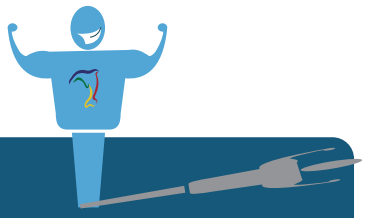
autentia

Avenida de Castilla,1 - Edificio Best Point - Oficina 21B
28830 San Fernando de Henares (Madrid)
tel./fax: +34 91 675 33 06
info@autentia.com - **www.autentia.com**

¿**Qué ofrece** Autentia Real Business Solutions S.L?

Somos su empresa de **Soporte a Desarrollo Informático**.
Ese apoyo que siempre quiso tener...

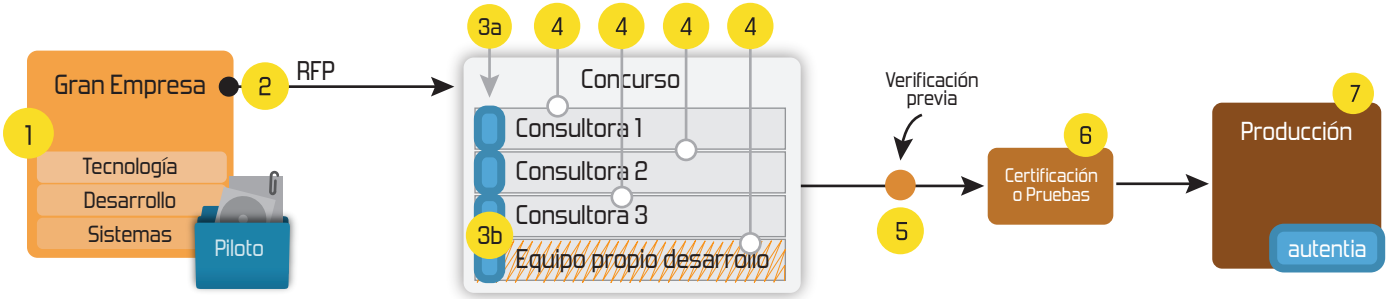
1. Desarrollo de componentes y proyectos a medida



2. Auditoría de código y recomendaciones de mejora

3. Arranque de proyectos basados en nuevas tecnologías

1. Definición de frameworks corporativos.
2. Transferencia de conocimiento de nuevas arquitecturas.
3. Soporte al arranque de proyectos.
4. Auditoría preventiva periódica de calidad.
5. Revisión previa a la certificación de proyectos.
6. Extensión de capacidad de equipos de calidad.
7. Identificación de problemas en producción.



4. Cursos de formación (impartidos por desarrolladores en activo)

Spring MVC, JSF-PrimeFaces /RichFaces, HTML5, CSS3, JavaScript-jQuery

Gestor portales (Liferay)
Gestor de contenidos (Alfresco)
Aplicaciones híbridas

Control de autenticación y acceso (Spring Security)
UDDI
Web Services
Rest Services
Social SSO
SSO (Cas)

JPA-Hibernate, MyBatis
Motor de búsqueda empresarial (Solr)
ETL (Talend)

Dirección de Proyectos Informáticos.
Metodologías ágiles
Patrones de diseño
TDD

Tareas programadas (Quartz)
Gestor documental (Alfresco)
Inversión de control (Spring)

BPM (jBPM o Bonita)
Generación de informes (JasperReport)
ESB (Open ESB)

Compartimos nuestro conocimiento en:
www.adictosaltrabajo.com

Para más información visítenos en:
www.autentia.com

Tutorial desarrollado por:

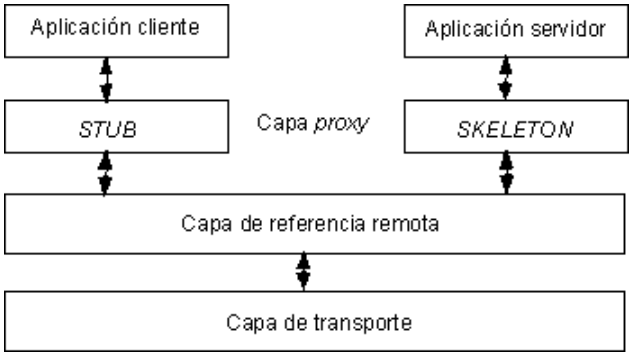
Enrique Medina Montenegro

Puedes ver mi CV y contratarme en:

PROGRAMACIÓN DISTRIBUIDA CON RMI

RMI (*Remote Method Invocation*) es un mecanismo que permite realizar llamadas a métodos de objetos remotos situados en distintas (o misma) máquinas virtuales de Java, compartiendo así recursos y carga de procesamiento a través de varios sistemas.

