**RMI (Remote Method Invocation)**

The **RMI** (Remote Method Invocation) is an API that provides a mechanism to create distributed application in java. The RMI allows an object to invoke methods on an object running in another JVM.

**RMI stands for Remote Method Invocation.** **It is a mechanism that allows an object residing in one system (JVM) to access/invoke an object running on another JVM**. RMI is used to build distributed applications; it provides remote communication between Java programs. It is provided in the package java.

## Architecture of an RMI Application

## C:\Users\ADMIN\Desktop\rmi_architecture.jpg

In an RMI application, we write two programs, a **server program** (resides on the server) and a **client program** (resides on the client).

* Inside the server program, a remote object is created and reference of that object is made available for the client (using the registry).
* The client program requests the remote objects on the server and tries to invoke its methods.

The following diagram shows the architecture of an RMI application.

Let us now discuss the components of this architecture.

* **Transport Layer** − This layer connects the client and the server. It manages the existing connection and also sets up new connections.
* **Stub** − A stub is a representation (proxy) of the remote object at client. It resides in the client system; it acts as a gateway for the client program.
* **Skeleton** − This is the object which resides on the server side. **stub** communicates with this skeleton to pass request to the remote object.
* **RRL(Remote Reference Layer)** − It is the layer which manages the references made by the client to the remote object.

The RMI provides remote communication between the applications using two objects *stub* and *skeleton*.

**Understanding stub and skeleton**

RMI uses stub and skeleton object for communication with the remote object.

A **remote object** is an object whose method can be invoked from another JVM.

**stub**

The stub is an object, acts as a gateway for the client side. All the outgoing requests are routed through it. It resides at the client side and represents the remote object. When the caller invokes method on the stub object, it does the following tasks:

1. It initiates a connection with remote Virtual Machine (JVM),
2. It writes and transmits the parameters to the remote Virtual Machine (JVM),
3. It waits for the result
4. It reads the return value or exception, and
5. It finally, returns the value to the caller.

**skeleton**

The skeleton is an object, acts as a gateway for the server side object. All the incoming requests are routed through it. When the skeleton receives the incoming request, it does the following tasks:

1. It reads the parameter for the remote method
2. It invokes the method on the actual remote object, and
3. It writes and transmits the result to the caller.

**Understanding requirements for the distributed applications**

If any application performs these tasks, it can be distributed application.

.

1. The application need to locate the remote method
2. It need to provide the communication with the remote objects, and
3. The application need to load the class definitions for the objects.

The RMI application have all these features, so it is called the distributed application.

**Java RMI Example**

The is given the 6 steps to write the RMI program.

1. Create the remote interface
2. Provide the implementation of the remote interface
3. Compile the implementation class and create the stub and skeleton objects using the rmic tool
4. Start the registry service by rmiregistry tool
5. Create and start the remote application
6. Create and start the client application

//Program Example

//Creating the remote Interface

import java.rmi.\*;

public interface Calc extends Remote

{

int multiply(int a,int b) throws RemoteException;

}

//Implement the remote Interface

import java.rmi.\*;

import java.rmi.server.\*;

public class CalcRemote extends UnicastRemoteObject implements Calc

{

public CalcRemote() throws RemoteException{

super();

}

public int multiply(int a,int b)

{

return(a\*b);

}

}

//Creating Client

import java.rmi.\*;

import java.rmi.registry.\*;

public class Client{

public static void main(String[] args)

{

try{

Registry rgsty=LocateRegistry.getRegistry(9000);

Calc obj=(Calc)rgsty.lookup("multiply");

int value=obj.multiply(6,5);

System.out.println("The product is: "+value);

}

catch(Exception e)

{

System.out.println("Failed to connect Server!!");

}

}

}

//Creating Server

import java.rmi.\*;

import java.rmi.registry.\*;

public class Server{

public static void main(String[] args)

{

try{

CalcRemote obj=new CalcRemote();

Registry rgsty=LocateRegistry.createRegistry(9000);

rgsty.rebind("multiply",obj);

System.out.println("Server Ready");

}

catch(Exception e)

{

System.out.println("Server Failed");

}

}

}