Java Remote Method Invocation (RMI)

Specification

- Java Remote Method Invocation Specification Standard Edition v1.5.0

Innehåll

1 Bakgrund

RMI är Javas version av Remote Procedure Call/Distributed Objects (RPC/DO). RMI kräver att alla applikationer är skrivna i Java. RMI ingår som standard i JSE och JEE (dock ej i JME ännu). Några målsättningar med RMI har varit;

- Göra det så enklelt som möjligt att skriva distribuerade applikationer.
- Metodanrop på objekt i andra JVM:er skall kunna göras på samma sätt som anrop på objekt i samma JVM (= transparent, d.v.s. båda typerna av anrop görs på samma sätt i koden, samma syntax). Exempel?

```
object.doACall()    //Var finns objektet?? Kan vara fjärrobjekt!
```

  (dock man ser på excepetion-typerna om det handlar om fjärranrop)
- Bibehålla Javas typsäkerhet och säkerhetsmodell (security managers m.m.).
- Göra det möjligt för serverapplikationer att anropa applets (callbacks).

2 Begrepp

Lokalt objekt (Local object) Ett objekt som bara kan användas inom en JVM.

Lokalt anrop (Local call) Ett anrop på ett lokalt objekt.

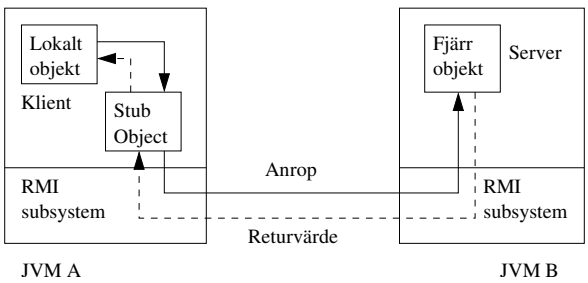
Fjärrgränssnitt (Remote interface) Alla fjärrobjekt beskrivs m.h.a. ett fjärrgränssnitt (Java interface) där objektets metoder deklareras. Det är fjärrgränssnittet klienten använder, se nedan.

Fjärrobjekt (Remote object) Implementeringen av fjärrgränssnittet. Ett objekt som accepterar anrop från andra JVMer (även från lokal objekt i samma JVM accapteras). Se nedan.

Fjärranrop (Remote call) Ett anrop på ett objekt i en annan JVM.

3 Arkitektur

Nedan visas ett vanligt objekt i en JVM som anropar en metod på ett fjärrobjekt i en annan JVM.



3.1 RMI subsystemet

JVM:er innehåller ett RMI-subsystem som hanterar allt på låg nivå (sockets, trådar, m.m.).

Det finns inga klasser som representerar själva RMI-systemet. Vi kan inte anropa metoder på systemet.

3.2 Stubbar

Klients anrop går via en s.k. stubbe (stub) i form av en Java-klass. Stubben representerar fjärrobjektet på klientens sida (Remote proxy pattern). Stubben har bl.a. till uppgifte att montera ner/upp (marshal/unmarshal) paramtrar/returvärden så att de kan skickas/tas emot över en nätverksförbindelse. På serversidan sker motsvarande m.h.a. fjärrobjektet. OBS! Att pararmetrar kan vara hela objektgrafer!

3.3 Protokoll

All kommunikation bygger i grunden på TCP/IP. Ovanpå detta används som standard "Java Remote Method Protocol", JRMP.

3.3.1 RMI-IIOP

RMI applikationer är helt inlåsta i Java världen. Det finns flera andra arkitekturer för RPC/DO bl.a. en industristandard som heter CORBA (med därom senare). CORBA använder "Internet Inter Orb Protocoll" (IIOP)

Man kan ändra högnivåprotokoll från JRMP till IIOP vilket möjliggör kommunikation med CORBA applikationer, kallas RMI-IIOP (man anger flaggor till rmic).

3.4 Distribuerad skräpsamling

Fungerar! RMI systemet använder sig av referensräknare för att skräpsamla objekt som ingen refererar till.

3.5 Dynamisk klassladdning

(Dynamik class loading) RMI tillåter att man använder vilka serialiserbara objekt som helst som parametrar eller returvärden (t.ex en stubbe). När det serialiserade objektet skall deserialiseras (skapa ett levande objekt) måste klassdefinitionen (.class-filen) för objektet finnas. Om definitionen inte finns på klienten kan RMI systemet dynamisk, från någon plats på nätet, ladda ned klassdefinitionen till JVMen (m.h.a. RMIClassLoader). Se vidare codebase. Utifrån detta kan man tänka sig flera olika konfigurationer;

Closed Alla klasser på klient och server laddas m.h.a. CLASSPTH (-cp). Ingen dynamisk klassladdning.

