TUSHAR DESHMUKH
PE27
1032201698

TITLE: Design a distributed application using RMI.

AIM: To implement a basic calculator (operations : +, - , /, *)using RMI.

OBJECTIVE:

To implement RMI for implementing basic mathematical operations.

To have a client and server execute the RMI.

THEORY:

RMI (Remote Method Invocation):

RMI (Remote Method Invocation) is a Java technology that enables communication between Java objects running on different Java Virtual Machines (JVMs), typically on different hosts across a network. It allows Java applications to invoke methods on remote objects as if they were local, abstracting away the complexities of network communication.

Key components of RMI include:

Remote Interface: Defines the methods that can be invoked remotely by clients. Remote interfaces extend the java.rmi.Remote interface and declare the remote methods that clients can call.

Remote Object: An object that implements a remote interface and provides the actual implementation of the remote methods. Remote objects are instantiated and registered with the RMI registry on the server-side, making them accessible to remote clients.

Stub and Skeleton: Stub and skeleton are automatically generated by the RMI runtime to facilitate communication between clients and server objects. The stub acts as a proxy for the remote object on the client-side, while the skeleton serves as a dispatcher for incoming remote method invocations on the server-side.

RMI Registry: A simple naming service provided by RMI for registering and looking up remote objects by their names. Clients use the registry to locate remote objects and obtain references to them.

TUSHAR DESHMUKH
PE27
1032201698

Types of Remote Classes for RMI:

There are two different types of remote classes in RMI:

UnicastRemoteObject: This class provides a simple way to create and export a remote object that can be accessed by clients using RMI. Objects of classes that extend UnicastRemoteObject can be accessed remotely by clients. This class handles the details of exporting the remote object, including generating a stub and registering it with the RMI registry.

RemoteObject: This is the abstract superclass of UnicastRemoteObject and provides the basic functionality for implementing remote objects. Unlike UnicastRemoteObject, RemoteObject does not automatically export the remote object. Instead, developers must manually export the object using the exportObject() method provided by the RemoteObject class. RemoteObject is typically used when developers need more control over the remote object's lifecycle and export process.

In summary, RMI allows Java objects to communicate across network boundaries, enabling distributed computing in Java applications. UnicastRemoteObject and RemoteObject are two types of remote classes used to implement remote objects in RMI, each offering different levels of control and convenience for developers.

PSEUDO CODE/STEPS OF ALGORITHM:

PLATFORM: Linux

PROGRAMMING LANGUAGE: C,compiler :gcc/cc

PLATFORM: Linux

LANGUAGE: C

INPUT: The values for mathematical operations

OUTPUT: The output based on +, - ,/ ,* operations

CONCLUSION: Thus, RMI has been studied and implemented on Linux platform.

TUSHAR DESHMUKH
PE27
1032201698

FAQs :

1. Differentiate between RPC and RMI

Answer:

RPC (Remote Procedure Call) and RMI (Remote Method Invocation) are both mechanisms for enabling communication between distributed systems, but they differ in their implementation and usage:

Communication Protocol:

RPC typically relies on a low-level communication protocol, such as TCP/IP, to transmit messages between client and server processes. It involves marshalling parameters, sending requests over a network, executing the procedure on the server, and unmarshalling the results.

RMI, on the other hand, is a higher-level abstraction built on top of Java's object-oriented features. It allows Java objects to invoke methods on remote objects running in different JVMs, using Java serialization for parameter passing and return values.

Language Dependency:

RPC is language-independent and can be implemented in various programming languages.

RMI is specific to Java and works only with Java objects running in JVMs.

Interface Definition:

In RPC, the interface between client and server processes is defined explicitly, often using an Interface Definition Language (IDL).

In RMI, the interface is defined using Java interfaces, making it more natural for Java developers.

Error Handling:

RPC typically relies on low-level error handling mechanisms, such as error codes or exceptions propagated across the network.

RMI leverages Java's exception handling mechanism, allowing developers to use try-catch blocks for error handling in a more object-oriented manner.

TUSHAR DESHMUKH
PE27
1032201698

2. What does rmic do?

Answer

rmic (RMI Compiler) is a tool provided by Java Development Kit (JDK) for generating stub and skeleton classes required for RMI communication. When developing RMI applications, developers define remote interfaces that extend java.rmi.Remote. The rmic tool processes these remote interfaces and generates stub (client-side proxy) and skeleton (server-side dispatcher) classes. These generated classes handle the communication between client and server JVMs, allowing remote method invocations to be performed seamlessly.