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



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 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.

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.

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.

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.

Toda aplicación RMI normalmente se descompone en 2 partes:

- Un **servidor**, que crea algunos objetos remotos, crea referencias para hacerlos accesibles, y espera a que el cliente los invoque.
- Un **cliente**, que obtiene una referencia a objetos remotos en el servidor, y los invoca.

Crear un servidor RMI

Un servidor RMI consiste en definir un objeto remoto que va a ser utilizado por los clientes. Para crear un objeto remoto, se define una **interfaz**, y el objeto remoto será una **clase** que implemente dicha interfaz. Veamos como crear un servidor de ejemplo mediante 3 pas

1. **Definir el interfaz remoto**

   Cuando se crea un interfaz remoto:

   - El interfaz debe ser **público**.
   - Debe extender (heredar de) el interfaz java.rmi.Remote, para indicar que puede llamarse desde cualquier máquina virtua Java.

- Cada método remoto debe lanzar la excepción java.rmi.RemoteException en su claúsula *throws*, además de las excepcior que pueda manejar.

Veamos un ejemplo de interfaz remoto:

```
public interface MiInterfazRemoto extends java.rmi.Remote
{
  public void miMetodo1() throws java.rmi.RemoteException;
  public int miMetodo2() throws java.rmi.RemoteException;
}
```

2. **Implementar el interfaz remoto**

```
public class MiClaseRemota
extends java.rmi.server.UnicastRemoteObject
implements MiInterfazRemoto
{
  public MiClaseRemota() throws java.rmi.RemoteException
  {
    // Código del constructor
  }

  public void miMetodo1() throws java.rmi.RemoteException
  {
    // Aquí ponemos el código que queramos
    System.out.println("Estoy en miMetodo1()");
  }

  public int miMetodo2() throws java.rmi.RemoteException
  {
    return 5; // Aquí ponemos el código que queramos
  }

  public void otroMetodo()
  {
    // Si definimos otro método, éste no podría llamarse
    // remotamente al no ser del interfaz remoto
  }

  public static void main(String[] args)
  {
    try
    {
      MiInterfazRemoto mir = new MiClaseRemota();
      java.rmi.Naming.rebind("//" + java.net.InetAddress.getLocalHost().getHostAddress() +
                             ":" + args[0] + "/PruebaRMI", mir);
    }
    catch (Exception e)
    {
    }
  }
}
```

Como se puede observar, la clase MiClaseRemota implementa el interfaz MiInterfazRemoto que hemos definido previamente. Además, hereda de UnicastRemoteObject, que es una clase de Java que podemos utilizar como superclase para implementar objetos remotos.

Luego, dentro de la clase, definimos un **constructor** (que lanza la excepción RemoteException porque también la lanza la superclase UnicastRemoteObject), y los **métodos** de la/las interfaz/interfaces que implemente.

Finalmente, en el método **main**, definimos el código para crear el objeto remoto que se quiere compartir y hacer el objeto remot visible para los clientes, mediante la clase Naming y su método rebind(...).

Nota: Hemos puesto el método main() dentro de la misma clase por comodidad. Podría definirse otra clase aparte que fuera la encargada de registrar el objeto remoto.

3. **Compilar y ejecutar el servidor**

Ya tenemos definido el servidor. Ahora tenemos que **compilar** sus clases mediante los siguientes pasos:

- Compilamos el interfaz remoto. Además lo agrupamos en un fichero JAR para tenerlo presente tanto en el cliente como el el servidor:

  ```
  javac MiInterfazRemoto.java
  jar cvf objRemotos.jar MiInterfazRemoto.class
  ```

- Luego, **compilamos las clases** que implementen los interfaces. Y para cada una de ellas generamos los ficheros Stub y Skeleton para mantener la referencia con el objeto remoto, mediante el comando rmic:

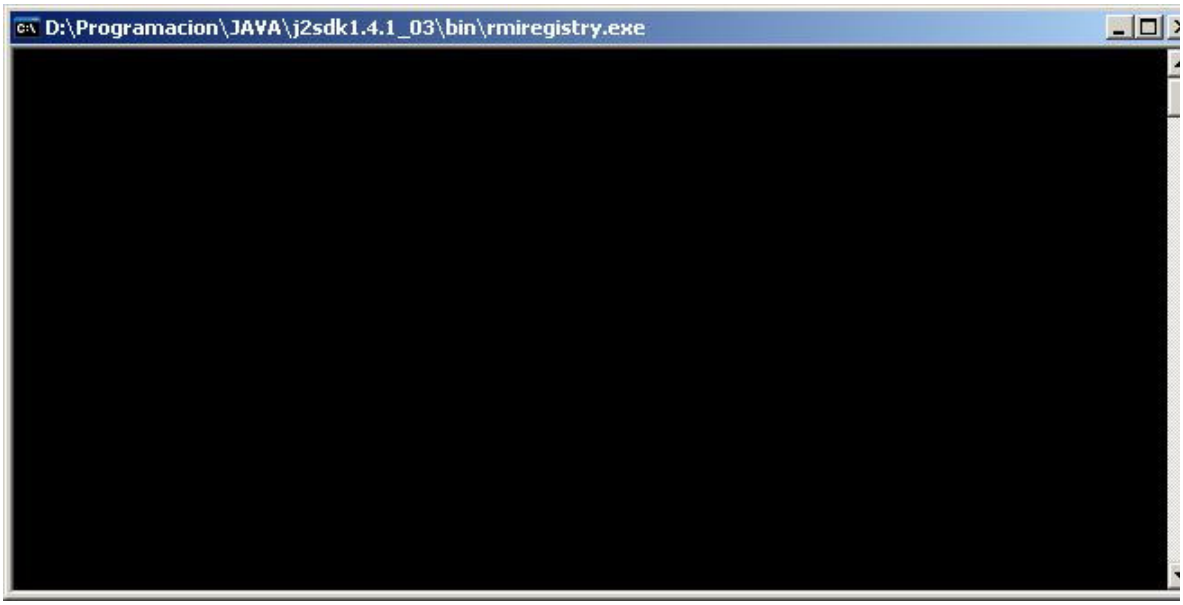
```
set CLASSPATH=%CLASSPATH%;.\objRemotos.jar;.
javac MiClaseRemota.java
rmic -d . MiClaseRemota
```

Observamos en nuestro directorio de trabajo que se han generado automáticamente dos ficheros .class (MiClaseRemota_Skel.class y MiClaseRemota_Stub.class) correspondientes a la **capa stub-skeleton** de la arquitectura RMI.

Para **ejecutar** el servidor, seguimos los siguientes pasos:

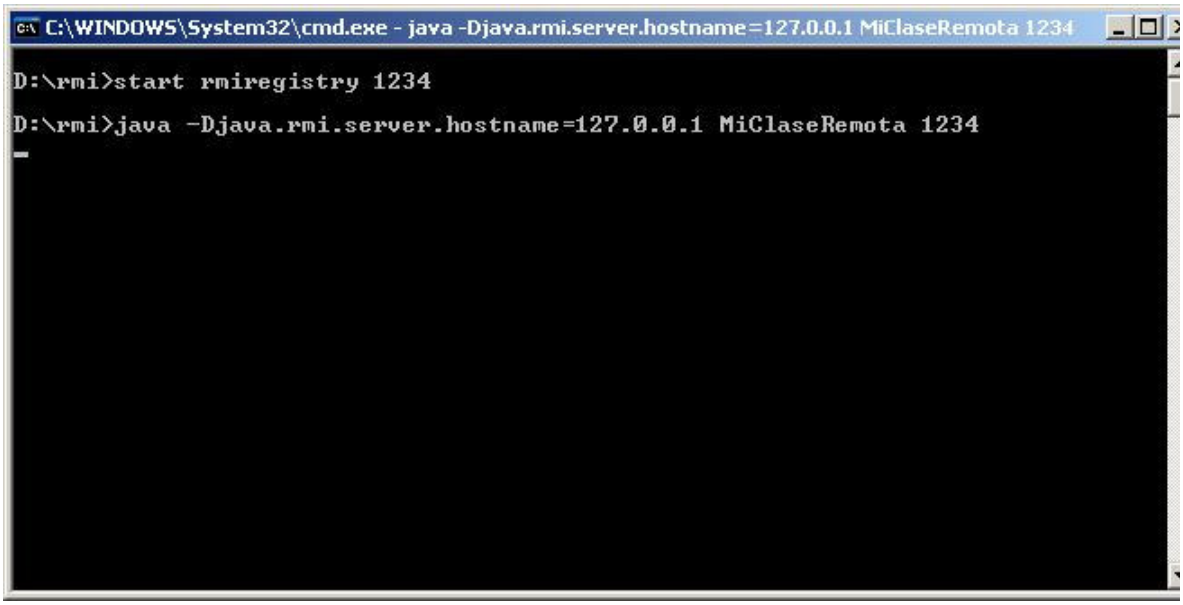
- Se arranca el **registro de RMI** para permitir registrar y buscar objetos remotos. El registro se encarga de gestionar un conjunto de objetos remotos a compartir, y buscarlos ante las peticiones de los clientes. Se ejecuta con la aplicación rmiregistry distribuida con Java, a la que podemos pasarle opcionalmente el puerto por el que conectar (por defecto, el 1099):

