Sistemas Distribuidos

Práctica 2: Invocación remota con RMI

Autor: Javier Celaya, modificada por Rafael Tolosana

Resumen

El objetivo de esta segunda práctica es el de comprender y utilizar la *llamada a procedimiento remoto* (RPC) a través de su correspondiente tecnología en Java: la *invocación de método remoto* (RMI). Deberéis construir la infraestructura necesaria para que un cliente pueda invocar un método en múltiples servidores que calcule los números primos de un cierto intervalo. El cliente deberá determinar los números primos existentes en un cierto intervalo, dividiéndolo en subintervalos y preguntando de manera concurrente a tantos servidores de cálculo como le indique el usuario.

Estas prácticas incluyen redactar una memoria y escribir código fuente. El texto de la memoria debe ser original, pero el código puede basarse en otro ya existente, siempre con el consentimiento del autor y citando la fuente. Por ejemplo, el código que encontréis en Internet puede ser utilizado siempre que citéis el nombre del autor y la URL de dónde lo habéis sacado. Sin embargo, las aportaciones personales siempre serán mejor valoradas. No citar la fuente se considera copia, y copiar supone un cero en la nota de prácticas.

1. Java RMI

Java Remote Method Invocation (RMI) es la respuesta de Java a RPC. Permite invocar métodos sobre objetos que se encuentran en otras máquinas virtuales, y posiblemente también en otros equipos. El esquema básico de RMI se compone de:

- Un programa servidor, que aloja una instancia de objeto sobre la que se invocarán los métodos remotos.
- Un programa cliente que realizará las invocaciones a método remoto.
- Y un servicio de nombres, llamado *registro RMI*, que permite a los clientes localizar a los servidores correspondientes.

El servidor declarará la clase de objeto que se quiere exportar. Esta clase debe implementar una interfaz, que llamaremos *interfaz remota*, y que determina qué métodos de la clase se van a exportar. Los programas cliente sólo deben conocer esa interfaz para realizar la llamada, no tienen por qué conocer la implementación concreta de cada método. El servidor indicará entonces al registro RMI que tiene una instancia de objeto que quiere exportar y le dará un nombre. Utilizando ese nombre, el cliente solicitará al registro una referencia a esa instancia. Lo que el registro le devuelve es un *stub*: un objeto que implementa la interfaz remota pero que únicamente se encarga de realizar la comunicación con el servidor, enviándole los parámetros de la llamada y devolviendo el resultado. En la documentación oficial del paquete `java.rmi` [1] tenéis todo lo que os hace falta para esta práctica. El tutorial básico os será especialmente útil.

1.1. Diferencias con RPC

RPC, en especial en su implementación en lenguaje C, deja a la vista del programador multitud de detalles. Generación de interfaces, de stubs, el lenguaje XDR, etc... Por el contrario, Java RMI los esconde para que el programador se centre en la aplicación que está desarrollando. Pero no debéis olvidar que están ahí.

Al igual que en C, comenzaréis definiendo la interfaz de las llamadas remotas, indicando los tipos de datos que se pasan como argumento y el tipo de valor devuelto. No es necesario que generéis ningún código, Java lo hace automáticamente mediante *introspección*. En pocas palabras, esta propiedad de Java le permite descubrir qué métodos y propiedades tiene un objeto del cual no conoce su clase. Igualmente, el stub del cliente y el wrapper del servidor, que gestionan la comunicación manteniendo la interfaz, se generan automáticamente en tiempo de ejecución, con la clase `java.rmi.server.UnicastRemoteObject`. Tanto el envío de los datos como del stub se realiza con la *serialización* de Java, que convierte un objeto en una representación binaria y viceversa. Vendría a ser el equivalente de XDR, pero también se realiza de manera transparente al programador. Por último, el equivalente al portmapper de RPC es el registro de RMI. Saber cómo resuelve Java todos esos problemas del diseño de llamada remota os permitirá sacarle todo el partido.

