Applicazione Worth

Di Federico Giannotti

MATRICOLA 545597

Descrizione generale e architettura dell'applicazione

Worth è un'applicazione per la gestione di progetti in modo condiviso. La funzione principale di Worth è quella di permettere ad ogni utente registrato la possibilità di creare un progetto e successivamente aggiungere task o membri ad esso. In aggiunta è presente una chat per ogni progetto nella quale i membri possono comunicare e visionare gli ultimi cambiamenti automaticamente segnalati da Worth. Queste funzioni sono la base per creare un ottimo ambiente di lavoro e sviluppo.

Worth implementa un'architettura Client/Server per permettere all'utente di comunicare con l'applicazione. Di seguito sono descritti i due componenti in maniera dettagliata.

Server

Il Server utilizza due connessioni per permettere al Client di utilizzare i suoi servizi, la prima utilizza la tecnologia RMI mentre la seconda una normale connessione TCP.

RMI

Il server crea un oggetto remoto utilizzato dal Client per registrarsi al servizio. Effettuata questa prima operazione, il client può utilizzare altri metodi come: correttezza dei dati di login, lista degli utenti registrati all'applicazione, registrazione ad un servizio di callback con relativo metodo di notifica. Alcuni di questi metodi possono essere richiesti dal Client solo dopo aver stabilito una corretta connessione TCP sull'altra porta messa a disposizione dal Server. Per effettuare un corretto login e stabilire la connessione TCP, il client dovrà utilizzare un metodo offerto da RMI che controllerà la validità dei dati inseriti. Superata la verifica verrà stabilita la connessione TCP sull'altra porta del Server. Questo permette di tenere i dati degli utenti all'interno di una sola classe e evitare l'apertura/chiusura istantanea delle connessioni TCP che ogni utente richiederebbe con dati errati. La tecnologia RMI callback permette al Client registrato al servizio di notifica di avere costantemente una lista aggiornata sullo stato online/offline degli utenti registrati all'applicazione. Il Client tiene in locale una struttura dati aggiornata, senza dover richiedere ogni volta al Server una copia della lista aggiornata. Unica eccezione viene fatta dopo il login dell'utente al servizio, in quel caso il Server invierà la copia attuale contenente lo stato degli utenti. Successivamente verrà aggiornata dal Client tramite il servizio di notifica. La scelta di inviare dopo il login una lista è stata considerata migliore rispetto ad una procedura forzata da parte del Server che attraverso il servizio di notifica facesse ricreare al Client la lista con lo stato attuale degli utenti.

TCP

Il Server crea una connessione TCP con il client nella quale avviene l'effettivo utilizzo dell'applicazione Worth. Il Server legge le richieste in arrivo dal Client ed elabora le risposte. Per fare ciò utilizza i metodi offerti dalla struttura dati dell'applicazione Worth con i parametri ricevuti dal Client. Il Server si occupa solamente di controllare la validità dei comandi ricevuti, e se validi, di mandare il risultato di tale operazione al Client. Il Server è implementato usando la tecnologia Mutiplexed I/O con JAVA NIO ed è single thread model. La motivazione di tali scelte è dovuta ad un ipotetico uso dell'applicazione Worth da parte dell'utente e dal tipo di operazioni che deve effettuare il Server. Il caso d'uso immaginato vede un utente loggarsi a Worth ad inizio della sua giornata lavorativa ed effettuare il logout prima a fine giornata. Durante questo periodo però verranno eseguite poche operazioni visto il tipo di servizio che offre Worth.

Infatti è facile immaginare che l'utente aggiungerà/aggiornerà un task circa un volta all'ora ed effettuerà altre operazioni come la creazione di un progetto ancora più raramente. Questo perché task o progetti troppo brevi non avrebbero la necessità di essere caricati su una piattaforma di gestione condiviso. Lo scenario che si crea vede un alto numero di utenti connessi all'applicazione ma un numero relativamente basso di richieste per il Server. Avere un thread per ogni utente connesso sarebbe uno spreco di risorse risultando poco efficiente. Inoltre le operazioni da eseguire sul Server hanno un carico di lavoro molto basso, quindi la scelta di un solo thread è stata ritenuta comunque più adatta rispetto all'utilizzo di un numero limitato di thread.

Client

Il Client inizialmente crea una connessione con il Server RMI per poi richiedere una connessione con il Server TCP. Oltre a connessioni TCP utilizza anche UDP per inviare pacchetti sugli indirizzi IP multicast dei progetti. La struttura dati del Client contiene informazioni sul suo stato attuale come: se è loggato a Worth, l'utente con il quale si è loggato a Worth, la lista aggiornata dello stato online/offline degli altri utenti. Il suo funzionamento è molto semplice: una volta avviato l'utente dovrà scrivere da terminale i comandi che desidera eseguire. Come risposta riceverà sempre una stringa stampata a schermo con il risultato dell'operazione. Utilizzando lo stesso Client è possibile effettuare il login/logout di diversi utenti. Dopo tale avvenimento tutti i dati della passata sessione verranno cancellati così da non alterare l'esperienza degli utenti.

