Relazione progetto Roberto Lorenzetti (531747)

Server

Il server ha una struttura NIO per gestire le connessioni con i client. All'avvio della classe Main, tale classe invoca immediatamente il metodo server.start appartenente alla classe ServerServ la quale implementa effettivamente la struttura NIO. Tramite questo metodo, come prima cosa si inizializzano tutte le variabili e le relazioni necessarie per implementare il sistema NIO (bind, configureBlocking(false), register ecc..) successivamente esporta l'oggetto remoto per le connessioni RMI. Prima ancora di entrare nel classico ciclo while del NIO, carica i dati dalla cartella Store presente nel package del progetto e avvia il thread che esegue il Task Console Reader. Quest'ultimo semplicemente legge cosa viene scritto in console, quando viene scritta la parola "exit" avviene la "corretta" terminazione del server. Corretta perché, uscendo dal programma in questo modo, viene effettuato il backup dei dati che sono stati aggiunti durante l'esecuzione. In particolare viene fatta una store dei dati nella cartella Store citata prima.

Finalmente si entra nel ciclo while del NIO che ha la classica struttura che abbiamo visto a lezione, in particolare ho utilizzato quasi le stesse notazioni presenti nella soluzione dell'assegnamento riguardante il NIO, proprio perché le ho trovate intuitive e semplici da "leggere".

Quando viene accettata una nuova connessione, ad essa viene associato un attachment di riferimento, oggetto della classe Attachment Info. Contiene al suo interno tre semplici informazioni: il buffer di lettura/scrittura, una stringa di

risposta da mandare al client e una stringa che rappresenta l'username del client associato a quella particolare connessione. Quando viene accettata una nuova connessione l'username viene messo ad ignoto. Quando invece la connessione è identificata (dopo un login con successo) l'username reale viene inserito nell'attachment. L'associazione esiste per gestire le chiusure improvvise da parte del client. In questo modo, se un client viene interrotto bruscamente (senza rispettare l'iter standard di chiusura) il server riesce a capire chi si è disconnesso, in modo tale da aggiornare la lista degli utenti online, levare la registrazione dalle callback e cancellare la chiave. Così il server rimarrà persistente in qualsiasi caso e gestirà le callback in modo corretto.

Il ciclo while interessato legge cosa viene richiesto da parte del client e, una volta ricevuta la richiesta nel completo, esegue il metodo elaborateRequest di un oggetto della classe ManagerRequest. Come dice il nome, un oggetto di tale classe si occupa di capire cosa è stato richiesto dal client, interrogare la struttura interna e ritornare una stringa di risposta. La stringa verrà poi mandata al client dal chiamante (il while relativo NIO).

La classe ManagerRequest possiede tre variabili di istanza: l'oggetto remoto esportato, una concurrent Hash Map che conserva le associazioni tra utenti registrati e password e una variabile di tipo Data, quest'ultima variabile contiene quasi tutta la struttura dati del sistema. Le tre variabili sono utilizzate dall'unico metodo presente nella classe (ElaborateRequest) che si occupa semplicemente di fare un parsing della richiesta e ottenere tutte le informazioni necessarie per interrogare le tre variabili di istanza.

La seconda variabile di istanza citata, è una concurrent HashMap perché è la stessa utilizzata dall'oggetto esportato per il metodo RMI di registrazione, la terza è , come detto precedentemente un oggetto per mantenere le strutture dati.

Un oggetto Data contiene quattro campi, una lista di progetti projects, una lista di nomi dei progetti projectsList, una lista di utenti online una insieme di indirizzi ip. Questa classe ha diversi metodi, divisibili un tre categorie. La prima riguarda la gestione dei progetti, la seconda riguarda gli utenti online e la terza riguarda il backup.

La prima contiene i metodi per creare un progetto, ottenere la lista dei progetti, ottenere un progetto tramite nome. La seconda per aggiungere membri per aggiungere utenti alla lista online ed ottenerla, la terza tutto il necessario per il backup(metodi load store).

Quando un progetto viene creato, viene scelto un indirizzo ip multicast da associargli. Questo indirizzo ip viene scelto casualmente nel range di indirizzi ip per poi essere associato al progetto. Si tiene traccia di tutti gli indirizzi già associati tramite un insieme. La vastità di indirizzi ip permette di effettuare questa scelta.

La classe project contiene tutte le informazioni relative ad un progetto, quindi i vari campi come il nome, il creatore, la lista dei membri ecc. Implementa inoltre tutti i vari metodi di getting, setting e adding.

Analogamente la classe Card contiene tutti i campi relativi alla card e i metodi per gestire le operazioni su di esse.

L'ultima classe da citare è la UserServerImp, la classe che implementa l'interfaccia per l'esportazione dell'oggetto

remoto. Questa classe utilizza due concurrent hash map, userRegisterMap e userReference. La prima mantiene l'associazione (user->passw) la seconda mantiene l'associazione (user->NotifyEventInterface) per gestire le terminazioni improvvise dei client e per supportare i metodi di registrazione , non registrazione per le callback. La concorrenza è gestita tramite le collezioni concurrent e una variabile, syncObj, di tipo object utilizzata come oggetto di riferimento per le lock (synchronized(syncObj)). Le concurrent hash map mi garantiscono la gestione corretta della concorrenza per quanto riguarda le operazioni sulle mappe, la variabile syncObj per gestire gli accessi e le modifiche alla lista di NotifyEventInterface clients, utilizzata per effettuare il ciclo di callback tramite un iteratore.

