RMI DEMO

Davide Lissoni Mat.179878
06/10/2016

1.Introduction:

This assignment consist in a simple Java Rmi (Remote method invocation) application.
Java Rmi applications refers to distributed applications in which it is made possible communication between remote objects, through the invocation of methods between the objects themselves. In particular for this project we were asked to implement two classes “ Document” and “server” which has to act respectively as client and server.

The custom class Document must be able to keep a set of strings in a “Document” instance and to print them.
The server class instead contain a remote method “addTimestamp” that, taking as parameter a Document type, will concatenate the string “Viewed on < current-timestamp>” at the end of the “Document” instance.
Despite these requirements the main topic of the assignment was to create a server and a client which implements some basic remote methods using Java Rmi.

2.Implementation:

The server consists in an interface “ServerInterface”, which defines the methods that the
client will be able to call (addTimestamp()), and its implementation.
The client part instead, consist in another interface “Document” which define the addString() remote method that, the server, will use in order to perform a remote callback on the document object, situated in the client (I.e. add the String created by the addTimestamp()).
Both the interfaces must extend `java.rmi.Remote,` and are present both in the client and server side.

Once started the server, a “ServiceInterface” object will be created and exported.
A rmiregistry will be created with the port specified as argument (1099 by default).

When the client is started, the server rmiregistry just created will be located and it will be made the “Document” object stub.
In addition the client has to handle the strings, if any, passed by program argument, adding them to the “Document” instance.

At this point the client will call the “addTimestamp()” remote method, passing the “Document” instance as parameter.
The server will run the client call, will create the string to add on the document and then will call in turn the“addString()” remote method situated on the client, passing as parameter the string just created. The client will perform the addString() call and then will print all the strings present in the Document instance through the toString() method (which override the toString method of the Object class).

3.Deployment:

In order to run the server from command line type the command:

java -Djava.rmi.server.hostname=<local ip> -jar RmiServer.jar <port>

If the argument missed, the port in which the server will create the registry is the 1099, which is the default port for rmiregistry.

In order to run the client from command line type the command:

java -Djava.rmi.server.hostname=<local ip> -jar RmiClient.jar <rmiregestry-address:port> <String1><String2>...

The rmiregistry-address with the port number included, has to be the address where the server created the rmiregistry (i.e. 193.168.0.1:1099).
Furthermore the number of strings passed as arguments is irrelevant, they can be one or more strings as well.

4.Comments and notes:

The command *-Djava.rmi.server.hostname=<local ip>* is used in order to control the effective IP address that RMI will use as the server hostname, that, by default is localhost.

Since it was the first time that I had to handle with java rmi, I encountered a bit of problems in the first step of the development part of this project especially in understanding the logic behind Java rmi.
The testing part has also taken me some time in particular when I tried to run the client and the server on different machines.
The program has been tested anyway both locally and on different machines using different operating systems (Ubuntu 15.10 and OSX El Capitan) and works properly.