Java RMI

Java has a Remote Method Invocation mechanism.

Remote Object Interface

```
// RMIExample.java
// Interface for the RMI remote object.
// Note:  Interface must extend from java.rmi.Remote
//          Methods must throw RemoteExcpetion

import java.rmi.*;

public interface RMIExample extends Remote
{
        public boolean PostMsg(String strMsg) throws RemoteException;
        public long Factorial(long lVal) throws RemoteException;
}
```

Note: the class `Remote` doesn't have any methods so none have to be implemented.

Server Side Code

This code implements the remote object.

```
// RMIExampleImpl.java
// Implements the remote object
// Note: The object must extend from UnicastRemoteObject
//       The object must implement the associated interface

import java.rmi.*;
import java.rmi.server.UnicastRemoteObject;
import java.io.*;

public class RMIExampleImpl extends UnicastRemoteObject
    implements RMIExample
{
    protected static String          m_strName;

    public RMIExampleImpl() throws RemoteException
    {
        super(); // call base class constructor
    }

    public boolean PostMsg(String strMsg) throws RemoteException
    {
        System.out.println("Server: PostMsg() invoked...");
        System.out.println("Server: Message > " + strMsg);
        return true;
    }

    public long Factorial(long lVal) throws RemoteException
    {
        long lRes = FactorialEx(lVal);
        System.out.println("Server: Factorial() invoked...");
        System.out.println("Server: Factorial("+lVal+") = " + lRes);
        return lRes;
    }

    protected long FactorialEx(long lVal)
    {
        if (lVal <= 1)
            return 1;
        else
            return lVal * FactorialEx(lVal-1);
```

```
    }

    public static void main(String argv[])
    {
        try
        {
            m_strName = "TheRMIExample";
            System.out.println("Server: Registering RMIExampleImpl as \"" + m_strName +"
            RMIExampleImpl Example = new RMIExampleImpl();
            Naming.rebind(m_strName, Example);
            System.out.println("Server: Ready...");
        }
        catch (Exception e)
        {
            System.out.println("Server: Failed to register RMIExampleImpl: " + e);
        }
    }
}
```

Client-Side Code

```
// RMIClient.java
//
// This sample Java RMI client can perform the
// following operations:
//   (1)  Send a message to a remote object.  This
//        is done by using the -m command line switch.
//        Example:  java RMIClient server -m "My message in quotes"
//   (2)  Calculate the factorial of a given number via
//        a method of the remote object.
//        Example:  java RMIClient server -f 5

import java.rmi.*;
import java.rmi.server.*;

public class RMIClient
{
    public static void main(String argv[])
    {
        // Validate command line parameters
        if (argv.length < 2)
        {
        System.out.println("Usage: java RMIClient serverhost [-m \"MESSAGE\"] [-f INTEGE
            System.exit(1);
        }

        // Command line option flags
        boolean bMessage = false;
        boolean bFactorial = false;

        String strMsg = "No message.";
        String srvr = "localhost";
        long   lVal = 1;

        // Determine data to be processed
        for (int i=0; i<argv.length; i++)
        {
            if (argv[i].equals("-m"))
            {
                bMessage = true;
                strMsg = argv[++i];
            }
            else if (argv[i].equals("-f"))
```

```
            {
                bFactorial = true;
                lVal = Long.parseLong(argv[++i]);
            }
    else {
srvr = argv[i];
            }
         }

        // Install security manager.  This is only necessary
        // if the remote object's client stub does not reside
        // on the client machine (it resides on the server).
        System.setSecurityManager(new RMISecurityManager());

        // Get a remote reference to the RMIExampleImpl class
        String strName = "rmi://" + srvr + "/TheRMIExample";
        System.out.println("Client: Looking up " + strName + "...");
        RMIExample RemRMIExample = null;

        try
        {
            RemRMIExample = (RMIExample)Naming.lookup(strName);
        }
        catch (Exception e)
        {
        System.out.println("Client: Exception thrown looking up " + strName);
        System.exit(1);
    }

        // Send a messge to the remote object
        if (bMessage)
        {
            try
            {
                if (!RemRMIExample.PostMsg(strMsg))
                    System.out.println("Client: Remote PostMsg() call failed.");
            }
            catch (Exception e)
            {
            System.out.println("Client: Exception thrown calling PostMsg().");
            System.exit(1);
            }
        }

        // Calculate the factorial
```

```
        if (bFactorial)
        {
            try
            {
                long lRes = RemRMIExample.Factorial(lVal);
                System.out.println("Client: Factorial(" + lVal + ") = " + lRes);
            }
            catch (Exception e)
            {
                 System.out.println("Client: Excpetion thrown calling Factorial().");
                 System.exit(1);
            }
         }
    }
}
```

