International Journal of Applied Engineering Research
ISSN 0973-4562 Volume 10, Number 11 (2015) pp. 28411-28415

http://www.ripublication.com

A Study on Remote Method Invocation In JAVA

Asha Sukumaran
Lecturer in Computer Engineering, Govt.womens Polytechnic College, Kayamkulam, India, E-mail: ashaasasi@yahoo.co.in

Abstract

Java Remote method Invocation (Java RMI) is a java application Programming Interface. It allows the programmers to create distributed Java technology based applications which allow calling methods of objects located remotely on another computer. Java is an Object Oriented Programming language which contains a large number of built in classes that support transfer of data between client and server computers. RMI allows sharing of resources between client and server by passing object types. It is a powerful feature in java.

Keywords: RMI, serialization, rmiregistry, stub, skeleton.

Introduction

A distributed network is type of computer network that is spread over different network. In this resources are divided over a number of networks. It allows sharing of application, data, and other services between different nodes on the network. Distributed application is software that executes on two or machines in a network. Client server architecture is an example of distributed application. Client is the machine which request and get some data resides on another machine with high power of processing called sever.

Distributed applications provide communication with the remote objects and an application need to load the class definitions for the objects. The Java RMI application has all these features, so it is also called distributed application.

Sockets are used to connect two computers. There are two types of sockets in Java, client socket and server socket. In Java there are many classes that support socket, defined in java.net package. Using this we can transfer data between two computers. But we cannot access a method which resides in a computer by an object which resides in another computer using socket. We can achieve this by Remote method Invocation in Java .Socket classes are not part of RMI, but RMI uses Java's socket classes to handle communication between distinct processes. Remote Method Invocation classes are defined in java.rmi package.

RMI allows object function calls between Java virtual machines. Method can even pass objects as its parameters and allows dynamic loading of new classes as required. RMI uses object serialization for marshalling and unmarshalling the passed parameters. Serialization converts an object into a transportable form over network by converting it into a stream of bytes which contain all the information about objects type and type of data. Deserialization is the reverse of serialization. In RMI application, both client and server interact with a remote interface which declares the remote objects. The client application invokes methods on another machine. This will be processed by a client side program called stub for transmission and send the request to the server. A skeleton on the server side takes this request and send to the server. The return value is sent back and reaches the client. java.rmi is the package in java which helps the remote method invocation.

Components of RMI

RMI consists of client, server, registry, stub, skeleton and an interface as its components.

(1) Client

Client is a machine which needs some task to be performed with the help of a method that is placed in the local host itself or in a remote machine. Client tries to invoke that method which is to be performed by a remote object resides on a distant machine.

(2) Server

Server is an application that creates a number of objects and waits for the client to invoke its methods. In server there is a main method that creates the instance of the object and binds that name into a RMI registry using rebind() or bind() method.

(3) RMI registry

RMI registry is a place in which it contains name of all the remote objects which is binded by the server. Since the client has no idea about where actually the remote object resides it has to first lookup in the registry for it.

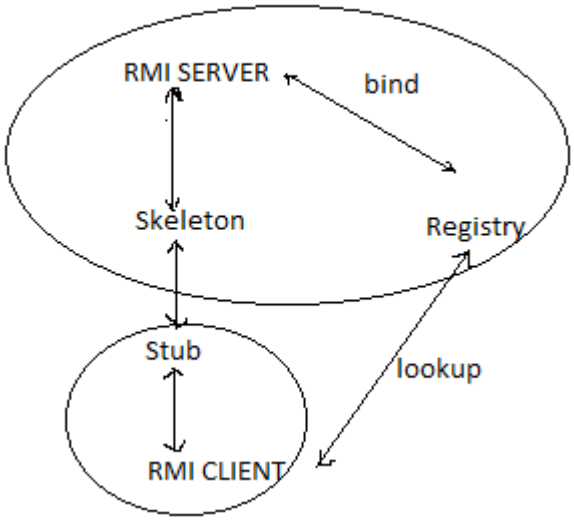
(4) Interface

Interfaces declare methods. Classes which like to use this method must implement this interface. In RMI this interface should extend the interface java. rmi. Remote so that both client and server can share it.

(5) Stub And Skeleton

Stub is a client side program which helps in marshalling the data. It helps to encode the sent parameters to a form which is transportable and send this to the server. Skeleton object helps to capture the requested data from the server and also to call the desired methods in the server. It helps in unmarshalling the parameters. Stub and Skeleton are created automatically when we use rmic. rmic is the rmi compiler

Working of RMI



Server must bind its name to a registry.
Client looks in the registry and sends it request to server through stub.
Skeleton invokes the remote method and serializes the result back to stub.

Writing RMI Service with Example

A**. Crete a remote interface to display largest of two numbers**

```
import java.rmi.*;
public interface biginterface extends Remote{
public int big(int x, int y) throws RemoteException;}
```

Here Remote keyword is used to make the interface remote .Save this file as biginterface.java

B. Implementing the interface

```
import java.rmi.*;
import java.rmi.server.*;
public class bigremote extends UnicastRemoteObject implements biginterface{
bigremote () throws RemoteException{
super();}
public int big(int x, int y){
if (x>y)
{return x;}
else { return y;}
}}
```

Unicast Remote Object makes objects available for remote machines. Save the above file as bigremote.

C. RMI Server

```
import java.rmi.*;
import java.rmi.registry.*;
public class bigserver
{
public static void main(String args[])
{
 try{ biginterface  s= new bigremote();
Naming.rebind("rmi://localhost:5000/as", s);}
catch (Exception e){}
}
}
```

Here both server and client are stored in the same machine. So localhost is used. 5000 is the port number. (Instead of writing RMI sever in separate file as above we can write the code of it in the implementing the interface file itself). Save the file as bigserver.java

D. RMI Client

```
import java.io.*;
import java.rmi.*;
public class client{
 public static void main(String args[])
{ try
{
biginterface   s=(bigremote)Naming.lookup("rmi://localhot:5000/as");
System.out.println(s.big(25,100));}catch (Exception e)
{}
}}
```

Here Naming.lookup() returns the reference of the remote object.
Save this file as client.java

E. Runnung client and Server

- In Windows First create a directory. In this store all the above java files. Then compile all the files in that directory as javac *.java
- Do rmic for creation of stub and skeleton rmic bigremote
- Start rmi registry rmiregistry 5000 or start rmiregistry
- Open a command prompt and Run sever in it java bigserver
- Run client in another command prompt java client

We will get the output in the client side as 100

Conclusions

RMI makes it simple to write remote java clients and java servers. It allows full power of object oriented programming in distributed application. It allows user system to be

safe with its security mechanism. RMI has a distributed garbage collection feature and it exists in almost all java virtual machines. But it is slow, unreliable and cannot communicate with non-java system.

References

[1] Java RMI by William Grosso: O'REILLY publications Indian reprint by SHROFF publishers and Distributors Pvt.Ltd.
[2] Java Network Programming by Elliote Rusty Harold.
[3] Java Network Programming by Hughes Et Al.
[4] http://www.Javapoint.com/RMI