Java RMI Evaluation

# Página web

Página web de formato vertical con el menú en la parte de arriba. Al ser la página web de Oracle, es una web muy compleja con muchos productos. Java RMI no está ubicado desde el inicio. Pensado para programadores de Java que conocen bien los productos y no quieres salirse del ámbito Java.

# Documentación

## Información en la web

## Manuales

## Instalación

Java RMI viene con el paquete de instalación de Java SE.

# Software

Java RMI usa varios servicios para encontrar los objetos remotos. Por ejemplo un servidor puede registrar usando la aplicación *RMI registry* su objeto remote y entonces un cliente puede encontrarlo.

La ventaja de Java RMI es que si la clase de un objeto no se encuentra definida en el cliente, su definición se puede descargar.

Java RMI trata los objetos remotos de diferente forma en el cliente. En vez de hacer una copia de la implementación del objeto, RMI pasa un stub remote del objeto remoto. Este stub implementa el mismo conjunto que la interfaz.

La arquitectura RMI puede verse como un modelo de cuatro capas.

### Primera capa

La primera capa es la de aplicación y se corresponde con la implementación real de las aplicaciones cliente y servidor. Aquí tienen lugar las llamadas a alto nivel para acceder y exportar objetos remotos. Cualquier aplicación que quiera que sus métodos estén disponibles para su acceso por clientes remotos debe declarar dichos métodos en una [interfaz](http://es.wikipedia.org/wiki/Interfaz_%28Java%29) que extienda java.rmi.Remote. Dicha interfaz se usa básicamente para "marcar" un objeto como remotamente accesible. Una vez que los métodos han sido implementados, el objeto debe ser exportado. Esto puede hacerse de forma implícita si el objeto extiende la clase UnicastRemoteObject (paquete java.rmi.server), o puede hacerse de forma explícita con una llamada al método exportObject() del mismo paquete.

### Segunda capa

La capa 2 es la capa proxy, o capa stub-skeleton. Esta capa es la que interactúa directamente con la capa de aplicación. Todas las llamadas a objetos remotos y acciones junto con sus parámetros y retorno de objetos tienen lugar en esta capa.

### Tercera capa

La capa 3 es la de referencia remota, y es responsable del manejo de la parte semántica de las invocaciones remotas. También es responsable de la gestión de la replicación de objetos y realización de tareas específicas de la implementación con los objetos remotos, como el establecimiento de las persistencias semánticas y estrategias adecuadas para la recuperación de conexiones perdidas. En esta capa se espera una conexión de tipo stream (stream-oriented connection) desde la capa de transporte.

### Cuarta Capa

La capa 4 es la de transporte. Es la responsable de realizar las conexiones necesarias y manejo del transporte de los datos de una máquina a otra. El protocolo de transporte subyacente para RMI es JRMP (Java Remote Method Protocol), que solamente es "comprendido" por programas Java.

## API

Para que un objeto se convierta en remote, la interfaz debe heredar de java.rmi.Remote y cada método de la interfaz declara java.rmi.RemoteException en su clausula throws.

## Lenguajes de programación soportados

Permite programar aplicaciones distribuidas solamente en Java.

## Plataformas

Windows, Linux in 32bit and 64bit.  
Solaris.

## Tipos soportados

Todos los tipos Java, incluyendo las clases.

## Modelo de programación

La programación con Java RMI consiste en cuatro pasos:

1. Definir los la interfaz remota.
2. Implementar los objetos remotos.
3. Implementar los clientes.

Desde el cliente, la forma de conseguir el objeto a usar es:

if (System.getSecurityManager() == null) {

System.setSecurityManager(new SecurityManager());

}

try {

String name = "Compute";

Registry registry = LocateRegistry.getRegistry(args[0]);

Compute comp = (Compute) registry.lookup(name);

Pi task = new Pi(Integer.parseInt(args[1]));

BigDecimal pi = comp.executeTask(task);

System.out.println(pi);

} catch (Exception e) {

System.err.println("ComputePi exception:");

e.printStackTrace();

}

La forma de crear el objeto en el servidor es:

if (System.getSecurityManager() == null) {

System.setSecurityManager(new SecurityManager());

}

try {

String name = "Compute";

Compute engine = new ComputeEngine();

Compute stub =

(Compute) UnicastRemoteObject.exportObject(engine, 0);

Registry registry = LocateRegistry.getRegistry();

registry.rebind(name, stub);

System.out.println("ComputeEngine bound");

} catch (Exception e) {

System.err.println("ComputeEngine exception:");

e.printStackTrace();

}

La configuración de Java RMI se hace por parámetros de la máquina virtual:

* java.rmi.server.codebase
* java.rmi.server.hostname
* java.security.policy

## Instalación

# Detalles técnicos

## Rendimiento

En la siguiente web:

<http://daniel.gredler.net/2008/01/07/java-remoting-protocol-benchmarks/>

Tiene mejor rendimiento que un Web Service por XML, pero serializa más lento que Protocol Buffer o Thrift.

# Modelo de negocio

## Licencia

## Proyectos que lo usan

# Notas