core WEB programming

Remote Method Invocation

© 2001-2003 Marty Hall, Larry Brown http://www.corewebprogramming.com

Agenda

- Steps to build and RMI application
- Running and compiling an RMI program
- Eample: Retrieving a String remotely
- Example: Performing numerical integration remotely
- Enterprise RMI configuration
- RMI Applets

RMI: Remote Method Invocation

- **Idea**
  - Distribute objects across different machines to take advantage of hardware and dedicated software
  - Developer builds network service and installs it on specified machine
  - User requests an instance of a class using URL syntax
  - User uses object as though it were a regular, local object
    - Network connections happen automatically behind the scenes
    - Java "serialization" lets you pass complex data structures over the network without writing code to parse and reconstruct them

RMI Operations

- **Stub Operation**
  - Package identifier of remote object
  - Package method identifier
  - Marshall parameters
  - Send package to server skeleton

- **Skeleton Operation**
  - Unmarshall Parameters
  - Calls return value or exception
  - Marshall method return
  - Send package to client stub

RMI Details

1. Starting: Build Four Required Classes

a. An interface for the remote object
   - Used by both the client and the server

b. The RMI client
   - This will look up the object on the remote server, cast it to the type of the interface from Step 1, then use it like a local object.
   - Note that as long as there is a "live" reference to the remote object, an open network connection is maintained. The connection will be automatically closed when the remote object is garbage collected on the client.

c. The object implementation
   - This object needs to implement the interface of Step a, and will be used by the server

d. The RMI server
   - This will create an instance of the object from Step c and register it with a particular URL

RMI Details, cont.

2. Compile and Run the System

a. Compile client and server.
   - Compiles the remote object interface and implementation automatically

b. Generate the client stub and the server skeleton
   - Use the `rmic` compiler on the remote object implementation for this.
     - The client system will need the client class, the interface class, and the client stub class
     - If the client is an applet, these three classes must be available from the applet's home machine
     - The server system will need the server class, the remote object interface and implementation, and the server skeleton class

RMI Details, cont.

2. Compile and Run the System, cont.

c. Start the RMI registry
   - This only needs to be done once, not for each remote object
   - The current version of RMI requires this registry to be running on the same system as server

d. Start the server
   - This step must be on the same machine as the registry of step c

e. Start the client
   - This step can be done on an arbitrary machine

A Very Simple RMI Example: The Four Required Classes

1. The Interface for the Remote Object

- The interface should extend `java.rmi.Remote`, and all its methods should throw `java.rmi.RemoteException`

```
import java.rmi.*;

/** The RMI client will use this interface directly.
 *  The RMI server will make a real remote object that
 *  implements this, then register an instance of it
 *  with some URL.
 */

public interface Rem extends Remote {
  public String getMessage() throws RemoteException;
}
```

Simple Example, Required Classes, cont.

2. The RMI Client

- Look up the object from the host using `Naming.lookup`, cast it to the appropriate type, then use it like a local object

```
import java.rmi.*; // For Naming, RemoteException, etc.
import java.net.*; // For MalformedURLException
import java.io.*;  // For Serializable interface

public class RemClient {
  public static void main(String[] args) {
    try {
      String host = (args.length > 0) ? args[0] : "localhost";
      Rem remObject = (Rem)Naming.lookup("rmi://" + host + "/Rem");
      System.out.println(remObject.getMessage());
    } catch(RemoteException re) {
      System.out.println("RemoteException: " + re);
    } catch(NotBoundException nbe) {
      System.out.println("NotBoundException: " + nbe);
    } catch(MalformedURLException mfe) {
      System.out.println("MalformedURLException: " + mfe);
    }
  }
}
```

Simple Example, Required Classes, cont.

