Universidad Pública de Navarra
Curso 2021/2022

Gestión de Sistemas de Información

Guión de la Práctica 04

Marilin Vega León, Dpto. de Automática y Computación

1. Contenidos del documento

Este documento recoge la Práctica 04 de la asignatura Gestión de Sistemas de Información correspondiente al curso 2021/2022.

Cada grupo de alumnos debe completar los ejercicios de la Sección 3, tal y como se describe en la Sección 4.

La asistencia al laboratorio es únicamente obligatoria el primero de los días asignados para esta práctica.

Las fechas de interés para el desarrollo de la práctica son las que siguen:

- 11 y 18 de Noviembre: Días de laboratorio asignados para completar la prácticas.
- 23 de Noviembre (14h): Fecha límite de entrega.

2. Introducción a la práctica

A día de hoy no tiene sentido construir un sistema de información que no sea accesible de manera remota, más si cabe con la deslocalización de la lógica de negocio y/o los datos a servicios *en la nube*. Sin esta capacidad, un sistema de información sólo sería accesible por un operador humano en la máquina que ejecute, con todas las desventajas que esto supone. Existen muchas maneras de implementar la accesibilidad de un sistema de información, cada una de ellas aportando una serie de ventajas e inconvenientes. En esta práctica vamos a experimentar con las tecnologías de acceso remoto a objetos de la plataforma Java, en concreto el Remote Method Invocation (RMI).

Java proporciona varias tecnologías nativas de acceso remoto, la más sencilla de las cuales es RMI. Con esta tecnología una máquina cualquiera puede hacer sus objetos accesibles a cualquier otra máquina, y hacer que dichas máquinas sean capaces de invocar sus métodos. No es este el lugar para hacer una descripción detallada de la tecnología, pero cabe mencionar los siguientes hechos destacados:

- Aunque RMI permite publicar objetos, la visibilidad pública de la misma se da en forma de interfaces. Es decir, aunque se publica un objeto instanciable, las máquinas remotas sólo conocerán que es una implementación de un `interface` dado. De esta manera, se dan a conocer los métodos que se quieren publicar, pero nunca los detalles de su implementación (ni siquiera la clase concreta que los implementa). Esta decisión es sumamente práctica, y tiene influencias en seguridad y buen diseño.
- La invocación de los métodos remotos requiere el paso de argumentos, y por lo tanto la serialización de los mismos. La serialización se da usando las tecnologías de serialización de Java, por lo cual hay que ser cuidadosos en la elección de los argumentos, en el uso de las clausulas `trasient` correspondientes, etc.

- Al usar RMI, la codificación de la invocación de métodos remotos se asemeja a la de métodos locales. Sin embargo, la invocación remota tiene problemáticas asociadas a la naturaleza distribuida del proceso, y esto deriva en el uso recurrente de excepciones. Es muy importante saber manejarlas, e incluso generar código capaz de reintentar invocaciones en caso de `RemoteException`.
- La localización de objetos compartidos se puede dar de muchas formas, e incluso se pueden crear registros personalizados al efecto. Sin embargo, es común utilizar el RMI Registry proporcionado por la JVM, que permite publicar objetos identificables por un *tag* concreto. Este registro se puede colocar en cualquier puerto, si bien el 1099 es la opción por defecto.

El uso de RMI es relativamente sencillo, y existen abundantes ejemplos accesibles en Internet, tanto en páginas oficiales de desarrolladores como en páginas independientes. Si tiene dudas acerca del uso de RMI, puede consultar el proyecto NetBeans RemoteCalc disponible en la sección de Recursos de esta misma asignatura en MiAulario.

3. Lista de ejercicios

En esta práctica vamos a generar dos accesos remotos diferentes a nuestro sistema de información. Uno de ellos está dirigido a administrativos, de manera que puedan acceder a la información acerca de conciertos y otros eventos, y modificarla a su conveniencia. El segundo acceso se dedicará al personal de ventas, y se centra en el registro y actualización de información acerca de clientes y sus interacciones con la empresa.

Ejercicio 1. Descargue de la carpeta correspondiente de MiAulario los *interfaces*, `ClientGateway` y AdminGateway , ambos pertenecientes al paquete `GSILabs.connect`. Ejecute una de las siguientes dos opciones:

a) Haga que la clase `BusinessSystem` los implemente.

b) Extienda la clase `BusinessSystem` con una clase `PublicBusinessSystem` y haga que dicha clase los implemente.

En cualquier caso, genere una pequeña justificación de su decisión (máximo 300 palabras), que deberá adjuntarse en la entrega, dentro de un archivo llamado GrXXP04Ej01.pdf.

□

Hay varias ventajas en la generación de diferentes interfaces de acceso. Principalmente, tenemos el hecho de que no todas las aplicaciones clientes tendrán la misma visibilidad del sistema de información, y así podremos reducir la funcionalidad a la que cada una puede acceder. Aunque en el lado servidor tengamos un único objeto sirviendo toda la información, es una buena idea hacerlo aparecer bajo diferentes identidades, en este caso representadas por `interface`'s.

