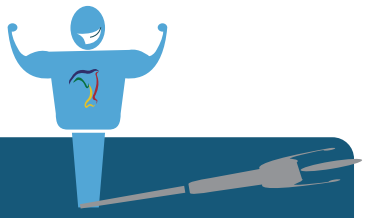
autentia

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...

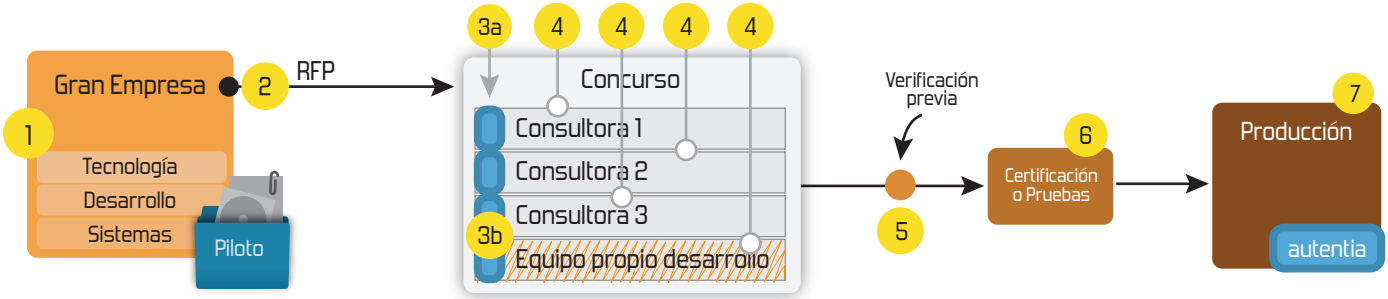
1. Desarrollo de componentes y proyectos a medida



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 (2)

En el anterior tutorial, Aplicación básica con RMI, vimos como usar RMI para trabajar con objetos remotos registrando el objeto servidor mediante el servicio de nombres RMI desde una máquina virtual de Java en ejecución, de forma que si la máquina virtual Java terminase, el objeto dejaría de existir y provocaría una excepción al invocarlo.

En este 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 va a permitir optimizar recursos, ya que el objeto servidor sólo se crea cuando se necesita.

Además, vamos a utilizar la clase MiParametro para ilustrar el paso de parámetros por la red mediante serialización de los datos.

Servidores activables

Hasta Java 1.2, el objeto remoto a compartir se creaba al principio, y tenía que estar siempre a disposición de los clientes. A partir de la versión 1.2 se pueden crear objetos remotos cuando un cliente lo solicite, con el consiguiente ahorro en el consumo de recursos. Esto se consigue partiendo de la clase java.rmi.activation.Activatable y del demonio rmid, que se encarga de gestionar el objeto remoto. Veamos los pasos para construir este tipo de servidores:

1. **Definir el interfaz remoto**

   Idéntico al servidor normal.

2. **Implementar el interfaz remoto**

   La clase que implementa el interfaz remoto tiene algunos cambios con respecto a la definida para el servidor normal. Los pasos a seguir son:

   - Debe implementar la interfaz remota (esto no cambia), y heredar de java.rmi.activation.Activatable (en lugar de UnicastRemoteObject)
   - El constructor debe tener dos parámetros: un objeto ActivationID y un objeto MarshalledObject, porque los utiliza el constructor de la clase padre Activatable.

   Un ejemplo de servidor de este tipo sería:

```
import java.rmi.*;
import java.rmi.activation.*;

public class MiClaseRemota2 extends java.rmi.activation.Activatable
                        implements MiInterfazRemoto
{
   public MiClaseRemota2(ActivationID a, MarshalledObject m)
                        throws java.rmi.RemoteException
   {
      super(a, 0);
   }

   public void miMetodo1() throws java.rmi.RemoteException
   {
      ... // Aquí pondremos el código que queramos
   }

   public int miMetodo2() throws java.rmi.RemoteException
   {
      ... // Aquí pondremos el código que queramos
   }

   public static void main(String[] args) throws Exception
```

```
    {
        System.setSecurityManager(new RMISecurityManager());

        Properties p = new Properties();
        p.put("java.security.policy", "/rmi/servidor2/java.policy");


        ActivationGroupDesc.CommandEnvironment ace = null;
        ActivationGroupDesc ejemplo = new ActivationGroupDesc(p, ace);


        ActivationGroupID agi = ActivationGroup.getSystem().registerGroup(ejemplo);
        MarshalledObject m = null;


        ActivationDesc desc = new ActivationDesc (agi, "MiClaseRemota2",
                                        "file://C:/rmi/servidor2/", m);

        MiInterfazRemoto mir = (MiInterfazRemoto)Activatable.register(desc);
        Naming.rebind("//" + java.net.InetAddress.getLocalHost().getHostAddress() +
                  ":" + args[0] + "/PruebaRMI");
        System.exit(0);
    }
}
```

