



Universidad Isabel I
Programación concurrente y
distribuida

Estudio de caso 5

RMI

Autor:
Tomas Garijo Lopez

Madrid Noviembre 2020

Introducción.

Para este estudio de caso se implementan varias soluciones de ejemplo para comprender como funciona la librería Java Remote Method Invocation (rmi) en java.

En este documento voy a explicar la implementación solución 1.5.1 que es la más extensa y más compleja de implementar, pero no por ella complicada, es el proyecto más completo porque hago uso de rmi, ejecución de rmiregistry por medio de código y no por ejecución directa del sistema operativo, expresiones regulares, acceso y escritura a ficheros y permisos de ejecución. La arquitectura de los demás ejemplos son iguales, con menor complejidad y sin la utilización de algunas de las implementaciones que acabo de exponer.

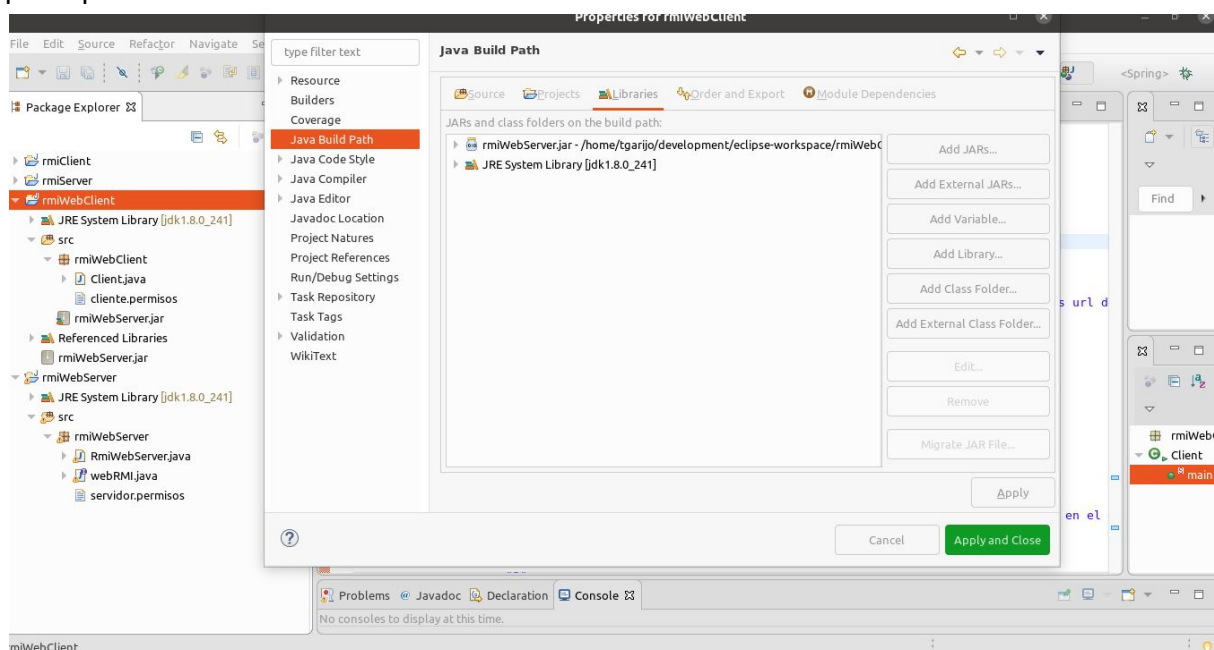
Se ha utilizado un sistema operativo Ubuntu 20 y el entorno de desarrollo de eclipse Versión: 2019-09 R (4.13.0) y el jdk 1.8.0_241.

El proyecto esta ubicado en <https://github.com/tgarijo/javaSistemaDistribuidos.git>

Todos los ejemplos tiene un shell script de ejecución sobre path relativos por lo cual no debería haber ningún problema en la ejecución, los scripts tienen nombres descriptivos, por ejemplo borrarDatos.sh, consultarWebLinks.sh, consultarWeb.sh en la parte cliente y startServer.sh en la parte servidora.

