

CmpE 473 Internet Programming

Pinar Yolum
pyolum@cmpe.boun.edu.tr

Department of
Computer Engineering
Boğaziçi University

Chapter 4 RMI

Examples from java.sun.com

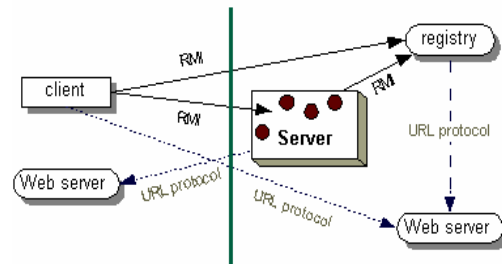
RMI

- Remote Method Invocation
 - Allows an object to call methods of a remote object
 - Both objects should run in JVM
 - Used sockets based transport for communication by default
- Three steps
 - Locate the object
 - Communicate with the object
 - Load bytecodes of transferred objects

Spring 2005— Pinar Yolum

3

RMI



Spring 2005— Pinar Yolum

4

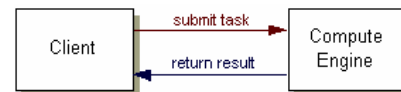
Remote Object

- Remote objects
 - Whose methods can be called by objects on other machines
 - Implement remote interface
 - Extends `java.rmi.Remote`
 - All methods in the interface must throw `java.rmi.RemoteException` + other specific exceptions
- Remote interface
 - Contains methods that can be called remotely by the client

Spring 2005— Pinar Yolum

5

RMI Server



```
package compute;  
  
import java.rmi.Remote;  
import java.rmi.RemoteException;  
public interface Compute extends Remote {  
    public Object executeTask (Task t) throws RemoteException;  
}
```

Spring 2005— Pinar Yolum

6

RMI Server

```
package compute;
import java.io.Serializable;
public interface Task extends Serializable {
    public Object execute();
}
```

- Any object that implements Task can be executed on the server
- Primitive return types should be wrapped

Spring 2005— Pinar Yolum

7

Passing Objects

- Local object
 - Passed by value
 - The object must be serializable
 - Default serialization
 - Copies all fields (except static or transients)
 - Can be overridden
- Remote object
 - Passed by reference
 - Stub: Representative (or proxy) for the remote object
 - Stub implements the same set of remote interfaces as the remote object
 - Stub can be cast to any of the remote interfaces implemented by the remote object

Spring 2005— Pinar Yolum

8

Finding classes

- Downloading bytecodes
 - When an object is sent, its class location (URL) is sent in the class description
 - Useful if the client doesn't have the class for an object
 - If the object is remote, only download a stub

Spring 2005— Pinar Yolum

9

RMI Server

```
package engine;
import java.rmi.*; import java.rmi.server.*; import compute.*;

public class ComputeEngine extends UnicastRemoteObject implements Compute {
    public ComputeEngine() throws RemoteException {
        super();
    }
    public Object executeTask(Task t) {
        return t.execute();
    }
}

public static void main(String[] args) {
    if (System.getSecurityManager() == null) {
        System.setSecurityManager(new RMISecurityManager());
    }
    String name = "/host/Compute";
    try {
        Compute engine = new ComputeEngine();
        Naming.rebind(name, engine);
        System.out.println("ComputeEngine bound");
    } catch (Exception e) {
        System.err.println("ComputeEngine exception: " + e.getMessage());
        e.printStackTrace();
    }
}
```

Spring 2005— Pinar Yolum

10

RMI Server

- UnicastRemoteObject
 - java.rmi.server.UnicastRemoteObject
 - Implements Remote and Serializable
 - Implementations for equals, toString, hashCode
- exportObject(Remote obj, int port)
 - Make RMI runtime aware of the object
 - Supports point-to-point remote communication
 - Load a stub class
 - Construct an instance of the stub

Spring 2005— Pinar Yolum

11

Constructing a Stub

- Naming creates a stub and stores it in the registry
- Identify the root class (getClass())
 - If the remote object implements an interface that extends Remote, then root class is remote object's class
 - Otherwise, the highest superclass that implements an interface that extends Remote is the root class
- Name of stub=concat(binary_name_root_class, "_Stub")
- Classloader of root class will load the stub to JVM
- Stub must extend RemoteStub and must have a constructor whose one parameter is a RemoteRef
 - RemoteRef ref=engine.getRef();
- Construct the class with RemoteRef
 - ComputeEngine_Stub stub=new ComputeEngine_Stub(ref);

Spring 2005— Pinar Yolum

12

Stub-to-Remote

- When the client calls a method on the stub
 - Stub connects the remote JVM
 - Writes (marshals) parameters to the remote JVM
 - Reads (unmarshals) the result (return value, warning, exception)
 - Gives the result to the client

Spring 2005— Pinar Yolum

13

Security Managers

- Checks accesses to
 - System resources (e.g., local file system)
 - Privileged operations
 - Based on policy implemented
 - Methods
 - Are of type `checkXXX`; (`checkAccess`; `checkConnect`)
 - Ex: `checkWrite(String filename)`
 - Ex: `checkConnect(String host, int port)`
 - Throw an exception if there is a problem
- All RMI programs must install one
- `java.rmi.RMISecurityManager`
 - Extends `java.lang.SecurityManager`