Server based En klient applet laddas från serverns CODEBASE (se nedan) tillsammans med alla nödvändiga klasser.

Client dynamic Initiala klasser (Main) laddas m.h.a. CLASSPATH övriga m.h.a. java.rmi.server.RMIClassLoader från en URL (HTTP eller FTP server) specificerad av RMIServern.

Bootstrap client. All klientkod laddas från URL (HTTP eller FTP server) bara en liten bootstrap-klass finns på klienten.

3.5.1 Bootstrap client exempel

```
// En bootstrapklass som laddar ner klienten
// Se vidare codebase nedan
p = System.getProperties();
url = new URL(p.getProperty("java.rmi.server.codebase"));
clientName = "RMIClient";
clientClass = RMIClassLoader.loadClass(url, clientName);
client = (Runnable)clientClass.newInstance();
client.run();
```

3.6 Trådar

Ett fjärrobjekt kan anropas av flera klienter. Det är upp till utvecklarna att se till att fjärrobjekt är trådsäkra (antingen inga klassattribut eller synchronized).

3.7 RMI och brandväggar

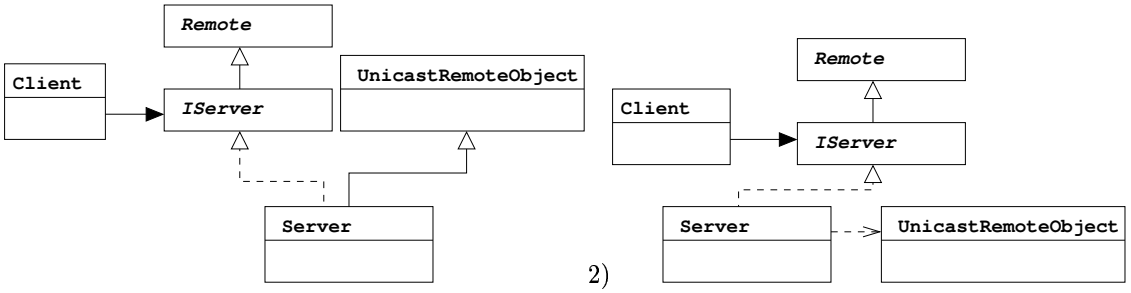
RMI försöker normalt att öppnar direkta socket-anslutningar mellan maskiner på Internet. Kan leda till problem med brandväggar. Man kan konfigurera systemet så att det ungefär fungerar som XML-RPC (d.v.s. anrop/returvärden bakas in i HTTP anrop). Ger sämre prestanda och framförallt ger det inte möjligheter till callbacks från server till klient (finns kommersiella lösningar).

4 Klasser

Paketet java.rmi. innehåller de klasser och gränssnitt som behövs för att utveckla RMI applikationer.

För att det skall vara möjligt att anropa fjärrobjekt måste klassen "exporteras" till RMI-subsystemet. Till detta används klassen java.rmi.server.UnicastRemoteObject[1]. Följande två alternativ kan användas;

1. Om man låter servern (fjärrklassen) ärva UnicastRemoteObjekt får men en del gratis, bl.a. kommer konstruktorn i UnicastRemoteObjekt att exportera servern till RMI-subsystemet
2. Server kan alternativt ha en referens till UnicastRemoteObject men då måste man explicit anropa UnicastRemoteObjekt.exportObject(server) för att exportera m.m.



[1] UnicastRemoteObject ärver/implementerar dessutom en hel del klasser/gränssnitt bl.a. Remote.

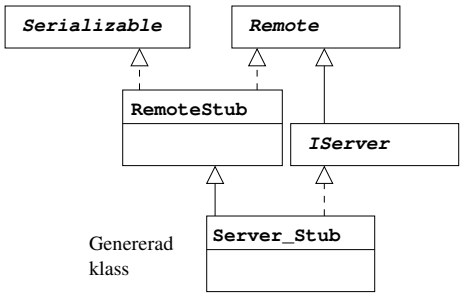
- Alla fjärrobjekt måste delas upp i en specifikation (fjärrgränssnittet) och en implementation (fjärrklassen). Gränssnittet IServer innehåller alla fjärrmetoder i klassen Server. Det är gränssnittet klienten känner till och arbetar mot, implementationen, Server, ligger i serverapplikationen.
- Alla fjärrobjekt måste implementera (det tomma) gränssnittet java.rmi.Remote. Vanligen låter man serverns gränssnitt ärva Remote enligt ovan.
- För samtliga metoder i IServer måste anges att de kan kasta java.rmi.RemoteException.