Makefile

```
RMIExampleImplStubs: RMIExampleImpl
        rmic RMIExampleImpl

RMIExampleImpl: RMIExample RMIClient
        javac RMIExampleImpl.java

RMIExample: RMIExample.java
        javac RMIExample.java

RMIClient: RMIClient.java
        javac RMIClient.java

clean:
        rm RMIClient.class RMIExample.class RMIExampleImpl.class RMIExampleImpl_Stub.cla
```

Sample Script

```
< 1 >ls
Makefile              RMIExample.java       java.policy
RMIClient.java        RMIExampleImpl.script
< 2 >make
javac RMIExample.java
javac RMIClient.java
javac RMIExampleImpl.java
rmic RMIExampleImpl
< 3 >rmiregistry&
[1] 70
< 4 >ls
Makefile                    RMIExampleImpl.class
RMIClient.class             RMIExampleImpl.java
RMIClient.java              RMIExampleImpl_Skel.class
RMIExample.class            RMIExampleImpl_Stub.class
RMIExample.java             java.policy
< 5 >java RMIExampleImpl&
[2] 19426
Server: Registering RMIExampleImpl as "TheRMIExample"
Server: Ready...
< 6 > java RMIClient ccc1 -m "How is the weather over there?"
Client: Looking up rmi://..edu/TheRMIExample...
Client: Exception thrown looking up rmi://..edu/TheRMIExample
< 8 >java -Djava.security.policy=java.policy RMIClient ccc1 -m "hello from the client"
Client: Looking up rmi://..edu/TheRMIExample...
Server: PostMsg() invoked...
Server: Message > hello from the client
< 9 >java -Djava.security.policy=java.policy ccc1 RMIClient -f 5
Client: Looking up rmi://..edu/TheRMIExample...
Server: Factorial() invoked...
Server: Factorial(5) = 120
Client: Factorial(5) = 120
< 10 >jobs
[1]  + Running                       rmiregistry
[2]  - Running                       java RMIExampleImpl
< 11 >kill %1 %2
< 12 >
[1]    Terminated                    rmiregistry
< 13 >
[2]    Terminated                    java RMIExampleImpl
```

Remote Object Requirements

1. A remote object's interface MUST be written as extending the java.rmi.Remote interface. This serves to mark remote objects for the RMI system. No methods are introduced by java.rmi.Remote.

2. A remote object's interface must be public.

3. A remote object's interface SHOULD extend java.rmi.server.UnicastRemoteObject. This servers to replace several Object class methods so that they work properly in a distributed environment (examples: hashCode(), equals(), toString()). Essentially, precautions are taken so that each client receives the same result when calling certain remote object methods.

4. All methods MUST be declared as throwing java.rmi.RemoteException. This could be seen as an RMI drawback - existing Java interfaces must be modified in order to function in a distributed environment.

5. Register the object using the java.rmi.Naming interface (implemented in the object's code). Example: Naming.rebind("ObjectName", new MyObjectImpl()); Note the RMI registry must be running before the object can be launched to register itself - see below.

6. Once the object's interface is defined and an implementation is derived, the object is compiled into bytecode using the javac compiler. A client and server stub is then created from the bytecode using the RMI stub compiler, rmic. The client stub serves to provide hooks into the object serialization subsystem in RMI for marshaling method parameters.

7. The RMI registry must be running on the server. The RMI registry is launched with the command rmiregistry [PORT_NUMBER] (included with the standard JDK distribution). The default port number is 1099. Multiple RMI registries may be running concurrently on different ports.

Client Requirements

1. Initialize the RMI security manager with the System object. Example: System.setSecurityManager(new java.rmi.RMISecurityManager()); The security manager enforces Java security upon remote (client) stubs retrieved from a network source. If no security manager is initialized, then stub classes can only be loaded from the local file system. Note that client stubs may reside on the server OR the client machine. If the client stub resides on the server, it is transmitted to the client in the lookup() call - see below. Copy file java.policy to where your client is.

2. The client must use the java.rmi.Naming method lookup() to retrieve a remote object reference. Calling lookup() causes the server's RMI registry to be queried. The string passed to lookup() uses a URL-like format in which the server and object are specified. Note that a specific port corresponding to a specific RMI registry may be specified. Example1: lookup("rmi://garden.wpi.edu/ObjectName"). Example2: lookkup("rmi://garden.wpi.edu:1099/ObjectName").

Quick Guide

- The target server must be specified on the command line to this client and is included in the "rmi://.../..." string in RMIClient.java.
- Put all files to the same directory on the server machine and type 'make' to build all of the Java classes.
- If the client will run on a machine using a different file server, then repeat the step above on the client machine.
- Launch the RMI registry on the server machine with: `rmiregistry &` (remember to kill -9 the process when your done using it)
- Launch the remote object with java RMIExampleImpl
- Now launch the client (same or remote machine) with java -Djava.security.policy=java.policy RMIClient -m "A Message to send"
- See the RMIClient.java file for the client command line options and a description of what they do.