Java Programming II

Java Network II
(Distributed Objects in Java)
Contents

- Distributed Objects
- Java RMI
- Communication of Remote Object and Client
- Writing Java RMI Application
- Hello Example
- RMI Structure
- If room (RMI Callback)
Distributed Objects

- Objects that can communicate with objects on heterogeneous run-time environments
- Distribute Objects Standard Protocol – ex: JRMP
- Robust
- Reliable
- Transparent

Distributed Objects Technology
- Multi-Platform
- Transparent access to distributed objects
- Language Neutral: RMI, CORBA, DCOM
Java Remote Method Invocation (RMI)

- Can use objects on remote different run-time environments as like objects on a local run-time environment.

- Abstraction of low-level network code on distributed network to provide developers an environment where they focus on their application development.
Introduction to RMI

- Distributed Processing on Network
- Define of Remote Interface
- Object Serialization
- java.rmi and java.rmi.server
- Create Stub and Skeleton
Communication of Remote Object and Client

- **Client** interacts with **Stub** in the **RMI Client Application**.
- **Remote Object** interacts with **Skeleton** in the **RMI Server Application**.
- The interaction occurs via the **Network**.
Writing Java RMI Application

- **Writing RMI Application**
  - Definition of Remote Interface
  - Definition of Remote Implementation Class
  - Write RMI Server Application
  - Write Client Application

- **Compile and Run the Application**
  - Compilation of the Implementation Class
  - Creation of Stub and Skeleton using “rmic” command
  - Compilation of the Server Application
  - Run the RMI Registry and Start the Server Program
  - Compilation of the Client Program
  - Run the Client
import java.rmi.Remote;
import java.rmi.RemoteException;

public interface Hello extends Remote {
    String sayHello() throws RemoteException;
}
import java.rmi.Naming;
import java.rmi.RemoteException;
import java.rmi.RMISecurityManager;
import java.rmi.server.UnicastRemoteObject;

public class HelloImpl extends UnicastRemoteObject implements Hello {
    public HelloImpl() throws RemoteException {
        super();
    }
    public String sayHello() {
        return "Hello World!";
    }
    public static void main(String args[]) {
        try {
            HelloImpl obj = new HelloImpl();
            // Bind this object instance to the name "HelloServer"
            Naming.rebind("HelloServer", obj);
            System.out.println("HelloServer bound in registry");
        } catch (Exception e) {
            System.out.println("HelloImpl err: " + e.getMessage());
            e.printStackTrace();
        }
    }
}
Hello Example : RMI

A Client Application

```java
import java.rmi.Naming;
import java.rmi.RemoteException;

public class HelloClient {

    public static void main(String args[]) {
        String message = "Hello: This is my test message";

        // "obj" is the identifier that we'll use to refer
        // to the remote object that implements the "Hello"
        // interface
        Hello obj = null;

        try {
            obj = (Hello)Naming.lookup("//" + "/HelloServer");
            message = obj.sayHello();
        } catch (Exception e) {
            System.out.println("HelloClient exception: " + e.getMessage());
            e.printStackTrace();
        }

        System.out.println("Message = " + message);
    } // end of main
} // end of HelloClient
```
Hello Example: RMI

Start Registry Server & Run Server and Client

% rmiregistry &
% java -Djava.security.policy=policy HelloImpl
% javac examples/hello/HelloClient.java
% java [-Djava.security.policy=policy] HelloClient

Please ensure there is the “policy” file in the current directory

Java Programming II
RMI Structure

◆ Protocol
  ● Java Remote Method Protocol (JRMP)
    • For distribute object environment by pure Java environment
  ● RMI/IIOP: Can communicate with CORBA

◆ JRMP
  ● Communication Mechanism between Stub and Skeleton
  ● Object Serialization: Types of parameters and return type of a remote method should follow Java serialization mechanism
Stub and Skeleton

RMI Client Application

Client

RMI Client Application

JVM

Remote Method Invocation

Data Transformation

Marshalling

Object Serialization

Network

JVM

Remote Object

Skeleton

RMI Server Application

Remote Method Invocation

Remote Method Return

Unmarshalling

Java Programming II

Remote Method Invocation

Unmarshalling
Stub and Skeleton

Stubs and Skeleton

- When RMI client invokes a remote method of a remote object, it uses stub reference of the remote object instead of remote object reference.

- For marshalling and unmarshalling of stub and skeleton, object serialization and deserialization are used.

Condition of Object for Object Serialization

- Object should implement java.io.Serialization interface
- Built-in types can be serialized basically
- Member variables should implement the Serializable interface or be declared as transient
- Static member variables can not be serialized.
JRMP Class

Marshall/Unmarshall (Object Serialization)

Managing connection to remote object

Creation of reference to remote object
Callback between Client and Server

- Implementation of callback between a client and a server through reference passing

- What is callback?
  - Usually, a server provides service objects with methods. When a client invokes a method of the server, the server executes the invocation and returns the result to the client.
  - For the callback, a client (or invoker) provides services (methods) for the server (or program to be invoked). The callback is that the server (callee) invokes the client (caller)'s methods.

- If you are interested in an example for callback to a client’s method, refer to the “/home/course/java2/codes/ByTopics/RMI/RMI-ComputeEngine”

- A Simple Callback Example
import java.rmi.Remote;
import java.rmi.RemoteException;

public interface Callback extends Remote {
    public String speakJapanese() throws RemoteException;
    public String speakEnglish() throws RemoteException;
    public String greeting(String s) throws RemoteException;
}
RMI Callback Example

RMI Server (CallbackImpl)

```java
import java.rmi.Naming;
import java.rmi.RemoteException;
import java.rmi.RMISecurityManager;
import java.rmi.server.UnicastRemoteObject;

public class CallbackImpl extends UnicastRemoteObject implements Callback {

    public CallbackImpl() throws RemoteException {
        super();
    }

    public String greeting(String lang) throws RemoteException {
        CallbackServer server = new CallbackServer();
        return server.sayHello(this, lang);
    }

    public String speakJapanese() {
        return new String("Konnichiwa!");
    }

    public String speakEnglish() {
        return new String("How are you!");
    }
}

public static void main(String args[]) {
    // Create and install a security manager
    if (System.getSecurityManager() == null) {
        System.setSecurityManager(new RMISecurityManager());
    }
    try {
        CallbackImpl obj = new CallbackImpl();
        // Bind this object instance to the name "MyCallbackServer"
        Naming.rebind("MyCallbackServer", obj);
        System.out.println("MyCallbackServer bound in registry");
    } catch (Exception e) {
        System.out.println("CallbackImpl err: " + e.getMessage());
        e.printStackTrace();
    }
}
```

Call `sayHello` with "this" for callback routine
import java.rmi.Remote;
import java.rmi.RemoteException;

public class CallbackServer {
    public String sayHello(Callback callback, String lang) throws RemoteException {
        String message = null;
        if (lang.equals("JAPANESE")) {
            message = callback.speakJapanese();
            System.out.println("In CallbackServer, " + message);
            return message;
        } else {
            message = callback.speakEnglish();
            System.out.println("In CallbackServer, " + message);
            return message;
        }
    }
}