Denne guide er oprindeligt
udgivet på Eksperten.dk

COMPUTERWORLD

RMI avanceret

Denne artikel beskriver nogle mere avancerede features i RMI.

Den gør det muligt at lave mere realistiske applikationer.

Den forudsætter godt kendskab til Java og til basal RMI.

Skrevet den **16. Feb 2010** af **arne_v** I kategorien **Programmering / Java** | ★★★☆☆

Historie:
V1.0 - 14/03/2004 - original
V1.1 - 16/02/2010 - smårettelser

Forudsætning

For det basal RMI se:
http://www.eksperten.dk/guide/25

Problemerne

Følgende praktiske problemer skal ofte løses for virkelige applikationer:

1) starte RMIRegistry sammen med server for at undgå at skulle starte 2 processer på server

2) kontrollere hvilke porte serveren lytter for at kunne lukke op for disse i firewall/router

3) bruge pakker og bundte klient og server i hver sin jar filer for nem distribution

4) sikre server og client mod misbrug

5) gøre det muligt for serveren at kalde klienten

Løsningerne

Problemerne løses ved:

1) lade serveren create registry

2) lade serveren bruge en socket factory med kendte porte

3) have 3 pakker common + client + server og 2 jar filer client.jar (med common + client) og server.jar (med common + server)

4) bruge security manager og policy fil

5) bruge callback mekanismen

Eksempel

Her kommer et eksempel som implementerer alle 5 løsninger.

Eksemplet er et simpelt halvfærdigt chat framework.

Der er kommentarer i koden hvor det interessante sker.

MyClient.java

```
package test.common;

import java.rmi.Remote;
import java.rmi.RemoteException;

// client interface
public interface MyClient extends Remote {
   public void notify(String msg) throws RemoteException;
}
```

MyClientImpl.java

```
package test.client;

import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;

import test.common.MyClient;

// client implementation
public class MyClientImpl extends UnicastRemoteObject implements MyClient {
   public MyClientImpl() throws RemoteException {
   }
   public void notify(String msg) {
      System.out.println(msg);
   }
}
```

MyClientMain.java

```
package test.client;

import java.net.MalformedURLException;
import java.rmi.Naming;
```

```
import java.rmi.NotBoundException;
import java.rmi.RemoteException;
import java.rmi.RMISecurityManager;

import test.common.MyClient;
import test.common.MyServer;

// client hoved program
public class MyClientMain {
   public static void main(String[] args) {
      try {
         // enable security
         System.setSecurityManager(new RMISecurityManager());
         // lookup remote server
         MyServer srv = (MyServer) Naming.lookup("rmi://" + args[0] +
":60000/MyServer");
         // create client object og registrer på server
         MyClient cli = new MyClientImpl();
         srv.register(cli);
         // test call
         srv.send("a");
         srv.send("bb");
         srv.send("ccc");
         // afregistrer
         srv.unregister(cli);
         // afslut
         System.exit(0);
      } catch (MalformedURLException e) {
         e.printStackTrace();
      } catch (RemoteException e) {
         e.printStackTrace();
      } catch (NotBoundException e) {
         e.printStackTrace();
      }
   }
}
```

MyServer.java

```
package test.common;

import java.rmi.Remote;
import java.rmi.RemoteException;

// server interface
public interface MyServer extends Remote {
   public void send(String msg) throws RemoteException;
   public void register(MyClient c) throws RemoteException;
   public void unregister(MyClient c) throws RemoteException;
}
```

MyServerImpl.java

```
package test.server;

import java.util.List;
import java.util.ArrayList;
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;

import test.common.MyServer;
import test.common.MyClient;

// server implementation
public class MyServerImpl extends UnicastRemoteObject implements MyServer {
   private List list;
   public MyServerImpl() throws RemoteException {
      list = new ArrayList();
   }
   // send message til alle client
   public void send(String msg) throws RemoteException {
      synchronized(list) {
         for(int i = 0; i < list.size(); i++) {
            MyClient c = (MyClient)list.get(i);
            c.notify(msg);
         }
      }
   }
   // registrer client
   public void register(MyClient c) {
      synchronized(list) {
        list.add(c);
        System.out.println("Number of clients: " + list.size());
      }
   }
   // afregistrer client
   public void unregister(MyClient c) {
      synchronized(list) {
        list.remove(c);
        System.out.println("Number of clients: " + list.size());
      }
   }
}
```