3. The Remote Object Implementation

- This class must extend UnicastRemoteObject and implement the remote object interface defined earlier
- The constructor should throw RemoteException

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

public class RemImpl extends UnicastRemoteObject
                     implements Rem {
  public RemImpl() throws RemoteException {}

  public String getMessage() throws RemoteException {
    return("Here is a remote message.");
  }
}
```

Simple Example, Required Classes, cont.

4. The RMI Server

- The server builds an object and register it with a particular URL
- Use `Naming.rebind` (replace any previous bindings) or `Naming.bind` (throw AlreadyBoundException if a previous binding exists)

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

public class RemServer {
  public static void main(String[] args) {
    try {
      RemImpl localObject = new RemImpl();
      Naming.rebind("rmi:///Rem", localObject);
    } catch(RemoteException re) {
      System.out.println("RemoteException: " + re);
    } catch(MalformedURLException mfe) {
      System.out.println("MalformedURLException: " + mfe);
    }
  }
}
```

Simple Example: Compiling and Running the System

1. **Compile the Client and the Server**

```
Prompt> javac RemClient.java
```

   - This compiles the `Rem` interface automatically

```
Prompt> javac RemServer.java
```

   - This compiles the `RemImpl` object implementation automatically

2. **Generate the Client Stub and Server Skeleton**

```
Prompt> rmic RemImpl
```

   - This builds `RemImpl_Stub.class` and `RemImpl_Skeleton.class`
   - The client machine needs `Rem.class`, `RemClient.class`, and `RemImpl_Stub.class`
   - The server machine needs `Rem.class`, `RemImpl.class`, `RemServer.class`, and `RemImpl_Skeleton.class`

Simple Example: Compiling and Running the System, cont.

3. **Start the RMI Registry**

```
Server> rmiregistry
```

   - On Unix systems you would probably add "&" to put the registry process in the background
   - You can also specify a port number; if omitted, port 1099 is used

4. **Start the Server**

```
Server> java RemServer
```

   - Again, on Unix systems you would probably add "&" to put the process in the background

5. **Start the Client**

```
Client> java RemClient hostname
Here is a remote message.
```

A Better RMI Example, Numerical Integration

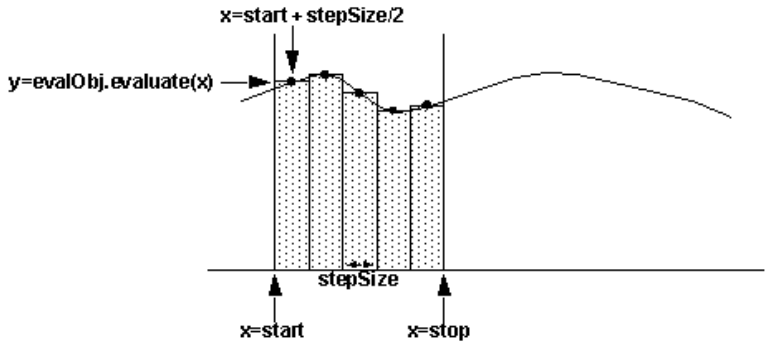
1. **Simple Iterative Program to Calculate Sums:**

$$\sum_{x=start}^{stop} f(x)$$

2. **Use to Approximate Numeric Integrals of the Form:**

$$\int_{start}^{stop} f(x)dx$$

3. **MidPoint Rule:**



4. **Motivation for RMI**
   - Since smaller rectangles typically give better results, this can often be very cpu-intensive
   - RMI can make it available on a fast floating-point box

Numerical Integration, Example, cont.

```
public class Integral {
  /** Returns the sum of f(x) from x=start to x=stop, where the function f
   *  is defined by the evaluate method of the Evaluatable object.
   */

  public static double sum(double start, double stop,
                           double stepSize,
                           Evaluatable evalObj) {
    double sum = 0.0, current = start;
    while (current <= stop) {
      sum += evalObj.evaluate(current);
      current += stepSize;
    }
    return(sum);
  }

  public static double integrate(double start, double stop,
                                 int numSteps,
                                 Evaluatable evalObj) {
    double stepSize = (stop - start) / (double)numSteps;
    start = start + stepSize / 2.0;
    return(stepSize * sum(start, stop, stepSize, evalObj));
  }
}
```

