WordQuizzle

RELAZIONE

Realizzato da: Matteo D'Onofrio (561901)

1) ARCHITETTURA

Per la realizzazione del progetto è stata scelta un'architettura Client-Server con server multithreaded, in cui abbiamo un processo unico che è il server centrale, il quale gestisce le richieste in arrivo tramite un Thread pool. Nello specifico, il server centrale apre un socket TCP sul quale attende richieste di connessione; per ognuna di esse quindi, crea un thread gestore del client, che viene eseguito dal ThreadPool (di tipo newFixedThreadPool).
I thread che gestiscono le richieste dei client sono differenti dai thread che si occupano di gestire le sfide tra client, infatti come vedremo, si alterneranno durante l'esecuzione.
Il processo server centrale, è colui che ha in tempo reale le informazioni riguardanti tutti i thread, sia i gestori dei client che i gestori delle sfide, permettendo la coordinazione dei diversi gestori.
Per il lato client, le informazioni trattate e gestite sono davvero minimali, demandando tutto il carico al server, e lasciando ad esso il solo compito di inviare, ricevere e stampare risposte.
Si è scelto tale approccio, che consta quindi di tanti piccoli thread in esecuzione, e quindi molti dati da computare, bilanciato con la suddivisione del carico di lavoro, ottenendo un servizio in cui ogni componente fa un semplice e specifico lavoro.

2) SUDDIVISIONE CARICO DI LAVORO (CHI FA COSA/CLASSI)

2.1) Word Quizzle Server

Come detto in precedenza il server centrale è costituito da un processo unico, il quale si occupa di:

- ricevere le connessioni in ingresso da parte dei client, generando un gestore per loro
- coordinare la comunicazione tra i diversi gestori
- avviare/terminare le sfide

- assegnare punteggi extra dopo la sfida

Per fare questo esso usa un semplice meccanismo tramite **Concurrent Hash Map**: nello specifico il server centrale ha le 5 seguenti tabelle hash:

1) Socket Tcp Map
2) Socket Udp Map
3) Client Handler Map
4) Client Name Map
5) Challenge Handler Map

Funzionamento:
Il server dispone di un Id incrementale, settato a 0 inizialmente. Quando un Client si connette al server, quest'ultimo genera un Client Handler (thread), che si occuperà di gestire tale Client. Il Client Handler riceve come parametro l'**Id**, così che nelle future operazioni con il server centrale possa identificarsi. Il server centrale, dopo questa operazione preliminare, provvede ad inserire e registrare via via, le informazioni riguardo tale client (nome, socket Tcp, porta Udp), inserendole nelle rispettive tabelle hash: in ognuna la chiave per accedere al valore è l'**Id,** così da permettere un accesso efficiente.

2.2) Database Handler (singleton)

Esso si occupa di leggere e scrivere sul database (DataBase.json). Le altre classi si occupano di manipolare i file JSON, demandando a questa il compito di operare sul file.

2.3) Translation Service (singleton)

Esso si occupa di estrarre le parole dal dizionario (Dizionario.txt), e successivamente tradurle, tramite una GET HTTP.
Le parole sono gestite all'interno di una **Concurrent Hash Map** (Dictionary), in cui le chiavi sono le parole in italiano e i valori le relative traduzioni.
Su un totale di 16 parole, ne sceglie randomicamente 4 ad ogni sfida.
Quando viene richiesto, restituisce l'intera tabella ai gestori delle sfide.

2.4) UDP Server Sender

Esso viene istanziato dai gestori delle sfide (Client Handler), al fine di inviare l'invito alla sfida in UDP al proprio client. Si occupa solo di inviare richieste, in quanto le risposte sono inviate dal client al Client Handler in TCP. Viene quindi distrutto dopo aver inviato la richiesta.

2.4) Challenge Handler

Esso si occupa di gestire la sfida, inviando ricevendo e valutando la correttezza della risposta fornita. Ogni volta che viene lanciata una sfida, il server centrale genera due thread Challenge Handler, uno per ogni client.
Esso verifica anche la durata della partita, con un timer settato a 10 secondi; quando vengono superati la partita termina e il punteggio ottenuto fino a quel momento viene valutato.

