Università degli Studi di Trento

Web Architectures

RMI DEMO

Gianvito Taneburgo

October 3, 2016

1 Introduction

The following report describes the procedure that has been followed to develop a simple client-server application using Java Remote Method Invocation (RMI) technology. In particular, the program shows how a client can manipulate a local object by calling a method that is implemented and executed on a server, possibly running on a different machine.
The Implementation paragraph provides details both on the server and on the client code. Some details about the RMI have been omitted: the reader is supposed to know the benefits of the technology and the basic mechanisms that make it possible.
The RMI technology has evolved during the years and the process to compile and run client-server applications relying on it has changed as well. For the remainder of this report it will be considered only the 8^{th} version of the Java tool set (language, *javac*, *rmic*, *jre*, etc).

2 Implementation

The application is made up of three classes (*Document*, *Client* and *Server*) and one interface (*ServerI*). *Document* is used by the client to create POJOs that will be manipulated via RMI by the server. For the purpose of this demonstration, an instance of *Document* is only able to contain a set of strings, thus the class will not be further described. *ServerI* declares the methods implemented by the server. In the provided example, a client will only be allowed to invoke the *addTimestamp* server method to append a string to the

input document. *Document* and *ServerI* are shared between the client and the server hosts. Finally, *Client* and *Server* contains the code specific to the components of the architecture.
In order to make the application compliant with the RMI specifications some *ad hoc* changes have been applied:

- the *ServerI* interface extends the *Remote* [1] interface;
- the *addTimestamp* method of *ServerI* can throw a *RemoteException* [2];
- the *Document* class implements the *Serializable* [3] interface in order to make its instances exchangeable via network.

Server main method has to be executed at the beginning. It creates a registry on a user-defined port (default is 1099). On this registry it binds to a user-defined name (default is *rmi:///rmiserver*) an instance of the *Server* class, implementing the *ServerI* interface. Such object is exported with the *exportObject* method [4] of the class *UnicastRemoteObject* [5] to make it available to receive incoming calls by clients. Stubs will be generated at runtime using dynamic proxy objects.
Client main method can now be run. The RMI registry created by the server is first located and then queried (client is thus required to know the proper registry port and resource name) to fetch the stub required for remote invocations. An object of class *Document* is created by the client and then modified by the server with a remote method invocation issued via the *ServerI* stub.

3 Deployment

The *src.zip* file contains the source code of both the client and the server. For convenience, the entire folder is supposed to be downloaded on both the hosts, even if only some of the classes are required by each component. At compile time only those required will be compiled. Please note that the *rmic* tool will not be explicitly used, as it is no more required [6]:

> As with any Java program, you use the javac compiler to compile the source files. The source files contain the declarations of the remote interfaces, their implementations, any other server classes, and the client classes.
>
> > Note: With versions prior to Java Platform, Standard Edition 5.0, an additional step was required to build stub classes, by using the rmic compiler. However, this step is no longer necessary.

[1] https://docs.oracle.com/javase/8/docs/api/java/rmi/Remote.html
[2] https://docs.oracle.com/javase/8/docs/api/java/rmi/RemoteException.html
[3] https://docs.oracle.com/javase/8/docs/api/java/io/Serializable.html
[4] https://docs.oracle.com/javase/8/docs/api/java/rmi/server/UnicastRemoteObject.html#exportObject
[5] https://docs.oracle.com/javase/8/docs/api/java/rmi/server/UnicastRemoteObject.html
[6] http://docs.oracle.com/javase/tutorial/rmi/overview.html

Download the *src.zip* file on the server machine and open a terminal in the download location folder. To unzip and compile the server program just type:

```
$ unzip src.zip; javac src/rmidemo/!(Client.java)
```

To create the RMI registry on port 11111 and to bind the service on the name *rmidemo* just type:

```
$ java -classpath src/ -Dport=11111 -Dname=rmidemo rmidemo.Server
```

The server is now running and can accept remote method invocations:

```
Trying to bind server on RMI port 11111 on the address rmidemo
RMI server ready...
```

Similarly, download the *src.zip* file on the client machine and open a terminal in the download location folder. To unzip and compile the client program just type:

```
$ unzip src.zip; javac src/rmidemo/!(Server.java)
```

To run the client providing the proper parameters just just type:

```
$ java -classpath src/ -Dport=11111 -Dname=rmidemo rmidemo.Client
```

The client terminal will show how the client document has been modified by the server:

```
Creating empty document.
Querying registry to build server stub...
Calling remote server method...
Document:
Viewed on 2016/10/03 19:54:28
```