Ref. No.:

Date : / /

Assignment No:1

Aim: Implement multithreaded client/server process communication using RMI

Obj: Develop multithreaded client/server process communication using java RMI

software used: Java, Eclipse IDE, JDK

Theory: RMI allows java object to invoke method on an object running on another machine. RMI provides remote communication between java program. used for distributed application.

RMI has client & server

1) A typical server program creates some remote objects, makes references to objects accessible, waits for clients to invoke methods on these objects.

2) A typical client program obtains a remote reference to one or more remote objects on server and invokes methods on them.

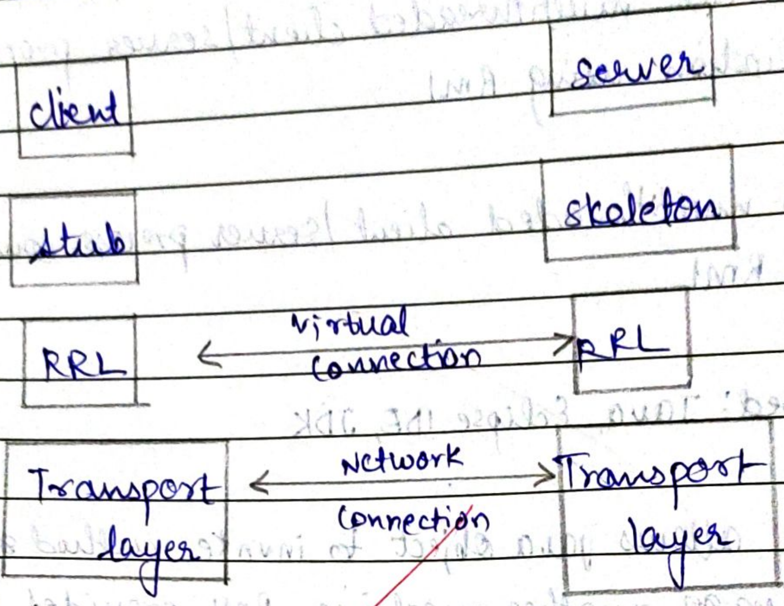
Distributed object applications need to do the following:

- locate remote object
- Communicate with remote objects.
- load class definations for object passed around.

Ref. No.:

Date :    /    /

Architecture :



Transport layer: This layer connects the client and server. It manages the existing connection and al sets up new connections.

Stub: A stub is a proxy of remote object at clie It resides in client system; acts as gateway for client program.

Skeleton: resides on server side. Stub communica with skeleton to pass request to remote object.

RRL: layer that manages references made by client remote object.

Ref. No.: 2

Date : / /

How RMI provides remote communication ?

- The RMI provides remote communication by using two objects stub & skeleton.

1) stub:
The stub is an object, acts as gateway for client side.
when caller invokes method, it does foll tasks:
• initiates connection with remote JVM.
• writes and transmits parameters to remote JVM.
• waits for result.
• Reads the return value.
• returns value to caller.

2) skeleton:
acts as gateway to server side object. It does foll tasks:
• reads parameter for remote method.
• invokes method on actual remote obj
• writes / transmits result to caller.

→ Remote interfaces, objects and methods:
The distributed application built using Java RMI is made up of interfaces and classes. Interfaces declare method and classes implement method
An object becomes remote by implementing a remote interface which has following characteristics:

Ref. No.:

Date : / /

- A remote interface extends interface java.rmi.R
- Each method of interface declares java.rmi. RemoteException in its throws clause, in addition any application specific exceptions.

RMI treats remote object differently from non re object when object is passed from one JVM to another JVM. Rather than making a copy of implementation object RMI passes a remote stub for remote object. The client invokes a method on local stub, which is responsible for carrying out method invocation on remote object.

A stub for remote object implements the same set of remote interfaces that remote object implements However only those methods defined in remote interface are available to be called from JVM wo

→ Marshalling and unmarshalling:
whenever a client invokes a method that accepts parameters are bundled into a message before be sent over network. In case the parameters are objects, then they are serialized. This process i known as marshalling.
At the server side, the packed parameters are unb and then required method is invoked. This proce is called unmarshalling.

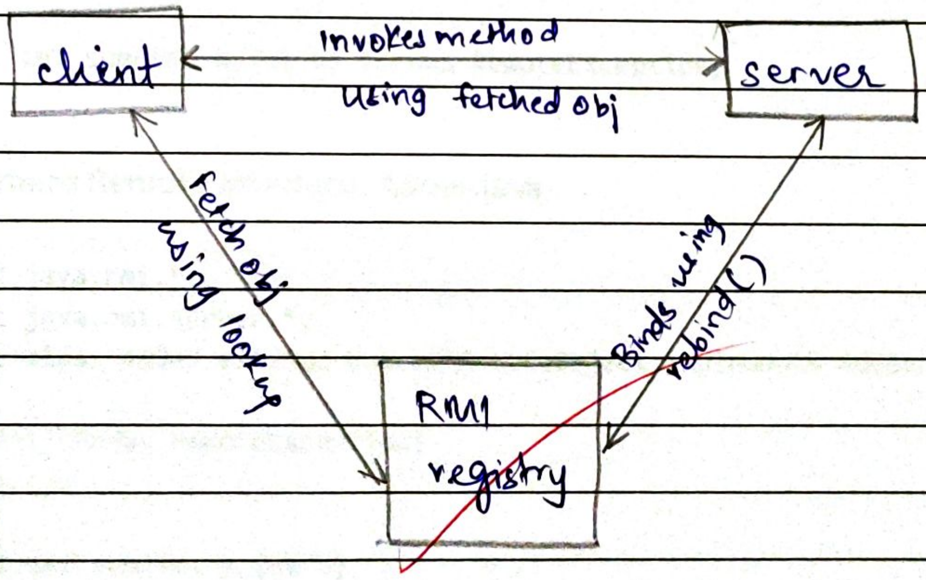
Ref. No.:

Date : / /

→ RMI Registry:

RMI registry is namespace on which all server objects are placed. Each time the server creates an object, it registers the object with RMI registry using bind() or reBind() methods.

To invoke remote obj, the client needs reference of that object. At that time, the client fetches object from registry using lookup() method.



→ Creating distributed applications using RMI:

1) Designing and implementing components of distributed application.

2) Compile sources

3) Making classes network accessible

4) starting application.

Date : / /

→ Goals of RMI:
- minimize complexity of application.
- preserve type safety
- distribute garbage collection
- minimize diff between working with local & remote objects.

Conclusion: We implemented a multi thread client/process communication using RMI.