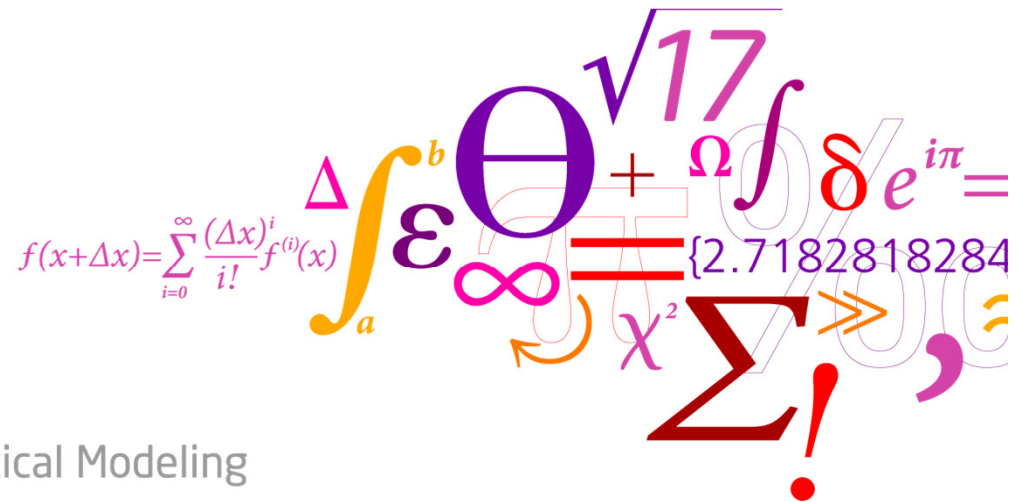


Java RMI Tutorial

Xenofon Fafoutis <xefa@dtu.dk>

Alessio Di Mauro <adma@dtu.dk>



DTU Informatics

Department of Informatics and Mathematical Modeling

Java RMI Concept

A way to invoke methods from a different address space, typically remotely but also locally.

Example

Local Machine (client)

```
RemoteObject remoteobject;  
int sum;
```

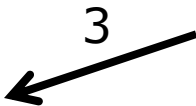
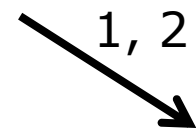
```
sum = remoteobject.add(1,2);
```

```
System.out.println(sum);
```

Remote Machine (server)

```
RemoteObject remoteobject;
```

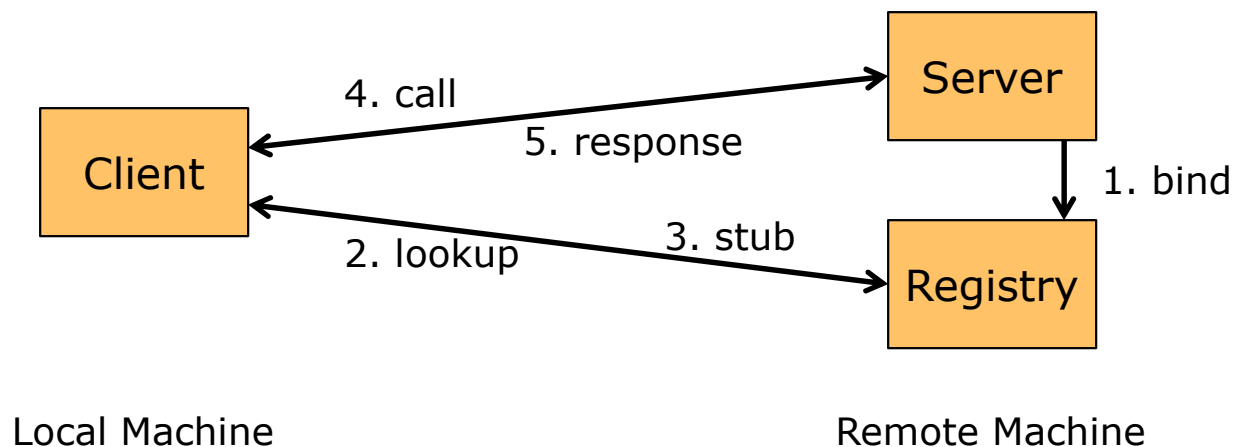
```
Public int add(int a, int b){  
    return a+b;  
}
```