Ejercicio 2. Cree dentro del paquete `connect` una clase ejecutable `BusinessServer` que cree una instancia de la clase elegida en el ejercicio anterior para implementar los interfaces. Luego, debe poblar dicha instancia de la manera que prefiera (programáticamente, vía ods/xml, etc.) y publicarlo en un registro RMI en el puerto 1099 bajo dos identificadores diferentes `ClientGateway` y `AdminGateway` (uno por interfaz). En caso de que el uso de un mismo puerto sea problemático, puede usar dos puertos diferentes.

Para ello, debe hacer lo siguiente.

1. Cree una instancia del objeto a publicar;

2. Genere un stub del objeto usando el método `exportObject` de la clase `UnicastRemoteObject`.
3. Cree un registro en el puerto 1099 usando el comando `LocateRegistry.createRegistry`. En caso de que sea necesario, haga lo propio en el puerto 1100.
4. Dentro de ese (esos) registro(s), asocie el *stub* a los identificadores mencionados anteriomente.

Si la consecución de este ejercicio requiere la modificación de las clases del paquete `BModel`, enumere los cambios realizados en un breve informe (máximo 300 palabras). Justifique en el mismo informe la necesidad de dichos cambios.

□

Con el ejercicio anterior colocamos un servidor de información de manera totalmente transparente. De hecho, no necesitamos preocuparnos de las interioridades de la comunicación y, amén de implementar ciertos interfaces y gestionar las posibles excepciones, podemos generar toda la lógica de nuestro sistema en términos de programa. Al lanzar el servidor, hemos abierto una puerta a la comunicación con objetos remotos para el servicio de información, pero no hemos realizado gestión alguna referente a las conexiones, representación de información etc.

Ejercicio 3. Cree dentro del paquete `connect` dos clases ejecutables `ClientHub` y `AdminHub` que sean capaces de conectarse a un servidor como el creado en el ejercicio anterior. Para ello, ambas clases ejecutables deben seguir los siguientes pasos:

1. Pedir al usuario, por línea de comandos, la dirección en la que está el servidor;
2. Pedir al usuario, por línea de comandos, el puerto en el que el servidor aguarda conexiones;
3. Pedir al usuario, por línea de comandos, el *tag* del objeto remoto a contactar, en términos del interfaz remoto correspondiente;
4. Ejecutar las funcionalidades proporcionadas por dicho objeto (es decir, ejecutar alguno de sus métodos) para comprobar que la conexión se realiza con éxito.

No es necesario que los clientes sean interactivos o gráficos en el punto 4, basta con que realicen operaciones predefinidas en el código y muestren por pantalla el resultado. El objetivo de estos clientes no es tanto ejercer de clientes funcionales como visualizar las facilidades que aporta RMI para la invocación de métodos en objetos remotos (almacenados en el servidor) como si fueran locales.

Pruebe su funcionamiento usando máquinas diferentes y testee las siguientes funcionalidades:

- Ambos pueden conectarse al servidor remoto;
- Si el servidor no está disponible o se cae, ambos son capaces de manejar la situación, generar un aviso y permanecer a la espera;
- Cualquier otra prueba que se os ocurra.

□

Al generar los clientes, cerramos el círculo de la invocación remota de objetos. Desde el punto de vista del programador, la situación ideal es aquella en la que el proceso de comunicación es totalmente transparente, para así poder centrarse en escribir el programa. Al usar RMI, y tras una gestión inicial (conexión con registro y objeto remoto), podemos invocar métodos sobre un objeto remoto como si fuera local. Cada vez que lo hacemos, los argumentos son serializados hacia el servidor, ejecutados en el mismo, y el resultado nos es devuelto. Sin embargo, todo esto es gestionado por la JVM, y por tanto transparente al programador. Hasta cierto punto.

-
-
-

4. Metodología de entrega

La nota se asignará, sobre un máximo de 9 puntos, en función de los siguientes criterios:

e1.1 Se ha completado con éxito el Ejercicio 1 (3 puntos), y se ha incluido una justificación adecuada para la decisión tomada (1 punto);

e1.2 Se ha completado con éxito el Ejercicio 2 (3 puntos);

e1.3 Se ha completado con éxito el Ejercicio 3 (3 puntos);

Cada uno de los criterios podrá evaluarse como *fallido* (0 %), *incompleto* (25 %, 50 %, 75 %) o *completo* (100 %). Téngase en cuenta que, dentro de la completitud de los mismos, se consideran cosas como la adecuada documentación, la limpieza del código, etc. Además, como en prácticas anteriores, el informe deberá estar escrito siguiendo ciertas convenciones y criterios de calidad, y recibirá automáticamente una puntuación de 0 si contiene faltas de ortografía.

Control de versiones

Este documento no ha sufrido modificaciones desde su publicación (xx/xx/2021).