UFAZ AOOP Mini-Project

Presented by: Kanan Jafarli and Nezir Ahmedli

RMI(Remote Method Invocation)

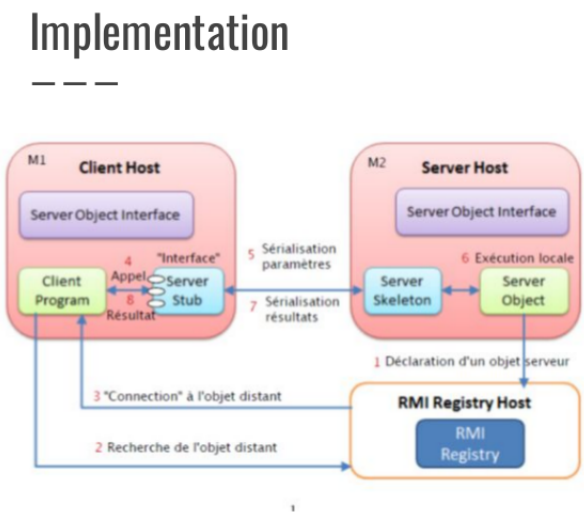
ATTENTION !!! For Run Program:

1. open /bin folder of RMIServeur
2. start the name service `rmiregistry 2019&`. **(because in our code port is 2019)**
3. run **CalculatorServer** class
4. run **CalculatorClient** class

---

1 Server

In this phase, server will accept tasks from client, run the tasks and returns as result.



5. The stub on the M1 machine
   a. packages the method identifier and its arguments (serialization) ;
   b. the request is transmitted over the network;
6. The skeleton on the M2 machine
   a. receives and unpacks the message (deserialization);
   b. calls the requested method;
   c. receives the result of the method;
7. The skeleton
   a. packs this result;
   b. transmits the result to the proxy on the M1 machine;
8. The proxy on the M1 machine
   a. receives and unpacks the message;
   b. returns the result as an ordinary method.

The server code consists of an interface and a class. The interface defines methods that can be invoked from the client. The class provides the implementation.

We created **RMIServeur** project. Inside project we created **Calculator** interface that extends `Remote` class. And we imported `java.rmi.Remote`. Interface has one `calculator` method that has one String attribute and returns integer. This method throws `RemoteException` and above we imported `java.rmi.RemoteException`.

Then we created **CalculatorImpl** class that inherit from `UnicastRemoteObject` class and implements `Calculator` interface. We created default constructor that throws `RemoteException`. Then we used `calculator` method which is the method of interface.

Inside method we used `StringTokenizer` for to break string that is given by client to operand and operation.

- StringTokenizer st = new StringTokenizer(str);
  int int1 = Integer.parseInt(st.nextToken()); //parseInt converts string to integer
  String operation = st.nextToken();
  int int2 = Integer.parseInt(st.nextToken()); //parseInt converts string to integer

Above we imported UnicastRemoteObject, RemoteException and StringTokenizer. We worked positive integers. If both of operands are bigger than or equal to zero at the same time, will continue. Else will print `"Please,enter positive integer"` and will return -1. We used `switch` for operation:

- In case of "+" : int1 + int2
- In case of "-" : int1 - int2
- In case of "*" : int1 * int2
- In case of "/" : int1 / int2
  If int2 equals to zero, it will print `"Error: division by zero"` and will return -1.
- In other cases : will print `"WRONG operation"` and will return -1.

Then we check result: If result is bigger than or equal to zero, method will return answer. Else print `"WRONG operation"` and return -1.

We created **CalculatorServer** class. Before the security manager is disabled. Therefore we invoked setProperty() method of the System, with parameters the name and the value of a system property.The method invocation will throw an AccessControlException, since the security manager is now enabled and the access to the system property is now not allowed. Then a security manager is enabled. It will protect access to system resources from untrusted downloaded code running within JVM. Then we created remote object and assigned to our constructor. java.rmi.registry.LocateRegistry class provides static methods for synthesizing a remote reference to a registry at a particular address (host and port) and with `LocateRegistry.getRegistry(port)` we create new registry in the current JVM.

- registry.rebind("Calculator",stub);

will makes a remote call to the RMI registry on the local host. Line any remote call, this can be end by a RemoteException being thrown that's why we get it inside catch block.

Then we created **security.policy** file in RMIServeur project and added a permission.

security.policy

```
1 grant {
2     permission java.security.AllPermission;
3 };
```

2 Client

We created **RMIClient** project. The client must know the interface of the service proposed by the server. Therefore we copied `Calculator` interface from server. Then we created **secclient.policy** file in RMIClient project and added a permission as in security.policy file.

We created **CalculatorClient** class. In client part we will be doing look up and invokes the remote method. So we are getting input(operands and operation, in order to calculate) using java.util.Scanner. And again, the client use security manager because the process of receiving the server remote object's stub could require downloading class definition from the server. The client also uses the LocateRegistry.getRegistry to synthesize a remote reference to the registry on the server's host. With the local host in command line on which calculator object runs. The client then invokes the `lookup` method on the registry to look up the remote object by name in the server host's registry. Then client reads input using `nextLine` method and prints result.

RMI Registry

RMI registry is a namespace on which all server objects are placed. Each time the server creates an object, it registers this object with the RMIregistry (using **bind()** or **reBind()** methods). These are registered using a unique name known as **bind name**.

To invoke a remote object, the client needs a reference of that object. At that time, the client fetches the object from the registry using its bind name (using **lookup()** method).

The following illustration explains the entire process –