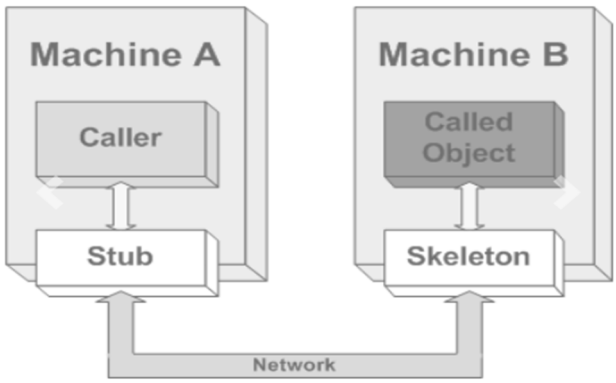
Introduction to Distributed Objects:

Distributed objects refers to the software modules that are designed to work together but reside either in multiple computers connected via a network or in different processes inside the same computer.

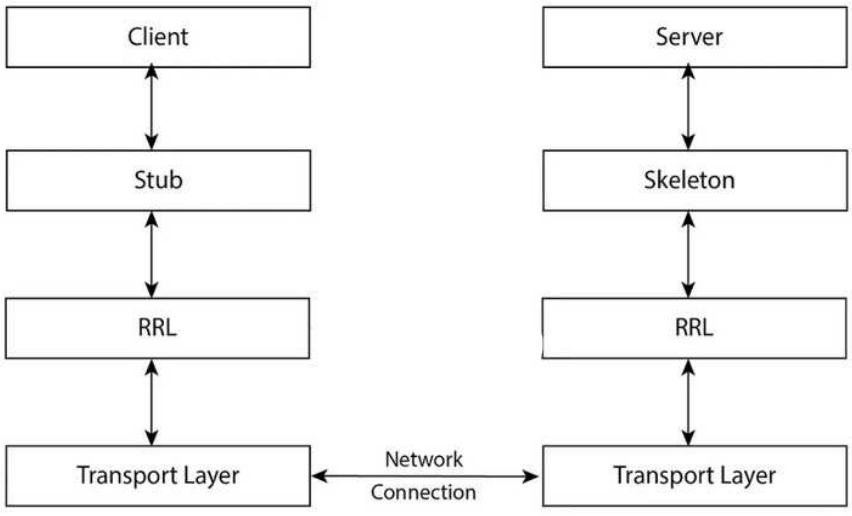


- A stub is defined on client side (machine A).
- Then the stub passes caller data over the network to the server skeleton (machine B).
- The skeleton then passes received data to the called object.
- Skeleton waits for a response and returns the result to the client stub (machine A).

RMI:

The RMI stands for Remote Method Invocation is an API mechanism. It is a mechanism that allows to access on object running on a certain JVM from another different JVM. The communication between client and server is treated using two intermediate objects, Stub object (on the client side) and Skeleton object (on the server side).

Architecture of a RMI application:



Transport Layer: Connect the client and the server. Also have the responsibility to manage new connections.

Stub: Representation of remote object at client. Act as the gateway.

Skeleton: Stub communicates with this to pass request to the remote object.

RRL (Remote Reference Layer): Manage the references made by client to the remote object.

Advantage of RMI:

- Portable across the platforms.
- Code can execute on remote JVMs.
- Existing systems can adapt RMI as this technology is available from JDK 1.02

Disadvantage of RMI:

- Can use only the java supported platforms.
- Limited functionality because of security restrictions.
- No support for legacy systems.

Stub:

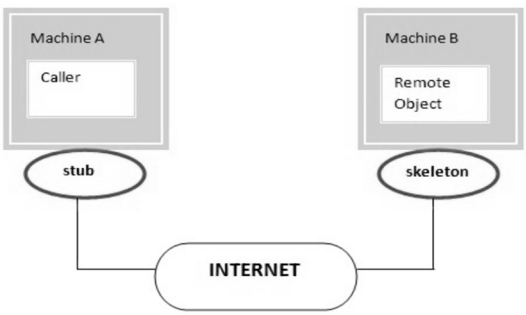
The stub is an object, acting on the client side as a gateway. All the outgoing request are sent through it. When a client invokes the method on the stub object following things are performed internally.

- A connection is established using Remote Virtual Machine.
- It writes and transmits (marshals) the parameters to the remote Virtual Machine (JVM)
- After the 2nd step, it waits for the output.
- Now it reads the value or exception which is come as an output.
- It finally, returns the value to the caller.

Skeleton:

The skeleton is an object that acts as a gateway to the side object of the server. All the incoming request are sent through it. When a Server invokes the method on the skeleton object following things are performed internally.

- All the Parameters are read for the remote method.
- The method is invoked on the remote object.
- It writes and transmits the parameters for the result. This is also known as Marshals.