5 Kompilering av fjärrklasser

En fjärrklass kompileras som vanligt med javac. Efter detta (på .class-filen) kör man en speciell rmi-kompilator (rmic). Kompilatorn kommer att generera stubben. Om man kompilerar en klass Server enligt;

```
$ rmic Server
```

kommer man att få en stubbe som ges namnet Server_Stub. Stubben ärver och implementerar enligt;



Stubben är som synes serialiserbar (kan skickas på nätet)[2].

5.1 Vilka klasser måste vara kända för klienten

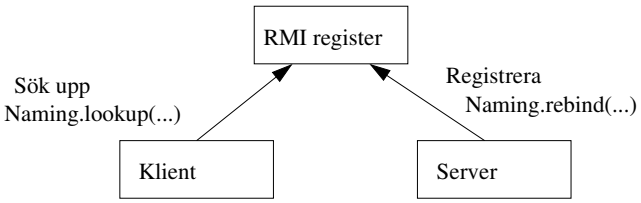
(= vad skall packas i jar-fil för distribution) Klienten måste känna till samtliga fjärrklassers gränssnitt och stubbar plus klasser som används av dessa. Klasserna måste antingen distribueras ut till klienten eller kunna laddas ner dynamisk.

6 RMI registret

En klient kan anropa ett fjärrobjekt och erhålla ett fjärrobjekt som returnvärde (en stubbe) och kan på så sätt anropa metoder på det nya fjärrobjektet. Har man bara ett fjärrobjekt kan man lätt enkelt få tag i fler. Frågan är bara hur man får tag på det första fjärrobjektet?

För att lösa detta används ett speciellt "hjälp"-program som heter rmiregistry som måste köras på samma maskin som servern. Man kan se registret som en avbildningstabell där ett program (server) kan registrera fjärrobjekt under ett namn (HashMap<String, Object>). Servern registrerar ett fjärrobjekt och klienten kan sedan sök upp objektet. Klienten måste alltså veta vart registret körs (måste ha en känd URL) och måste veta namnet på objektet (konfigurationsfiler!). Klient och server använder klassmetoder för Naming-klassen för att komma åt registret.

[2]Lägg märke till att RemoteStub impementerar Remote och IServer ärver Remote!



OBS! Det är bara "den första" (bootstrap) objektet som behöver hittas m.h.a. registret.

> "Note: Before you start the rmiregistry, you must make sure that the shell or window in which you will run the registry, either has no CLASSPATH set or has a CLASSPATH that does not include the path to any classes that you want downloaded to your client, including the stubs for your remote object implementation classes".

6.1 Fjärrobjektfabriker

Det är vanligt att servern bara har ett enda fjärrobjekt som fungerar som en fabrik för att få tag på fler fjärrobjekt (Factory pattern).

7 Codebase

När ett objekt (stubbe) skickas mellan JVMer i ett fjärranrop noteras (annotates) objektet med URLen till objektets klassdefinition.

Från SUN tutorial;

"A codebase can be defined as a source, or a place, from which to load **classes** into a virtual machine.....you can think of a codebase as the directions that you give to a VM, so it can find your [potentially remote] classes.

You can think of your CLASSPATH as a "local codebase", because it is the list of places on disk from which you load local classes. When loading classes from a local disk-based source, your CLASSPATH variable is consulted. Your CLASSPATH can be set to take either relative or absolute path names to directories and/or archives of class files. So just as CLASSPATH is a kind of "local codebase", the codebase used by applets and remote objects can be thought of as a "remote codebase"."

