USAGE & IMPLEMENTATION IN REMOTE METHOD ALLOCATION

Himanshu Sethi[1], Harshit Punn[2]
Dronacharya College of Engineering of Engineering
Maharishi Dayanand University, Gurgaon, Haryana, India

Abstract: Java Remote Method Invocation (RMI) allows the programmer to execute remote methods using the same semantics as local Functions calls. RMI is the Java version is Remote Procedure Call (RPC). RMI internal implementation is outside the scope of the client and only make exposed interface with the remote server object. The purpose of RMI is to allow programmers to rely on remote services from distant objects. The paper explains the architectural layers RMI and its mechanisms. Paper deals with the performance of all layers of RMI and how they are implemented. This paper has considered an example to explain the proper performance of RMI.

Index Terms: Marshaling, Naming, RMI Registry, Object Serialization Stub & Skeleton.

I. INTODUCTION

The Java RMI is Java's native theme for making and using remote objects over network. It allows us to distribute java object instances across network on completely different machines, which can be invoked from native machine. To act with the ways of objects on remote machines using JVM (Java Virtual Machine), RMI is employed. This process permits the exchange of data/statistics victimization multiple JVMs. It provides the placement transparency by making the ways being accessed domestically. RMI is that the Java version of Remote Procedure decision (RPC), but has the flexibility to pass quite one remote object along with the request. This object being passed has the ability to vary the service that's performed on the remote pc. This property of java is named "Moving Behavior" by Sun Microsystems. For example, once a user at a foreign laptop fills out associate degree expense account, the Java program interacting with the user could communicate, using RMI, with a Java program in another laptop that forever had the most recent policy concerning expense coverage. In reply, that program would challenge an object associate degreed associated methodology info that will modify the remote laptop program to screen the user's travel and entertainment account information during a means that was in line with the most recent policy. The user and also the company each would save time by catching mistakes early. Whenever the corporate policy modified, it'd need a change to a program in exactly one laptop. Object parameter-passing mechanism is thought as object serialization. Associate RMI request may be a request to invoke the method of a distant object. The request is of identical syntax as letter of invitation to invoke associate object technique within the same computer. In general, RMI is intended to preserve the object model and its blessings across a network.

II. REMOTE METHOD INVOCATION

The Remote technique Invocation (RMI) model represents an Evaluation distributed object application. It permits an object within a JVM, acting as a shopper, to invoke a method on an object running in a foreign JVM, actions as a server, and come the results to the shopper. Therefore, RMI implies a shopper and a server. RMI uses a stratified architecture; every of the layers may be enhanced or replaced while not poignant the remainder of the model. For example, a UDP/IP layer might replace the transport layer while not poignant the higher layers. RMI design explains the communication between two Java Virtual Machines, wherever the strategies are invoked from native machine. The RMI implementation consists of essentially three abstraction layers. the primary is that the Stub and Skeleton layer, which lies in a lower place the read of the developer. This layer intercepts technique calls created by the shopper in the interface reference variable and redirects these calls to a foreign RMI service. Remote Reference Layer comprehends a way to infer and manage references made up of shoppers to the remote service objects. In JDK 1.1, this layer provides a unicast affiliation from clients to remote service objects that area unit running and export them onto a server. The transport layer is predicated on TCP/IP connections between machines in an exceedingly network. Java RMI provides the subsequent elements:

- Remote object implementations.
- User interfaces, or stubs, to remote objects.
- A written record for remote object for locating objects over the network.
- A network protocol for communication between remote objects and their user that is Java Remote method protocol.

A. LAYERS OF RMI

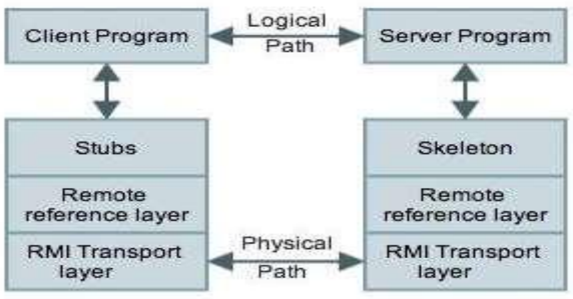

Figure 1: RMI Architecture

The Stub Layer (& Skeletons):
A stub may be a program on the client facet of the client/server relationship. The stub layer resides between the application layer and also the remainder of the RMI system and acts as an interface. Stub is the agreeable substitute protest that delivers the interface on client viewpoint to address the server perspective. It is produced by a JDK took, rmic, from the server item ordered code, and is conveyed to the client. The network-related code dwells within the stub and skeleton, so that the client and the server won't need to subsume the network and sockets in their code. A skeleton may be a remote object at the server-side. This stub consists of strategies that invoke dispatch calls to the remote implementation of objects.