Java RMI Architecture

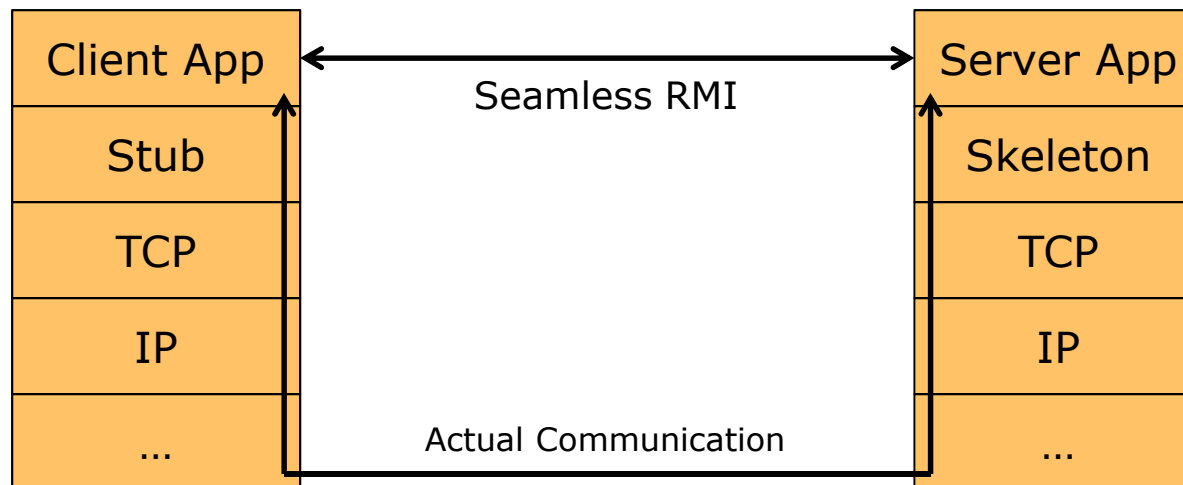
Basic components

- Client
 - Invokes a remote method on a remote object
- Server
 - Owns the remote objects and implements the remote methods
- Registry
 - Relates remote objects with names in plaintext



Java RMI in the TCP/IP Stack

- Middleware between transport and application layer
- Seamless to the programmer (remote object is handled as local object)
- Runs over TCP (reliable communication)



- Stub
 - Pretends to be the remote object
- Skeleton
 - Handles requests from stub / Talks to real remote object

Key Programming Elements

Classes, Interfaces & Methods

- *java.rmi.Remote*
 - Needs to be extended by the classes that contain RMI methods
- *java.rmi.registry.Registry*
 - Associates a name to a remote object
 - Key methods:
 - *bind(string,Remote)*
 - *lookup(string)*
- Static Methods
 - *LocateRegistry.getRegistry([string],[int])*
 - Static method to get the registry
 - *UnicastRemoteObject.exportObject(Remote,[int])*
 - Exports the remote object to JRE to receive remote calls

More in the java docs under the "*java.rmi.**" packages!

Developing an Java RMI System

Implement

- Step 1: Define the interface of the remote object
- Step 2: Implement the server including the remote object
- Step 3: Implement the client

Compile

- Step 4: Compile source files normally

Run

- Step 5: Run *rmiregistry*
- Step 6: Run server
- Step 7: Run client

Step 1: Remote Object Interface

```
import java.rmi.Remote;  
import java.rmi.RemoteException;  
  
public interface Hello extends Remote {  
    String sayHello() throws RemoteException;  
}
```

- It defines the remote object and the input/output of the remote methods
- It needs to extend "java.rmi.Remote" class

Step 2: Implement the Server (pt.1)

```
public class Server implements Hello {  
  
    public Server() {}  
    public String sayHello() {  
        return "Hello, world!";    }  
    public static void main(String args[]) {...    }  
}
```

- The Server needs to implement the methods defined in the interface

Step 2: Implement the Server (pt.2)

```
import java.rmi.server.UnicastRemoteObject;
...
public static void main(String args[]) {
    ...
    Server obj = new Server();
    Hello stub = (Hello) UnicastRemoteObject.exportObject(obj, 0);
    ...
}
```

- The remote object must be *exported* to the Java RMI runtime so that it may receive incoming remote calls
- Method *exportObject* takes care of the Server socket
- Second argument defines the *port* number (optional)
 - 0 for letting the OS choose the port

Step 2: Implement the Server (pt.3)

```
import java.rmi.registry.Registry;
import java.rmi.registry.LocateRegistry;
...
public static void main(String args[]) {
    ...
    Registry registry = LocateRegistry.getRegistry();
    registry.bind("Hello", stub);
    ...
}
```

- Locate and get the name registry
 - Registry by default operates on port 1099
 - Unless another ip/port is specified in the arguments of *getRegistry*, it looks for the registry on *localhost* (127.0.0.1) in the default port (1099)
- Method *bind* registers the remote object (*stub*) with a name in plaintext

Step 3: Implement the Client

```
import java.rmi.registry.Registry;
import java.rmi.registry.LocateRegistry;
...
public static void main(String args[]) {
    ...
    Registry registry = LocateRegistry.getRegistry(host);
    Hello stub = (Hello) registry.lookup("Hello");
    String response = stub.sayHello();
    ...
}
```

- Locate and get the name registry
- Retrieve the remote object using its plaintext name (method *lookup*)
- Invoke the remote methods

Steps 4-7: Compile and Run

- Compile normally
 - > *javac Hello.java Server.java Client.java*
- Run registry
 - > *rmiregistry* *[runs on default port 1099]*
 - > *rmiregistry 2001* *[runs on chosen port, 2001]*
- Run server
 - > *java -Djava.rmi.server.codebase=[url]/ Server*
- Run client
 - > *java -Djava.rmi.server.codebase=[url]/ Client*

The *java.rmi.server.codebase* property specifies the location, a codebase **URL**, from which the definitions for classes originating *from* this server can be downloaded.

If the classes are in the local file system use the file URL scheme (e.g. *file:/%CD%/* for Windows).

References

Example is based on “Getting Started Using Java™ RMI” by Oracle

<http://docs.oracle.com/javase/6/docs/technotes/guides/rmi/hello/hello-world.html>

Exercise

The German mathematician Gottfried Leibniz developed the following method to approximate the value of π :

$$\pi/4 = 1 - 1/3 + 1/5 - 1/7 + \dots$$

Write a Java RMI program that allows the user to specify the number of iterations used in this approximation, invokes a remote method in a RMI server that executes the algorithm, and displays the received value.