Sistemas Distribuidos

Remote Method Invocation

Arturo Díaz Pérez

Sección de Computación
Departamento de Ingeniería Eléctrica
CINVESTAV-IPN
Av. Instituto Politécnico Nacional No. 2508
Col. San Pedro Zacatenco
México, D. F. CP 07300

Tel. (5)747 3800 Ext. 3352
e-mail: adiaz@cs.cinvestav.mx

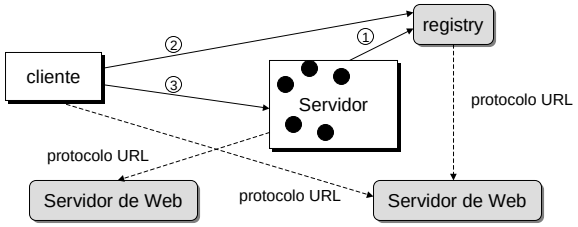
Introducción

- El sistema de *invocación de métodos remotos* de Java (Java RMI) permite que un objeto ejecutándose en una máquina virtual invoque métodos de un objeto ejecutándose en otra máquina virtual.
  - RMI proporciona comunicación remota entre programas escritos en el lenguaje de programación Java.
- El contenido de esta presentación es el siguiente:
  - Una revisión rápida de la estructura de una aplicación RMI
    - Describe el sistema RMI y presenta sus ventajas
    - Describe conceptos básicos
    - La secuencia de ejecución de una aplicación RMI
  - Los pasos en la creación de una aplicación RMI
    - Creación de una aplicación RMI simple
      - ✓ Diseño de la interfaz, servidor y cliente
      - ✓ Compilación y ejecución de un ejemplo
    - Creación de una aplicación RMI típica
      - ✓ Diseño de la interfaz, servidor y cliente
      - ✓ Transferencia de objetos
      - ✓ Compilación y ejecución de un ejemplo

La Estructura de Aplicaciones RMI

- Las aplicaciones basadas en RMI se componen básicamente de dos programas separadas: un cliente y un servidor
  - Un servidor típico crea algunos objetos remotos, hace accesible las referencias a ellos y espera a que los clientes invoquen los métodos de los objetos remotos
  - Un cliente típico obtiene una referencia a uno o más objetos remotos en el servidor e invoca los métodos en ellos.
- RMI proporciona un mecanismo para la comunicación entre clientes y servidores: *aplicación de objetos distribuidos*.
  - Localización de objetos
    - facilidad de nombramiento (rmiregistry)
    - la aplicación puede pasar y regresar referencias a objetos remotos como parte de su operación normal
  - Comunicación con objetos remotos
    - Controlado por RMI
  - Cargar el código intermedio (bytecodes) de los objetos que son pasados en la aplicación
    - RMI proporciona los mecanismos necesarios para cargar el código de un objeto así como para transmitir sus datos

Aplicaciones RMI



- Los pasos que se siguen para ejecutar una aplicación RMI son los siguientes:
  - Se inicia el servidor y registras sus objetos y métodos a invocarse remotamente en un servicio de nombramiento
  - Se inicia el cliente y localiza los objetos a invocar de manera remota en el directorio de nombres del servidor obteniendo la referencia para el objeto
  - El cliente invoca a los métodos remotos mediante la referencia al objeto obtenida
- Es posible transferir el código en tiempo de ejecución del objeto objeto invocado remotamente.
  - Se utiliza un servidor Web para cargar el código intermedio de una clase de un objeto cuando es necesario

Características de Aplicaciones RMI

- Interfaces, objetos y métodos remotos
  - Las aplicaciones RMI se construyen a partir de interfaces y clases
    - las interfaces definen métodos y las clases los implantan
  - Los objetos que tienen métodos que son invocados desde máquinas virtuales diferentes son *objetos remotos*.
  - Un objeto se convierte en un objeto remoto si implanta una *interfaz remota*.
    - Extiene la clase `java.rmi.Remote`
    - Cada método de la interfaz declara a `java.rmi.RemoteException` en su cláusula `throws` adicionalmente a cualquier excepción de la aplicación.
  - Cuando un objeto se pasa de una máquina virtual a otra, en lugar de hacer una copia de su implantación en la máquina receptora, se pasa un "stub" para el objeto remoto
    - El stub actúa como un representante local del objeto remoto el cual es básicamente la referencia remota para quien lo invoca
- Ventajas del cargado dinámico
  - Si la clase de un objeto invocado de manera remota no está definida en el máquina virtual invocada, entonces, el código de la clase del objeto invocado es transferido a la máquina virtual
  - Dado que el tipo y comportamiento de un objeto es transferido a una máquina virtual remota, se pueden extender los objetos de manera dinámica.

