Remote Method Invocation (RMI) de Java

Concurrencia y Distribución
Programación Avanzada
Posgrado en Ciencia e Ingeniería de la Computación, UNAM

1. Introducción

El mecanismo RMI (*Remote Method Invocation*) permite que una aplicación o applet se comunique con objetos que residen en programas que se ejecutan en máquinas remotas. En esencia, en lugar de crear un objeto, el programador liga el objeto remoto con un representante local, conocido como *stub*. Los mensajes dirigidos al objeto remoto se envían al *stub* local, como si fuera el objeto real. El *stub* acepta los mensajes que se le envíen, y a su vez, los envía al objeto remoto, el cual invoca sus métodos apropiados. El resultado de la invoación de los métodos en el objeto remoto se envía de regreso al *stub* local, que los remite al emisor original de la llamada.

Aparte de ligar el *stub* con el objeto remoto, el código escrito por el programador para comunicar con el objeto remoto es igual al código que se utilizaría si el objeto se encontrara en una aplicación o applet local.

2. Resumen del proceso

La Figura 1 muestra una aplicación cliente ejecutándose en una máquina A, que envía un mensaje a un objeto remoto contenido en una aplicación servidor que se ejecuta en una máquina B.

Cuando la aplicación cliente envía un mensaje al *stub* local del objeto remoto, la petición se transmite a la máquina que contiene al objeto real, donde el método es invocado y cualquier resultado retornado al *stub* local, de modo que la aplicación cliente puede obtener la respuesta apropiada.

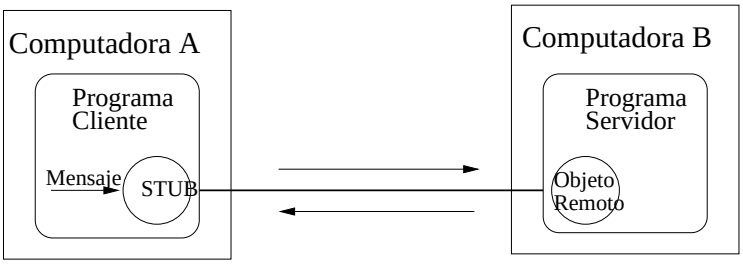

Figura 1: Enviando un mensaje a un objeto remoto.

2.1. La clase `java.rmi.Naming`

La clase `Naming` contiene los siguientes métodos estáticos que permiten el acceso a objetos remotos utilizando un URL para especificar el nombre y lugar del objeto remoto.

Método	**Responsabilidad**
`bind(url,object)`	Liga un nombre a un objeto remoto. El nombre se especifica como un URL.
`list(url)`	Retorna un arreglo de cadenas representando los URLs en el registro.
`lookup(url)`	Retorna un objeto remoto (un *stub*) asociado con el URL.
`rebind(url, object)`	Similar al `bind()`, pero reemplaza la asociación hecha.
`unbind(url)`	Remueve la asociación entre el objeto remoto y el URL.

El URL se presenta en la forma `rmi://host:port/objectName`, donde:

Componente	**Valor por defecto**	**Especificación**
`rmi`	`rmi`	El método de acceso (debe ser `rmi`).
`host`	`localhost`	La máquina servidor (`host`).
`port`	`1099`	El puerto utilizado.
`objectName`	`-`	El nombre del objeto remoto.

2.2. La aplicación `rmiregistry`

La aplicación servidor `rmiregistry` se utiliza para:

- Registrar un nombre y un lugar de un objeto remoto. Esto se realiza a partir de un servidor que contiene un objeto remoto. El código en el servidor es de la forma:

  ```
  Naming.rebind("rmi://host/name", object);
  ```

  Donde `rmi://host/name` es el lugar donde se encuentra el objeto remoto y `object` es el objeto que se llama remotamente. Además la aplicación `rmiregistry` debe estarse ejecutando en la misma máquina servidor que la aplicación servidor que contiene el objeto remoto.

- Permitir a un cliente ligar un *stub* local que le de acceso al objeto remoto contenido en la aplicación servidor. La aplicación cliente se liga al objeto remoto mediante el método `lookup()`, que retorna un objeto que permite el acceso vía un *stub* al objeto remoto. El código en la aplicación cliente debe ser de la forma:

  ```
  Naming.lookup("rmi://host/name");
  ```

En esencia, la aplicación `rmiregistry` actúa como un registro de objetos que pueden ser accesados remotamente. Los objetos se registran y ligan a un *stub* local usando los métodos de la clase `Naming`, del paquete `java.rmi`.

