JAC444 - Introduction to Java for C++ Programmers

Lesson 10: Remote Method Invocation (RMI)

Agenda

- Introduction to RMI
- Building RMI Applications
- Running RMI Applications
- RMI Architecture Layers
- Workshop 9

Introduction to RMI

- RMI (Remote Method Invocation) is a framework for building distributed Java systems.
- The RMI system allows an object running in one Java virtual machine to invoke methods on an object running in another Java virtual machine.
- What is a distributed system/application?
  - A collection of cooperative remote Java objects reside on DIFFERENT computers connected by the Internet.
- application development questions
  - how to use these remote objects? how to create the remote objects?
  - Solutions
    - use of socket-based client/server programming model
    - use of RMI

Stand-alone vs Distributed Application

- a stand-alone Java application
  - all Java objects reside on ONE computer
  - example:

    ProductApplication.java
- a distributed application
  - What if the Java objects are created on another computer?
  - How can a Java program running on a different computer use these Java objects?

Socket-based vs RMI

- The Limitation of Using Java Sockets
  - the complexity of coding application-level protocols
    - developers have to deal with encoding and decoding messages for exchange
    - time-consuming and error-prone
- Java-to-Java Distributed Applications
  - a distributed application is split across the client and the server
  - RMI: Java support for distributing objects across multiple computers
    - Java objects run on different computers over the Internet
    - Java objects communicate with one another via remote method calls

Remote Method Invocation (RMI)

- a Java-based distributed programming model

  "making remote methods calls across the network"

- a technology for creating and using distributed Java objects (i.e. Java objects that are spread over the network)
  - invocation of Java methods on remote objects

Remote Method Invocation (RMI)

- an abstraction of network communication
  - network communication is transparent to a Java programmer
  - a Java programmer does not write any code to handle remote method calls and data communication

Remote Method Invocation (RMI)

- a remote object / a server object is created and registered on a server
- a client object uses a remote object by obtaining a remote reference to the object and making remote method calls
- seamless integration: same Java syntax and semantics as in non-distributed Java applications

Building RMI Applications

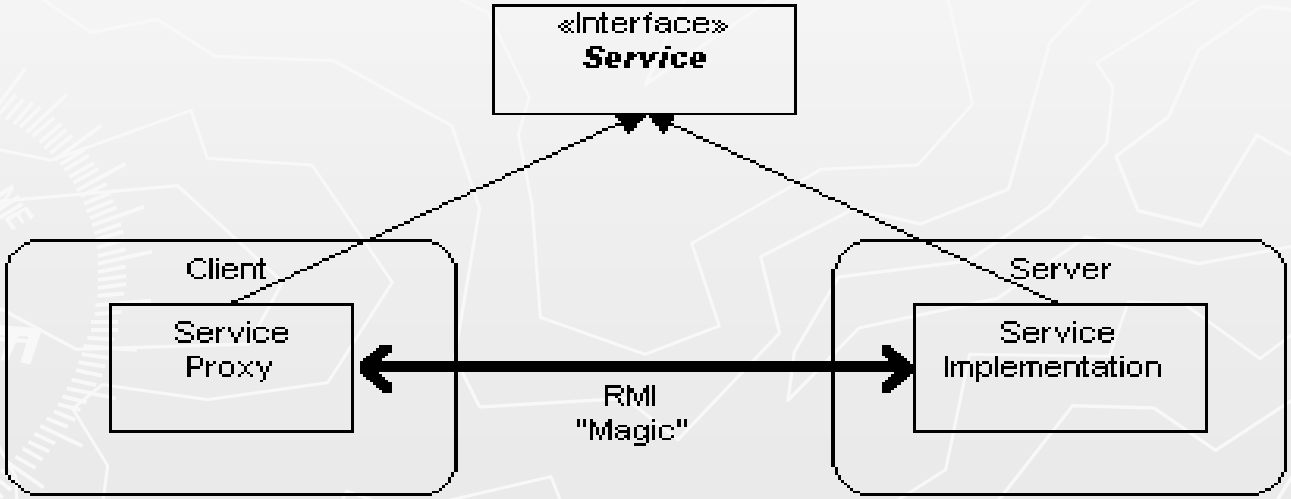
1. develop the "Remote" interface of a remote object (i.e. declarations of remote methods)
   - the interface shared by the client and the server
2. develop the remote object implementation of the "Remote" interface
   - server-side programming
3. develop a server application to bind the remote object to the RMI registry
   - server-side programming
4. develop a client application that uses remote objects
   - client-side programming

Example 1

- interface: Product.java
- implementation: ProductImpl.java
- server application: ProductServer.java
- client application: ProductClient.java

The Heart of RMI

- RMI allows the code that defines the behavior and the code that implements the behavior to remain separate and to run on separate JVMs.