2. Objetivo de la práctica

En esta práctica debéis programar un sistema basado en llamadas remotas con Java RMI que permita a un programa cliente localizar y utilizar varios servidores de cálculo. En concreto, debéis programar los siguientes tres componentes:

1. Un servidor de cálculo que sea capaz de obtener los números primos en un intervalo dado como parámetro.

2. Un servidor de asignación que sea capaz de encontrar tantos servidores de cálculo como se le pida.

3. Un cliente al que se le pasará un número n y un intervalo. Entonces, tendrá que solicitar al servidor de asignación n servidores de cálculo, y dividir el intervalo en subintervalos contiguos para que cada servidor calcule los números primos de una parte diferente de manera concurrente.

La sección 3 expone todos los detalles del sistema. Como resultado de la práctica entregaréis un fichero JAR que se llamará `SSDDp2.jar`.

Como en la práctica 1, el fichero JAR debe contener dos cosas:

- una memoria, en la que resumiréis vuestro diseño del sistema, explicando el funcionamiento de cada componente y sus interacciones. **Poned especial atención en explicar las funciones de comunicación por RMI.** Ceñíos a lo que se os pide, sed claros y concisos. Además, debéis realizar y documentar una traza de pruebas para validar su correcto funcionamiento, abarcando todos los casos que se os ocurran. No os extendáis más de tres páginas en total.
- el código fuente que hayáis desarrollado. Comentadlo conveniente, si fuera necesario. Poned atención a la organización del código, la modularidad, la legibilidad, la encapsulación, etc.

Así, tiene que ser posible invocar la ejecución de cada componente con las ordenes:

```
java -jar SSDDp2.jar -c [IP_registro]
java -jar SSDDp2.jar -a [IP_registro]
java -jar SSDDp2.jar -u min max n [IP_registro]
```

La opción "-c", "-a" y "-u" establece si queremos ejecutar un servidor de cálculo, el servidor de asignación o el cliente. En el caso de ejecutar el cliente, los parámetros "min", "max" y "n" se corresponden con los extremos del intervalo y el número de servidores de cálculo a utilizar. El parámetro opcional "IP_registro" dará la IP (o el nombre) de la máquina donde se encuentra el registro RMI. Por defecto, se buscará el registro en la dirección local.

3. Sistema distribuido de cálculo de números primos

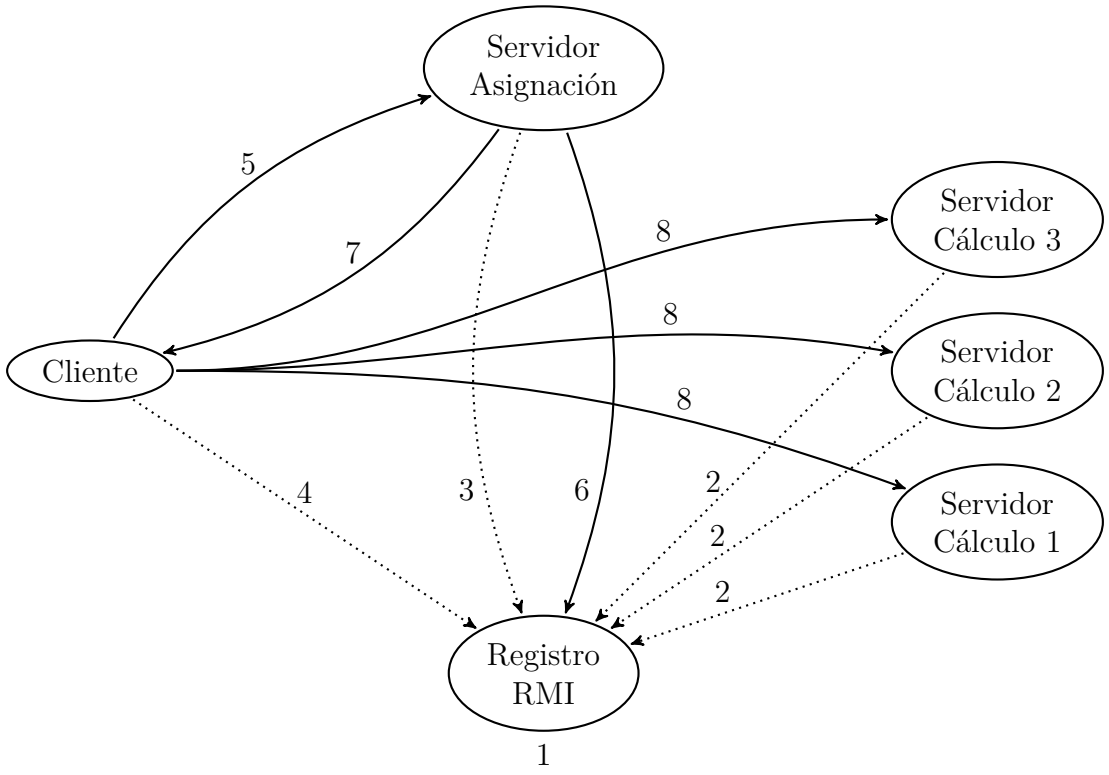
El sistema que vais a desarrollar permite a un cliente localizar varios servidores de cálculo y utilizarlos todos a la vez para encontrar los números primos de un cierto intervalo. Al ejecutar los cálculos en paralelo, se finaliza el trabajo en una fracción del tiempo que se requeriría con una sola máquina. Consiste en los siguientes componentes:

- Un servidor de registro RMI.
- Varios servidores de cálculo.
- Un servidor de asignación.

- Un cliente que realizará la llamada remota.

Cada uno puede ejecutarse en un equipo distinto, siempre que todos sepan dónde se ejecuta el registro RMI. Sería interesante que al menos los servidores de cálculo se ejecutaran en equipos distintos, para que se aprecie el reparto de trabajo, pero de cara a la validación de la práctica podéis ejecutarlos todos en el mismo equipo. La interacción entre ellos es la siguiente:

1. Se lanza el registro RMI.
2. Se lanzan uno o más servidores de cálculo, que se registran en el registro RMI.
3. Se lanza un servidor de asignación, que se registra en el registro RMI.
4. Se lanza el cliente, quien busca al servidor de asignación en el registro RMI.
5. El cliente le pide al servidor de asignación n servidores de cálculo.
6. El servidor de asignación busca en el registro RMI los n servidores de cálculo.
7. El servidor de asignación le devuelve el resultado al cliente.
8. El cliente divide el intervalo de búsqueda en subintervalos y le pide a cada servidor de cálculo que encuentre los números primos de uno o más subintervalos.



3.1. Registro RMI

El servidor de registro RMI es una herramienta estándar de Java. Se ejecuta con el comando `rmiregistry`. Debe ser capaz de encontrar las clases que queráis registrar, así que vuestro fichero JAR debe estar en el classpath. En general, esto se consigue añadiendo la ruta de vuestro JAR a la variable de entorno **`CLASSPATH`**. En una terminal Unix podéis hacerlo en el mismo momento de ejecutar el registro:

```
CLASSPATH=/ruta/a/vuestro/fichero.jar rmiregistry
```

3.2. Servidor de cálculo

Lo primero que vais a implementar es un servidor que acepte peticiones RMI para calcular los números primos en un intervalo. Este servidor debe crear y exportar un objeto con un método que cumpla esa función, y registrarlo en el registro RMI. Para ello, lo primero es definir su interfaz remota. Los objetos en RMI deben implementar una interfaz que derive de `java.rmi.Remote`, así que vuestra primera interfaz `Worker` deberá ser de la siguiente forma:

```
import java.rmi.Remote;
import java.rmi.RemoteException;
import java.util.ArrayList;

public interface Worker extends Remote {
    // Devuelve un vector con los primos entre min y max.
    java.util.ArrayList<Integer> encuentraPrimos(int min, int max)
        throws RemoteException;
}
```