2.3. El proceso completo

La Figura 2 muestra el proceso completo, con una aplicación cliente ejecutándose en la máquina A, que accesa a un objeto remoto que se ejecuta en una aplicación servidor en una máquina B.

La secuencia de pasos se realiza como sigue:

1. Inicialización:
   - Se ejecuta la aplicación `rmiregistry` en la computadora B.
   - La aplicación servidor que contiene al objeto remoto se inicia en la computadora B.

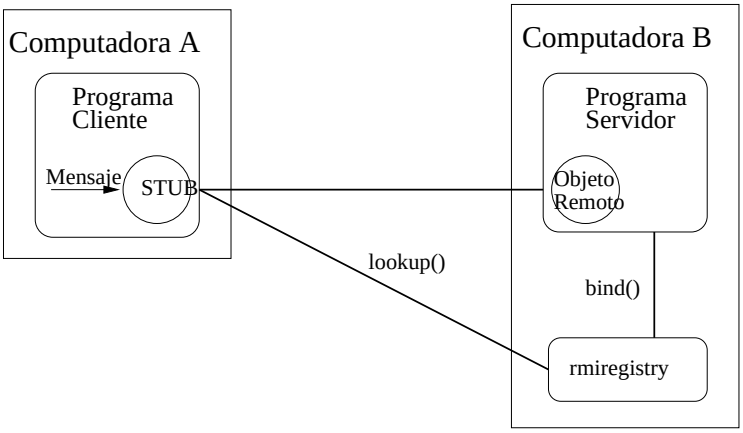

Figura 2: Resumen del accdeso a un objeto remoto utilizando `rmiregistry`.

- La aplicación servidor liga (usando `bind()` ó `rebind()`) al objeto remoto con la aplicación `rmiregistry`.
- La aplicación cliente se inicia en la computadora A.
- La aplicación cliente busca (`lookup()`) al objeto remoto y lo liga a un *stub* local.

2. Acceso:

   - Los mensajes se envían al *stub* local en la computadora A, que son re-enviados al servidor donde se encuentra el objeto real. El método correspondiente al mensaje se invoca, y el resultado se retorna.

Nótese que la aplicación `rmiregistry` debe estar ejecutándose en la misma computadora donde se ejecuta la aplicación servidor que contiene al objeto remoto.

Una clase importante utilizada en el proceso RMI es la clase `Naming`, cuyas responsabilidades son:

Método	Responsabilidad
`bind(str,o)`	Liga el nombre `str` con el objeto remoto `o`.
`lookup(str)`	Retorna un *stub* asociado con el nombre `str`.
	El objeto remoto se accesa mendiante enviar mensajes al *stub*.
`rebind(str,o)`	Similar al `bind()`, pero reemplaza la asociación.
`unbind(str)`	Remueve la asociación entre el objeto remoto
	y el nombre `str`.

3. La implementación – acceso remoto a una cuenta de banco

Desarrolle la aplicación de acceso remoto a una cuenta bancaria, en forma de una relación cliente-servidor, donde la aplicación servidor contiene un objeto de tipo cuenta bancaria que se accesa remotamente por la aplicación cliente.

3.1. La clase `RAccount`

El comportamiento de la cuenta de banco se basa en una clase `RAccount`. Esta clase contiene las variables de instancia `theBalance` y `theMinBalance`, ambas de tipo `double` que mantienen información sobre el estado de la cuenta y la cantidad mínima que ésta debe contener.

Además, debe contener los métodos para el manejo de la cuenta, como `accountBalance()`, que permite obtener el valor de `theBalance`; el método `withdraw(double)`, que permite "retirar" una cantidad siempre y cuando no resulte menor a la cantidad mínima que debe haber en la cuenta; el método `deposit(double)`, que permite incrementar el valor de `theBalance`; y el método `setMinBalance(double)`, que permite establecer la cantidad mínima que debe haber en la cuenta.

Como una instancia de esta clase debe accesarse remotamente, ésta debe extender la clase `UnicastRemoteObject` del paquete `java.rmi.server` e implementar la interfaz `RemoteAccount`, que se presenta más adelante.

Además, cada método de la clase debe ser capaz de arrojar la excepción `RemoteException`.

