UNIT –V

REMOTE OBJECT INVOCATION

A Remote Object: a remote object is an object whose method can be invoked from another JVM lets understand the Stub and Skeleton objects.

STUB: stub is an object, acts as a gateway for the client side and represents the remote object when caller invokes method on the stub objects, 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 the (UnMarshals) the return value or exception and
5. It finally, returns the value to the caller

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

1. It reads the parameter of 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.

Understanding the Requirements for the distributed applications:

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

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

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

Steps to write RMI program:

The given 6 steps to write the RMI program

1. Create the remote interface
2. Provide the implementation of the remote interface
3. Create the remote interface:

For creating the Remote Interface, extend the Remote Interface and declare the Remote exception with all the methods of the remote interface there is only one name add() and it declares Remote Exception.

1. Import java.rmi;
2. Public interface Adder extend Remote {
3. Public int add(int x, int y) throws Remote Exception;
4. }
5. Provide the implementation of the remote interface.

Now provide the implementation of the remote interface. For providing the implementation of the remote interface we need to

\*Either extend the UnicastRemoteObject class

\* or use the exportObject() method of the UnicastRemoteObject class.

Incase , you extend the UnicastRemoteObject class, you must define a constructor that declares RemoteException.

1.import java.rmi.\*;

2.import java.rmi.server.\*;

3.public class AdderRemote extends UnicastRemoteObject implements Adder {

4. AdderRemote() throws RemoteException{

5. Super();

6. }

7. public int add(int x, int y){return x+y};

8.}

1. Create the STUB and SKELETON objects using rmic tool

Next step is to create stub and skeleton objects using rmi compiler. The rmic tool invokes the RMI compiler and creates stub and skeleton objects.

1. Rmil AdderRemote
2. Start the registry service by the rmi registry tool

* Now start the registry service by using the rmiregistry tool . If you don’t specify the port number, it uses a default port number 1099 .in the above example we are using the Port number 5000 the remote object from another machine change the local host to the hostname (or IP address) where the remote object is located.

1. Import java.rmi.\*;
2. Public class MyClient {
3. Public static void main(String args[]){
4. Try{
5. Adder stub=(Adder)Naming.lookup(“rmi://localhost:5000/sonoo”);
6. System.out.println(stub,add(344,4));
7. }catch(Exception e){}
8. }

For executing RMI steps:

1. Compile all java files
2. Java \*.java
3. Create stub and skeleton object by rmic tool
4. Rmic AdderRemote
5. Start rmi registry in one command prompt: rmiregistry 5000.
6. Start the server in another command prompt: java MyServer
7. Start the client application in another command prompt : java MyClient
8. Rmiregistry 5000
9. Create and run the server application now RMI services need to be hosted in a server process. The Naming class provides methods to get and store the remote object the naming class provides 5 methods

* In this example we are binding the remote object by the name sonoo

1. Import java.rmi.\*;
2. Import java.rmi.registry.\*;
3. Public class MyServer{
4. Public static void main(String args[]){
5. Try{
6. Adder stub=new AdderRemote();
7. Naming .rebind(“rmi://localhost:5000/sonoo”,stub);
8. }catch(Exception e){System.out.println€;}
9. }
10. }
11. Create and run the client Application.

* At the client we are getting the stub object by the lookup() method of the naming class and invoking the method on this object. In this example we are running the server and client application in the same machine se we are using local host . If you want to access .

Java Mail API: it is fairly high level representation of any email system

* The components are represented by abstract classes in the javax.mail
* It declares abstract methods to get and set various kinds of envelope information for the message such as the sender and the addressee, the date sent and the subject of the message.
* These classes are all abstract because they don’t make many assumptions about how the email is stored and transferred between machines.
* The java mail API roughly follows the abstract factory design pattern
* Java mail API class specializes the general API to a particular protocol and mail format
* The purpose of java mail API is to shield you from low level details
* The java mail API uses the service provider to speak to the server using the Native protocol
* The java mail API is a standard extension to java not part of the core JDK or JRE
* Since mail arrives from the network at unpredictable times the Java Mail API relies on an event based call-back mechanism to handle incoming mail

**Write a simple calculator program using RMI**

1 \* Calculation.java

Import java.rmi;

Public interface calculation extends java.rmi Remote{

Public long add(long a, long b) throws java.rmi.RemoteException;

Public long sub(long a, long b) throws java.rmi.RemoteException;

Public long mul(long a,long b)throws java.rmi,RemoteException;

Public longdiv(long a, long b) throws java.rmi.RemoteException;

}

2 \* calculator impl.java

Import java.rmi.\*;

Import java.rm.server.\*;

Public class calculatorImpl extends java.rmi.server.UnicastRemoteObject implements calculator{

//implementations must have an explicit constructor in order to declare the remote exception

Public calculatorImpl() throws java.rmi.RemoteException{

Super();

}

Public long add(long a, long b) throws java.rmi.RemoteException{ System.out.println(“doing addition”);

Return a+b;

}

Public long sub(long a, long b)throws java.rmi.RemoteException{System.out.println(“doing subtraction”);

Return a-b;

}

Public long mul(long a, long b)throws java.rmi.RemoteException{System.out.println(“doing multiplication”);

Return a\*b;

Public long div(long a, long b) tfhrows java.rmi.RemoteException{System.out.println(“doing division”);

Return a/b;

3 \* calculatorserver.java

Import java.rmi.Naming;

Public class calculatorServer(){

Try{

Calculator 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();

}

}

4 \* calculatorclient.java

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{

Calculator c=(calculator)Naming.lookup(“rmi://localhost/calculatorservice”);

System.out.println(c.sub(4,3));

System.out.println(c.add(4,5));

System.out.println(c.mul(3,6));

System.out.println(c.div(9,3));

}

Catch(MalformedURLException mu){

System.out.println();

System..out.println(“MalformedURLException”);

System.out.println(mu);

}

Catch(Remote Exception re) {

System.out.println();

System.out.println(“not bound exception”);

System.out.println(nbe)

Catch java.lang.Arithmetic Exception ae){ system.out.println();

System.out.println(“java.lang.ArithmeticException”);

System.out.println(ae);

}

}

]