© 2014 IJIRT | Volume 1 Issue 6 | ISSN : 2349-6002

The RMI Mechanism by JAVA

Tarun Yadav, Rahul Yadav, Vishal Sharma
Computer Science and Engineering, Dronacharya College of Engineering, Haryana, India

***Abstract-* The team wants to describe about the RMI that is Remote Method Invocation feature that Java programming language offers to its programmers. Java is one of the mostly used programming languages in today's world. It has almost all the features that are required to make a simple program like say “hello” to making GUI components for the Operating Systems. Hence one of the feature in it is RMI, which is basically used over a Network. Basically, the team wants to describe about this RMI feature used in Java by a programmer who wants to access an object available at some other Computer. We want to elaborate on this Feature of Java and its effectiveness in the transferring of data across a Network. It basically depends on the RMI Layer model . The client appears to talk directly to the server. In reality, the client program talks only to the a stub. The stub passes that conversation along to the remote reference layer. The remote reference layer talks to the transport layer. The transport layer on the client passes the data across the Internet to the transport layer on the server. The transport layer on the server then communicates with the server’s remote reference layer. The remote reference layer talks to the skeleton. The skeleton communicates with the server. We access the RMI mechanism whenever we have to access objects remotely, this happens when the classes to be used are not present In the built-in environment. The RMI is an improvement in a programming language because Java happens to be the first of them all Providing the mechanism for solving the accessing of a remote object.**

***Index Terms-* Stub, Skeleton, Remote Object.**

I. INTRODUCTION

Java is one of the mostly used programming languages in today's world. It has almost all the features that are required to make a simple program like say “hello” to making GUI components for the Operating Systems. Hence one of the feature in it is RMI, which is basically used over a Network.

Basically, the team wants to describe about this RMI feature used in Java by a programmer who wants to access an object available at some other Computer. We want to elaborate on this Feature of Java and its effectiveness in the transferring of data across a Network because some memory would be taken by tag array that is cache directory.

II.RMI- HISTORY

Basically all the computer application used in todays world comprises of three basic fields.
* Client
* Server
* Database
Initially a Monolithic approach was subjected to the use of all the three fields of the Computer Application. In this field, all these fields were into a single place.
For example- MS Word.
Then came Two-Tier Architecture of the Computer Application. Where a client would have different place and Server/ Database would have a same place.

Example- In a Database, a client needs to update his information.
Finally, came a true to its purpose architecture that is a three-tier Architecture for the computer applications.
In this architecture Client, Server and Database, all had different positions. Where Client would always contact the respective Server to seek information through an Object. And the Remote Object whose Interface has already been declared or initialized in the Server can be accessed.

2.1Introduction to RMI

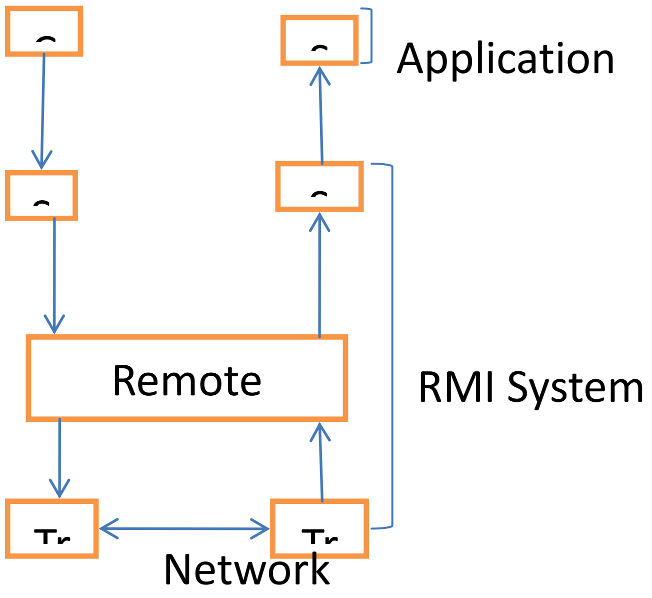
RMI basically stands for Remote Method Invocation, which means it is a mechanism through which a program running on a computer can access another method from some other computer easily.
It is a personification of the Three-tier Architecture of the Computer Application. It is effectively used between clients and Servers to exchange information according to their needs.