  start rmiregistry 1234



- Por último, **se lanza el servidor**:

  java -Djava.rmi.server.hostname=127.0.0.1 MiClaseRemota 1234



Crear un cliente RMI

Vamos ahora a definir un cliente que accederá a el/los objeto/s remoto/s que creemos. Para ello seguimos los siguientes pasos:

1. **Definir la clase para obtener los objetos remotos necesarios**

La siguiente clase obtiene un objeto de tipo MiInterfazRemoto, implementado en nuestro servidor:

```
public class MiClienteRMI
{
  public static void main(String[] args)
  {
    try
    {
      MiInterfazRemoto mir = (MiInterfazRemoto)java.rmi.Naming.lookup("//" +
                        args[0] + :" + args[1] + "/PruebaRMI");

      // Imprimimos miMetodo1() tantas veces como devuelva miMetodo2()
      for (int i=1;i<=mir.miMetodo2();i++) mir.miMetodo1();
      }
      catch (Exception e)
      {
        e.printStackTrace();
      }
    }
  }
}
```

Como se puede observar, simplemente consiste en **buscar el objeto remoto** en el registro RMI de la máquina remota. Para ello usamos la clase Naming y su método lookup(...).

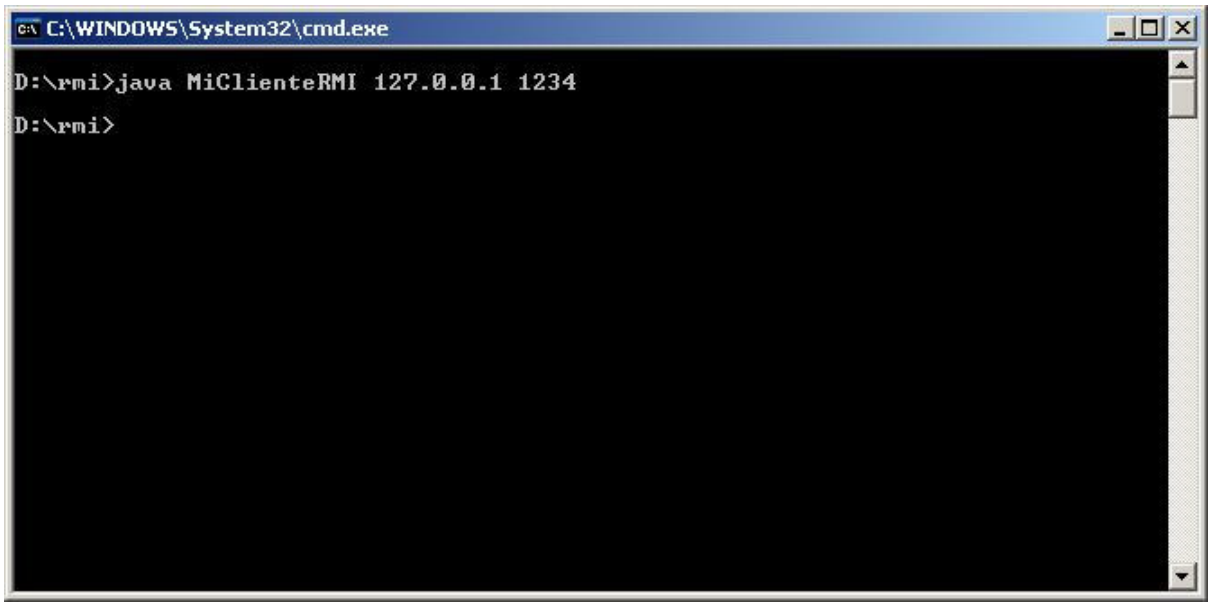
2. **Compilar y ejecutar el cliente**

Una vez que ya tenemos definido el cliente, para compilarlo hacemos:

```
set CLASSPATH=%CLASSPATH%;.\objRemotos.jar;.
javac MiClienteRMI.java
```