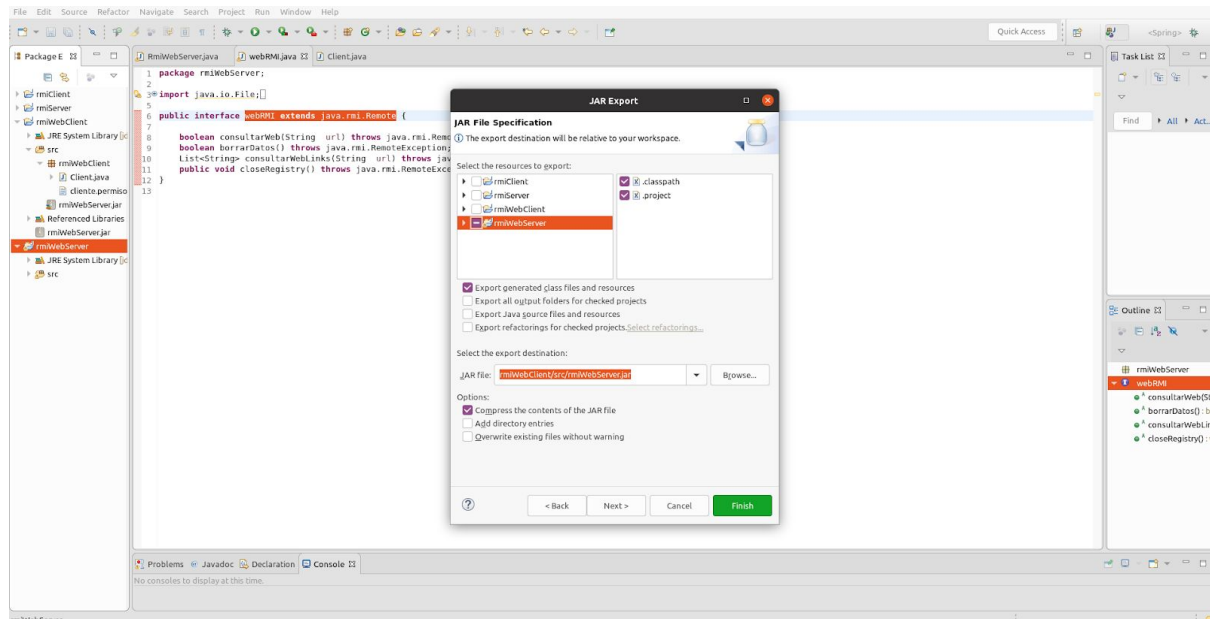
Para la ejecución de estos scripts podemos utilizar `sh ./nombreDeScript.sh` o darle permisos de ejecución una sola vez a los scripts con el comando `chmod +x nombreDeScript.sh` y ejecutarlos `./nombreDeScript`.

Las partes servidoras de los ejemplos son exportadas a un jar que es donde están las implementaciones del servidor. En el entorno de desarrollo (Eclipse) se deberán importar para que el cliente en desarrollo encuentre el interface rmi.



Para exportar un proyecto a un jar en eclipse podemos utilizar jar export del menú file. en este caso se exporta al directorio src, pero en otras implementaciones podríamos utilizar un directorio lib o establecer el path donde se ubica el jar en el classpath.

Exportación de un proyecto a jar



1.5. Cuestiones

1. Crea una solución RMI que ofrezca la siguiente interfaz e implemente un objeto remoto para dicha funcionalidad:

- consultarWeb: Permite obtener el código fuente de una web y guardar el código en un archivo de texto.
- borrarDatos: Permite borrar el archivo descargado.
- consultarWebLinks: Dada una URL permite obtener todos los enlaces de esa web.