El servidor es muy parecido al anterior, salvo porque en el constructor se pasan dos parámetros, y porque el método main() tiene una forma distinta de registrarlo:

- Con el objeto Properties se establece el fichero de política de seguridad.
- El objeto ActivationGroupDesc es un descriptor de un grupo de activación. En un grupo de activación podremos activar objetos remotos para que puedan acceder los clientes.
- El objeto ActivationGroupID es un identificador para un grupo de activación. Se registra el grupo creado con el ActivationGroupDesc, y se obtiene el identificador del registro.
- El objeto MarshalledObject se emplea para indicar datos de inicialización, si se requieren.
- El objeto ActivationDesc registra el objeto en el demonio rmid. Al crearlo, se pasan como parámetros:
  - El identificador del grupo creado (el objeto ActivationGroupID).
  - La clase del objeto remoto que se quiere registrar.
  - La localización de dicha clase.
  - Un objeto MarshalledObject con datos de inicialización, si se necesitan.

Con eso, se obtiene un objeto remoto (MiInterfazRemoto), llamando al método register() de Activatable, pasándole como parámetro el ActivationDesc creado con todos los pasos anteriores. Después ya se tiene el método rebind() visto en el servidor primero.

3. **Compilar y ejecutar el servidor**

Ya tenemos definido el servidor activable. 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 clien en 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 fiche y Skeleton para mantener la referencia con el objeto remoto, mediante el comando rmic:

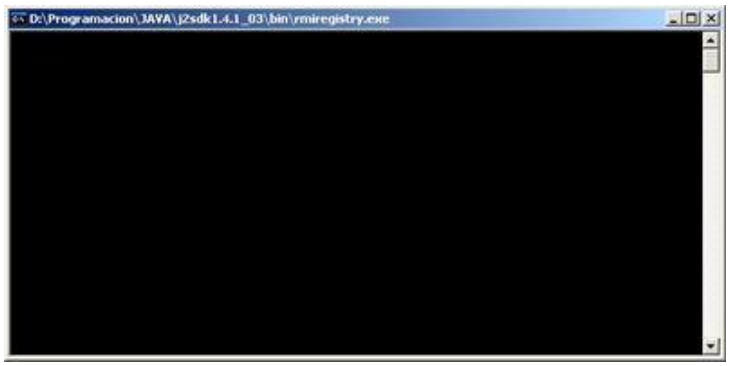
```
set CLASSPATH=%CLASSPATH%;.\objRemotos.jar;.
javac MiClaseRemota2.java
rmic -d . MiClaseRemota2
```

  Observamos en nuestro directorio de trabajo que se han generado automáticamente dos ficheros .class (MiClaseRemota2_Skel.class y MiClaseRemota2_Stub.class) correspondientes a la **capa stub-skeleton** de la arquitectura RMI.

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

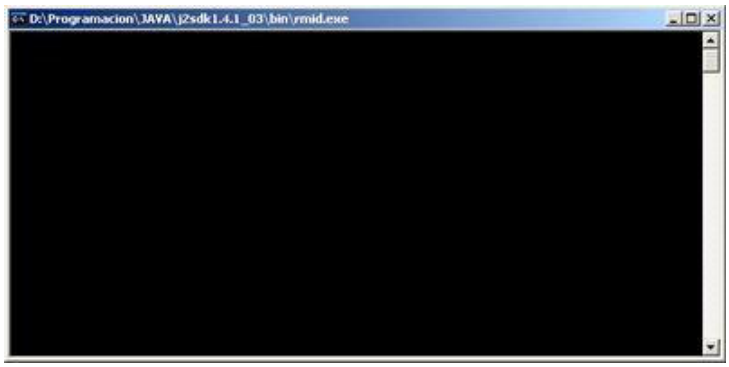
- Se arranca el **registro de RMI** para permitir registrar y buscar objetos remotos:

```
start rmiregistry 1234
```