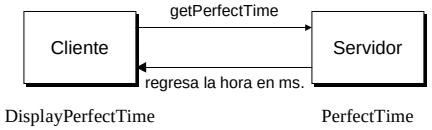
Creación de una Aplicación Distribuida

- Los pasos para la creación de una aplicación distribuida son:
  - Diseñar e implantar las componentes de la aplicación distribuida
    - Interfaz, servidor y cliente
  - Compilar los fuentes y generar los "stubs"
  - Hacer que las clases sean accesibles por la red
  - Ejecutar la aplicación
- Diseño e implantación de las componentes
  - Definición de las interfaces remotas
    - Una interfaz remota especifica los métodos que son invocados de manera remota
      - ✓ Determinación de los objetos que serán usados como parámetros y como valores de regreso de esos parámetros
    - La implantación de objetos remotos incluye la implantación de una o más interfaces remotas.
      - ✓ Puede incluir implantaciones de otras interfaces y otros métodos no necesariamente remotos
      - ✓ Si alguna clase local es usada como parámetro o valor de regreso, entonces esa clase debe ser implantada también
    - Los clientes que usan los objetos remotos pueden ser implantados en cualquier momento después de que las interfaces son definidas, inclusive después de que los objetos remotos han sido desplegados (*creados y registrados por el servidor*).

Creación de una Aplicación Distribuida

- Compilación y creación de "stubs"
  - Se usa el compilador de java, javac, para compilar el código fuente
    - La implantación de interfaces remotas
    - Implantaciones de las clases del servidor y del cliente
  - Se usa el compilador rmic para crear los "stubs" para los objetos remotos
    - RMI usa una clase stub para cada objeto remoto como su representante local
    - Un stub para un objeto remoto implanta el mismo conjunto de interfaces remotas que el objeto remoto
      - » Inicia una conexión con la máquina virtual remota
      - » Escribe y transmite los parámetros a la máquina virtual remota
      - » Espera por los resultados de la invocación remota
      - » Lee el valor de regreso y regresa de la invocación local
- Hacer accesibles las clases a la red
  - Todos los archivos asociados con los interfaces remotas, stubs y otras clases necesarias para transferir código se hacen accesibles por medio de un servidor de Web.
- Iniciar la aplicación
  - En esta etapa se inicia el registro de objetos remotos de RMI y los programas para el servidor y el cliente.

Aplicación RMI Simple



Interfaz entre el Cliente y el Servidor

```
//: PerfectTimeI.java
// The PerfectTime remote interface
package c15.ptime;
import java.rmi.*;

interface PerfectTimeI extends Remote {
  long getPerfectTime() throws RemoteException;
} ///:~
```

Parte del paquete c15.ptime

extiende a la clase Remote

reporta la excepción RemoteException

Implantación de la Interfaz Remota

Código para el Servidor

```
//: PerfectTime.java
// The implementation of the PerfectTime
// remote object
package c15.ptime;
import java.rmi.*;
import java.rmi.server.*;
import java.rmi.registry.*;
import java.net.*;

public class PerfectTime extends UnicastRemoteObject implements PerfectTimeI {
  // Implementation of the interface:
  public long getPerfectTime() throws RemoteException {
    return System.currentTimeMillis();
  }
  // Must implement constructor to throw RemoteException
  public PerfectTime() throws RemoteException {
    // super(); // Called automatically
  }
  // Registration for RMI serving
  public static void main(String[] args) {
    System.setSecurityManager( new RMISecurityManager() );
    try {
      PerfectTime pt = new PerfectTime();
      Naming.bind( "//sigma:2005/PerfectTime", pt);
      System.out.println("Ready to do time");
    } catch(Exception e) {
      e.printStackTrace();
    }
  }
} ///:~
```

implantación del método remoto

es necesario incluir un método main para registrar los métodos implantados

registra el objeto pt con el nombre PerfectTime en la computadora sigma y asocia el puerto 2005