Numerical Integration, Example, cont.

```
/** An interface for evaluating functions y = f(x) at a specific
 *  value. Both x and y are double-precision floating-point
 *  numbers.
 */

public interface Evaluatable {
  public double evaluate(double value);
}
```

Integration Example: Four Required Classes

1. The RemoteIntegral Interface

- The interface shared by the client and server

```
import java.rmi.*;

public interface RemoteIntegral extends Remote {

  public double sum(double start, double stop, double stepSize,
                    Evaluatable evalObj)
    throws RemoteException;

  public double integrate(double start, double stop,
                          int numSteps, Evaluatable evalObj)
    throws RemoteException;
```

Integration Example: Four Required Classes, cont.

2. The Remote Integral Client

- Sends the `RemoteIntegral` an `Evaluatable` to integrate

```
public class RemoteIntegralClient {
  public static void main(String[] args) {
    try {
      String host = (args.length > 0) ? args[0] : "localhost";
      RemoteIntegral remoteIntegral =
        (RemoteIntegral)Naming.lookup("rmi://" + host + "/RemoteIntegral");
      for(int steps=10; steps<=10000; steps*=10) {
        System.out.println("Approximated with " + steps + " steps:" +
          "\n  Integral from 0 to pi of sin(x)=" +
          remoteIntegral.integrate(0.0, Math.PI, steps, new Sin()));
      }
      System.out.println("'Correct' answer using Math library:" +
                         "\n  Integral from 0 to pi of sin(x)=" +
                         (-Math.cos(Math.PI) - -Math.cos(0.0)));
    } catch(RemoteException re) {
      System.out.println("RemoteException: " + re);
    } catch(NotBoundException nbe) {
      System.out.println("NotBoundException: " + nbe);
    } catch(MalformedURLException mfe) {
      System.out.println("MalformedURLException: " + mfe);
    }
  }
}
```

Integration Example: Four Required Classes, cont.

2. The Remote Integral Client, cont.

- Evaluatable Sin function

```
import java.io.Serializable;

class Sin implements Evaluatable, Serializable {
  public double evaluate(double val) {
    return(Math.sin(val));
  }

  public String toString() {
    return("Sin");
  }
}
```

Integration Example: Four Required Classes, cont.

3. The Remote Integral Implementation

- Remote object that calculates the integral value

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

public class RemoteIntegralImpl extends UnicastRemoteObject
                                implements RemoteIntegral {

  public RemoteIntegralImpl() throws RemoteException {}

  public double sum(double start, double stop, double stepSize,
                    Evaluatable evalObj) {
    return(Integral.sum(start, stop, stepSize, evalObj));
  }


  public double integrate(double start, double stop, int numSteps,
                          Evaluatable evalObj) {
    return(Integral.integrate(start, stop, numSteps, evalObj));
  }
}
```

Integration Example: Four Required Classes, cont.

4. The Remote Integral Server

- Creates the `RemoteIntegral` and registers it with the rmi registry

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

public class RemoteIntegralServer {
  public static void main(String[] args) {
    try {
      RemoteIntegralImpl integral =  new RemoteIntegralImpl();
      Naming.rebind("rmi:///RemoteIntegral", integral);
    } catch(RemoteException re) {
      System.out.println("RemoteException: " + re);
    } catch(MalformedURLException mfe) {
      System.out.println("MalformedURLException: " + mfe);
    }
  }
}
```

Integration Example: Compiling and Running the System

1. **Compile the Client and the Server**

```
Prompt> javac RemoteIntegralClient.java
Prompt> javac RemoteIntegralServer.java
```

2. **Generate the Client Stub and Server Skeleton**

```
Prompt> rmic -v1.2 RemoteIntegralImpl
```

- Client requires: `RemoteIntegral.class`, `RemoteIntegralClient.class` and `RemoteIntegralImpl_Stub.class`
- Server requires: `RemoteIntegral.class`, `RemoteIntegralImpl.class`, and `RemoteIntegralServer.class`
- If the server and client are both running JDK 1.1, use the `-v1.1` switch to produce the RMI 1.1 skeleton stub, `RemoteIntegralImpl_Skeleton`, required by the server

Integral Example: Compiling and Running the System, cont.