Key RMI Programming Concept

remote service

= a remote interface +

an implementation

- a client object uses a remote interface
- a server (JVM) runs the implementation
  (i.e. provides the remote service)
- a stub class: a proxy for the remote service
  and runs on the client (JVM)

Building RMI Applications

1. Developing A Remote Interface

- a declaration of remote methods
- the Remote interface: a tagging interface
- the method arguments and the return values must be "serializable"
  - checked at run time

- Interface: Product.java

Building RMI Applications

2. Remote Object Implementation

- a subclass of UnicastRemoteObject
  - an abstraction of communication details
- an implementation of a "remote" interface
  - Product
- the constructor "exports" a remote object
- the RemoteException
  - a superclass of all exceptions that can occur in the RMI run time

- Implementation: ProductImpl.java

Locating/Naming Remote Objects

- Question: How does a client object locate an RMI remote object ?
- Answer: Use a naming or directory service that runs on a host (i.e. server machine) and a port number (default: 1099).
- the RMI registry
  - an RMI naming service
  - a remote object that serves as a directory service for the client objects
  - java.rmi.registry.Registry

Building RMI Applications

3. develop a server application

- the java.rmi.Naming class
- the server side
  - create a local object that implements a remote service
  - register the object with the registry
  - rebind( ) *

    * a thread is started to listen for a client request

❖Server application: ProductServer.java

Building RMI Applications

- the syntax of an RMI object URL

```
rmi://<host_name>
[:<service_port_number]
        /<remote_service_name>
```

e.g. `"rmi://localhost:6666/toaster"`

Building RMI Applications

4. develop a client application
   - the client side
     - use an URL to access the RMI registry on a host and obtain a remote reference to a remote object
       - lookup( )

Client application: ProductClient.java

Running an RMI Application: the server side

- Compile:
  - compile the remote object implementation

```
javac    ProductImpl.java
```

  - RMI compile: generate the stub class (v 1.2 and above)

```
rmic       -v1.2    ProductImpl
```

  - compile the Java application that registers a remote object

```
javac     ProductServer.java
```

Running an RMI Application: the server side

- Run
  - start the RMI registry **

```
rmiregistry 6666
```

** Run the command in the directory that contains the class files. The default port number is 1099.

  - run the Java application

```
java ProudctServer
```

Running an RMI Application: the client side

- compile the client application

```
javac     ProductClient.java
```

- run the client application *

```
java     ProductClient
```

* the stub class must be available locally or loaded remotely

Example - Calculator

- RMI Calculator

  https://scs.senecac.on.ca/~wei.song/jac444/workshops/Calculator.zip

- Compile:

