

Definitions and examples

RMI

Remote Method Invocation

By **Hamid Mosavi-Porasi**

Implementing the RMI Server on the Remote Host

Remote objects are referenced via interfaces. In order to implement a remote object, we must first create an interface for that object. This interface must be *public* and must extend the `Remote` interface. Define the remote methods that you want to invoke within this interface. These methods must *throw* `RemoteException`. Look at the [MyServer.java](#) . `MyServer` defines two methods: `getDataNum()` and `getData()`. The `getDataNum()` method returns an integer indicating the total number of data strings that are available on the server. The `getData()` method returns the *n*th data string.

Compile `MyServer.java`
e.g. `>javac MyServer.java`

```
import java.rmi.*;

public interface MyServer extends Remote{
    int getDataNum() throws RemoteException;
    String getData(int n) throws RemoteException;
}
```

- **Create a class that implements the remote interface**

After creating the remote interface, we must create a class that implements the remote interface. This class typically extends the `UnicastRemoteServer` class.

The implementation class should have a structure that creates and initialize the remote object. It should also implement all of the methods defined in the remote interface. It should have a `main()` method so that it can be executed as a remote class. The `main()` method should use the `securityManager()` method of the `System` class to set an object to be used as the remote object's security manager. It should register a name by which it can be remotely referenced with the remote registry.

`MyServerImpl` class provides the implementation class for the *myServer* interface. We should change the *hostname* value to the name of the host where the remote object is to be located. The data array contains five strings that are retrieved by the client object via the `getDataNum()` and `getData()` methods. The `getDataNum()` method returns the length of data, and the `getData()` method returns the *n*th element of the data array.

The `main()` method sets the security manager to an object of the `RMISecurityManager` class. It creates an instance of the `MyServerImpl` class and invokes the `rebind()` method of `Naming` to register the new object with remote registry. It registers the object with the name `MyServer` and informs you that it has successfully completed the registration process.

Compile [MyServerImpl.java](#) .

```
import java.rmi.*;
```

```

import java.rmi.server.*;

public class MyServerImpl extends UnicastRemoteObject
    implements MyServer{
    static String hostname ="h76n2fls32o927.telia.com";
    static String data[]={ "Remote", "Method", "Invokation", "is", "Great!"};

    public MyServerImpl() throws RemoteException{
    super();
    }
    public int getDataNum() throws RemoteException{
    return data.length;
    }
    public String getData(int n) throws RemoteException{
    return data[data.length];
    }
    public static void main(String args[]){
    System.setSecurityManager(new RMISecurityManager());
    try{
    MyServerImpl instance = new MyServerImpl();
    Naming.rebind("//"+hostname+"/MyServer", instance);
    System.out.println("I'm registered!");
    }catch(Exception ex){
    System.out.println(ex);
    }
    }
}

```

- **Create Stub and Skeleton Classes**

Once we have created the class that implements the remote interface, use the *rmic* compiler to create the stub and skeleton classes:

rmic MyServerImpl

The *rmic* compiler creates the files *MyServerImpl_stub.class* and *MyServerImpl_Skel.class* in the directory.

- **Copy the remote interface and stub file to the client host**

We'll need the *MyServer.class* interface file to compile our client software, and we'll need *MyServer.class* and *MyServerImpl_stub.class* to run our client. Copy these files to an appropriate location on your client host. If you run both the client and server on the same computer, the directory structure and files should already be in position.

- **Start Up the remote registry**

Now we must start our remote registry server. This program listens on the default port 1099 for incoming requests to access named objects. The named objects must register

themselves with the remote registry program in order to be made available to requesters. You start up the remote registry server as follows:

Start rmiregistry

- **Create and Register the Remote Object**

We're almost done with the remote server. The last thing to do is to execute the MyServerImp program to create an object of the MyServerImpl class that registers itself with the remote registry. We do this as follows:

Java MyServerImpl

I'm registered!

The program displays the I'm registered! String to let us know that it has successfully registered itself. Leave the server running (don't exit the server program by pressing Ctrl+C) while you start the client. If we the client and server on the same computer, we'll need to open up a separate command line window for the client.

- **Implementing the RMI Client on the Local Host**

Now that we have the remote server up and running, let's create a client program to remotely invoke the methods of the MyServer object and display the results it returns. You must change the hostname variable, in the [client](#) program, to the name of the remote server host where the object is registered. Compile this program and copy it to a client directory that is accessible from the CLASS PATH of the client host. Once we have compiled it, we can run it as follows:

Java MyClient

Remote
Method
Invocation
Is
Great!

The [MyClient](#) program remotely invokes the methods of the server object and displays the data returned to the console window.

MyClient consists of a single main() method that invokes the lookup() method of the Naming class to retrieve a reference to the object named MyServer interface. It then invokes the getDataNum() method of the remote object to retrieve the number of available data items, and the getData() method to retrieved data items are displayed in the console window.

```
import server_packet, e.g the packet containing server code
import java.rmi.*;
```

```
public class MyClient{
    static String hostname ="h76n2fls32o927.telia.com";
    public static void main(String args[]){
```

```
try{
    MyServer server = (MyServer) Naming.lookup("//"+hostname+"/MyServer");
    int n = server.getDataNum();
    for (int i =0; i < n;++i){
        System.out.println(server.getData(i));
    }
}catch (Exception ex){
    System.out.println(ex);
}}
```