Practical No: 02

Title: Design a distributed application using RMI for remote computation where client submits two strings to the server and server returns the concatenation of the given strings.

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.

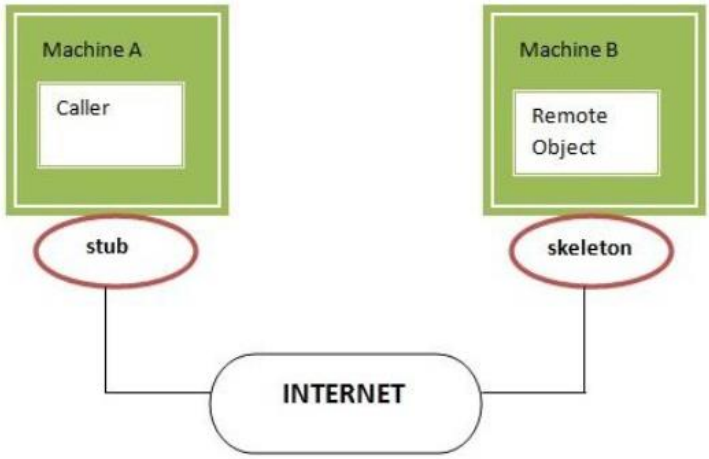
skeleton

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 rmi registry 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.

1) create the remote interface

For creating the remote interface, extend the Remote interface and declare the RemoteException with all the methods of the remote interface. Here, we are creating a remote interface that extends the Remote interface. There is only one method named add() and it declares RemoteException.

```
1.  import java.rmi.*;

2. public interface Adder extends Remote{

3. public int add(int x,int y)throws RemoteException;

4. }
```

2) 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

In case, 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.}
```

3) create the stub and skeleton objects using the rmic tool.

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

1. rmic AdderRemote

4) Start the registry service by the rmiregistry 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. In this example, we are using the port number 5000.

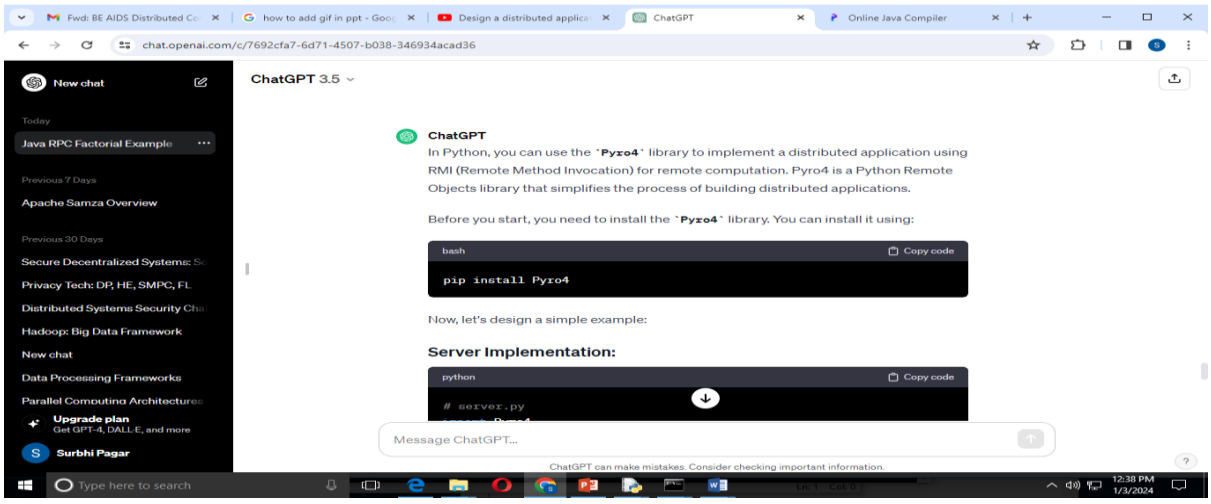
rmiregistry 5000

5) 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.

In Python, you can use the **Pyro4 library to implement a distributed application using RMI (Remote Method Invocation) for remote computation. Pyro4 is a Python Remote Objects library that simplifies the process of building distributed applications.**

Before you start, you need to install the Pyro4 library. You can install it using:



Server Implementation:
 # server.py

```
import Pyro4
@Pyro4.expose
class StringConcatenationServer:
    def concatenate_strings(self, str1, str2):
        result = str1 + str2
        return result

def main():
```

```
    daemon = Pyro4.Daemon()  # Create a Pyro daemon
    ns = Pyro4.locateNS()  # Locate the Pyro nameserver

    # Create an instance of the server class
    server = StringConcatenationServer()

    # Register the server object with the Pyro nameserver
    uri = daemon.register(server)
    ns.register("string.concatenation", uri)

    print("Server URI:", uri)

    with open("server_uri.txt", "w") as f:
        f.write(str(uri))

    daemon.requestLoop()

if __name__ == "__main__":
    main()
```

Client Implementation:

 # client.py

```
import Pyro4

def main():
    with open("server_uri.txt", "r") as f:
        uri = f.read()

    server = Pyro4.Proxy(uri)  # Connect to the remote server

    str1 = input("Enter the first string: ")
    str2 = input("Enter the second string: ")

    result = server.concatenate_strings(str1, str2)

    print("Concatenated Result:", result)

if __name__ == "__main__":
    main()
```

Steps to Run:

Install Pyro4 library

Then Use command Pyro4-ns

And use following steps

1)Save the server code in a file, e.g., server.py

2)Save the client code in a file, e.g., client.py

3)Open a terminal and run the server: python server.py

4)you will get server uri paste it inserver_uri.txt file.keep it in same folder where you have stored python files.

5) Open another terminal and run the client: python client.py

6) enter the values for concatenation.

This example demonstrates a basic setup for using Pyro4 to create a distributed application for string concatenation. Adjust the strings in the client code as needed. Note that this is a simple example, and in a real-world scenario, you might want to handle exceptions, error checking, and security considerations.

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.