3. **Start the RMI Registry**

```
Prompt> rmiregistry
```

4. **Start the Server**

```
Prompt> java RemoteIntegralServer
```

5. **Start the Client**

```
Prompt> java RemoteIntegralClient
Approximated with 10 steps:
 Integral from 0 to pi of sin(x)=2.008248407907945
Approximated with 100 steps:
  Integral from 0 to pi of sin(x)=2.000082249070987 7
Approximated with 1000 steps:
  Integral from 0 to pi of sin(x)=2.000000822467298 3
Approximated with 10000 steps:
  Integral from 0 to pi of sin(x)=2.000000008224362
    ...
```

Enterprise RMI Configuration

- **Stub files need to be placed on a HTTP server for downloading**
  - In Java 2, the RMI 1.2 protocol does not require the skeleton
- **Client must install an RMISecurityManager to load the RMI classes remotely**

```
System.setSecurityManager(new RMISecurityManager());
```

- **Client requires a policy file to connect to registry and HTTP server**

Policy File for Client

```
grant {
  // rmihost - RMI registry and the server
  // webhost - HTTP server for stub classes
  permission java.net.SocketPermission
    "rmihost:1024-65535", "connect";
  permission java.net.SocketPermission
    "webhost:80", "connect";
};
```

- Need to grant permission to ports 1024-65535 on the server
  - The server communicates with the rmiregistry (and client) on a randomly selected source port
- Alternatively, can set policies in `java.policy` located in `JAVA_HOME/lib/security/`

Enterprise RMI, Remote Integral, Example

```
public class RemoteIntegralClient2 {

    public static void main(String[] args) {
    try {
      System.setSecurityManager(new RMISecurityManager());
      String host = (args.length > 0) ? args[0] : "localhost";
      RemoteIntegral remoteIntegral =
        (RemoteIntegral)Naming.lookup("rmi://" + host +
                                      "/RemoteIntegral");
      for(int steps=10; steps<=10000; steps*=10) {
        System.out.println
          ("Approximated with " + steps + " steps:" +
          "\n  Integral from 0 to pi of sin(x)=" +
          remoteIntegral.integrate(0.0, Math.PI,
                                   steps, new Sin()));
      }
      ...
    } catch(RemoteException re) {
      System.out.println("RemoteException: " + re);
    }
     ...
  }
```

Enterprise Example: Compiling and Running the System

1. **Compile the Client and the Server**

```
Prompt> javac RemoteIntegralClient2.java
Prompt> javac RemeteIntegralServer.java
```

2. **Generate the Client Stub and Server Skeleton**

```
Prompt> rmic -v1.2 RemoteIntegralImpl
```

3. **Place the files on the correct machines**

Client	*Server*	*HTTP Server*
RemoteIntegralClient2	RemoteIntegralServer	RemoteIntegralImpl_Stub
RemoteIntegral	RemoteIntegralImpl	RemoteIntegral
Evaluatable	RemoteIntegralImpl_Stub	Evaluatable
Sin	RemoteIntegral	
Cos	Integral	
Quadratic	Evaluatable	
	Sin	
	Cos	
	Quadratic	

Enterprise Example: Compiling and Running the System, cont.

4. Start the HTTP Server

- Place `RemoteIntegral_Stub.class`, `RemoteIntegeral.class`, and `Evaluatable.class` on an HTTP server
- Verify that you can access the files through a browser

5. Start the RMI Registry

```
Server> /somedirectory/rmiregistry
```

- Make sure that none of the class files are in the directory in which you started the registry or available through the classpath

6. Start the Server

```
Server> java -Djava.rmi.server.codebase=http://webhost/rmi/
              RemoteIntegralServer
```

- Server must be started on same host as rmiregistry

Enterprise Example: Compiling and Running the System, cont.