III. RMI LAYER MODEL

The client appears to talk directly to the server
In reality, the client program talks only to the a stub.
*The stub passes that conversation along to the remote reference layer

*The remote reference layer talks to the transport layer
*The transport layer on the client passes the data across the
*Internet to the transport layer on the server
*The transport layer on the server then communicates with the server's remote reference layer
*The remote reference layer talks to the skeleton
*The skeleton communicates with the server
RMI is an implementation of the of the *Distributed Object* programming model—similar to CORBA, but simpler and specialized to the Java language.

3.1.RMI- The Architecture



It's architecture basically has 2 layers:
* Application : which comprises of the client and the server.
*RMI System: Which comprises of the system that performs the functioning of the RMI.

Application layer
Client: Client is the one who sends the request to the server.

Server: Server is the end point that receives the request and outputs the results.

RMI System Layer
Stub: The request sent by the Client is serialized for further processing and is sent to the sending end of the RRL layer.

RRL: Remote Reference Layer, accepts the request from the Stub and marshal it so that it can go through the Transport layer without getting compromised.

Transport end is the point where the connection is established between the client and server.

RRL(receiving): It Receives the message and it unmarshal it.

Skeleton: it receives the unmarshalled request and then it de-serializes it so that the server can understand it.

IV. RMI WORKING

Now comes the difficult part where we would have to actually develop an RMI.

There are a few steps that are involved in the developing of the RMI.

Steps are:
* Create a Remote Interface
* Create a Remote Class that not only implements the Remote interface but also extends the Remote class, so that we can establish the Remote Object.
*Create a Remote Server
*Create a Remote Client

Step By Step working of the RMI.

Step-1. Creating a Remote Interface

```
//Must extend java.rmi.Remote Interface
//The Remote Interface must be declared public
//Each method must throw the RemoteException

//Hello Interface

import java.rmi.*;

public interface Hello extends  Remote
{
public String sayHello() throws RemoteException
}
```

Step-2 Define the Remote Interface Implementation

```
import java.rmi.*;
import java.rmi.server.*;
public class HelloImpl extends UnicastRemoteObject
implements Hello  //implementing remote server
{
public HelloImpl() throws RemoteException
{
super();  //Defining the constructor for the remote Object
}
public String sayHello() throws RemoteException
{
return "Hello! Neha malik";
}  //Providing Implementation for Remote methods
}
```

Step-3 Creating RMI Server class

```
import java.rmi.*;
import java.rmi.server.*;
public class HelloServer
{
public static void main(String args[])
{
try
{
System.setSecurityManager(new RMISecurityManager());
//  creating and installing security manager
Hello h=new HelloImpl();
 //creating instance of remote object
Naming.rebind("server",h);
//Registering the Remote object using rebind() method
System.out.println("object is registered.");
System.out.println("Now server is waiting for client
request...");
}
catch(Exception e)
{
System.out.println("Error: "+e);
}}}
```

Step-4 Create Remote Client class

```
import java.rmi.*;
public class HelloClient
{
public static void main(String args[])
{
try
{
Hello h = (Hello)Naming.lookup("rmi://localhost/server");
// Obtaining reference of remote object from RMI registry
//using lookup() method
System.out.println("Client: Hello!");
System.out.println("Server: "+h.sayHello());
// invoke remote methods
}
catch(Exception e)
{
System.out.println("Error: "+e);
}}}
```

Step-5 Executing of the RMI Program

RMI is an implementation of the of the *Distributed Object* programming model—similar to CORBA, but simpler and specialized to the Java language.

4.1.RMI Layer model

The client appears to talk directly to the server
In reality, the client program talks only to the a stub.
The stub passes that conversation along to the remote reference layer
The remote reference layer talks to the transport layer
The transport layer on the client passes the data across the Internet to the transport layer on the server
The transport layer on the server then communicates with the server's remote reference layer
The remote reference layer talks to the skeleton
The skeleton communicates with the server.

```
import java.rmi.*;
import java.rmi.server.*;
public class HelloImpl extends UnicastRemoteObject
implements Hello  //implementing remote server
{
public HelloImpl() throws RemoteException
{
super();  //Defining the constructor for the remote Object
}
public String sayHello() throws RemoteException
{
return "Hello! Neha malik";
}  //Providing Implementation for Remote methods
}
import java.rmi.*;
import java.rmi.server.*;
public class HelloServer
{
public static void main(String args[])
{
try
{
System.setSecurityManager(new RMISecurityManager());
//  creating and installing security manager
Hello h=new HelloImpl(); //creating instance of remote
object
Naming.rebind("server",h); //Registering the Remote object
using rebind() method
System.out.println("object is registered.");
System.out.println("Now server is waiting for client
request...");
}
catch(Exception e)
{
System.out.println("Error: "+e);
}}}
```

Step-1:Compile the four source files
Step 2: Generate the Stub & Skelton using:

```
rmic HelloImpl
```

// As the Stub and Skeleton are used for the Serializing and de-serializing of the request submitted by the Client.