Lista classi server:

Server Main: utilizzata per avviare il server e fare il parsing degli argomenti

ServerServ: utilizzata per implementare il ciclo NIO e tutte le varie operazioni di setup

ManagerRequest: utilizzata per elaborare le richieste del client

Data: per gestire tutte le strutture dati e per fare la load e la store

Project: per concretizzare l'oggetto che rappresenta il progetto

ProjectData: contiene alcuni informazioni sui progetti

UserServerImp: classe per l'oggetto remoto

Card: concretizza l'oggetto card

MulticastAddress: classe utilizzata per la scelta dell'indirizzo ip.

Client

Il client si avvia tramite la classe ClientMain, tale classe avvia semplicemente il programma facendo il parsing degli argomenti passati e poi avvia un oggetto della classe LoginPage. Tale classe implementa la prima interfaccia grafica visibile, quella di login. L'interfaccia è molto semplice e presenta un text field per il nome utente e un text field per la password. Questi text field servono per inserire l'username e la password. I campi non possono essere vuoti e non possono contenere asterischi.

Se il login viene effettuato con successo si entra nella schermata WelcomePage, in questa schermata si possono fare tutte le operazioni chiaramente esplicitate dalle scritte presenti a schermo e che quindi non avrebbe senso ripetere in questa relazione. Ci sono due cose che però è doveroso dire, la prima è che i bottoni refresh accanto ai combobox servono per ottenere la lista corrente da visualizzare.Per esempio, appena entrati in questa pagina tutti i combo box saranno vuoti finché non viene premuto il bottone refresh accanto. Quindi, sempre per esempio, se si preme il bottone refresh accanto al combo box relativo al bottone open project, si otterrà la lista dei progetti di cui l'utente è membro. Dopodiché si può selezionare il progetto da aprire per entrarci. Se qualcun altro nel frattempo mi aggiunge ad un nuovo progetto, devo riaggiornare la lista tramite il bottone refresh per visualizzarlo tra le opzioni possibili di apertura. Stessa cosa varrà anche per le pagine successive.

La seconda cosa doverosa da citare sono il modo con cui sono gestite le notifiche di callback. Quando siamo nella schermata welcome page, comparirà una campanella accanto ai bottoni per aprire le liste degli utenti registrati/ online. Una volta cliccata la campanella, comparirà la lista colorata con i nuovi utenti registrati o rispettivamente, online. Il colore blu indica un nuovo utente registrato, colore verde nuovo utente online, il colore grigio utente andato offline. Tutte queste differenze di colore si riferiscono alla lista che si ottiene premendo i bottoni get list. Quindi le differenze di colore si presenteranno rispetto alla lista nera ottenuta tramite quel bottone. Per essere più chiari se, arriva una notifica di un nuovo utente registrato, e io clicco sulla campanella compariranno in blu i nuovi “x” utenti registrati. Se ora non clicco su get list, e si attiva di nuovo la campanellina perché nel frattempo si sono registrati “y” utenti visualizzerò in blu x+y utenti. Gli x di prima e gli y nuovi. Quindi per resettare i colori occorre premere il bottone get list che refresha la lista per la differenza dei colori.

Se si apre un progetto si entra nella schermata ProjectPage, con le stesse regole di quella precedente. Ora se arriva una notifica di callback, non si vedrà nulla a schermo ma se si torna alla schermata precedente tramite il bottone undo, la campanella sarà attiva e sarà possibile cliccarci. Questa cosa vale per tutte le pagine successive, quindi quelle relative alla chat e alla card.(ChatPage e CardPage).

Una cosa da aggiungere è la gestione della concorrenza. Infatti quando si utilizza la libreria swing, utilizzata per l’interfaccia grafica, il thread event dispatcher gestisce le interazioni grafiche. Proprio per questo motivo per

aggiornare l'interfaccia grazie alle callback occorre utilizzare il metodo invoke later per far sì che sia l'event dispatcher ad l'unico e il solo thread ad interagire con le rappresentazioni e le risorse grafiche. Il metodo invoke later utilizza un task che implementa un metodo runnable.

La stessa cosa avviene quando si apre la schermata di chat, infatti lì avremo un task chat reader utilizzato da un thread che si occupa di leggere i messaggi. La lista dei messaggi a schermo viene aggiornata analogamente.

Lista classi client

ClientMain

ClientServices: avvio

LoginPage: schermata login

MyListCellRender: rendering celle per i colori

NotifyEventImpl: oggetto callback

TaskCallback

TaskMessage : task per aggiornare la chat con invwokelater

TaskReader

UpateComboBox

UpdateListBox

WelcomePage

CardPage

ChatPage

Istruzioni

Per avviare il programma correttamente bisogna eseguire i seguenti passi in ordine:

Estrarre l’archivio consegnato

Importare su eclipse il file progetto.zip

Spostare/copiare manualmente le cartelle Imm e Store nella cartella progettoserver presente nella workspace di eclipse (una volta importato l’oggetto, bisogna andare nella workspace di eclipse dove sarà comparsa la cartella relativa al progetto. Dopodiché bisogna inserire le due cartelle in Progetto>progettoReti>src>progettoServer)

(Purtroppo l’importazione automatica modifica i path, non sono riuscito a trovare un modo migliore, la cartella Imm contiene le immagini dei bottoni e la cartella Store server per il backup)

Eseguire ServerMain(se si usa senza argomento verrà utilizzata la porta 9999)

Eseguire ClientMain(con argomenti 127.0.0.1 9999)