La solución se compone de dos programas, el servidor llamado rmiWebserver, que implementa los servicios solicitados en el caso de estudio y un cliente llamado rmiWebClient que hace llamadas a los servicios que expone es servidor.

Servidor.

Los servicios se exponen por medio de un interface webRMI que extiende e la clase java.rmi.Remote y a su vez este interface está implementado en la calse RmiWebServer que es la clase principal del programa que extiende de la clase java.rmi.server.UnicastRemoteObject.

Para superar la excepción en la ejecución del tipo AccessControlException: access denied se ha de establecer la política de seguridad que permite la escucha de puertos sin privilegios y permite que el código acceda a recursos del sistema a los cuales en circunstancias normales no tendría permisos.

Esta seguridad se establece en un fichero llamado `servidor.permisos`, para este ejemplo se han dado todos los permisos, pero se podría establecer mayor atomicidad y ser más específicos de que recursos del sistema puede acceder nuestro código.

Una vez comprobado los permisos el servidor rmi se implementa por medio de una url y un puerto tcp, se puede implementar simplemente con un nombre de servicio, pero particularmente me parece más relevante que se establezca con una `url:port`. En este caso el puerto elegido es el 5080 y empieza la escucha.

Todo estos datos se podrían parametrizar en sucesivas versiones, en este ejemplo se ha harcoreado la parametrización por términos de simplicidad en la ejecución, pero no es una buena practica a la hora de codificar.

Otro punto a destacar para otras versiones sería un sistema de log.

Para el método `consultaWeb`, se espera una url, que viene del cliente y se hace una llamada por medio de la implementación `java.net.URL` para acceder a la url traer el código fuente de la página esto se hace a traves del metodo `getData(url)` el cual devuelve un `BufferedReader`. Una vez traídos los datos se intera el `BufferedReader`, se añade en un `StringBuilder` y se graba en el directorio de ejecución en un fichero llamado `data.html`. Una mejor implementación sería guardar el fichero en un directorio temporal como `tmp`.

La siguiente implementación pedida en la de borrar el fichero de la implementación `consultaWeb`, no tiene ninguna complejidad, ya que lo único que hace es comprobar que el fichero exista y borrarlo.

Para la implementación `consultarWebLinks` que recibe una url, sigue la misma implementación que `consultaWeb`, de hecho unas sucesivas refactorizaciones podrían simplifica llamada a implementaciones iguales, como por ejemplo la interacción de `StringBuilder`. Para obtener las url del sitio configurado guardamos en un fichero llamado `pattern.txt` y por medio de una expresión regular estándar sacamos las urls como se solicita, este fichero no sé borrar por lo cual si hay una nueva llamada a este servicio se sobreescribe el fichero.

Por último y a modo de ejemplo se ha implementado un servicio para que el servicio rmi se cierre si lo solicita el cliente, en este ejemplo siempre se cierra el servicio en el servidor. A modo de log de ejecución se establece una serie de mensajes por consola para que el usuario tenga una idea de lo que ha hecho el programa.

Cliente.

