Name-Nirmal Avhad

Roll No:- 71

Subject :- CL3 ( CI)

**Practical 2**

Title:- Distributed String Concatenation Application using RMI

Objective:

1. Develop a client-server application using Remote Method Invocation (RMI) for remote computation.

2. Enable the client to submit two strings to the server for concatenation.

3. Ensure that the server concatenates the given strings and returns the result to the client program.

• Jupyter Notebook, any Java IDE

• A machine with at least 8GB of RAM is recommended for model training.

• A multi-core CPU is suitable, and for faster training, a GPU (Graphics Processing Unit) is

highly recommended.

Prerequisities -

• Basic understanding of Java programming

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.

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. Let's understand the stub and

skeleton objects:

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 (marshals) the parameters to the remote Virtual Machine (JVM),

3. It waits for the result

4. It reads (unmarshals) 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

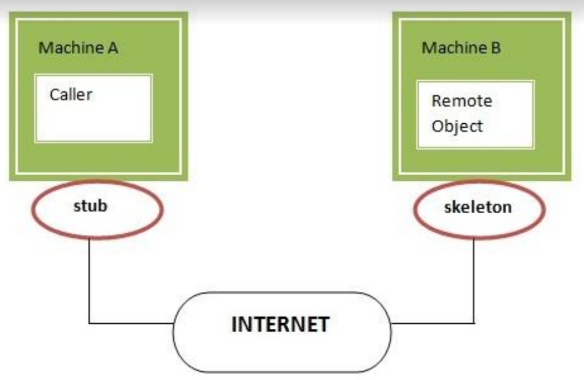
through it. When the skeleton receives the incoming request, it does the following tasks:

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 (marshals) the result to the caller.In the Java 2 SDK, an stub protocol was

introduced that eliminates the need for



skeletons.

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

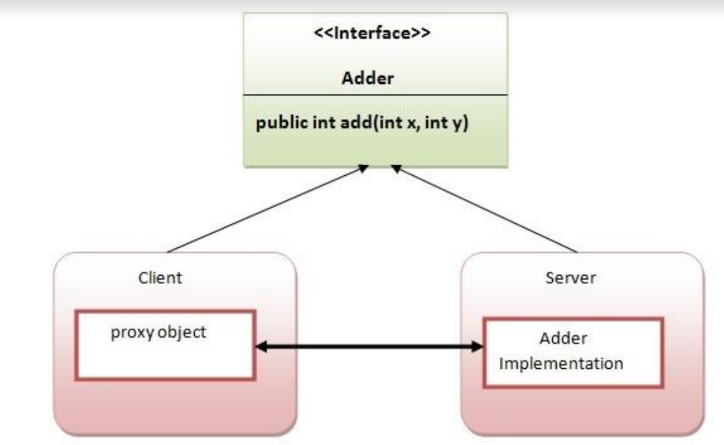
RMI Example

In this example, we have followed all the 6 steps to create and run the rmi application. The client application

need only two files, remote interface and client application. In the rmi application, both client and server

interacts with the remote interface. The client application invokes methods on the proxy object, RMI sends the

request to the remote JVM. The return value is sent back to the proxy object and then to the client application.



Conclusion: Thus This application demonstrates the use of RMI to create a distributed application for string

concatenation, where the server receives two strings from the client, concatenates them, and returns the result

to the client.

Code:-

1. interface AddRem.java

import java.rmi.\*;

public interface AddRem extends Remote{

public int addNum(int a,int b) throws RemoteException;

}

2. class AddRemImpl.java

import java.rmi.\*;

import java.rmi.server.UnicastRemoteObject;

public class AddRemImpl extends UnicastRemoteObject implements AddRem {

public AddRemImpl() throws RemoteException{}

public int addNum(int a,int b)

{

return(a+b);

}

}

3. Client.java

import java.rmi.\*;

import java.net.\*;

import java.io.\*;

import java.util.\*;

public class Client {

public static void main(String[] args) {

String host="localhost";

Scanner sc=new Scanner(System.in);

System.out.println("Enter 1st number: ");

int a=sc.nextInt();

System.out.println("Enter 2st number: ");

int b=sc.nextInt();

try{

AddRem remoobj=(AddRem)Naming.lookup("rmi://"+host+"/AddRem");

System.out.print("Nirmal ID:");

System.out.println(remoobj.addNum(a,b));

}

catch (RemoteException re)

{

re.printStackTrace();

}

catch (NotBoundException nbe)

{

nbe.printStackTrace();

}

catch (MalformedURLException mfe)

{

mfe.printStackTrace();

}

}}

4. Server.java

import java.rmi.\*;

import java.net.\*;

public class Server {

public static void main(String[] args) {

try{

AddRemImpl locobj=new AddRemImpl();

Naming.rebind("rmi:///AddRem",locobj);

}catch (RemoteException e)

{

e.printStackTrace();

}

catch (MalformedURLException mfe)

{

mfe.printStackTrace();

}}}

Output:

