Lab 5 –Remote Method Invocation in Java

This lab will introduce a simple example that works with Remote Method Invocation (RMI). It will cover how to define an API, how to implement it, and how to compile and run the client and server code, and have them communicate.

Important Note

There are many tutorials available on the use of RMI in Java, many referring to a classical way to build the RMI stubs, run the registry. This method involves a tool called *rmic* to create a stub class and running the registry by calling *rmiregistry* from command line. **This tutorial uses an alternative method**, which automatically includes the stub, and incorporates creating of the registry as part of the server.

Overview

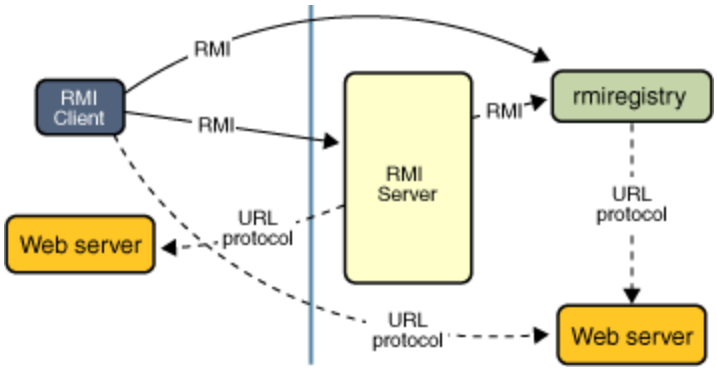
The following description is taken from: http://docs.oracle.com/javase/tutorial/rmi/overview.html

RMI applications often comprise two separate programs, a server and a client. A typical server program creates some remote objects, makes references to these objects accessible, and waits for clients to invoke methods on these objects. A typical client program obtains a remote reference to one or more remote objects on a server and then invokes methods on them. RMI provides the mechanism by which the server and the client communicate and pass information back and forth. Such an application is sometimes referred to as a distributed object application.

Distributed object applications need to do the following:

> ***Locate remote objects.*** *Applications can use various mechanisms to obtain references to remote objects. For example, an application can register its remote objects with RMI's simple naming facility, the RMI registry. Alternatively, an application can pass and return remote object references as part of other remote invocations.*
>
> ***Communicate with remote objects.*** *Details of communication between remote objects are handled by RMI. To the programmer, remote communication looks similar to regular Java method invocations.*
>
> ***Load class definitions for objects that are passed around.*** *Because RMI enables objects to be passed back and forth, it provides mechanisms for loading an object's class definitions as well as for transmitting an object's data.*

The following illustration depicts an RMI distributed application that uses the RMI registry to obtain a reference to a remote object. The server calls the registry to associate (or bind) a name with a remote object. The client looks up the remote object by its name in the server's registry and then invokes a method on it. The illustration also shows that the RMI system uses an existing web server to load class definitions, from server to client and from client to server, for objects when needed.



The RMI example

Please download the two netbeans projects. Use netbeans 7.* to open the applications.

The first project contains the source code for the server, the second for the client.

Exercise 1

Compile the two projects and first run the server, then the client.

Question 1: What does the file api.java contain? Why do both projects contain a file api.java?

Question 2: Why does only one of the projects contain a file *ApiImpl.java*?

Question 3: Describe what happens when the client calls method *setBalance(1000)?*

Question 4: In which line does the server register its service, in which line does the client connect to the registry?

Question 5: Run the client several times. What is the output?

Question 6: Open the task manager in windows, and locate the process running the server. Why is the server running continuously?

Question 7: Where is the process for the client?

Exercise 2

Add three new remote methods.

- One should add an integer amount to the balance.
- One should withdraw an integer amount from the balance.
- The third should add interest given an interest rate to the balance.

Change the client code such that it uses all three methods.

Question 8: Run the client several times. What is the output?

Exercise 3

The project files use integers as method parameters, and the remote methods have either integer or double return types. It is also possible to use objects as parameters and return type.

Add the following class to the api of server and client, and change the code such that it uses objects of type *Data* as parameters and also as return type of the methods.

```
package api;

import java.io.*;

public class Data implements Serializable {
    private static final long serialVersionUID = 1L;
    private int value;

    public Data(int value) {
        this.value = value;
    }

    public int getValue() {
        return value;
    }

    public void setValue(int value) {
        this.value = value;
    }
}
```

Question 9: Build and restart the server. Run the client several times. What is the output?

Question 10: *class Data implements Serializable*. What does it mean to serialize an object, and why is this necessary in the context of RMIs?

Other tutorials

Note: Some tutorials use the classical way to create stubs and start the registry.

- The Java tutorial
- Another Java tutorial
- Simple Example by Edwin
- RMI in 10 minutes
- An RMI chat application