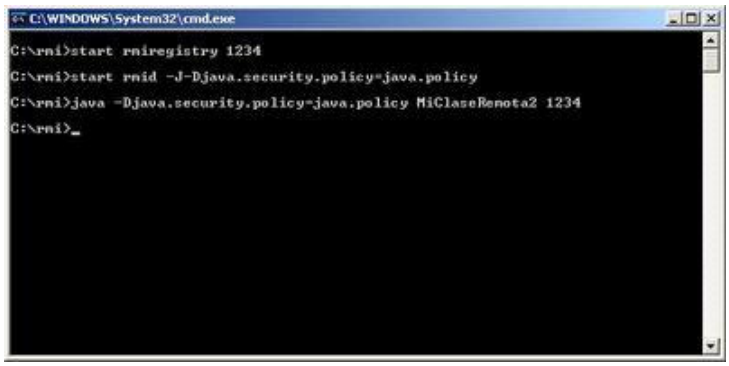
- Luego, lanzamos también el demonio rmid (se puede hacer antes o después de lanzar el rmiregistry):

  start rmid -J-Djava.security.policy=java.policy



- Por último, **se lanza el servidor**. Pero a diferencia del servidor normal, el servidor activable no se queda esperando peticiones del cliente. Simplemente registra la clase como "*activable*" y nos devuelve al prompt del sistema:

  java -Djava.security.policy=java.policy MiClaseRemota2 1234



Crear un cliente RMI

El cliente es similar al ejemplo anterior del servidor normal, pero con el código adaptado a nuestro nueva clase remota. Los pasos a seguir son:

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)
   {
      if (System.getSecurityManager() == null)
         System.setSecurityManager(new java.rmi.RMISecurityManager());

      try
      {
         MiInterfazRemoto mir = (MiInterfazRemoto)java.rmi.Naming.lookup("//" +
                        args[0] + ":" + args[1] +"/PruebaRMI");
```

```
            mir.miMetodo1();
            int i = mir.miMetodo2();

            System.out.println ("Valor de miMetodo2: " + i);

            MiParametro mp = mir.getParametro();

            System.out.println ("MiParametro: " + mp.getS());
         }
         catch (Exception e)
         {
            System.out.println ("Error, no encuentro: " + e.getMessage());
         }
      }
   }
```

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(...).

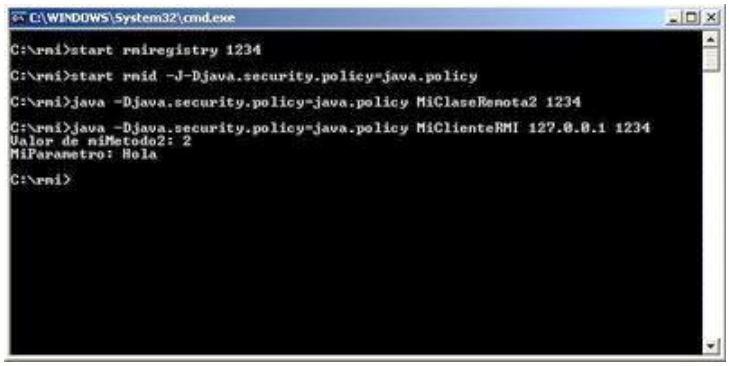
2. **Compilar y ejecutar el cliente**

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

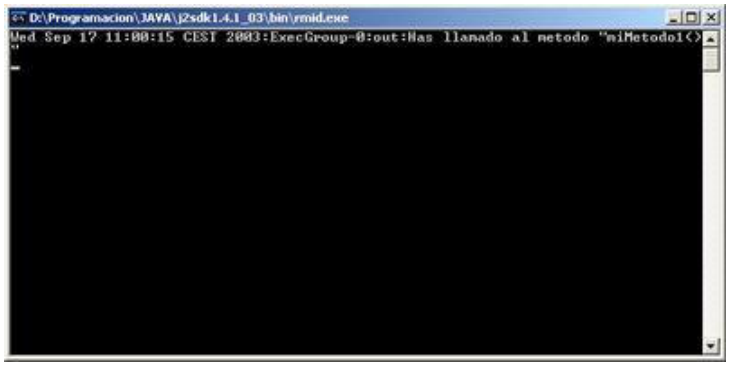
```
set CLASSPATH=%CLASSPATH%;.\objRemotos.jar;.
javac MiClienteRMI.java
```

Luego, para ejecutar el cliente hacemos:

