Hemuppgift 3: RMI and Databases

Nätverksprogrammering, ID1212

Evan Saboo
saboo@kth.se

2017-12-03

https://github.com/ZpeedX/Network-Programming/tree/master/Homework%203

1 Introduktion

Målet tredje uppgiften är kunna utveckla en applikation som använder ”Remote Method Invocation” för att kommunicera mellan olika processorer och man ska kunna spara, hämta och ta bort data från en databas.
Applikationen ska ha en strukturerad arkitektur för att kunna dela upp logik och användargränssnitt.
Samma som i förgående hemuppgifterna ska serven vara multi-trådad för att kunna kommunicera med flera klienter.

Applikationen jag utvecklade är en klient-server applikation som gör det möjligt att lagra till, hämta från och ta bort filer från en filkatalog. Varje klient hanterar varsin användare med hjälp av användargränssnitt och Server hanterar olika operationer som klienter vill utföra.

2 Litteraturstudie

Uppgiften utfördes med hjälp av dessa källor:

* Jag fick mest hjälp av Objekt-Orienterad Middleware- och Databas hanterings föreläsningarna. Min applikation har fått inspiration från chatt- och bank applikations exemplaren.

* Jag använde StackOverflow hemsidan för att få hjälp med problem som uppstod och för att få kort och enkel information om några java syntaxer.[1]

* Java dokumentationshemsidan var också hjälpsam för att hitta Java syntaxförklaringar.[2]

[1] https://stackoverflow.com
[2] https://docs.oracle.com/javase/7/docs/api/

3 Metod

För denna hemuppgift behövde jag gå igenom två stora delar, RMI (Remote Method Invocation) och JDBC Databasåterkomst:

- RMI möjliggör objektfunktionssamtal mellan noder som kör Java Virtuella Maskiner. En JVM kan anropa metoder som hör till ett objekt som är lagrat i en annan JVM och då kan man returnera ett svar som skickas tillbaka till första JVM.
  I första och andra hemuppgiften använda vi sockets för att kommunicera mellan två noden men nu kan vi istället använda RMI för kommunikation och hantering av kommunikationsproblem, vilket underlättar applikationsutvecklingen.

- JDBC APIn används för att koppla, lägga till, ändra på och ta bort data i databaser. Jag behövde gå igenom hur APIn fungerar och vad för java kod man behöver skriva för att utföra databas operationer.

RMI var jobbigare att förstå än databasåterkomst. Eftersom jag har tidigare jobbat med databaser och SQL så var inte jobbigt att implementera funktionaliteterna, men jag behövde gå igenom RMI två gånger för att förstå hur två noder kopplade- och kommunicerade med varandra.

Applikationen jag utvecklade tog mycket mer tid än förväntat. En faktor var att jag behövde gå igenom två stora delar jämfört med de förgående hemuppgifterna. En annan faktor var att jag behövde implementera flera operationer som klienten ska kunna utföra, där en av dem var ganska jobbig att implementera.

4 Resultat

I figur 1 visas en demonstration när applikationen kör. Klienten kopplas automatiskt till serven när klientsidan startas och användaren då kan börja använda programspecifika operationer.

Arkitektur:
Applikationen använder MVC arkitekturen för att separera användargränssnitt och logik (Figur 2).
Vi kan se att klientsidan har 3 paket, vilket är View, startup och Net. I startup finns klassen Main för att starta klienten och koppla till serven. Eftersom vi använder RMI och serven hanterar all logik behöver vi bara View för att anropa metoder från servern. Med hjälp av RMI kan View se servens metoder som lokala metoder, då vi behöver bara kalla på en metod för att den ska dirigeras till serven.
Net paketet används av view bara för att skicka till och ta emot filer från server genom TCP sockets.
Då vi använder RMI behöver vi bara server controller paketet för att tillhandhålla metoderna klienten behöver. Man kan tänka sig att view (klient) kallar på metoder från controller (server) och controller hämtar metoderna från model (server) för att returnera till klienten, vilket följer MVC arkitekturen.
Model paketet används för att hantera användardata och användaroperationer.
Klassen User används för att hantera en användare medans UserManager klassen hanterar alla inloggade användare.
Serven net paket har samma funktionalitet som klientens paket, dvs. den används för att skicka till- och ta emot filer från klienten via TCP sockets.
Integrations paketet i serven används för att hämta data från databasen för att sedan bli hanterad av modell paketet.
Common paketet har klasser som båda servern och klienten behöver och har gränssnitts klasser som RMI behöver för att överföra metoder mellan serven och klienten.