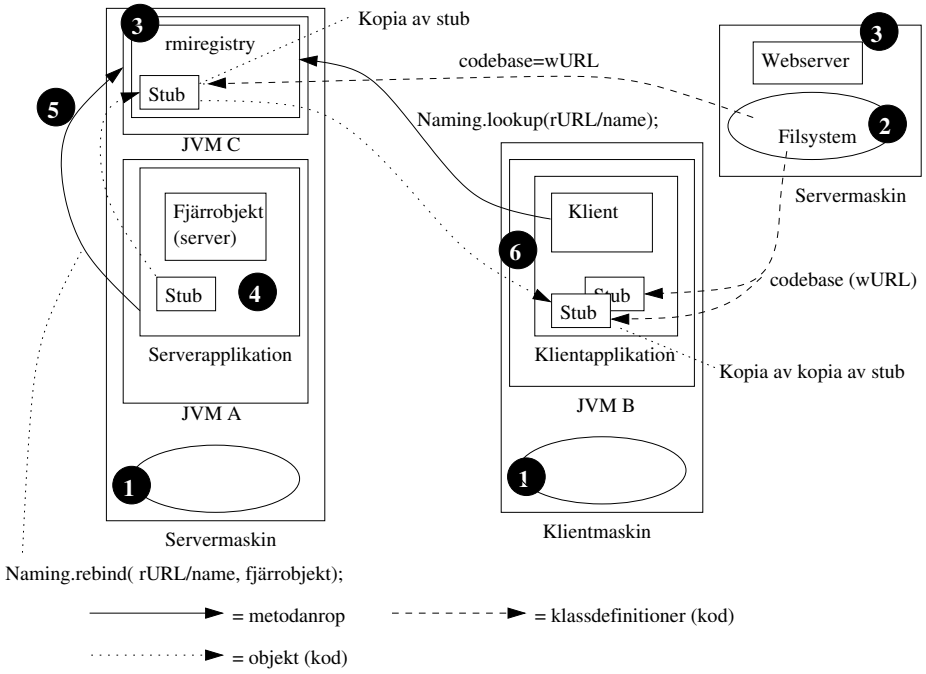
Obs! att codebas handlar om att ladda klassdefinitioner (.class-filer)

OBS! Om man anger CLASSPATH till ett bibliotek (inte en jar-fil)

MÅSTE den avslutas med /,

se kodexempel (där gick en timme ;-))

8 Start av systemet



1. Fjärrobjektet och stubbens klassdefinitioner ligger på servernmaskinens filsytem. Klienten känner bara till serverns gränssnitt (IServer.class måste finnas på klient)

2. Klienten kan inte komma åt klassfilerna på servermaskinen (om den inte körs på samma maskin eller klienten kan koppla sig till serverns filsystem, eller dylikt). Man distribuerar därför ut de klassdefinitioner (.class) som klienter kan behöva till en web- eller ftp-server eller liknande (stubbe, plus alla klasser som refereras av denna).

3. RMI-registret och web-serverna startas.

4. När serverapplikationen startas laddas klasserna för fjärrobjektet och stubben från klassfilerna på servermaskinen till JVM A. Därefter skapas en instans av fjärrobjektet och stubben (objekt).

Då serverapplikationen startas anges dessutom fjärrobjektets "codebase" d.v.s. vart klassdefinitionerna för all objekt som klienter kan använda finns. I bilden ovan anger man här webserverns URL (wURL).

---

5. Serverapplikationen anropar Naming.rebind med URL till register (rURL), ett namn och instansen av fjärrobjektet som parameter. Detta innebär att *stubben* kopieras över från serverapplikationen till rmiregistret och sparas under angivet namn. Stubben innehåller sin URL (vart den kom ifrån) och sin codebase. För att kunna instansiera stubben måste registret ha tillgång till klassdefinitionerna, dessa laddas från objektets codebase (wURL). Om registret inte kan hitta klasserna slänger det en ClassNotFoundException som fångas i serverapplikationen.

6. När klientapplikationen startas anropar den Naming.lookup med rURL och namnet på fjärrobjektet. Klienten måste känna till dessa. Om detta lyckas kommer en kopia av stubben (med typen Object) att skickas till klientapplikationen som typomvandlar den till IServer. Eftersom stubben vet var klassdefinitionen finns (wURL) kan JVM B ladda denna d.v.s klienten kan skapa en instans stubben

Klienten kan nu anropa metoder på fjärrobjektets gränssnitt . Alla anrop kommer att gå via stubben.

9 Anrop

Anropen görs "at most once" d.v.s. anropet sker en enda gång. Anropet kan antingen lyckas eller misslyckas (RemoteException). Jmf senare med CORBA. Följande gäller;

- Parametrar och returvärden kan vara primitiva typer, lokala objekt eller fjärrobjekt (objektgrafer).
- Primitiva typer skickas alltid som kopior (call by value).
- Lokala objekt skicka som kopior (call by value). Klientens ändringar av objektet påverkar inte objektet i servern. Lokala objekt som skall användas som parametrar och/eller returvärden måste implementera Serializable.
- Fjärrobjekt (alla som implementerar Remote) skickas och returneras alltid som stubbar. Klienten kan i detta fall ändra ett objekt på servern (call by reference). Det finns alltså bara ett objekt, det på servern.
- Om två parametrar i ett anrop refererar till samma objekt i anropande JVM kommer de att referera till samma objekt (kopia) också i anropad JVM.