```
java MiClienteRMI 127.0.0.1 1234
```

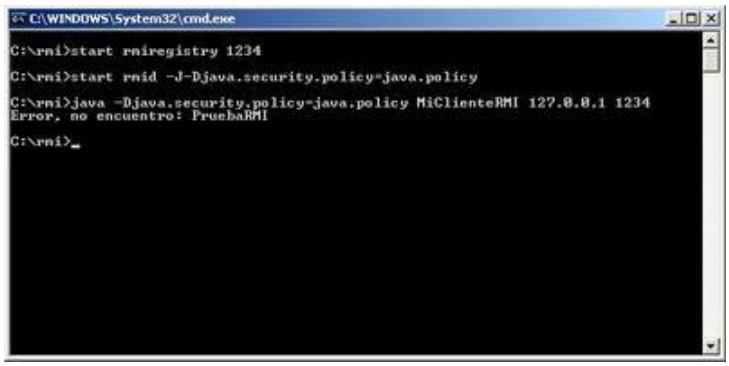


Si echamos un vistazo a la ventana donde está ejecutándose el servidor RMI, veremos como se ha encontrado el objeto remoto, se ha activado mediante el ExecGroup-0 y y se han ejecutado sus métodos:



Pero, ¿qué pasaría si hubiésemos lanzado los demonios rmiregistry y rmid, pero no hubiésemos registrado la clase en el registro de RMI? Veamos qué ocurre:

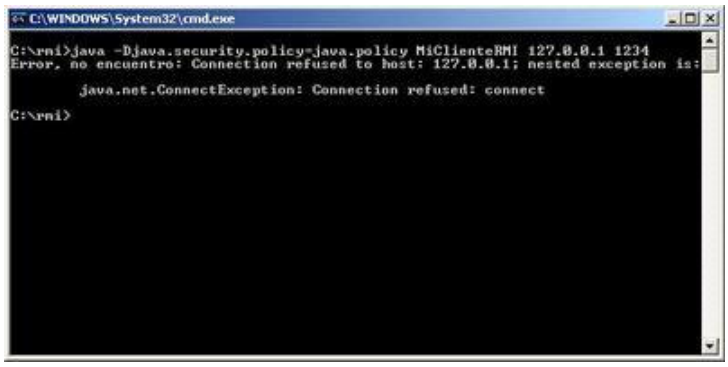
```
start rmiregistry 1234
start rmid -J-Djava.security.policy=java.policy
java -Djava.security.policy=java.policy MiClienteRMI 127.0.0.1 1234
```



Obviamente, el cliente intenta activar el objeto remoto, conecta al demonio del sistema de activación RMI (rmid), pero al no haber sido registrada la clase, no la encuentra y muestra un error por pantalla.

Resumen

En este nuevo ejemplo de RMI hemos visto como se puede activar un objeto remoto desde el cliente sólo en el momento que se necesita. Para ello, además del demonio rmiregistry, hay que lanzar el demonio rmid. Si no se siguen estos pasos, el cliente no podrá ni siquiera conectarse al servicio RMI:



Otro tipo de servidores son los denominados servidores *IOOP*. *IOOP* es el protocolo de transporte adoptado por el *Object Management Group* (OMG) para *CORBA* (Common Object Request Architecture), proporcionando interoperatividad con las implementaciones de objetos *CORBA* en varios lenguajes. *RMI* sobre *IOOP* permite a los programadores Java programar las interfaces *RMI*, pero utilizar *IOOP* como capa de transporte subyacente. De esta forma, los programadores Java pueden participar de los servicios de red de *CORBA* con servicios y clientes implementados en cualquier lenguaje soportado por *CORBA*.

Pero esto es algo que hablaremos en otra ocasión...

Autor: Enrique Medina Montenegro (c) 2003

Otros Tutoriales Recomendados (También ver todos)

Nombre Corto	Descripción
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.
Aplicación básica con RMI	Gracias a este tutorial, podreis aprender paso a paso como crear una aplicación cliente-servidor con RMI