Invocación de la Interfaz Remota

Código para el Cliente

```
//: DisplayPerfectTime.java
// Uses remote object PerfectTime
package c15.ptime;
import java.rmi.*;
import java.rmi.registry.*;

public class DisplayPerfectTime {
  public static void main(String[] args) {
    System.setSecurityManager(new RMISecurityManager());
    try {
      PerfectTimeI t =
        (PerfectTimeI )Naming.lookup("//sigma:2005/PerfectTime");
      for(int i = 0; i < 10; i++)
        System.out.println("Perfect time = " +
          t.getPerfectTime());
    } catch(Exception e) {
      e.printStackTrace();
    }
  }
} ///
```

Parte del paquete c15.ptime

Invoca al manejador de seguridad

Obtiene la referencia al objeto remoto llamado PerfectTime que está sigma en el puerto 2005

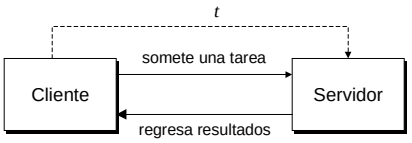
Invoca al método remoto

Compilación y Ejecución

- Acciones a realizar
  - mkdir $HOME/mysrc/c15/ptime
    - la ruta de acceso para el paquete c15.ptime
  - setenv CLASSPATH .:/usr/local/java/jdk117_v3/lib/classes.zip:$HOME/mysrc
    - se define la ruta de acceso a las clases en java
  - javac -d $HOME/mysrc/c15/ptime PerfectTime.java PerfectTimeI.java DisplayPerfectTime.java
    - se compila cada uno de los archivos fuente y su código intermedio se coloca en $HOME/mysrc/c15/ptime
  - rmic c15.ptime.PerfectTime
    - se compila el método remoto PerfectTime y se generan los stubs
      - ✓ PerfectTime_Skel.class
      - ✓ PerfectTime_Stub.class
  - Servidor: rmiregistry 2005&
    - Se arranca el servidor de nombres en el lado del servidor el cual permite a clientes remotos obtener una referencia a un objeto remoto.
  - Servidor: java c15.ptime.PerfectTime
  - Cliente: java c15.ptime.DisplayPerfectTime

Aplicación de Cómputo Distribuido

- Una máquina de cómputo: un objeto remoto en un servidor
  - Toma tareas sometidas por un cliente, las ejecuta y regresa los resultados
  - Las tareas se ejecutan en la computadora en donde se ejecuta el servidor
  - Varios clientes pueden someter tareas a un servidor
  - Las tareas a ejecutarse no necesitan definirse en la máquina de cómputo en donde se van a ejecutar
  - Dinámicamente, se pueden crear tareas nuevas y someterlas a la máquina de cómputo
    - El código de la tarea es transferido a la máquina de cómputo
  - Se requiere que la clase de las tareas implemente una *interfaz* particular



Diseño de la Interfaz

- Interfaz compute.Compute

```
package compute;

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

public interface Compute extends Remote {
     Object executeTask(Task t) throws RemoteException;
}
```

Se define un método remoto

- Interfaz compute.Task

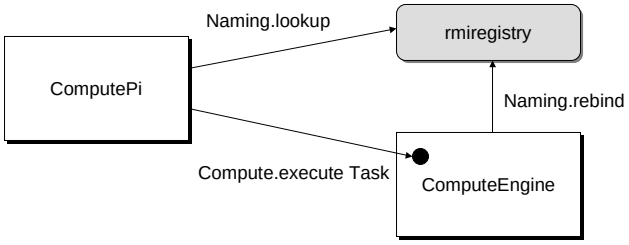
```
package compute;

import java.io.Serializable;

public interface Task extends Serializable {
     Object execute();
}
```

Se define un método execute() que regresa una instancia de Object

RMI usa el mecanismo de objetos serializables para transportar objeto por valor entre máquinas virtuales Java.
Una implantación "Serializable" marca a la clase como capaz de convertirse en un secuencia de bytes que la describe y que se pueden usar para reconstruir una copia exacta del objeto cuando éste es leído sobre un canal de lectura.