7. Start the Client

```
Client> java -Djava.security.policy=rmiclient.policy
             RemoteIntegralClient2 rmihost

Approximated with 10 steps:
 Integral from 0 to pi of sin(x)=2.008248407907974 5
Approximated with 100 steps:
  Integral from 0 to pi of sin(x)=2.000082249070987 7
  ...
```

- The rmihost is where server in which the `rmiregistry` was started

An RMI Applet

- **Applet does not require a RMI Security Manager**
- **Applet can only access server in which class files were loaded**
  - RMI Registry and remote object server must be the same HTTP host in which the applet was loaded
- **RMI 1.1 stub protocol not properly supported in IE**
- **RMI 1.2 stub protocol require Java Plug-In or Netscape 6**

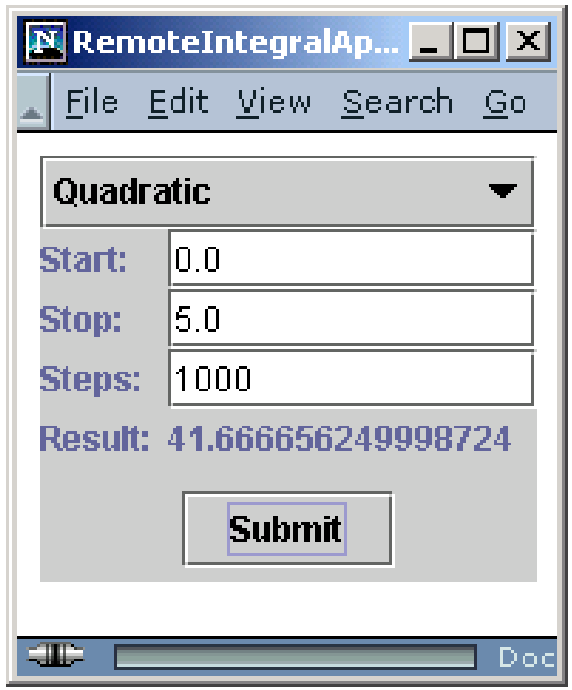
RMI Applet, Example

```
...
import javax.swing.*;

public class RemoteIntegralApplet extends JApplet
                                  implements ActionListener {
  private Evaluatable[] shapes;
  private RemoteIntegral remoteIntegral;
  private JLabel result;
  private JTextField startInput, stopInput, stepInput;
  private JComboBox combo;

  public void init() {
    String host = getCodeBase().getHost();
    try {
      remoteIntegral =
        (RemoteIntegral)Naming.lookup("rmi://" + host +
                                      "/RemoteIntegral");
    } catch(RemoteException re) {
      reportError("RemoteException: " + re);
    }
    ...
```

RMI Applet, Example

```
...

public void actionPerformed(ActionEvent event) {
  try {
    int steps = Integer.parseInt(stepInput.getText());
    double start = Double.parseDouble(startInput.getText());
    double stop = Double.parseDouble(stopInput.getText());
    showStatus("Calculating ...");
    Evaluatable shape = (Evaluatable)combo.getSelectedItem();
    double area = remoteIntegral.integrate(start, stop,
                                           steps, shape);
    result.setText(Double.toString(area));
    showStatus("");
  } catch(NumberFormatException nfe) {
    reportError("Bad input: " + nfe);
  } catch(RemoteException re) {
    reportError("RemoteException: " + re);
  }
}
}
```

RMI Applet, Result



Applet that communicates to a
remote object through RMI in Netscape 6

Summary

- RMI is a pure Java-based protocol for communicating with remote objects
- Register (bind) and look-up remote objects in a registry
- Java 2 no longer requires the skeleton class needed with the RMI 1.1 protocol
- Enterprise RMI configuration requires a RMI Security Manager and client policy file for permissions

core WEB programming

Questions?

© 2001-2003 Marty Hall, Larry Brown http://www.corewebprogramming.com