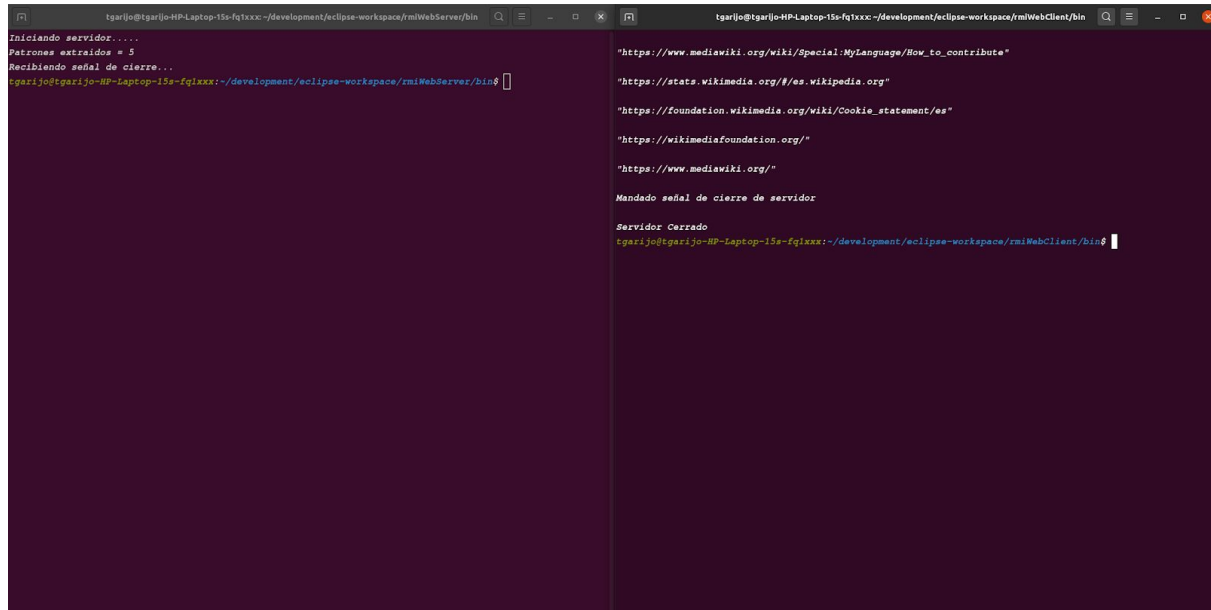
El cliente es una clase que no extiende ni implementa ningún interface. Pero si tiene una llamada al interface `webRMI` del servidor que está contenida en el jar que describimos al principio del documento.

Se hace también uso de los roles de seguridad, para evitar la excepción `AccessControlException: access denied`.

Los scripts de ejecución pasan al programa cliente por medio de parámetros los servicios a ejecutar, para simplificar se ha optado pasar un número que hace referencia al servicio a ejecutar: Uso (1: Serializa web, 2: Borra fichero Serializado, 3: Saca las url del sitio)

El cliente se conecta al servidor rmi por medio de una url:port hace la llamada al los servicios por medio de la instancia webRMI server y saca por consola un log de ejecución.

Veamos unas capturas de la opción 3 : Saca las url del sitio



```
tgar1jo@tgar1jo-HP-Laptop-15s-fq1xxx:~/development/eclipse-workspace/rmiWebServer/bin$
Iniciando servidor....
Patrones extraidos = 5
Recibiendo señal de cierre...
tgar1jo@tgar1jo-HP-Laptop-15s-fq1xxx:~/development/eclipse-workspace/rmiWebServer/bin$

tgar1jo@tgar1jo-HP-Laptop-15s-fq1xxx:~/development/eclipse-workspace/rmiWebClient/bin$
"https://www.mediawiki.org/wiki/Special:MyLanguage/How_to_contribute"
"https://stats.wikimedia.org/es.wikipedia.org"
"https://foundation.wikimedia.org/wiki/Cookie_statement/es"
"https://wikimediafoundation.org/"
"https://www.mediawiki.org/"
Mandado señal de cierre de servidor
Servidor Cerrado
tgar1jo@tgar1jo-HP-Laptop-15s-fq1xxx:~/development/eclipse-workspace/rmiWebClient/bin$
```

Fuentes:

<https://www.adictosaltrabajo.com/2008/02/07/rmi-cdr/>

<https://stackoverflow.com/questions/2427473/java-rmi-accesscontrolexception-access-denied>

<https://mkyong.com/java/java-rmi-hello-world-example/>

<https://stackoverflow.com/questions/237061/using-regular-expressions-to-extract-a-value-in-java>

https://es.wikipedia.org/wiki/Java_Remote_Method_Invocation