Schema generale dei threads e delle strutture dati

L'analisi verrà fatta utilizzando la divisione del paragrafo precedente:

RMI

La politica della JAVA RMI prevede l'implementazione automatica del multithreading portando il server a non essere thread safe. È necessario sincronizzare l'accesso all'oggetto remoto offerto da RMI per far sì che più client con richieste concorrenti non portino ad uno stato inconsistente. L'implementazione del Server RMI all'interno di Worth prevede l'utilizzo di lock implicite utilizzando metodi synchronized nell'oggetto remoto.

TCP

Il Server TCP è single thread model quindi il problema della concorrenza non viene sollevato. Anche la struttura dati di Worth non necessita di alcun tipo di sincronizzazione dato che le operazioni su di essa possono essere richiesta solamente da un thread alla volta (può esserci al massimo un thread nelle operazioni di scrittura o lettura di un SocketChannel).

CLIENT

Il Client è un processo single thread e tutte le strutture dati gestite da esso non hanno bisogno di meccanismi di sincronizzazione. L'unica eccezione è presente nella classe che implementa la chat di gruppo dei progetti. Questa classe contiene un thread secondario che ha il compito di creare una connessione UDP e unirsi al gruppo multicast del relativo progetto. Dunque per ogni progetto ci sarà un thread secondario. Oltre a ricevere i pacchetti spediti sul canale, il thread deve aggiungerli ad una lista che verrai poi letta in maniera asincrona dal client. I metodi di scrittura/lettura dei messaggi sono sincronizzati utilizzando synchronized. In questo modo si evita che il Client provi a leggere i messaggi mentre il thread secondario sta ancora scrivendo, e viceversa.

Elenco delle classi definite

L'elenco verrà riportato in ordine alfabetico:

- Card: rappresenta un task all'interno del progetto. Viene creata ogni volta che un utente desidera aggiungerla ad un progetto oppure recuperata dal file system durante l'avvio del server.

- Chat: classe utilizzata per implementare la chat del progetto. Contiene i metodi per unirsi al gruppo ip multicast e ricevere i messaggi, leggere i messaggi ricevuti e terminare l'ascolto sul canale UDP.

- ChatClient: rappresenta la chat di progetto utilizzata dal client per mandare e ricevere messaggi. Contiene un'istanza della classe Chat c, un thread t che utilizza il metodo start() di c per unirsi al gruppo multicast e ricevere i messaggi e infine una stringa che identifica il progetto.

- Client: la classe si occupa di stabilire le connessioni con il Server di Worth e di comunicare con esso attraverso l'invio di stringhe digitate dall'utente tramite terminale. Il Client contiene anche una lista di ChatClient relativi ai vari progetti di cui l'utente è membro. Prima di inviare un'operazione esegue un controllo sulla validità di tale operazione e sul numero di parametri passati. Questo controllo più quello successivo eseguito lato Server permettono di garantire l'esecuzione di operazioni legittime. Client ha un loop nel quale l'utente richiede le funzioni e rimane in attesa della risposta da parte del server. Non è possibile fare più di una richiesta senza aver ricevuto prima la risposta dal Server. Un utente può effettuare l'operazione di login/logout a suo piacimento. Lo stato del Client verrà aggiornato e di conseguenza anche le nuove operazioni permesse. È possibile chiudere il Client una volta disconnessi con il comando "quit".

- EventManager: gestisce gli eventi per la registrazione a Worth, la richiesta di login, la presenza di un utente in Worth, la registrazione al servizio di notifica e il metodo per inviare le notifiche ai client. Implementa EventManagerInterface.

- EventManagerInterface: interfaccia offerta al client per RMI.

- Info: rappresenta le informazioni generali del progetto. Contiene nome progetto, indirizzo ip multicast e la lista dei membri. Viene generata con la creazione di un nuovo progetto o recuperata dal file system quando il Server viene avviato.

- ListaUtenti: classe che contiene l'elenco degli utenti registarti al servizio. Viene utilizzata per il parsing del file contenente gli utenti registarti all'avvio del Server.

- MainClassClient: classe contenente il metodo main. Utilizzata per avviare il Client.

- MainClassServer: classe contenente il metodo main. Utilizzata per avviare il Server.

- NotifyEventImpl: classe che implementa i metodi di notifica che deve utilizzare il server per informare il client. Lo stato online/offline degli utenti viene notificato al Client attraverso l'uso di questi metodi. Contiene anche i metodi per leggere i dati aggiornati via notifica. Implementa NotifyEventinterface.