Step 3: Start the RMI registry

```
start rmiregistry
```

// This step registers the Object of the RMI interface Implimentation class to the Server, so that it gets legitimate to access it through the client.

Step 4: start the server

```
java HelloServer
```

//This step executes the Server class and runs all the objects declared in it.

Step 5: Start the client

```
java HelloClient
```

//Finally the client access the objects that it needs to.

There are few points that needs to be understood whenever we are performing the RMI mechanisms.

V. SECURITY MANAGER

A *security manager* is an object that defines a security policy for an application. This policy specifies actions that are unsafe or sensitive. Any actions not allowed by the security policy cause a SecurityException to be thrown. An application can also query its security manager to discover which actions are allowed.

Typically, a web applet runs with a security manager provided by the browser or Java Web Start plugin. Other kinds of applications normally run without a security manager, unless the application itself defines one. If no security manager is present, the application has no security policy and acts without restrictions.

5.2.INTERACTING WITH THE SECURITY MANAGER

The security manager is an object of type SecurityManager; to obtain a reference to this object, invoke System.getSecurityManager.

SecurityManager appsm = System.getSecurityManager();

If there is no security manager, this method returns null.

Once an application has a reference to the security manager object, it can request permission to do specific things. Many classes in the standard libraries do this. For example, System.exit, which terminates the Java virtual machine with an exit status, invokes SecurityManager.checkExit to ensure that the current thread has permission to shut down the application.

The SecurityManager class defines many other methods used to verify other kinds of operations. For example, SecurityManager.checkAccessverifies thread accesses, and SecurityManager.checkPropertyAccess verifies access to the specified property. Each operation or group of operations has its own check*XXX*() method.

In addition, the set of check*XXX*() methods represents the set of operations that are already subject to the protection of the security manager. Typically, an application does not have to directly invoke any check*XXX*() methods.

5.3.RECOGNIZING A SECURITY VIOLATION

Many actions that are routine without a security manager can throw a SecurityException when run with a security manager. This is true even when invoking a method that isn't documented as throwing SecurityException. For example, consider the following code used to read a file:

```
reader = new FileReader("xanadu.txt");
```

In the absence of a security manager, this statement executes without error, provided xanadu.txt exists and is readable

5.4.PERMISSIONS

A permission represents access to a system resource. In order for a resource access to be allowed for an applet (or an application running with a security manager), the corresponding permission must be explicitly granted to the code attempting the access.

Java uses Permission abstract class for representing access to a system resource.

public abstract class **Permission** extends Object implements Guard, Serializable

A permission typically has a name (often referred to as a "target name") and, in some cases, a comma-separated list of one or more actions. For example, the following code creates a FilePermission object representing read access to the file named abc in the /tmp directory:

perm = new java.io.FilePermission("/tmp/abc", "read");

In this, the target name is "/tmp/abc" and the action string is "read".

The policy for a Java application environment is represented by a Policy object. In the Policy reference implementation, the policy can be specified within one or more policy configuration files. The policy file(s) specify what permissions are allowed for code from specified code sources. A sample policy file entry granting code from the /home/sysadmin directory read access to the file /tmp/abc is

```
grant codeBase "file:/home/sysadmin/" {
   permission java.io.FilePermission "/tmp/abc", "read";
};
```

Permission objects are similar to String objects in that they are immutable once they have been created. Subclasses should not provide methods that can change the state of permission once it has been created.

VI. CONCLUSION

After reviewing about RMI mechanism, we concluded that RMI is although a difficult task to implement but it gets easier through the JAVA programming language.

ACKNOWLEDGEMENT

The team acknowledges that the review given about the above topic is purely out of our own views and that we preferred only the References when needed.

REFERENCES

[1]. Power Point Presentation by Mrs. Neha Malik
[2]. Professional Java programming by Brett Spell, WROX Publication.
[3]. Advance Java by Gajendra Gupta, Firewall Media.
[4]. Advance java 2 platform, how to program, 2nd edition, Harvey M. Deital, Prentice Hall.