

Relazione Reti

Fausto Frasca

February 10, 2020

1 Premesse

Come punto di riferimento per il progetto si è scelto un computer di ultima generazione non troppo potente, ma comunque con risorse hardware minime come 8gb di ram, un processore quad-core e un ssd. Su tale base si è cercato di fare delle scelte che non richiedessero un uso eccessivo delle risorse al fine di avere un sistema efficiente e che non gravasse troppo sull'hardware su cui gira.

2 Introduzione generale

Il progetto è costituito da tre directory:

- Client
- Server
- Commons

In questo paragrafo verrà riportata una breve introduzione di quella che è l'architettura generale del sistema. Il tutto verrà approfondito nei paragrafi successivi.

2.1 Server

Il server è composto da un thread con selettore principale per la gestione della connessione TCP, un threadpool per svolgere operazioni "lunghe" delegate dal selettore e un oggetto RMI esportato per l'operazione di registrazione.

Sotto al server è presente un *user dispatcher*, con cui comunica richiedendo gli oggetti *User* in modo tale che sia il thread principale, sia i thread del threadpool operino sul medesimo oggetto.

Sotto allo user dispatcher è presente un *DBMS* che si occupa della gestione della memoria, sia in lettura che in scrittura.

Come riportato nell'immagine sotto, il server comunica esclusivamente con lo user dispatcher e quest'ultimo comunica col DBMS.

PROVA

Figure 1: Schema server

2.2 Client

Il client fornisce una GUI per l'interazione con l'utente.

Una volta connesso al server, il client avrà un processo principale per gestire la comunicazione client-server su socket TCP e un thread a parte che sta in ascolto su una socket UDP per le richieste di sfida.

2.3 Commons

Nella cartella *Commons* sono presenti quelle interfacce comuni sia al client che al server

3 Descrizione approfondita

3.1 Avvio

3.1.1 Server

Una volta avviato il main della classe *MainClassServer.java*, il server configura il registry RMI sulla porta *8000* ed esporta l'oggetto della classe *RMIRegistrationImpl.java* sotto il nome di *"RemoteRegister"*.

Tali parametri sono definiti in *Commons/src/main/java/RMIRegistrationInterface.java*.

Vengono gestiti diversi casi di errore, come per esempio se la porta è già occupata o avviene un errore nella configurazione.

Successivamente si passa alla configurazione per la connessione TCP.

Viene creata una socket non bloccante a cui viene associato un selettore, il cui compito sarà quello di gestire le varie interazioni client-server che avvengono su tale socket.

La socket viene configurata su *localhost* sulla porta *5000*.

I parametri sono definiti su *Commons/src/main/java/TCPConnection.java*.

Anche in questo caso vengono gestiti eventuali errori.

Considerazioni: Si è preferito scegliere il selettore anziché un'architettura dove vi è un thread per ogni client connesso, per ridurre quello che è il numero di thread avviati e evitando al tempo stesso di avere thread che non facessero nulla nel caso in cui il client non eseguisse alcuna operazione.

In fine si passa alla creazione del threadpool. Si è optato per un *newCachedThreadPool*.

Al threadpool sono delegate tutte quelle operazioni dove magari è richiesto

l'accesso in memoria o si opera su una risorsa condivisa e che perciò rallenterebbero l'operato del selettore principale.

Le operazioni delegate sono:

- Login (creare la classe e farlo)
- Aggiungi amico
- Mostra amici
- Mostra classifica
- Sfida

Considerazioni: Poichè la funzione principale del sistema è quella della sfida la scelta è ricaduta su `newCachedThreadPool` per il semplice fatto che garantisce di non imporre un limite al numero di thread e di conseguenza non si impone un limite superiore al numero di sfide che possono essere fatte contemporaneamente.

Una volta finita la fase di setup, viene creata un'istanza della classe *Server.java* a cui vengono passati il selettore e il threadpool.

Da qui in poi il server è pronto ad accettare richieste di connessione da parte dei client.