RMI:
Hur kopplas klienten till serven?
När servern startas kollar den först om det finns något register i RMI register som den kan binda till. Detta görs genom att först hämta alla register från RMI registers standard port (1099) med hjälp av metoden getRegistery().list(), om ingen register hittas skapas då en ny register i samma port.
Sedan är det bara att binda ett register namn (String) som både serven och klienten känner till genom att använda metoden Naming.rebind(), man ska också binda ett

objekt som ska köras av RMI, vilket i vårt fall är ett nytt objekt av Controller klassen. När en ny klient vill koppla till servern så letar klienten först efter registernamnet med hjälp Naming.lookup(). När registret hittas hämtas gränssnittsklassen som server controller implementerar. Gränssnittet heter FCInterface och finns i common paketet, vilket ”extendar” Java RMI Remote gränssnittet.
Då controller klassen implementerar gränssnittet vet RMI att metoderna i gränssnittet ska användas av noder som ska kalla på dem. På så sett kan klienten använda metoderna i gränssnittet som lokala metoder.

Eftersom RMI använder flera trådar för att hantera flera kommunikationer så blir serven multi-trådad.

Databasåterkomst:
Jag använde mig av JDBC för att koppla till och utföra databas operationer på Netbeans lokala databas. Databasen heter Java DB och är en implementation av Apache Derby.
I FileCatalogDAO klassen finns alla databas operationer. För att koppla till databasen används JDBC Driver Manager med port nummer 1527 och databasnamnet.
För att exekvera SQL kommandon i databasen kan man använda Objektet ”Statement” eller PreparedStatement. Jag använde PreparedStatement eftersom den förkompilerar SQL påståendet och jag behöver bara ersätta alla frågetecken”?” i påståendet med värden för att kunna sedan köra den. Med PreparedStatement kan man också skydda sig från SQL-injektioner.

Nedan beskrivs applikationens funktionaliteter:

Registera ny användare:
En ny användare kan registrera sig i fil katalogen genom att mata in användarnamn och lösenord.
När serven får in användaruppgifterna kollas först om användarnamnet redan finns i databasen eftersom användarnamnet måste vara unikt, annars läggs uppgifterna i databasen.

Avregistrera användare:
Om en användare finns i databasen kan den då tas bort genom att användaren förser med korrekt användarnamn och lösenord. Alla privata filer som användaren äger i databasen tas också bort vid avregistreraringen.

Användarinloggning:
När användaren har registrerat sig kan hen logga in. Vid inloggning skapas ett nytt objekt av klassen User (modell) och läggs in i hashmappen (finns i UserManager klassen) som håller koll på alla inloggade användare. Med User objektet läggas också

en ny slumpad long som nyckel i hashmappen. Nycken skickas tillbaka till klienten för användas som en unik användaridentifikation varje gång användaren vill göra en annan operation.

Användarutloggning
Om användaren är inloggad kan hen logga ut och då kommer User objektet tas bort från hashmappen med hjälp av user ID:n.

Hämta en lista av alla filer och des egenskaper:
En inloggad användare kan hämta en lista av alla filer hen har tillgång till. Användarens privata filer och alla publika filer hämtas som en lista av FileDTO objekt. I modell paketet finns en klass (FileHandler) som har information på en fil, vilket implementerar gränssnittet FileDTO, som finns i common paketet. Gränssnittet ”extendar” Serializable vilket betyder att objekt av gränssnittet kan omvandlas av stream av bytes eller byte array för att sedan kunna skickas över RMI.
När klienten får en lista av objekt från serven kan hen använda objektens metoder för att ta ut varje fils egenskaper såsom filnamn, filägare, filstorlek, tillgång, behörighet. En annan anledning till varför FileDTO används är att man vill bara tillåta klienten kunna hämta metoder som gränssnittet tillåter. Detta hjälper också utvecklaren att inte kalla på metoder vid fel ställe i koden.