10 Remote object activation

Det kan vara resurskrävande för servern att hålla (ett potentiellt mycket) stort antal objekt i minne. Man kan ordna så att fjärrklasser automatisk startar då anrop sker (Activation). Kan spara resurser.

10.1 Transienta och persistenta referenser

Remote object activation löser även ett annat problem. Ofta är distribuerade applikationer tänkta att vara mer eller mindre i kontinuerlig drift, men om serverna går ner så är inte klientens referens (variabel) inte giltig längre, även om servern startas om innan klienten hinner göra något anrop (får exception vid nästa anrop). Man säger att referensen är transient.

Med remote object activation kan man skapa referenser som är giltiga även om serverna skulle krascha (och därefter starta om).

11 Säkerhet

- Att ladda ner kod innebär säkerhetsrisker, applikationer som laddar ner måste ha en "security manager".
- Vanligen vill man att servern skall kunna anropa klienten d.v.s. klienten måste vara ett fjärrobjekt. Man kan tänka sig att servern har en metod;

```
// IServer.java
void registerClient( String name, IClient client );
```

som klienten anropar för att registrera sig. IClient är då klientens gränssnitt (extends Remote). Servern sparar undan referensen (stubben till klienten) och kan vid behov anropa denna.

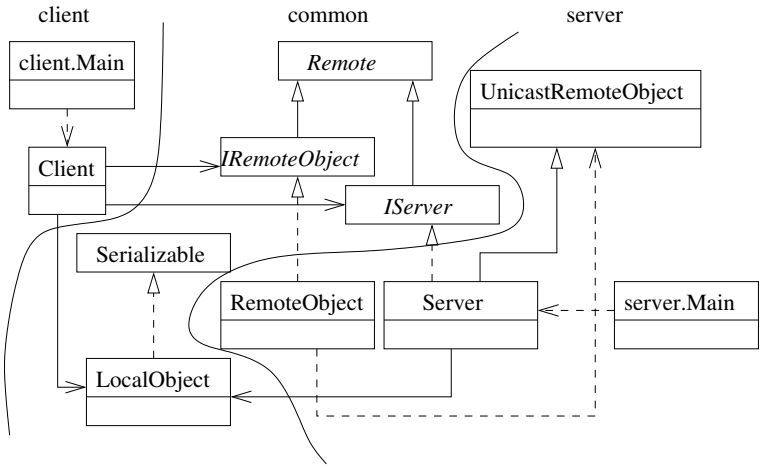
12 Kodexempel

Detta är en kort översikt över kodexemplet på kurssidans. Exemplet visar hur en klient kan anropa en sett fjärrobjekt (Serverklassen).

- Servern kan returnera ett booleskt värde, ett lokalt objekt och ett (annat) fjärrobjekt.
- Klienten och servern ändrar ett värde i det lokala objektet och i fjärrobjektet. Exemplet visar att det lokala objektet är en självständig kopia (värdet oberoende i klient och server). Fjärrobjektets värde däremot påverkas av både server och klient (det finns bara ett objekt, det på servern).

12.1 Klasser

Klientens och serverns klasser ligger i ett klient respektive serverpaket. Klasser som är gemensamma för klient och server ligger i ett common-paket (Java-klasser ligger naturligtvis i sina paket).



12.2 Serverkod

```
// Skapa en ett fjärrobjekt Server och bind den i registret
Naming.rebind("rmi://127.0.0.1:1099/Server", new Server());

// Metoden contact i serverobjektet
public boolean contact() throws RemoteException {
   return true;
}
```

12.3 Klientkod

```
// Leta rätt på serverstubben (känd URL och känt namn)
// Omvandla till IServer
server = (IServer) Naming.lookup("rmi://127.0.0.1;1099/Server");

//Anrop på servern (stubben)
boolean isAlive = server.contact();
if( isAlive ){
 :
```

12.4 Övriga intsällningar

För att det skall fungera behövs en hel del flaggor m.m. då man skall starta systemet. Se skript i kodexemplen.