The Remote Reference Layer:
The second layer of RMI design deals with the interpretation of references made of client remote objects to server remote objects. This layer deals the lower level transport interfaces. With the assistance of Remote Object Activation it activates the latent remote object for unicast communication.

The Transport Layer:
The third layer of RMI design provides the affiliation between two JVMs. The transport layer sets up the connections to remote address areas, manages them, monitors the affiliation liveliness, and listens the incoming calls. For incoming calls, the transport layer establishes a connection. It locates the target, dispatches the remote calls and passes the affiliation to the dispatcher.

B. RMI MECHANISM

- The client Program utilizes the stub for making a demand for a remote item. The server program gets this appeal from the skeleton.
- RMI invocation is initiated by career a technique on stub object, that maintains an inside reference to the remote object it represents.
- The stub forwards the strategy invocation request through Remote Reference Layer with the assistance of marshaling method. This layer forwards the request to an acceptable remote object.
- Marshaling: This method transforms the native Objects of an acceptable moveable type, in order that they will be Easily broadcasted to a remote method.
- Each Array, string or user-defined object is checked While being marshaled to conclude whether or not it Implements Java RMI. Remote interface. If it is a Remote object, then that reference is employed for Marshaling.
- If it's a Serializable object, then initial it's serialized into bytes that square measure sent to the remote object and then they're reassembled to make a copy of local object. If in case the object is neither then it throws a Java. Rmi. Marshal Exception to the client.
- The remote reference layer then receives the marshaled arguments from the stub, which then converts the client request into single network level requests.
- The remote reference layer at the server aspect receives transport-level request and transforms it into a request for skeleton to match the documented object.
- The skeleton converts the remote request into suitable methodology decision and carries out the method of un-marshaling the theistic arguments applicable for servers. The arguments sent as remote objects are converted into native stubs and those sent as serialized objects are converted into native copies of originals.
- In case a return quality is made then the article is marshaled by the skeleton and sent fresh to the client through server remote reference layer.
- The final result is transmitted again to client through a suitable transport convention.

C. STEPS TO CREATE RMI-BASED CLIENTS AND SERVER
Creation of RMI-based Server:
Make the remote interface and the servant segment class. To have these servant classes make the RMI Server. Compile the class records and create the Skeletons and IDL Record. Start up the Server.

Creation of RMI-based Client:
Make the Client class. Empower the Stub Generation and compile the Client. Verify the Server is running and at that point startup the Client process.

III. RMI REGISTRY

The server application makes an object and makes it available remotely. For making the object remotely, the server needs to register the RMI-empowered questions that are accessible to the customers. The clients will realize these Remote Services on a naming service that is procurable on publicly outlined port. RMI defines its own naming service, the RMI registry, possesses a standard port of 1099. a regular JDK tool, rmi registry, handles the registry. In the event that an item executes the Java. rmi. remote interface, then it is limited to registry setting. The interface that is, no doubt referenced is executed by every registry context.

A. METHODS FOR REGISTERING REMORE OBJECT
The method for remote articles are summoned by actualizing the Java. rmi. remote interface. Emulating are the method for enrolling the remote articles:

- bind(): It binds the determined name of the remote object. The parameter of this system ought to be in a URL form.
- unbind(): Demolishes the binding for a particular name of a remote method in the registry.
- rebind(): It again binds the pointed out name to the remote object. The current binding will be supplanted by rebinding.
- list(): It returns the names that were certain to the

registry in associate array kind. These names are within the form of a URL-formatted string.
- lookup(): A stub, a reference comes back to the remote object that is expounded with a such name.