Ejercicio 1. Completar el código de la clase `WorkerServer`, que implementa el interfaz `Worker`. Esa clase deberá encontrar el registro RMI en la dirección que le pasáis al JAR al ejecutarlo y registrar una instancia con un nombre tal que, más tarde, el servidor de asignación sea capaz de encontrarlo. Pero tened en cuenta lo siguiente: como vais a tener varios objetos `Worker` registrados en el mismo registro, cada servidor de cálculo debe registrar su objeto con un nombre diferente. Debéis generar un nombre que no haya sido usado aún... Investigad el método `java.rmi.registry.Registry.list()`.

3.3. Servidor de asignación

Una vez que tengáis vuestros objetos `Worker` registrados, vais a implementar un segundo tipo de servidor que se encargará de asignarlos a los clientes. Es decir, este servidor debe crear y exportar un objeto remoto con un método que devuelva un conjunto de referencias a objetos `Worker`. El conjunto contendrá tantas referencias como le sean solicitadas, pero sin sobrepasar el número de servidores de cálculo que hayan sido registrados. Su interfaz `WorkerFactory` será la siguiente:

```
import java.rmi.Remote;
import java.rmi.RemoteException;
import java.util.ArrayList;

public interface WorkerFactory extends Remote {
    // Devuelve un vector de hasta n referencias a objetos Worker.
    ArrayList<Worker> dameWorkers(int n) throws RemoteException;
}
```

Ejercicio 2. Completar el código de la clase `WorkerFactoryServer`, que implemente el interfaz `WorkerFactory`. Su método `dameWorkers` deberá buscar en el registro RMI `n` referencias a objetos `Worker` y guardarlas en un vector, que devolverá como resultado. Este servidor debe descubrir los objetos `Worker` que habéis registrado anteriormente a través de su nombre.

3.4. Cliente

Ejercicio 3. Implementar un programa cliente que utilice este sistema. El cliente contactará con el registro RMI y le pedirá la referencia a objeto remoto de tipo `WorkerFactory` que habéis registrado con el servidor de asignación. A este objeto le solicitará `n` referencias de objeto remoto `Worker`, y a cada referencia devuelta le pedirá que calcule los números primos de uno o más subintervalos del intervalo de búsqueda. Podéis dividir el intervalo de búsqueda como queráis, en `n` o más subintervalos no necesariamente del mismo tamaño, pero **debéis justificar vuestra decisión en la memoria**. Deberéis hacer que el cliente pida los cálculos a todos los servidores de cálculo de forma concurrente, de la manera que creáis más conveniente. Tras recibir todos los resultados, el cliente mostrará por pantalla cuantos primos hay en el intervalo y la lista de números primos en orden ascendente.

4. Realización y Evaluación (Rúbrica)

La realización de las prácticas es por parejas, pero los dos componentes de la pareja deberán entregarla de forma individual. En general estos son los criterios de evaluación:

- Deben entregarse todos los programas, se valorará de forma negativa que falte algún programa / alguna funcionalidad.
- Todos los programas deben compilar correctamente, se valorará de forma muy negativa que no compile algún programa.
- Todos los programas deben funcionar correctamente como se especifica en el problema.

- Todos los programas tienen que seguir el manual de estilo de Java propuesto por Oracle, disponible en moodle2. Además de lo especificado en el manual de estilo, cada fichero fuente deberá comenzar con la siguiente cabecera:

```
/*
 * AUTOR: nombre y apellidos
 * NIA: n'umero de identificaci'on del alumno
 * FICHERO: nombre del fichero
 * TIEMPO: tiempo en horas de codificaci'on
 * DESCRIPCI'ON: breve descripci'on del contenido del fichero
 */
```

4.1. Rúbrica

Con el objetivo de que, tanto los profesores como los estudiantes de esta asignatura por igual, puedan tener unos criterios de evaluación objetivos y justos, se propone la siguiente rúbrica en el Cuadro 1. Los valores de las celdas son los valores mínimos que hay que alcanzar para conseguir la calificación correspondiente y tienen el siguiente significado:

- A+ (excelente). En el caso de software, conoce y utiliza de forma autónoma y correcta las herramientas, instrumentos y aplicativos software necesarios para el desarrollo de la práctica. Plantea *correctamente* el problema a partir del enunciado propuesto e identifica las opciones para su resolución. Aplica el método de resolución adecuado e identifica la corrección de la solución, *sin errores*. En el caso de la memoria, se valorará una estructura y una presentación adecuadas, la corrección del lenguaje así como el contenido explica de forma precisa los conceptos involucrados en la práctica. En el caso del código, este se ajusta exactamente a las guías de estilo propuestas.

- A (bueno). En el caso de software, conoce y utiliza de forma autónoma y correcta las herramientas, instrumentos y aplicativos software necesarios para el desarrollo de la práctica. Plantea *correctamente* el problema a partir del enunciado propuesto e identifica las opciones para su resolución. Aplica el método de resolución adecuado e identifica la corrección de la solución, *con ciertos errores* no graves. Por ejemplo, algunos pequeños casos (marginales) no se contemplan o no funionan correctamente. En el caso del código, este se ajusta *casi* exactamente a las guías de estilo propuestas.

- B (suficiente). En el caso de software, conoce y utiliza de forma autónoma y correcta las herramientas, instrumentos y aplicativos software necesarios para el desarrollo de la práctica. No plantea correctamente el problema a partir del enunciado propuesto y/o no identifica las opciones para su resolución. No aplica el método de resolución adecuado y / o identifica la corrección de la solución, pero *con errores*. En el caso de la memoria, bien la estructura y / o la presentación son mejorables, el lenguaje presenta deficiencias y / o el contenido no explica de forma precisa los conceptos

importantes involucrados en la práctica. En el caso del código, este se ajusta a las guías de estilo propuestas, pero es mejorable.

- B- (suficiente, con deficiencias). En el caso de software, conoce y utiliza de forma autónoma y correcta las herramientas, instrumentos y aplicativos software necesarios para el desarrollo de la práctica. No plantea correctamente el problema a partir del enunciado propuesto y/o no identifica las opciones para su resolución. No se aplica el método de resolución adecuado y/o se identifica la corrección de la solución, pero *con errores* de cierta gravedad y/o sin proporcionar una solución completa. En el caso de la memoria, bien la estructura y / o la presentación son *manifiestamente* mejorables, el lenguaje presenta *serias* deficiencias y / o el contenido no explica de forma precisa los conceptos importantes involucrados en la práctica. En el caso del código, hay que mejorarlo para que se ajuste a las guías de estilo propuestas.
- C (deficiente). El software no compila o presenta errores graves. La memoria no presenta una estructura coherente y/o el lenguaje utilizado es pobre y/o contiene errores gramaticales y/o ortográficos. En el caso del código, este no se ajusta exactamente a las guías de estilo propuestas.

Calificación	Solución al Problema	Aspectos Tecnológicos	Código	Memoria
10	A+	A+	A+	A+
9	A+	A+	A	A
8	A	A	A	A
7	A	A	B	B
6	B	B	B	B
5	B-	B-	B-	B-
suspenso	1 C			

Cuadro 1: Detalle de la rúbrica: los valores denotan valores mínimos que al menos se deben alcanzar para obtener la calificación correspondiente

5. Entrega y Defensa

Deberéis entregar el fichero JAR a través de moodle2 en la actividad habilitada a tal efecto, el *29 de octubre de 2015* para los grupos A y el *5 de noviembre de 2015* para los grupos B.

Los días 30 de octubre y 13 de noviembre durante la sesión de prácticas se realizará una defensa "in situ" de la práctica.

Referencias

[1] *Documentación oficial de Java RMI* `http://docs.oracle.com/javase/6/docs/technotes/guides/rmi/index.html`