- NotifyEventinterface: interfaccia del server

- Project: classe che rappresenta un progetto. Contiene un'istanza della classe Info e quattro liste rappresentanti ognuna uno degli stati del flusso di lavoro dei task. Contiene metodi per modificare lo stato dei task presenti.

- RandMulticastAddress: classe contenente un metodo statico che restituisce un indirizzo ip multicast globale random compreso tra 224.0.1.0 a 238.255.255.255.

- ServerRMI: Server che utilizza la tecnologia RMI per offrire vari servizi al client. Contiene il metodo start() per essere avviato all'interno del metodo MainClassServer.

- ServerWorth: la classe rappresenta il Server adibito ad accettare le connessioni tcp dai client ed a eseguire le loro richieste. Il server contiene un selettore con il compito di registrare i nuovi canali ed eseguire le operazioni di lettura/scrittura sul buffer. Durante l'operazione OP_READ viene controllata la validità dei parametri dell'operazione e in seguito eseguita sulla struttura dati Worth. Il risultato di tale operazione viene scritto durante l'operazione di OP_WRITE e inviato al Client.

- Utente: la classe rappresenta un utente all'interno dell'applicazione Worth. Viene creata dopo l'avvenuta registrazione a Worth oppure recuperata dal file system durante l'avvio del Server.

- UtenteStatus: classe rappresentate lo stato online/offline di un utente all'interno dell'applicazione.

- Worth: la classe rappresenta la struttura dati utilizzata dall'applicazione per gestire i progetti. Il metodo costruttore recupero sul file system tutti i progetti contenuti nella cartella Worth_project (se presente) e li aggiunge ad una lista. La struttura dati contiene tutti i metodi necessari per interagire con i progetti e i relativi task. Le funzioni restituiscono una stringa che verrà utilizzata dal Server come risposta alla richiesta del Client.

Sintassi dei comandi

Comandi validi se l'utente è disconnesso:

- **register [username] [password]** : registrazione all'applicazione Worth
- **login [username] [password]** : login sull'applicazione Worth
- **quit** : terminazione del client
- **help**: lista dei comandi disponibili

Comandi validi se l'utente è connesso:

- **list_users** : restituisce l'elenco degli utenti registrati e il loro stato online/offline
- **list_users_on** : restituisce l'elenco degli utenti registrati online
- **send [nome_progetto] [msg]** : invia alla chat di gruppo msg
- **read_chat [nome_progetto]** : legge i messaggi sulla chat di gruppo
- **add_member [nome_progetto] [nome_membro]** : aggiunge un nuovo membro al progetto
- **logout** : effettua il logout dall'applicazione
- **create_project [nome_progetto]** : crea un nuovo progetto
- **list_projects :** restituisce l'elenco dei progetti di cui l'utente connesso è membro
- **join_chat [nome_progetto] :** l'utente connesso si unisce alla chat del progetto. NOTA: prima di effettuare le operazioni di invio e lettura sulla chat bisogna aver effettuato l'operazione di join alla chat del progetto, altrimenti verrà restituito errore

- **show_members [nome_progetto] :** restituisce l'elenco di tutti i membri del progetto
- **show_cards [nome_progetto] :** restituisce tutte le card contenute nel progetto
- **show_card [nome_progetto] [nome_card] :** restituisce tutti i dettagli della card appartenente al progetto
- **add_card [nome_progetto] [nome_card] [descrizione] :** aggiunge una nuova card al progetto
- **move_card [nome_progetto] [nome_card] [destinazione] :** muove la card nel progetto verso un nuovo stato del flusso di lavoro
- **show_history [nome_progetto] [nome_card] :** restituisce la storia del flusso di lavoro della card
- **cancel_project [nome_progetto] :** elimina il progetto se tutti i suoi task sono terminati

Note

Il programma non prevede nessun argomento da dover passare da terminale.

L'unica libreria esterna utilizzata è Jackson 2.9.7 per serializzare oggetti Java in JSON e viceversa.

L'invio dell'indirizzo ip multicast del progetto avviene tramite connessione tcp con il Server. Dopo aver effettuato la richiesta, è possibile effettuare le operazioni di scrittura e lettura della chat. Questo permette ad un utente loggato che nel frattempo viene aggiunto come membro ad un nuovo progetto di poter entrare nella chat una volta eseguito il comando.

Per realizzare l'applicazione è stato utilizzato del codice contenuto nelle soluzioni alle esercitazioni proposte durante il corso.

Istruzioni

1 Estrarre i file nella cartella compressa

2 Compilare il codice con il comando **javac -cp lib/* *.java**

3 Avviare il server con il comando **java -cp lib/* MainClassServer**

4 Aprire un secondo terminale e avviare il client con il comando **java -cp lib/* MainClassClient**