2.5) Client Handler

Esso è un thread e si occupa di ricevere, analizzare e rispondere alle richieste effettuate dai client.
Per la normale interazione (tutto ciò che non riguarda la sfida), si affida a un dispatcher(metodo della classe), il quale analizza la stringa in ingresso e richiama il metodo corrispondente.
Per la sfida i passi sono questi:

- client invia la richiesta di sfida al suo Client Handler
- il Client Handler invia la richiesta al server centrale
- il server centrale segnala al Client Handler sfidato, di inviare la richiesta in UDP al suo client
- Client Handler sfidato invia, attende e analizza risposta del client (impostando un timer di 5 secondi: dopo essere trascorsi, qualsiasi risposta ricevuta viene interpretata come rifiuto dell'invito).
- In caso di risposta affermativa entrambi i Client Handler (sfidato/sfidante) si mettono in attesa (wait) sulla Challenge Handler Map, cedendo il controllo ai Challenge Handler (gestori di sfida).
- Quando la sfida termina, il server centrale "risveglia" (notify) i Client Handler, restituendo loro il controllo.

2.6) Rmi Implementation

Contiene l'implementazione del metodo che consente la registrazione al servizio.

2.7) Word Quizzle Client

Rappresenta il vero client; tuttavia il suo lavoro è suddiviso in 3 classi(seguenti), ed esso si occupa di coordinarle e gestire i dati in comune.

2.8) Client Sender
Componente che si occupa di inviare le richieste al server.

2.9) Client Receiver
Componente che si occupa di ricevere e stampare le risposte del server.

2.9.1) Client Receiver UDP
Componente che si occupa di ricevere e stampare le richieste di partecipazione alle partite, ricevute sul socket UDP.
Quando l'utente effettua il logout, viene lanciato un interrupt verso questo thread, per interromperlo.

3) GESTIONE THREAD/CONCORRENZA

3.1) Lato server
Come già accennato, il server gestisce le richieste con un ThreadPool, al quale sottomette i task. Ogni Client Handler è un task, quindi un thread per ogni client connesso. Inoltre abbiamo anche i Challenge Handler, di conseguenza due thread per gestire ogni sfida.
Quando ci sono più utenti online essi competono per risorse quali : DataBase Handler, Translation Service, Rmi Implementation (tutti singleton).
Per gestire il loro accesso concorrente non sono stati necessari lock espliciti, ma si è demandato il controllo dell'accesso mutuamente esclusivo al **synchronized statement**.

Un altro punto critico è quello del server centrale nella gestione/modifica delle varie tabelle (viste al punto 2.1), in quanto anche lì numerosi thread (Client Handler) cercano di accedere per leggere/modificare il valore del proprio stato corrente; di conseguenza è stato gestito con:

- uso del synchronized statement nei metodi che accedono/leggono/modificano tali tabelle
- sincronizzazione di blocchi di codice, in cui il thread acquisisce la lock sull'oggetto specifico, ed effettua su di esso operazioni
- utilizzo di wait/notify (meccanismo **monitor**) sulla tabella Challenge Handler Map, in cui i Client Handler si mettono in **wait**, quando lanciano/ricevono una sfida, finché:

- scade timer
- client accettano / rifiutano sfida
- termina sfida

I tre eventi descritti sopra fanno scattare una **notify.**

3.2) Lato client

Ogni client ha un processo centrale, che gestisce i dati e permette la sincronizzazione dei 3 thread di cui si costituisce: Client Sender, Client Receiver e Client Receiver UDP.
Word Quizzle Client tramite lo statement synchronized, apposto alla firma dei metodi, garantisce la thread safety.

4) ESECUZIONE

L'unica libreria esterna usata dal progetto è: **JSON simple,** per gestire la lettura/scrittura di file JSON. Inoltre non sono necessari parametri in ingresso, solamente la presenza del file Dizionario.txt, il quale viene fornito nella cartella del progetto.
Per l'esecuzione, nella cartella vengono forniti 3 progetti:

- WordQuizzleServer
- WordQuizzleClient
- WordQuizzleClient2

il fine è quello di agevolare l'interazione eliminando i parametri forniti in ingresso. Infatti i due progetti client sono identici, cambiano solamente i parametri di configurazione(porte TCP e UDP), che sono già settati.

Basta infatti eseguire i seguenti file:

- MainClassServer (contenuta in WordQuizzleServer/ src / wordquizzleserver)
- MainClassClient (contenuta in WordQuizzleClient / src / wordquizzleclient)
- MainClassClient2 (contenuta in WordQuizzleClient2/ src / wordquizzleclient2)

per testare tutte le funzionalità del progetto.