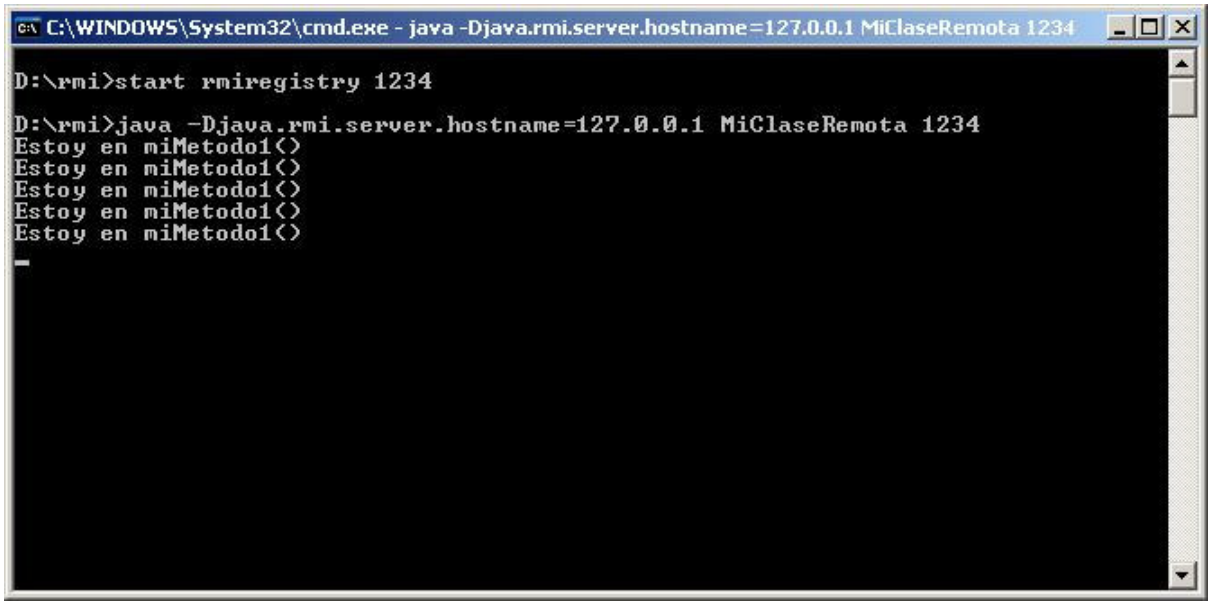
Luego, para ejecutar el cliente hacemos:

```
java MiClienteRMI 127.0.0.1 1234
```



