RMI REGISTRY

Equipo 6
Garcia Hernandez Manuel Hoscani
Rodriguez Sanchez Daniel Tapia
Delgadillo Diego Alejandro

¿QUÉ ES RMI?

La invocación de métodos remotos en Java (RMI, por sus siglas en inglés) es un ejemplo de mecanismo de computación distribuida con un protocolo bien definido. Uno de los objetivos principales de los diseñadores de RMI fue permitir a los programadores desarrollar programas Java distribuidos con la misma sintaxis y semántica que los programas no distribuidos. La arquitectura de RMI define el comportamiento de los objetos, cómo y cuándo pueden ocurrir excepciones, cómo se gestiona la memoria y cómo se pasan y devuelven los parámetros en los métodos remotos. El sistema RMI debe realizar la recolección de basura de objetos remotos y permitir extensiones como la replicación de servidores y la activación de objetos persistentes para atender una invocación. Estas extensiones son transparentes para el cliente y añaden requisitos mínimos de implementación para los servidores que las usan.

¿QUÉ ES RMI?

RMI permite a una computadora cliente acceder a un objeto ubicado en un servidor remoto como si fuera un objeto instanciado localmente en el cliente.

ARQUITECTURA DE RMI

RMI se compone de tres capas principales: la capa de stub/skeleton, la capa de referencia remota (RRL) y la capa de transporte. Estas capas permiten una sustitución independiente y son responsables de la comunicación entre el cliente y el servidor.

ARQUITECTURA DE RMI

- La capa de transporte usa conexiones TCP/IP para establecer y gestionar conexiones, manejar llamadas entrantes y mantener una tabla de objetos remotos locales.

- La capa de referencia remota (RRL) gestiona las referencias de los clientes a los objetos remotos, estableciendo y gestionando conexiones, y escuchando llamadas entrantes.

ARQUITECTURA DE RMI

- El stub actúa como proxy del lado del cliente, iniciando llamadas remotas, empaquetando argumentos, notificando a la RRL, procesando respuestas y finalizando llamadas.

- El skeleton, es una entidad del lado del servidor que despacha llamadas a la implementación real del objeto remoto, procesando argumentos entrantes, llamando al método correspondiente y empaquetando los valores de retorno.

¿QUE ES RMI REGISTRY?

Es una función de nombres simple que proporciona RMI que utiliza los metodos estaticos de la clase **java.rmi.Naming** la cual consta de los siguientes cinco métodos estáticos:

Los métodos del código de lanzamiento, bind(), unbind() y rebind(), tratan con la mecánica de insertar y eliminar servidores de un registro.

Los métodos del cliente, list() y lookup(), tratan de consultar un registro para encontrar un servidor.

BIND(), REBIND() Y UNBIND()

Su estructura es:

- public static void bind(String name, Remote obj)
- public static void rebind(String name, Remote obj)
- public static void unbind(String name)

Estos tres métodos se ocupan de vincular o desvincular un servidor en el registro.Lo primero que hacen es analizar name para averiguar dónde se está ejecutando el registro. Name es una combinación de la ubicación del registro RMI y el nombre lógico del servidor. Es decir, es una URL con el siguiente formato: //registryHost:port/nombre_lógico

BIND(), REBIND() Y UNBIND()

Después de analizar name, estos métodos forman una conexión de socket con el registro real. El registro es un servidor RMI que se ejecuta en la máquina host nombrada y escucha en el puerto indicado. El registro procede a:

- Rechazar la solicitud: si se solicitó una vinculación y otro objeto ya ha sido vinculado al nombre lógico, la solicitud será rechazada.

- Vincular el objeto: nombre_lógico se asociará con el objeto deserializado que se recibió.

