Relazione su WORTH

zhiqiang shi

October 20, 2021

1 Introduzione

Il server usa la porta 33333 per la connessione TCP e la porta 30000 per la UDP. Inoltre utilizza la porta 30001 per la registrazione degli nuovi utente e la porta 30002 per le notifiche su online e offline dei utenti.

Il server utilizza Json per salvare i progetti e gli utenti sul disco. Per gli utenti, il server salva sotto la directory .\Members, sottoforma di nickName.json, viene salvato il username e il password dell'utente, memtre il suo stato non viene salvato, quando il server fa il caricamento degli utenti dai file Json, automaticamente setta il suo stato a "offline". Per i progetti, il server salva sotto la directory .\Projects, creando per ogni progetto un nuovo directory con nome del progetto, dentro questo directory c'e il project.json che salva il projectName, multicastAddress e members, c'e anche uno directory con nome Cards, che salva i dati delle card sotto la forma di cardName.json, salvando i suoi cardName, description e history. Il project non salva i dati degli cards, perche usando la directory Cards e il loro ultimo stato di history riesce a ricostruire la lista di TODO, INPROGRESS, TOBEREVISED, DONE.

Viene usato ArrayList per salvare tutti i tipi di dati, solo per la parte di chat viene scelto di usare HashMap. Per la parte di Server viene scelto di usare il multiplexing gestito da Selector, per evitare troppi overhead causati da lock e troppi thread, dato che tutti i commandi richiedono uso dei ArrayList di utenti e progetti.

La comunicazione dei indirizzi multicast e utenti da server viene fatto solo se utente ha eseguito il login con successo.

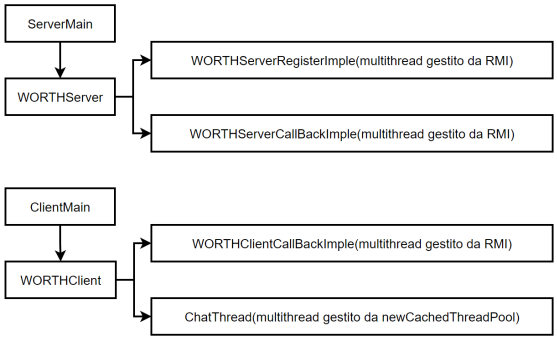
Dato che il progetto non ha un meccanismo di buttare tutti gli utenti dal chat di un certo progetto, percio indirizzo usato da quel progetto rimane inutilizzabile anche se viene cancelato, il indirizzo multicast ritorna ad utilizzabile solo dopo un riavvio del server.

2 Thread attivati e struttura dati

Il server si parte con ServerMain, crea il thread WORTHServer, il WORTHServer usa ArrayList<Member> e ArrayList<Project> per salvare gli utenti

registrati e i progetti, inoltre utilizza anche un generator che salva tutti gli indirizzi multicast utilizzati e crea degli indirizzi multicast per i nuovi progetti. Il WORTHServer crea WORTHServerRegisterImple e WORTHServerCallBackImple, il primo per ricevere delle nuove registrazione, dato che RMI è multithreading per la sua implementazione, per evitare che piu utenti usano la stessa nickName, viene usato "synchronized" per avere una correta registrazione, usa ArrayList<Member> del server. Il secondo per notificare gli utenti online il login, logout o la registrazione dei nuovi utenti, viene usato anche "synchronized" per avere una correta notifica, utilizza un suo ArrayList<WORTHClientCallBackInterface> per ricordare gli utenti iscritti alla notifica. Per ogni connessione, il server salva nel suo attachment un ServerAnswer per indicare nickName e la risposta da inviare.