3. What is the importance of RMI registry?

Answer

The RMI Registry serves as a simple naming service in the RMI architecture, allowing clients to locate and obtain references to remote objects. Its importance lies in the following aspects:

Object Lookup: Clients can query the RMI Registry to obtain references to remote objects by their registered names.

Object Binding: Remote objects are bound to names in the RMI Registry during their registration process, allowing them to be uniquely identified and accessed by clients.

Centralized Registry: The RMI Registry provides a centralized location for registering and looking up remote objects, facilitating communication between distributed components of an RMI-based application.

Service Discovery: Developers can use the RMI Registry for service discovery, allowing clients to dynamically locate and interact with available remote services at runtime.

TUSHAR DESHMUKH
PE27
1032201698

CODE:

CalculatorServer.java

```java
//Server Program
import java.rmi.Naming;

public class CalculatorServer {
public CalculatorServer()
{
try
{
CalculatorRI c = new CalculatorImpl();
Naming.rebind("rmi://localhost:1099/CalculatorService", c);
}
catch (Exception e)
{
System.out.println("Trouble: " + e);
}
}
public static void main(String args[])
{
new CalculatorServer();
}
}
```

CalculatorRI.java

```java
//Remote Interface : declares the
// remote services
public interface CalculatorRI
extends java.rmi.Remote
{
public long add(long a, long b)
throwsjava.rmi.RemoteException;
public long sub(long a, long b)
throwsjava.rmi.RemoteException;
public long mul(long a, long b)
throwsjava.rmi.RemoteException;
public long div(long a, long b)
throwsjava.rmi.RemoteException;
public long square(long a)
throwsjava.rmi.RemoteException;
public long modulo(long a,long b)

throws java.rmi.RemoteException;
```

```
}
```

CalculatorImpl.java

```
//Defines the services in the
//Remote Interface
public class CalculatorImpl
extends
java.rmi.server.UnicastRemoteObject
implements CalculatorRI
{
// Implementations must have an
//explicit constructor
// in order to declare the
//RemoteException exception
public CalculatorImpl()
throws java.rmi.RemoteException
{
super();
}
public long add(long a, long b)
throwsjava.rmi.RemoteException
{
return a + b;
}
public long sub(long a, long b)
throwsjava.rmi.RemoteException
{
return a - b;
}
public long mul(long a, long b)
throwsjava.rmi.RemoteException
{
return a * b;
}

public long div(long a, long b)
throwsjava.rmi.RemoteException
{
return a / b;
}
public long square(long a)
throws java.rmi.RemoteException
{
return a*a;
```

TUSHAR DESHMUKH
PE27
1032201698

```
}
public long modulo(long a,long b)
throwsjava.rmi.RemoteException
{
return a % b;
}
}
```

CalculatorClient.java

```
//Client Program which
//calls the remote services
//Client Program which
//calls the remote services
import java.rmi.Naming;
import java.rmi.RemoteException;
import java.net.MalformedURLException;
import java.rmi.NotBoundException;
public class CalculatorClient
{
public static void main(String[] args)
{
try
{
int a,b;
a=Integer.parseInt(args[0]);
b=Integer.parseInt(args[1]);
CalculatorRI c = (CalculatorRI)

Naming.lookup(
"rmi://localhost/CalculatorService");
System.out.println("a+b : "+ c.add(a, b) );
System.out.println("a-b : "+ c.sub(a, b) );
System.out.println("a*b : "+ c.mul(a, b) );
System.out.println("a/b : "+ c.div(a, b) );
System.out.println("a*a : "+ c.square(a) );
System.out.println("a%b : "+ c.modulo(a,b));
}
catch(ArrayIndexOutOfBoundsException e)
{
System.out.println("Sorry! insufficient arguments");
}
catch (MalformedURLException murle)
{
System.out.println();
```

TUSHAR DESHMUKH
PE27
1032201698

```
System.out.println(
"MalformedURLException");
System.out.println(murle);
}
catch (RemoteException re)
{
System.out.println();
System.out.println(
"RemoteException");
System.out.println(re);
}
catch (NotBoundException nbe)
{
System.out.println();
System.out.println(
"NotBoundException");
System.out.println(nbe);
}
catch (java.lang.ArithmeticException ae)
{
System.out.println();
System.out.println(
"java.lang.ArithmeticException");
System.out.println(ae);

}
}
}
```

TUSHAR DESHMUKH
PE27
1032201698