Networks and Database Systems

Professor Tom Ellman
Lecture 6
Remote Method Invocation

- A Java virtual machine runs program $P_a$ on host A.
- A Java virtual machine runs program $P_b$ on host B.
- $P_a$ invokes method on object $O_b$ in $P_b$.
- $P_b$ executes the invoked method on host B.
- $P_a$ gets result value of invocation from $P_b$. 
Remote Method Invocation

Host A

Program $P_a$

Invoking Method on object $O_b$

Object $O_b$

Program $P_b$

Host B

Return Value
Remote Method Invocation

Client Application

O_b Proxy

Stub

Server Application

O_b Object

Skeleton
Server Side Implementation

- Declaration of remote object interface.
- Implementation of the remote object class.
- Implementation of the remote server class.
package hellormi;

import java.rmi.*;

public interface Hello extends Remote {
    public String getGreeting() throws RemoteException;
}

HelloServer

In this simple application, the remote object will implement the Hello interface. This interface has just one method with no parameters that returns a greeting string. The interface must extend the Remote interface. Remote is an empty interface (analogous to Serializable).
The server includes the class **HelloImpl**, which extends the **UnicastRemoteObject** base class and implements the **Hello** interface. This is the class that implements the method (**getGreeting**) that will be invoked from the client.
package hellormi;
import java.rmi.*;

public class HelloServer {
    public static void main(String[] args) throws Exception {
        HelloImpl temp = new HelloImpl();

        String rmiObjectName = "Hello";
        Naming.rebind(rmiObjectName, temp);

        System.out.println("Binding complete...\n");
        temp.setGreeting("Yo! Hello!");
    }
}

The server main program begins by creating an object (temp) that is an instance of the class HelloImpl. Next it uses the Naming class to bind the (arbitrary) name "Hello" to the temp object. This allows remote clients to identify temp using the name "Hello".
Naming Remote Objects
(On Server Side)

rmi://<HostName>/<ObjectName>
//<HostName>/<ObjectName>
<ObjectName>

rmi://localhost/Hello
//localhost/Hello
Hello
Client Side Implementation

- Declaration of remote object interface.
- Implementation of the client class.
HelloClient

```java
package hellormi;

import java.rmi.*;

public interface Hello extends Remote {
    public String getGreeting() throws RemoteException;
}
```

The client application needs its own copy of the **Hello** interface. It tells the client how to interface with the remote object.
The client passes a URL to the `Naming.lookup` method. The URL identifies the protocol (`rmi`), the host (`localhost`) and the name (`Hello`) bound to the remote object. The object returned is a reference to the remote object. The client casts it to the `Hello` interface and then invokes the remote method `getGreeting` on it.
Naming Remote Objects
(On Client Side)

rmi://<HostName>/<ObjectName>
//<HostName>/<ObjectName>

rmi://localhost/hello
//localhost/hello
Compiling and Configuring the Client and Server

• Compile the client and server in the usual way.
• Run the **rmic** utility on the server’s remote object implementation class (**helloرمي.HelloImpl**) to obtain a stub class (**HelloImpl_Stub**).
• Put the stub class (**HelloImpl_Stub**) in the class path of the client application.
Running a Client/Server RMIApplication

1. Start the `rmiregistry` utility program.
2. Start the server program.
3. Run the client program.
The server contacts its registry process to bind the name "Hello" to the HelloImpl remote object. The client program interacts with its local HelloImplStub object. The stub communicates with the server’s registry process to access the HelloImpl remote object.
Passing Objects through RMI Interfaces

- RMI can pass Java objects to/from remote method.
- Objects passed must implement `Serializable` interface.
- Examples:
  - Bank1/Bank2 Client/Server: Passing `Account` objects to/from remote method.
  - Fibonacci Client/Server: Passing `BigInteger` objects to/from remote method.
- Terminology:
  - Marshalling: Passing parameters to remote objects.
  - UnMarshalling: Returning result values from remote objects.
Command Line Parameters for Starting the Server JVM

java -Djava.rmi.server.codebase=https://vspace.vassar.edu/thellman/web/375/
-Djava.rmi.server.hostname=192.168.10.121

By passing suitable command line parameters to the Java Virtual Machine, the client can arrange to load the stub class file (HelloImpl_Stub) from a server on the host at run time.

The server must also inform the JVM of the host name or IP address that clients will use to connect to the server.
Advantages of RMI

• After initial setup, remote location of object is invisible to the application program.
• Compare to complexity of implementing a custom protocol using sockets.
Disadvantages of RMI

- Client needs access to server’s class files.
- Loading of remote class files is a security risk.
- RMI makes sense mainly when client and server are being developed by the same team of people.
Lab Exercise: Run the Fibonacci Client/Server RMI Demo

- Work in pairs, using different machines for client and server.
- Run in debug mode and put breakpoints in client and server to observe interaction.
- Provide server with VM arguments indicating server host name or IP address.
- Start `rmiregistry` in class directory of server application.