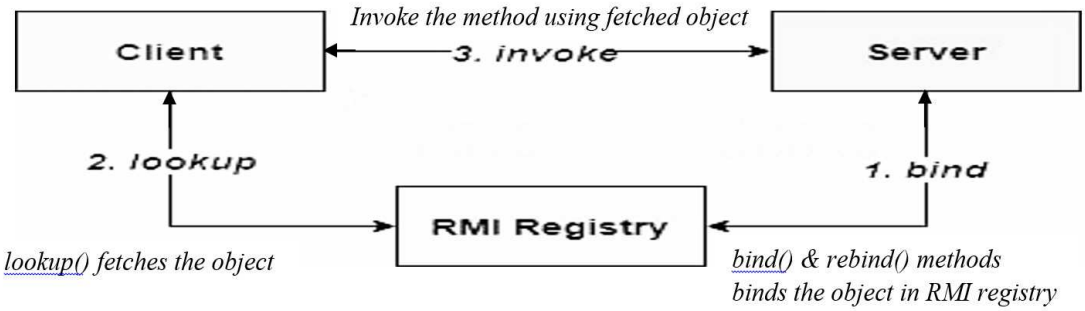


Working of an RMI Application:

- When the client makes a call to the remote object, it is received by the stub which eventually passes this request to the RRL.
- When the client-side RRL receives the request, it invokes a method called invoke() of the object remoteRef. It passes the request to the RRL on the server-side.
- The RRL on the server-side passes the request to the Skeleton (proxy on the server) which finally invokes the required object on the server.
- The result is passed all the way back to the client.

RMI Registry:

RMI Registry acts a broker between RMI servers and the clients. The server "registers" its services in the registry - hence a RMI Registry can act as a "directory" for many servers/services. The client does not need to know the location of individual servers, and does a lookup on the RMI Registry for the service it needs. The registry, being a naming directory returns the appropriate handle to the client to invoke methods on.



RMI Client:

The RMI client gets the reference of one or more remote objects from registry with the help of objects name. Now it can be invokes the methods on the remote object to access the services of the objects as per the requirements of the logic in RMI application.

RMI Server:

RMI server contains object whose methods are to be called remotely. It creates remote objects and applies the reference to these objects in the Registry, after that the registry registers these objects who are going to be called by client remotely.

Creating RMI Application

Steps helps you to create RMI Application:

- Defining a remote interface.
- Implementing the remote interface.
- Start the RMI registry.
- Create and execute the server application program.
- Create and execute the client application program.

Sum.java (interface class)

```
import java.rmi.Remote;
import java.rmi.RemoteException;

public interface Sum extends Remote{
	public int add(int a, int b) throws RemoteException;

}
```

ServerDemo.java

```
import java.rmi.RemoteException;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import java.rmi.server.UnicastRemoteObject;
public class ServerDemo extends UnicastRemoteObject implements Sum{

	public ServerDemo()throws RemoteException{

	}
	public static void main(String[] args) {

		try{
			Registry registry=LocateRegistry.createRegistry(9999);
			ServerDemo obj=new ServerDemo();
			registry.rebind("RMI searching", obj);
			System.out.println("Server is Ready...");
		}catch (RemoteException remoteException) {
			System.out.println("Remote Exception."+remoteException);
		}
	}
	@Override
	public int add(int a, int b) throws RemoteException {

		int c=a+b;
		return c;
	}
}
```

ClientDemo.java

```
import java.rmi.NotBoundException;
import java.rmi.RemoteException;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import java.util.Scanner;

public class ClientDemo {
```

```
    public static void main(String[] args)throws RemoteException, NotBoundException {

        try{
            Registry registry= LocateRegistry.getRegistry("localhost",1099);

            Sum sum=(Sum)registry.lookup("RMI searching");

            Scanner input=new Scanner(System.in);

            System.out.print("Please Enter your First No :");
            int a=input.nextInt();

            System.out.print("Please Enter your Second No :");
            int b=input.nextInt();

            System.out.println("Sum of two nos : "+sum.add(a, b));

        }catch (RemoteException e) {
            System.out.println("Remote Exception."+e);
        }
    }
}
```

CORBA:

CORBA stands for Common Object Request Broker Architecture. It was created by OMG (Object Management Group). OMG was created in 1989. CORBA is a platform which supports multiple programming languages to work together (i.e., it supports multiple platforms). It is a middleware neither 2-tier or 3-tier architecture. The object request broker (ORB) enables clients to invoke methods in a remote object. It is a technology to connect to objects of heterogeneous types.

Types of objects:

There are two types of objects in CORBA.

1. Service Provider Object
2. Client Object

Service provider object: Object that includes functionalities that can be used by other objects.

Client object: Object that requires services of other objects.

Difference between RMI and CORBA

	RMI		CORBA
1	RMI stands for Remote Method Invocation.	1	CORBA stands for Common Object Request Broker Architecture.
2	It uses java interface for implementation.	2	It uses Interface Definition Language (IDL) to separate interface from implementation.

3	RMI programs can download new classes from remote JVMs.	3	CORBA does not support this code-sharing mechanism.
4	RMI passes objects by remote reference or by value.	4	CORBA passes objects by reference.
5	Distributed garbage collection is available integrated with local collectors.	5	No distributed garbage collection is available.
6	RMI is slow in execution than CORBA.	6	CORBA is fast in execution than RMI.