  javac *.java

  rmic CalculatorImpl

- Run

  start rmiregistry 5566

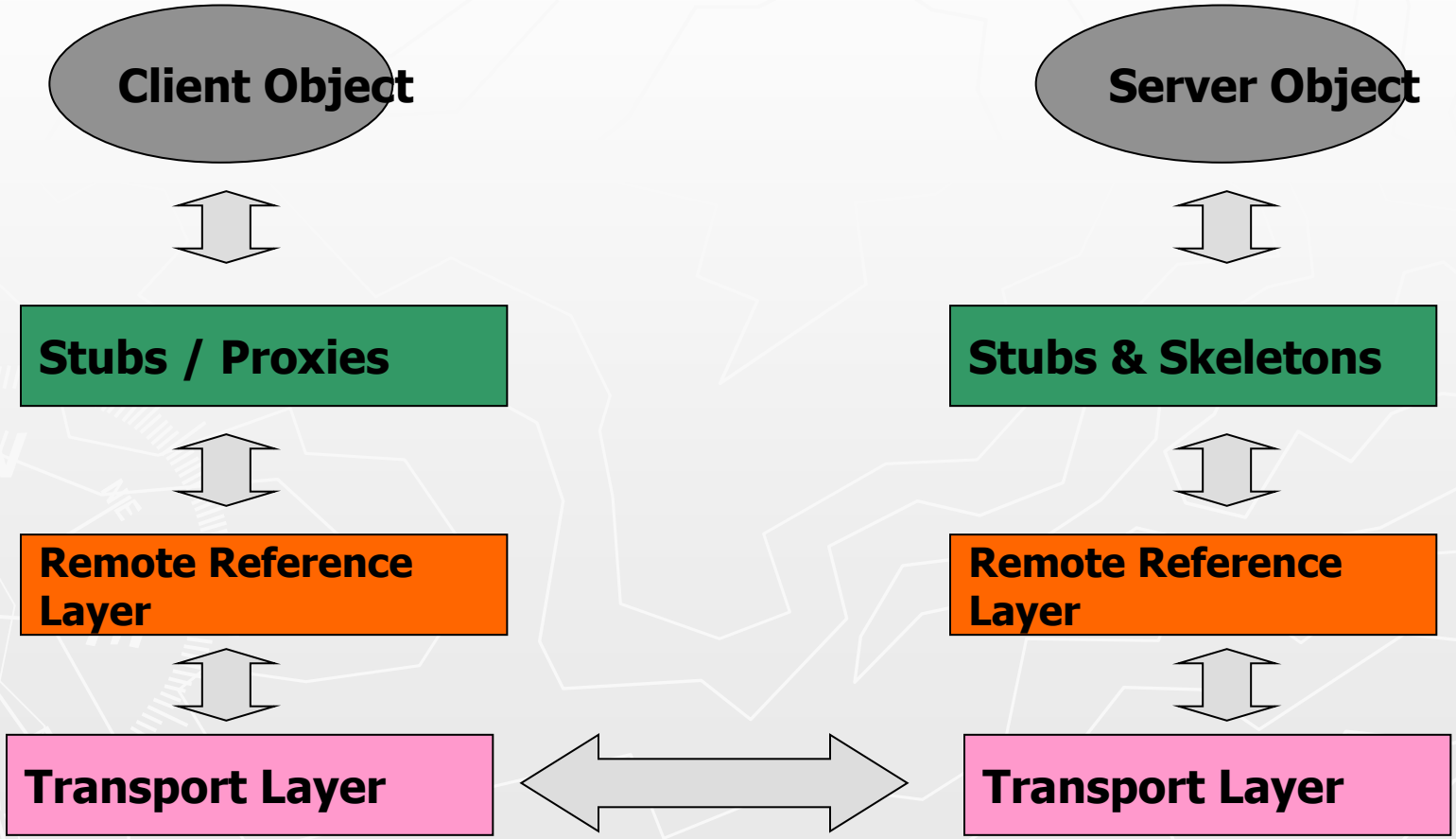
  java CalculatorServer

  java CalculatorClient

RMI Architecture Layers

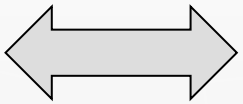
- the "invisible parts" of RMI technology
- 3 abstraction layers
  - the Stub and Skeleton layer
  - the Remote Reference layer
  - the Transport layer

RMI Architecture Layers



The Stub and Skeleton Layer

- the interface between an RMI application and the RMI system:

RMI client  RMI server

- the stub: a client-side proxy (i.e. an object that acts on behalf of the remote server object)
  - intercept remote method calls made by a client
  - redirect the calls to a remote RMI service (via the Remote Reference layer)

The Stub and Skeleton Layer

- the stub
  - marshall remote method arguments
  - unmarshall return values
- the skeleton (a mirror image of the stub): a server-side proxy
  - unmarshall remote method arguments
  - direct the method calls to the actual server objects
  - marshall return values and exceptions back to the client-side stub (via the Remote Reference layer)

The Remote Reference Layer

- connect client objects to remote objects by a one-to-one connection link
  - Java supports Remote Object Activation
- support the invocation semantics of the RMI connection

The Transport Layer

- make stream-based network connections over TCP/IP between the client-side JVM and the server-side JVM

The RMI Method Invocation Mechanism

1. The RMI server object is registered with an RMIRegistry server (that operates at the transport layer).
2. The RMI client finds the remote server object through the RMIRegistry (at the remote reference layer) and receives a **stub object** (on behalf of the remote object).

The RMI Method Invocation Mechanism

3. The RMI client invokes methods on a remote object. These invocations are handled by the stub object.
4. The remote method arguments are marshalled and sent across the network.
5. The remote method call is initiated through the stub object.
6. The remote method call is dispatched to the server-side skeleton object.

The RMI Method Invocation Mechanism

7. The remote method arguments are unmarshalled on the server side.
8. The remote method call is invoked on the actual server-side object.
9. Return values are marshalled by the server-side skeleton object.
10. Return values are sent to the client-side stub object.
11. Return values are unmarshalled by the client-side stub object and passed upward to the caller.

Parameters in RMI

- Question:

  How does RMI transfer parameters (and return values) between the client JVM and the server JVM?

- Answer: Use object serialization! **

** for a Remote object

- a remote reference for the object is generated, marshalled and sent over the network (i.e. a stub object)
- the actual Remote object is NOT copied

FAQ

1. Q: Can we find out all the service names that have been registered with the RMI Registry?

   A: Yes. java.rmi.Naming.list("odin.senecac.on.ca")

2. Q: Does RMI have built-in synchronization mechanism for remote method calls?

   A: No. One has to declare those methods as synchronized.

FAQ

3. Q: Can we use one RMI Registry to run
   two RMI servers?

   A: Yes. Put all the server code in one subdirectory.

About Workshop 7

- How to create a runnable jar file, e.g. ws7server.jar
  1. Create text file manifest.txt with the content:

     Main-Class: SocketServer

     (followed by a new line)
  2. Compile SocketServer.java
  3. Compress to jar file

     jar cvfm ws7server.jar manifest.txt SocketServer.class

     Or

     jar cvfm ws7server.jar manifest.txt *.class
  4. Run:

     java –jar ws7server.jar

Workshop for This Week

- Workshop 9

Resourceful Link

- Java Tutorial on RMI
- RMI System Overview
  http://docs.oracle.com/javase/6/docs/platform/rmi/spec/rmi-arch2.html
- Tutorials & Code Camps (Sun Developer Network - SDN)
  jGuru: Remote Method Invocation (RMI)
  http://www.dse.disco.unimib.it/ds/extra/rmiTutorial.pdf

Thank You!