Ladda upp en fil:
När en användare vill ladda upp en fil skapas först en sessions Id genom java metoden random.nextInt() och sedan skickas vidare till serven genom RMI metod kallelse. Serven kontrollerar först om filen finns redan i databasen innan den läggs till. När filens information har lagts i databasen returneras en bekräftelse (boolean) till klienten. När klienten får klartecknet så kallas en metod i klient net paketet för att skapa en ny socket och skicka sessions ID:et till serven via socketen. Klienten börjar sedan skicka fixerad antal av byte array hela tiden tills hela filen har skickats.
Servens socket tråd tar metod ny socket koppling genom ServerSocket.accept() och lägger klientens socket i en Hashmap där nykeln är session ID:et och värdet är socket. En annan tråd i serven skapas av RMI tråden genom att kalla på CompletableFuture och denna tråd får i uppgift att hämta klient socket från hashmapen genom att kolla med sessions ID:et och sedan hämtas all data från socket input stream. Varje gång nya data hämtas från input stream så läggs den i en ny fil som har samma namn som klientens fil.
Klienten och serven stänger kopplingen när filen har överförts.

Ladda ner en fil:
När användaren vill ladda ner en fil från filkatalogen utförs samma operationer som när man ska ladda upp en fil. Först kollar serven om filen finns i databasen och kollar

om filen tillhör användaren om den är privat, annars kan vem som helst ladda ner filen om den är publik. Sedan skickas filen genom sockets till klienten.

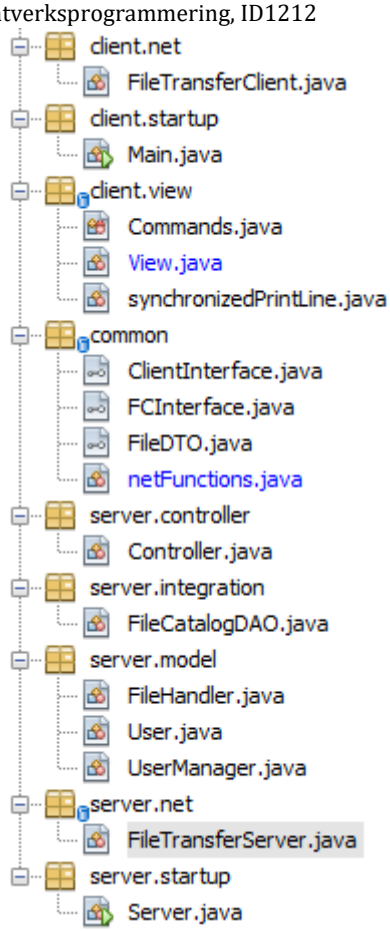
Uppdatera och ta bort en fil:
Bara ägaren till private filen kan ändra innehållet eller fil rättigheterna. Om filen är publik och skrivbar kan vem som helst ändra filinnehållet men inte rättigheterna, men om publika filen är bara läsbar så kan ingen förutom ägaren ändra på filen.