Il client si parte con ClientMain, crea il thread WORTHClient, il WORTHClient utilizza ArrayList<Member> per salvare localmente gli utenti senza il password, ha un nickname per vedere se il client ha fatto login, Client crea WORTHClientCallBackImple per ricevere i callbacks da parte di server, ma fa la registrazione al server solo quando utente ha fatto la login. Il client ha anche una newCachedThreadPool che ha il compito di creare i thread ChatThread per ricevere i messaggi UDP dopo il login. Il chat si appoggia su un hashMap<String, ChatHistory>, i suoi metodi sono tutti synchronized per evitare la perdita di messaggi causati dall'accesso contemporaneo del thread WORTHClient e ChatThread. Un boolean globale per fare la chiusura del client.



3 descrizione delle classi

3.1 Card

La classe composto da cardName, description e history.

3.2 Project

La classe composto da projectName, multicastAddress e le liste members, todo, inprogress, toberevised, done e un projectObjectMapper per scrivere dati permanenti sul disco.

3.3 Member

La classe composto da nickName, password e status.

3.4 ServerAnswer

La classe legato ad ogni connessione TCP, composto da nickName e answer, il primo per indicare quale utente è associato quel connessione e il secondo per salvare le risposte da inviare al client.

3.5 MulticastAddressGenerator

La classe composto da una lista che salva tutti gli indirizzi usati e genera i nuovi.

3.6 WORTHServerCallBackInterface e WORTHServerCallBackImple

WORTHServerCallBackInterface dichiara i metodi del RMI e viene implementato da WORTHServerCallBackImple, ha una lista clients.

3.7 WORHServerRegisterInterface e WORTHServerRegisterImple

WORTHServerRegisterInterface dichiara i metodi del RMI e viene implementato da WORTHServerRegisterImple, ha una lista di membri in condivisione con la classe WORTHServer e un WORTHServerCallImple che ha il compito di avvisare tutti gli utenti quando un nuovote utente si è registrato

3.8 ServerMain

Ha compito di creare il thread WORTHServer.

3.9 WORTHServer

La classe implementato con NIO Selector per la connessione TCP, ha due liste per salvare gli utenti e i progetti, un WORTHServerCallBackImple per notificare gli utenti online, un serverObjectMapper per leggere i dati sul disco durante l'avvio e un generator per gli indirizzi.

3.10 ChatHistory

La classe composto da una String con indirizzo del multicast e la chat, fa il supporto.

3.11 ChatThread

La classe composto da un hashmap condiviso con WORTHClient, e un projectName che questo thread è in ascolto, viene settato un timeout cosi il thread non viene bloccato su receive ma ogni 5 secondi controlla se il thread viene interroto.

3.12 WORTHClientCallBackInterface e WORTHClientCallBackImple

WORTHClientCallBackInterface dichiara i metodi per RMI e viene implementato da WORTHClientCallBackImple, ha una lista di membri.

3.13 ClientMain

Ha compito di creare il thread WORTHClient.

3.14 WORTHClient

La classe principale da parte del client, che riceve gli input dalla tastiera e in base ai commandi esegue le operazioni. Ha una lista di membri in condivisione con WORTHClientCallBackImple e una HashMap di progetti in condivisione con ChatThread, un ThreadPool per creare i thread che ascoltano i messaggi UDP, un nickName per controllare se questo client ha fatto il login e alla fine un boolean che serve per indicare se il client deve chiudere o no.

4 Istruzione d'uso

Assicurare che "jackson-databind-2.9.7.jar", "jackson-core-2.9.7.jar" e "jackson-annotations-2.9.7.jar" sono usati come libreria esterna. Compilare tutti i file, esegue prima il ServerMain, dopo che ha stampato "server pronto", esegue il ClientMain e si puo interagire con Client.
I commandi sono:
login username password
logout
listUsers
listOnlineUsers
listProjects
createProject projectName
addMember projectName memberName
showMembers projectName
showCards projectName

showCard projectName cardName
addCard projectName cardName, description usare _ per il spazio
moveCard projectName cardName source destination
cancelProject projectName
sencChat projectName viene chiesto dopo il messaggio da inviare
readChat projectName
Se il client dopo il login ha fatto un improviso chiusura, il client rimane online, percio si deve riavviare il server.