COMP2207
Programming with Distributed Objects

Professor Timothy Norman

Department of Electronics and Computer Science, University of Southampton
T.J.Norman@soton.ac.uk

Academic Year 2017/18
Programming with Distributed Objects

- **Server**
  - Creates a set of remote objects
  - Makes references to these objects accessible
  - Waits for client requests to invoke methods on these objects

- **Client**
  - Obtains remote references
  - Invokes methods on remote objects

- **Java.rmi provides**
  - Java API to support distributed object programming
  - Registry services for advertising and look-up of remote object references
  - Distributed garbage collection
Distributed Object Application

- **Lookup/locate**
  - RMI naming service, implemented as `rmiregistry` application
  - Remote references may be passed as parameters

- **Proxies**
  - Called a stub manages communication with server on client side
  - Method invocation appears identical to local method calls

- **Class loading**
  - Java objects are instantiated from compiled classes
    - They must be loaded into the local JVM
  - RMI allows class loading from remote sites
  - Needs serialization and transmission of class specifications
A Client

```java
class ShoutClient {
    public static void main(String[] args) {
        // Check args
        String host = args[0];
        try {
            Registry reg = LocateRegistry.getRegistry(host);
            ShoutInterface sb = (ShoutInterface) reg.lookup("Shout");
            BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
            System.out.print("Enter message:");
            System.out.println(sb.shout(in.readLine()));
        } catch (Exception e) {
            // Do something sensible
        }
    }
}
```
Registration, dynamic loading & invocation

- **Client Program**
  - load class

- **Web Server**
  - load class

- **Server**
  - Object
  - register
  - rmiregistry

- **method call**

- **look up**
Two fundamental concepts at the heart of the distributed object model

- **Remote object reference**
  - Methods on a remote object can only be invoked if the calling object has its **remote reference**
  - This reference is provided by a **registry** (or “white pages”)

- **Remote interface**
  - Every remote object has a **remote interface**
  - Specifies what **methods** can be invoked remotely
  - The class specification, from which remote objects are instantiated, must **implement** this interface
Outline

1. The Remote Interface
Remote Interfaces

```java
package comp2207.shout;

import java.rmi.Remote;
import java.rmi.RemoteException;

public interface ShoutInterface extends Remote {
    public String shout(String s) throws RemoteException;
}
```

- This method will be available remotely if an object implements this interface.
- Only these methods are available remotely.
- Must extend Remote.
- Any implementation can be a remote object.
The Stub

**Stub** = *Proxy for a remote object*

- A remote object is treated *differently* when it is passed from one *Java Virtual Machine* to another
  - A *remote reference* is passed to the receiving JVM (*not a copy*)
  - A *stub* is created by the *infrastructure* within the client’s JVM
  - Since Java 5.0: the *javac* compiler adds code for dynamic creation of stub *at runtime*
  - The *stub* implements the same methods as on the *remote object*, and only those defined on the *remote interface*
RMI Implementation

- RMI middleware consists of
  - Stub
  - Dispatcher (using Java reflection)
package comp2207.shout;

import java.rmi.RemoteException;

public class ShoutImpl implements ShoutInterface {
    public ShoutImpl() throws RemoteException {
    }

    public String shout(String s) throws RemoteException {
        return s.toUpperCase();
    }
}

- **Implements** the remote interface
- Requires an appropriate **constructor**
- Must provide behaviour to all methods on remote interface
Deployment of Remote Objects

To deploy our remote application, we need to

- Generate instances of remote objects to run on server
- Export them and obtain a stub
- Register them with the local registry
The Registry

- Runs of the **same machine** as the server
- On a separate **JVM**
- Listens on a specific **port** for registration/lookup (default is **1099**)
- Maintains a table mapping (“binding”) **URLs** to **references** of remote objects

**Methods:**

1. `void rebind(String url, Remote obj)`
2. `void bind(String url, Remote obj)`
3. `void unbind(String url, Remote obj)`
4. `Remote lookup(String url)`
5. `String[] list()`

- A **URL** is:
  `protocol://unique.computer.name:port/objectLocation/name`
- (although RMI will accept arbitrary strings)
Remote Implementation

```java
package comp2207.shout;

import java.rmi.RemoteException;
import java.rmi.registry.Registry;
import java.rmi.registry.LocateRegistry;
import java.rmi.server.UnicastRemoteObject;

public class ShoutMainline {
    public static void main(String args[]) {
        try {
            ShoutImpl serv = new ShoutImpl();
            ShoutInterface stub = (ShoutInterface) UnicastRemoteObject.exportObject(serv, 0);
            Registry reg = LocateRegistry.getRegistry();
            reg.rebind("Shout", stub);
        } catch (RemoteException e) {
            System.err.println(e.getMessage());
        }
    }
}
```

- Export to *anonymous port*; “Shout” rather than URL
Compiling & Running the Shout Example

https://secure.ecs.soton.ac.uk/notes/comp2207/shout.zip

- **Extract** the archive
  - Will generate directory `comp2207/shout` and you will find the following files in that directory:
    - `ShoutClient.java` — the client code
    - `ShoutInterface.java` — the remote interface
    - `ShoutImpl.java` — the implementation of the remote interface
    - `ShoutMainline.java` — the server mainline code

- **Compile** the code
  - Navigate to `comp2207/shout` and invoke the normal java compiler:
    `javac *.java`

- **Run** the system
  - Navigate to the place where you extracted the archive
  - Run `rmiregistry`
  - Run the server mainline:
    `java comp2207.shout.ShoutMainline`
  - Run the client:
    `java comp2207.shout.ShoutClient localhost`
Dynamic Loading from Web?

- In the previous slide, we had this set-up
  - Three JVMs on a single machine: rmiregistry; server mainline; and client
  - Stub code acquired from local CLASSPATH by registry & client
- RMI allows us to use a codebase via HTTP
- ...what are the implications of this?
Dynamic Loading from Web?

- In the previous slide, we had this set-up
  - Three JVMs on a single machine: rmiregistry; server mainline; and client
  - Stub code acquired from local CLASSPATH by registry & client
- RMI allows us to use a codebase via HTTP
- ...what are the implications of this?
- This involves downloading code from a remote server to run on your machine! — do you trust it?
- By default, a JVM can utilise code only in the local CLASSPATH
- In order to make use of dynamic class loading, we need two things:
  - A Security Manager; and
  - A sensible security policy
The Security Manager

- The security manager sets up a “sandbox”, in which code is run
- The SecurityManager class is in java.lang

```java
if (System.getSecurityManager() == null) {
    System.setSecurityManager(new SecurityManager());
}
```

- We now need to tell the JVM what the policy is
- Can be done in code:

```java
System.setProperty("java.security.policy", "mypolicy");
```

- Or via the -D switch when initiating the JVM:

```
java -Djava.security.policy=mypolicy
```
The Policy File

- In designing your policy file, you need to consider
  - Where is the code you’re using coming from (codebase)?
  - What do you want to deny and permit
  - Do you expect code to be digitally signed?

- A simple policy (stored in a file “mypolicy”):

```java
grant {
  permission java.net.SocketPermission "*:1024-65535",
      "connect,accept,resolve";
  permission java.net.SocketPermission "*:80", "connect";
};
```

- Now, we can run the client with stub classes obtained via HTTP:

```bash
java -Djava.security.policy=mypolicy
     -Djava.rmi.server.codebase=
     http://users.ecs.soton.ac.uk:80/tjn1f15/comp2207.jar
comp2207.shout.ShoutClient localhost
```