3.2. La interfaz `RemoteAccount`

La interfaz `RemoteAccount` define el protocolo que el objeto remoto debe implementar. Se define como sigue:

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

interface RemoteAccount extends Remote{
  public double accountBalance() throws RemoteException();
  public void   deposit(final double money) throws RemoteException();
  public double withdraw(final double money) throws RemoteException();
}
```

Esta interfaz debe definirse ya que no resulta correcto que la clase `RAccount` implementar directamente implemente directamente la interfaz `Remote`.

3.3. La aplicación cliente

La aplicacióin cliente se ejecuta a partir de la línea de comando con un URL que es el parámetro que representa al objeto remoto y su locación. Por ejemplo, si tanto la aplicación cliente como servidor se ejecutan en la misma computadora, entonces el URL sólo require especificar lo siguiente:

```
java Client Mike
```

El código de la clase `Client` debe extraer, como parte de su método `main()`, el URL como una cadena de tipo `string`, que se pasa como argumento a un método estático `process()`, cuya responsabilidad es contactar y accesar al objeto remoto.

Nótese que si se utiliza una versión anterior de Java, es necesario considerar un manejador de seguridad que permita el acceso al objeto remoto. Sin embargo, para versiones a partir de Java 2, esto no se requiere.

El método `process()` debe utilizar el método `lookup(url)`, retornando un *stub* del objeto remoto. Este *stub* se accesa en forma local como si se tratara del objeto mismo. Es decir, un mensaje enviado al *stub* se convierte en una forma que puede enviarse al objeto remoto mediante una conexión de red. Si se retorna un resultado por parte del objeto remoto, este se entrega como resultado de enviar el mensaje al *stub* local. De tal modo, el método `process()` debe realizar las siguientes operaciones sobre el objeto remoto:

- Lectura del balance (`accountBalance()`).
- Depósito de 100.00 (`deposit(100.00)`).
- Lectura del nuevo balance (`accountBalance()`).
- Retiro de 20.0 (`withdraw(20.0)`.
- Lectura del nuevo balance (`accountBalance()`).

3.4. La aplicación servidor

La aplicación servidor simplemente crea una instancia de la clase `RAccount`, y lo liga al URL que se da como parámetro de la línea de comando. Por ejemplo, de nuevo, si el cliente y el servidor se ejecutan en la misma computadora, entonces el URL solo debe especificar el nombre del objeto remoto.

```
java Server Mike
```

Sin embargo, el objetivo de este ejercicio es obviamente ejecutar cliente y servidor en diferentes computadoras.

La clase `Server` debe extraer la información del URL de la línea de comandos, y utilizarla en un método estático `process()` para crear un objeto de tipo `RAccount` y ligarlo al URL dado.

3.5. Compilando y ejecutando las aplicaciones

Las siguientes secciones describen cómo debe verse la compilación y ejecución de la aplicación distribuida, utilizando el JDK estándar. Los siguientes archivos deben contener las clases e interfaces:

Archivo	**Contenido**
`RemoteAccount.java`	La interfaz `RemoteAccount`.
`RAccount.java`	La clase `RAccount`.
`Client.java`	La clase `Client`.
`Server.java`	La clase `Server`.

3.5.1. Compilando las aplicaciones

Usando la línea de comando, los archivos deben compilar como sigue:

```
javac RemoteAccount.java
javac RAccount.java
javac Client.java
javac Server.java
```

Sin embargo, es necesario generar un *stub* para el objeto remoto. Esto se logra mediante el comando para el compilador rmi:

```
rmic RAccount
```

3.5.2. Ejecutando las aplicaciones

La ejecución debe seguir los siguientes pasos:

Primeramente, en la computadora servidor, la aplicación `rmiregistry` debe iniciarse mediante el comando:

```
rmiregistry
```

Ahora es posible iniciar la aplicación servidor mediante:

```
java Server rmi://MyMachine/MikeAccount
```

Y el cliente se inicia con:

```
java Client rmi://MyMachine/MikeAccount
```

Esto debe imprimir lo siguiente:

```
El balance de Mike  = 0.0
El balance de Mike  = 100.0
El balance de Mike  = 80.0
```

Si se vuelve a ejecutar el cliente, la salida debe ser:

```
El balance de Mike  = 80.0
El balance de Mike  = 180.0
El balance de Mike  = 160.0
```

Referencias

[1] Ken Arnold and James Gosling. *The Java Programming Language*. Addison-Wesley, 1996.

[2] Barry Boone. *Java Essentials for C and C++ Programmers*. Addison-Wesley, 1996.

[3] Brinch Hansen, P. *The Programming Language Concurrent Pascal*. In IEEE Transactions on Software Engineering, 1(2), 1975, pp. 199-207.

[4] Gary Cornell and Cay S. Horstsmann. *Core Java*. Prentice-Hall, 1996.

[5] David Flanagan. *Java in a Nutshell*. O'Reilly, 1996.