Skicka notis när ägarens fil ändras:
När en annan användare än filens ägare uppdaterar, laddar ner eller tar bort filen så skickas en notis till ägaren om hen är inloggad. I figur 1 kan man se att två personer gjorde totalt 3 ändringar på användarens publika fil.
När en ny användare försöker logga in skickas ett objekt (vars metod skriver meddelande till användaren) som parameter i inloggnings metoden för att sedan skickas till servern via RMI. Objektet sparas sedan i User klassen för att sedan kunna användas för notis.
Objektets klass implementerar gränssnittet ClientInterface som finns i common paketet. Gränssnittet implementerar också Remote vilket gör så att RMI kan skicka över utskriftmetoden som servern ska använda.
Server tråden som ändrar på filen tilldelar notis uppgiften till en ny tråd i tråd polen (CompletableFuture). Den nya tråden letar först efter ägarens User objekt genom att lopa genom hashmappen tills ägarens namns hittas i User objektet. Sedan används utskriftsmetoden från Objektet get från User objektet och lägger in notismeddelandet som parameter i metoden för att sedan skrivas ut till användaren via RMI.

```
------------------------------------------------------------------------
Building FileCatalog 1.0
------------------------------------------------------------------------

--- exec-maven-plugin:1.2.1:exec (default-cli) @ FileCatalog ---
> Welcome to file catalog. To list all commands type 'help'.
> help
> REGISTER user - usage: register <username> <password>
> UNREGISTER user - usage: unregister <username> <password>
> LOGIN user - usage: login <username> <password>
> LOGOUT user - usage: logout
> LIST files - usage: list
> UPLOAD file - usage: upload <filepath and/or filename> <choose: public/private> <if public: write/read>
> DOWNLOAD file - usage: download <filepath and/or filename>
> UPDATE file - usage: update <filepath and /or filename> <if owner choose: public/private> <if owner: write/read>
> DELETE file - usage: delete <filename>
> register root root
> Registration was successful, you can now login.
> login root root
> Welcome root
> list
Name:          Owner:         Size:          Is Public:     Is Writable
test2.txt      pass           24 B           true           false
> upload test2.txt private
> File already exists
> upload test3.txt private
> File uploaded successfully
> upload test4.txt public write
> File uploaded successfully
> list
Name:          Owner:         Size:          Is Public:     Is Writable
test2.txt      pass           24 B           true           false
test3.txt      root           24 B           false          false
test4.txt      root           24 B           true           true
> update test2.txt
> File is set to Read-Only.
> Your public file 'test4.txt' was updated by kalle.
> Your public file 'test4.txt' was downloaded by pass.
> Your public file 'test4.txt' was deleted by pass.
> list
Name:          Owner:         Size:          Is Public:     Is Writable
test2.txt      pass           24 B           true           false
test3.txt      root           24 B           false          false
> unregister root root
> You have to logout before unregistering.
> logout
> You have successfully logged out.
> unregister root root
> User unregistration successful
>
```

Figur 1: Användargränssnittet.



Figur 2: Applikationens filstruktur.

5 Diskussion

Kraven nedan uppfylldes:

- Programmet följer MVC arkitekturen för att ha bättre struktur på logiken och användargränssnittet och kunna kontrollera alla operationer som används mellan dem.
- En användare kan registrera sig, avregistrera, logga in och logga ut från katalogen.
- Användaren kan också ladda upp, ta bort och ladda ner filer från filkatalogen. Detta görs via TCP sockets.
- Serven tar hand om all data och logik medan klienten hanterar användarens inmatningar
- Klient och server kommunicerar via RMI och bara serven registrerar sig själv i en RMI register. När serven kallar på en klient utförs det via en fjärr referens som fås från klienten.
- Servern använder en databas för att spara användar- och fil information.

Med en sådan stor applikation kan det förkomma flera problem under utvecklingen. Jag stötte nästa alltid på små problem som tog högst 10 minuter att felsöka. En av dem var att socket sessions ID:et skickade alltid värdet 1 till serven och detta gjorde så att jag var tvungen att använda netbeans debug verktyget för att hitta felet. Men det visade sig att jag råkade skriva värdet 1 som parameter i view klassen när jag kallade på metoder från net.
Den störta svårigheten jag hade var filöverföringen via TCP sockets. Min första var att ServerSocket skapades varje gång en ny klient vill överföra en fil, vilket var ganska dålig implementation eftersom man inte kunde skapa två ServerSockets som lyssnar på samma port.
Jag förbättrade min implementation genom att skapa en ny tråd som lyssnar hela tiden på port numret och när en ny klient kopplades så la jag den i en hashmap som sedan användes av en annan tråd.

Jag är inte helt nöjd med applikationen eftersom koden är inte helt strukturerad och jag ville använda JPA istället för JDBC men jag hade inte mycket tid att gå igenom den då jag hade bara fyra dagar kvar på mig att utveckla hela programmet.

6 Kommentar om kursen

Jag har spenderad 10–12 timmar åt att gå igenom föreläsningarna innan jag började med hemuppgiften. Tiden det tog att utföra hemuppgiften tog 24–25 timmar. Rapporten tog ungefär 5 timmar att skriva.