LOOKUP() Y LIST()

Ambos son similares entre si. En el caso de lookup(), name es una URL con el mismo formato que la URL pasada a bind(), rebind() y unbind(). Este lanza una excepción o devuelve un solo stub a la aplicación que llama; el cliente puede usar este stub para llamar directamente a los métodos en el servidor, sin volver a usar el registro. Su estructura es:

- public static Remote lookup (String name)

En el caso de list(), el argumento debe ser más corto, especificando solo la máquina y el puerto para el registro RMI, pero no un servidor específico. Por otra parte list() devuelve un arreglo en forma de cadenas de caracteres con las URLs completas. Su estructura es:

- public static String [ ] list(String name)

SEGURIDAD EN RMI REGISTRY

Un hacker creo un programa que escanea Internet buscando registros RMI. Lo hace simplemente intentando conectar a cada puerto en cada máquina que encuentra. Siempre que el programa encuentra un registro RMI en funcionamiento, inmediatamente usa el método list() para encontrar los nombres de todos los servidores que se ejecutan en el registro. El programa llama a rebind() y reemplaza cada stub en el registro con un stub que apunta a su propio servidor.

Por lo tanto si no restringes el acceso a un servicio de nombres, tu red se vuelve increíblemente vulnerable. El propio servicio de nombres es un punto vulnerable y necesita ser protegido.

SEGURIDAD EN RMI REGISTRY

La solución que adoptó el registro RMI fue la siguiente:
Cualquier llamada que vincule un servidor en el registro debe originarse desde un proceso que se ejecute en la misma máquina que el registro. Esto no impide que los hackers descubran qué servidores están en funcionamiento, ni que llamen a métodos, pero sí impide que reemplacen cualquiera de los servidores y alteren la estructura de las aplicaciones cliente-servidor.

LIMITACIONES DE RMI REGISTRY

El Registro RMI tiene varias ventajas, como su fácil administración y uso, además de ser rápido y fácil de encontrar para los clientes. Sin embargo, presenta limitaciones significativas cuando se utiliza como un servicio de nombres.

VENTAJAS

1. Fácil de administrar: El JDK incluye la aplicación rmiregistry, que es fácil de lanzar y no requiere acciones adicionales.
2. Fácil de encontrar: Usa un puerto estándar (1099) y los clientes solo necesitan conocer la máquina que ejecuta el registro.
3. Fácil de usar: La interfaz tiene solo cinco métodos simples con argumentos predeterminados razonables.
4. Rápido: Los cinco métodos son muy rápidos.

PROBLEMAS Y LIMITACIONES

1. Ejemplo del cliente de impresoras: Una aplicación necesita encontrar un servidor de impresoras y enviar un documento. Los métodos list() y lookup() del registro pueden ser ineficaces.
2. Desperdicio de ancho de banda: Obtener nombres y recuperar información irrelevante puede resultar en un uso ineficiente de recursos. Descargar todo el contenido del registro para buscar una impresora es ineficiente.
3. Selección de impresora predeterminada: Es posible almacenar la elección del usuario localmente y reutilizarla para evitar descargar todas las entradas cada vez.
4. Diversidad de impresoras: Los usuarios pueden querer seleccionar impresoras según la calidad de impresión, tipo de archivo y ubicación, lo cual el registro RMI no maneja adecuadamente.

CONCLUSIONES

En conclusión, hemos explorado de manera exhaustiva el Registro RMI y su papel central en el desarrollo de aplicaciones distribuidas en Java. Hemos comprendido cómo el Registro RMI facilita la comunicación entre componentes distribuidos, permitiendo que los objetos se registren y actualicen dinámicamente en entornos de red. Además, hemos destacado la importancia de seguir las mejores prácticas para garantizar una implementación efectiva y segura del Registro RMI en nuestros proyectos. Al comprender mejor este componente crucial, estamos equipados para diseñar y desarrollar sistemas distribuidos más robustos y eficientes en el futuro.

REFERENCIAS

- Grosso, W. (2002). Java RMI. «O'Reilly Media, Inc.» Pag. 260 - 272

- Kaur, H. (2008). An alternative architecture for improving availability and flexibility of RMI registry (Tesis de Maestría). Punjab Technical University, Jalandhar, India, Pag. 7 - 14