Diseño de la Máquina de Cómputo

- La máquina de cómputo, creada a partir de la clase ComputeEngine, implanta la interfaz Compute
  - Permite que varias tareas sean sometidas a ella mediante la invocación del método executeTask
- El cliente
  - ComputePi: localiza a la máquina de cómputo y envía una tarea de la clase Pi
  - Pi: calcula una aproximación de Pi



Compilación de la Interfaz

- Se separa el código en tres paquetes
  - `compute` (interfaces Compute y Task interfaces): Interfaz
  - `engine` (Implantación de ComputeEngine y su stub): Servidor
  - `client` (Código para ComputePi y para la tarea Pi): Cliente
- Compilación de las interfaces
  - Las interfaces Compute y Task se empacarán en un archivo `compute.jar` (java archive)
    - cd /home/adiaz/java/rmi/example-1dot2
    - javac compute/Compute.java
      - ✓ Se obtiene Compute.class
    - javac compute/Task.java
      - ✓ Se obtiene Task.class
    - jar cvf compute.jar compute/*.class
  - El archivo `compute.jar` se distribuye a los desarrolladores de las aplicaciones cliente y servidor

Compilación del Código del Servidor

- Compilación del servidor
  - El paquete `engine` contiene únicamente la clase ComputeEngine que se ejecuta en el lado del servidor
    - Supongamos que el paquete `engine` se encuentra localizado en **/home2/adiaz/java/rmi/example-1dot2**
    - El archivo `compute.jar` se encuentra en **/home2/adiaz/public_html/classes**
  - Es necesario definir la variable CLASSPATH como sigue:
    - setenv CLASSPATH /home2/adiaz/java/rmi/example-1dot2:/home/adiaz/public_html/classes/compute.jar
  - La compilación del código del servidor se realiza mediante los siguientes pasos
    - cd /home2/adiaz/java/rmi/example-1dot2
    - javac engine/ComputeEngine.java
    - rmic -d . engine.ComputeEngine
      - ✓ se generan los archivos ComputeEngine_Stub.class y ComputeEngine_Skel.class
    - mkdir /home2/adiaz/public_html/classes/engine
    - cp engine/ComputeEngine_*.class /home2/adiaz/public_html/classes/engine
  - Es necesario desempacar el archivo compute.jar
    - cd /home/ann/public_html/classes
    - jar xvf compute.jar

Compilación del Código del Cliente

- Compilación del cliente
  - El paquete `client` contiene la clase ComputePi y Pi que se ejecuta en el lado del cliente
    - Supongamos que el paquete `client` se encuentra localizado en **/home3/adiaz/src/**
    - El archivo `compute.jar` se encuentra en **/home3/adiaz/public_html/classes**
  - Se realizan los siguientes pasos
    - setenv CLASSPATH /home3/adiaz/src:/home3/adiaz/public_html/classes/compute.jar
    - cd /home3/adiaz/src
    - javac client/ComputePi.java
      - ✓ Se obtiene ComputePi.class
    - javac -d /home3/adiaz/public_html/classes client/Pi.java
      - ✓ Se obtiene Pi.class y se coloca en /home3/adiaz/public_html/classes/client
      - ✓ Es necesario hacer el código de Pi accesible para que se transfiera al servidor

Ejecución de los Programas

- En el lado del servidor
  - Para forzar a que el programa siguiente tenga que transferir las classes se hace
    - unsetenv CLASSPATH
  - Se ejecuta el programa para resolver nombres y determinar referencias a objetos remotos
    - rmiregistry &
  - Se define la variable ambiente para que el servidor pueda localizar sus clases
    - setenv CLASSPATH /home2/adiaz/java/rmi/example-1dot2:/home2/adiaz/public_html/classes/compute.jar
  - Se ejecuta el servidor de la manera siguiente:
    - java -Djava.rmi.server.codebase=http://sigma2/~adiaz/classes/ -Djava.rmi.server.hostname=sigma2.cs.cinvestav.mx engine.ComputeEngine
- En el lado del cliente se hace lo siguiente:
  - Se define la variable ambiente para que el cliente pueda localizar sus clases
    - setenv CLASSPATH /home2/adiaz/src:/home3/adiaz/public_html/classes/compute.jar
  - Se ejecuta el cliente de la manera siguiente:
    - java -Djava.rmi.server.codebase=http://sigma3/~adiaz/classes/ client.ComputePi sigma2.cs.cinvestav.mx 20

Ambiente de Ejecución