software, soprattutto nella componente server, è stata utilizzata ripetutamente la serializzazione e deserializzazione tra classi ed oggetti JSON scritti su file. A tale scopo ho scelto di utilizzare la libreria Jackson, versione 2.9.7.