IV. IMPLEMENTING AN RMI SYSTEM

The steps concerned in building a distributed application with RMI include:

- Interface explanation for the remote methods.
- Stub files
- A server to host the remote services
- An RMI Naming service A client program that requires the remote services

A. DEFINE AN INTERFACE FOR DECLARING REMOTE METHODES

It includes actualizing a remote interface for between the client and the server. It characterizes the remote protests that are asked for by the client. We are making a simple application to add two numbers. So we pronounce the add() method in interface Addition.java.

```
import java.rmi.Remote;
import java.rmi.RemoteException;
public interface Addition extends Remote{
public long add(long x, long y) throws RemoteException;
}
```

The interface is augmented with the goal that it can be called remotely in the middle of the client and server. The Remote Exception happens when there is some failure in RMI process.

B. DEFINE THE CLASS AND IMPLEMENT REMOTE METHODES

```
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;
public class AdditionImpl extends Unicast Remote Object implements Addition
{protected AdditionImpl() throws Remote Exception
{super();}
public long add(long x,long y)throws Remote Exception
{return x+y; }}
```

We characterize a class Additionimpl.java and actualize the interface and characterize the body of the remote method. The Unicast remote object is a base class for user-defined remote objects having the general structure as, Public class Unicastremoteobject develops Remote server.

C. DEFINING THE SERVER PROGRAM

The primary Parameter is a URL to a registry that contains the name of the application, which here is "Addition service". The second parameter is an item name that is gotten to remotely in the middle of client and server. The rebind is a strategy of Naming class which is actualized in java.rmi.* package. 1099 is the default RMI port and 127.0.0.1 is a localhost-ip address

```
import java.rmi.Naming; public class AdditionServer
{
AdditionServer()
{
Try
{
Addition c= new AdditionImpl();
Naming.rebind("rmi://127.0.0.1:1099/AdditionService",c);
}
catch(Exception e){ e.printStackTrace();
}
}
public static void main(String[] args)
{
new AdditionServer();
}
}
```

To get to the remote object on client side, which is as of now binding at a server side by, reference URL, we utilize the lookup method, which has the same reference URL. This lookup is a technique for Naming class which is accessible in Java.rmi package. The name of the URL must be same as indicated on server side class.

```
import java.rmi.Naming; public class AdditionClient
{
public static void main(String[] args)
{
try
{
Addition c= (Addition)Naming.lookup
("//127.0.0.1:1099/AdditionService");
System.out.println("Addition of two digits is: "+c.add(10,15));
}
catch(Exception e)
{
System.out.println(e);
}}}
```

Compile all the source java files.

- javac Addition.java
- javac AdditionImpl.java
- javac AdditionClient.java
- javac AdditionServer.java

- The command rmic empowers the stub generation. Syntax: rmic AdditionImpl This command produces AdditionImpl_Stub.class file.

- Start the RMI remote Registry: The references of the remote objects are registered to RMI Registry. Syntax: rmiregistry & (which opens rmiregistry.exe)

- Run the server program and afterward run the client program on another terminal window.

V. CONCLUSION

Actualizes remote connection between them. The servers' job is to acknowledge demand from a client, perform services, and after that send the results to the client. The utilization of Registry and Naming classes is to bootstrap our distributed applications.

RMI implementation includes four product programs to be specific:

- Client program: does the request
- Server program: implements the request
- Stub Interface: Executed by client to know the remote functions
- Skeleton Interface: Executed by server.

Advantages of RMI:

- It's simple and clean to actualize and produces more robust and flexible applications.
- Distributed systems are created while decoupling the client and server objects.
- No client installment is needed with the exception of java environment.
- While changing database, only server objects are recompiled but not the interface and client program.

REFRENCES

[1] Java Remote Method Invocation: http://en.wikipedia.org/wiki/Java_remote_method_invocati on
[2] Java RMI Tutorial: http://www.eg.bucknell.edu/~cs379/DistributedSystems/rm i_tut.html#serial
[3] The JavaTM Tutorials-RMI: http://docs.oracle.com/javase/tutorial/rmi/
[4] Ninghui Li, John C. Mitchell and Derrick Tong, "Securing Java RMI-based Distribute Applications"
[5] Jason Maassen, Rob van Nieuwpoort, Ronald Veldema, Henri E. Bal and Aske Plaat, "An Efficient Implementation of Java's Remote Method Invocation" (1999)
[6] Remote Method Invocation: http://www.javacamp.org/moreclasses/rmi/rmi4.html
[7] Remote Method Invocation: http://www.javatpoint.com/RMI
[8] Naming Methods: http://www.cis.upenn.edu/~bcpierce/courses/629/jdkdocs/a pi/java.rmi.Naming.html