MyServerMain.java

```
package test.server;

import java.io.IOException;
import java.net.MalformedURLException;
import java.rmi.Naming;
```

```
import java.rmi.RemoteException;
import java.rmi.RMISecurityManager;
import java.rmi.server.RMISocketFactory;
import java.rmi.registry.LocateRegistry;

// server hoved program
public class MyServerMain {
   public static void main(String[] args) {
      try {
         // enable security
         System.setSecurityManager(new RMISecurityManager());
         // sæt socket factory
         RMISocketFactory.setSocketFactory(new FixedPortRMISocketFactory());
         // start registry
         LocateRegistry.createRegistry(60000);
         // bind server objekt
         Naming.rebind("rmi://localhost:60000/MyServer", new MyServerImpl());
      } catch (RemoteException e) {
         e.printStackTrace();
      } catch (MalformedURLException e) {
         e.printStackTrace();
      } catch (IOException e) {
         e.printStackTrace();
      }
   }
}
```

FixedPortRMISocketFactory.java

```

package test.server;

import java.io.IOException;
import java.net.Socket;
import java.net.ServerSocket;
import java.rmi.server.RMISocketFactory;

public class FixedPortRMISocketFactory extends RMISocketFactory {
   private static int socketNumber = 60000;
   // lav normal udgående socket
   public Socket createSocket(String host, int port) throws IOException {
      System.out.println("Connecting port : " + port);
      return new Socket(host, port);
   }
   // lav indgående socket
   // hvis eksplicit bedt om bestent port så
   //    brug den port
   // ellers
   //    vælg en fortløbende port udfra basis porten
   public ServerSocket createServerSocket(int port) throws IOException {
      if(port == 0) {
          socketNumber++;
```

```
            port = socketNumber;
        }
        System.out.println("Listening port : " + port);
        return new ServerSocket(port);
    }
}
```

socket.policy

```
// grant alle indgående og udgående socket (kunne godt restrictes mere med
//                                          specifik adresse og port !)
// grant mulighed for at sætte socket factory
grant {
    permission java.net.SocketPermission "*:*", "connect,accept,resolve";
    permission java.lang.RuntimePermission "setFactory";
};
```

build.bat

```
rem compile java -> class
javac -classpath . test\common\*.java
javac -classpath . test\server\*.java
javac -classpath . test\client\*.java
rem compile implementation class -> stub & skeleton class
rmic -classpath . test.server.MyServerImpl
rmic -classpath . test.client.MyClientImpl
rem create jar filer
jar cf client.jar test\common\*.class test\client\*.class
test\server\*stub.class
jar cf server.jar test\common\*.class test\server\*.class
test\client\*stub.class
```

server.bat

```
rem Angiv classpath til jar fil
rem Angiv policy fil
rem Angiv serverens eksterne navn
java -classpath server.jar -Djava.security.policy=socket.policy
-Djava.rmi.server.hostname=xxx.domain.dk test.server.MyServerMain
```

client.bat

```
rem Angiv classpath til jar fil
rem Angiv policy fil
rem Angiv serverens eksterne navn
java -classpath client.jar -Djava.security.policy=socket.policy
test.client.MyClientMain xxx.domain.dk
```

Noter

socket.policy kan strammes lidt mere op med en clientsocket.policy som kun tillader serverens adresse (xxx.domain.dk) og en serversocket.policy som kun tillader relevante port (60000-60010).

Da serveren connecter til RMIRegistry så er det vigtigt at navnet xxx.domain.dk også virker på indvendig side af firewall/router.

RMISocketFactory kan også bruges til at lade kommunikationen gå over SSL sockets.

Oprindeligt var RMI tænkt til at kunne bruges fra applets med enkeltstående class filer og derfor anbefaler SUN dokumentation normalt at man ikke putter stubs i normalt classpath men downloader dem fra en web server ved at angive -Djava.rmi.server.codebase=http://www.domain.dk/dir/. Det kan jeg ikke se den helt store pointe i ved en client applikation og brug af jar filer.
Og til applets vil jeg ofte anbefale HTTP fremfor RMI. Men hvis vi snakker applets så husk tricket.

Kommentar af simonvalter d. 15. Mar 2004 | 1

God artikel, besvarer en masse spørgsmål som jeg selv har siddet og rodet med tidligere.

Kommentar af dna d. 18. Mar 2004 | 2

Artiklen omhandler nogle kendte problemer fra Java! - Artiklen indeholder nok kode til, at man selv er igang på kort tid! Good job!