Spring 2005— Pinar Yolum

14

Permissions

- Granted to codes for access
- Create a permission

```
perm = new java.io.FilePermission("/tmp/abc", "read");
```
- Grant a permission

```
grant codeBase "file:/home/sysadmin/" {
    permission java.io.FilePermission "/tmp/abc", "read";
};
```
- Stored in a `.policy` file
- Difficult to construct by hand (policytool)
- Check out the default policy
`$JAVA_HOME/lib/security/java.policy`
- `java -Djava.security.policy=policy-file MyClass`

Spring 2005— Pinar Yolum

15

Remote Object

- `Compute engine = new ComputeEngine();`
 - `ComputeEngine` remote object is ready to accept calls
 - The accessible part is the `Compute` interface
- `String name = "//host/Compute";`
 - Host: Host name where the registry is
 - `Computer` is the name of the remote object

Spring 2005— Pinar Yolum

16

Registry

- `Naming.rebind(name, engine);`
 - Adds it to the registry (association between name and object)
 - Name: String formatted as a URL
 - Default port: 1099
- Clients can talk to the server only over the registry
- Keeps track of registered servers
- Wakes up the server if there is an incoming request

Spring 2005— Pinar Yolum

17

RMI Client

```
package client;
import compute.*; import java.math.*;

public class Pi implements Task { /** constants used in pi computation */
    private int digits;
    /** * Construct a task to calculate pi to the specified * precision. */
    public Pi (int digits) {
        this.digits = digits; } /** * Calculate pi. */
    public Object execute() {
        return computePi(digits);
    }
    public static BigDecimal computePi(int digits) {
        BigDecimal pi = ...;
        return pi.setScale(digits, BigDecimal.ROUND_HALF_UP); }
}
```

Spring 2005— Pinar Yolum

18

RMI Client

```

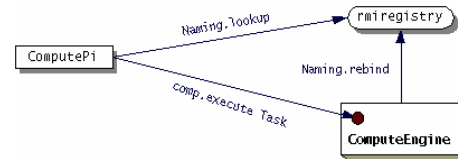
package client;
import java.rmi.*; import java.math.*; import compute.*;

public class ComputePi {

    public static void main(String args[]) {
        if (System.getSecurityManager() == null) {
            System.setSecurityManager(new RMISecurityManager());
        }
        try {
            String name = "/" + args[0] + "/Compute";
            Compute comp = (Compute) Naming.lookup(name);
            Pi task = new Pi(Integer.parseInt(args[1]));
            BigDecimal pi = (BigDecimal) (comp.executeTask(task));
            System.out.println(pi);
        }
        catch (Exception e) {
            System.err.println("ComputePi exception: " + e.getMessage());
            e.printStackTrace();
        }
    }
}
    
```

Spring 2005— Pinar Yolum 19

RMI Client



```

package compute;
public interface Task extends java.io.Serializable {
    Object execute();
}
    
```

Spring 2005— Pinar Yolum 20

Compiling

- Build a jar file of the interfaces
 - javac compute\Compute.java
 - javac compute\Task.java
 - jar cvf compute.jar compute*.class
 - Build the server side files (And Make them accessible over a Web server)
 - javac engine\ComputeEngine.java
 - rmic -d . engine.ComputeEngine
 - mkdir c:\home\ann\public_html\classes\engine
 - cp engine\ComputeEngine_*.class c:\home\ann\public_html\classes\engine
 - cd c:\home\ann\public_html\classes
 - jar xvf compute.jar
 - Build the client side files
 - set CLASSPATH=c:\home\jones\src;c:\home\jones\public_html\classes\compute.jar
 - cd c:\home\jones\src
 - javac client\ComputePi.java j
 - javac -d c:\home\jones\public_html\classes client\Pi.java
- Spring 2005— Pinar Yolum 21

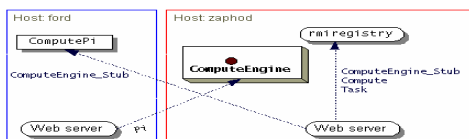
Running

- Unset CLASSPATH (no java classes should be reachable)
 - Run RMI registry first
 - start rmiregistry (optional port number)
 - Set classpath
 - It should see both compute.jar and remote object implementation
 - Start the server


```

java-
Djava.rmi.server.codebase=file:/c:/home/ann/public_html/classes/
-Djava.rmi.server.hostname=zaphod.east.sun.com
-Djava.security.policy=java.policy engine.ComputeEngine
            
```
- Spring 2005— Pinar Yolum 22

Running



- Start the client
 - set CLASSPATH c:\home\jones\src;c:\home\jones\public_html\classes\compute.jar
 - java -Djava.rmi.server.codebase=file:/c:/home\jones\public_html\classes/ -Djava.security.policy=java.policy client.ComputePi zaphod.east.sun.com 20
- Spring 2005— Pinar Yolum 23

Examples

- <http://javaalmanac.com/egs/java.rmi/pkg.html>
 - http://dev2dev.bea.com/codelibrary/code/examples_rmi.jsp
- Spring 2005— Pinar Yolum 24