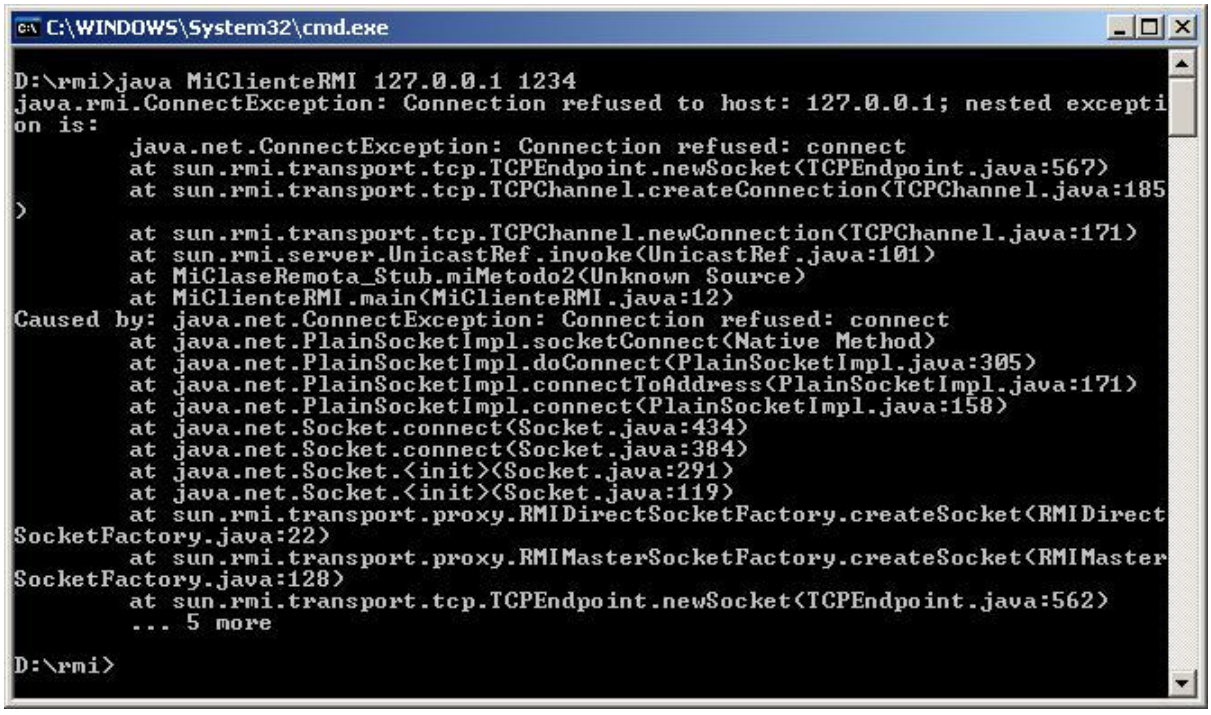
Se debe poder acceder al fichero Stub de la clase remota. Para ello, o bien lo copiamos al cliente y lo incluimos en su CLASSPATH o lo eliminamos del CLASSPATH del servidor e incluimos su ruta en el java.rmi.codebase del servidor (si no se elimina del CLASSPATH del servidor, se ignorará la opción java.rmi.codebase, y el cliente no podrá acceder al Stub)..

Si echamos un vistazo a la ventana donde está ejecutándose el servidor RMI, veremos como se ha encontrado el objeto remoto y ejecutado sus métodos:



Resumen

En este ejemplo sencillo hemos visto como utilizar la tecnología RMI para trabajar con objetos remotos que nos van a permitir crear aplicaciones distribuidas. Para ello, hemos registrado el objeto servidor mediante el servicio de nombres RMI (naming/registry service) desde una máquina virtual de Java en ejecución, de forma que si la máquina virtual Java termina, el objeto deja de existir y provocará u excepción al invocarlo:



En un próximo tutorial veremos cómo utilizar el servicio de activación de objetos directamente desde el cliente de forma automática (se crea el objeto servidor desde una máquina virtual existente o nueva), registrando un método de activación a través del demonio de RMI del host. Esto nos permite optimizar recursos, ya que el objeto servidor sólo es creado cuando se necesita.

Autor: Enrique Medina Montenegro (c) 2003

---

y muchas otras cosas

Nuevo servicio de notificaciones

Si deseas que te enviemos un correo electrónico cuando introduzcamos nuevos tutoriales, inserta tu dirección de correo en el siguiente formulario.

Subscribirse a Novedades	
e-mail	
	Enviar

Otros Tutoriales Recomendados (También ver todos)

Nombre Corto	Descripción
Activación automatica de servicios RMI	En este tutorial veremos cómo utilizar el servicio de activación de objetos RMI directamente desde el cliente de forma automática
Escritura log con Fichero UDP y JMS	Os mostramos ejemplos para cuantificar el coste de escritura de Logs por pantalla, fichero, UDP y JMS (describiendo como configurar el entorno)
Escritura log con Fichero UDP y JMS	Os mostramos ejemplos para cuantificar el coste de escritura de Logs por pantalla, fichero, UDP y JMS (describiendo como configurar el entorno)
Construir un Servidor Web en Java	En este tutorial os enseñamos los principios de las aplicaciones multi-hilo a través de la creación de un servidor web básico en Java. Podremos ver en un ejemplo real el uso de sockets, threads, excepciones, etc.
Instalar JBoss	Os mostramos como instalar en servidor gratuito de aplicaciones JBOSS así como a automatizar su arranque y parada.
Configuración y acceso a OpenLdap desde Java con JNDI	Con este tutorial, aprenderás como realizar la instalación de OpenLdap, así como la carga de un LDIFF básico, y a configurar el entorno Java para acceder a la información.
EJB´s y Orion	Recreación de la guía paso a paso de como crear una aplicación Web con EJB´s y Servlets y su despliegue con ANT sobre Orion

www.AdictosAlTrabajo.com Opimizado 800X600