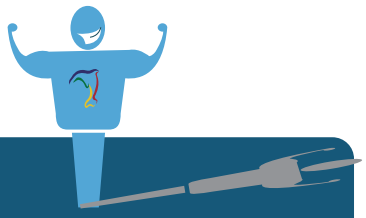
autentia

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...

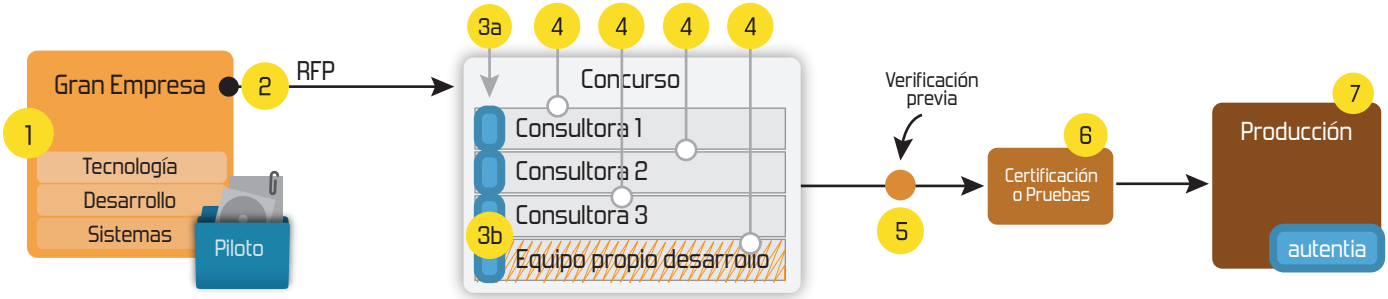
1. Desarrollo de componentes y proyectos a medida



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

EJBs para JMS

Si recordais nuestro tutorial sobre logs y JMS, creamos un cliente y un servidor capaces de gestionar mensajes a través de JMS.

Creamos en es caso un cliente y servidor simples.

Ahora vamos a hacer una cosa igual de rápida y sencilla..... vamos a crear un EJB de un nuevo tipo... especializado en gestionar este tipo de mensajes.... es decir, vamos a sustituir la aplicación que consuma los mensajes por un EJB.

Un Message-Driven Bean es un tipo de EJB nuevo el cual, a diferencia que los otros EJBs que hemos visto hasta ahora.... no tiene interfaces Home y Remoto ..... solo tiene un metodo (entre comillas) que es invocado cada vez que la cola a la que se liga en despliegue, recibe un mensaje (permitiendo filtros).

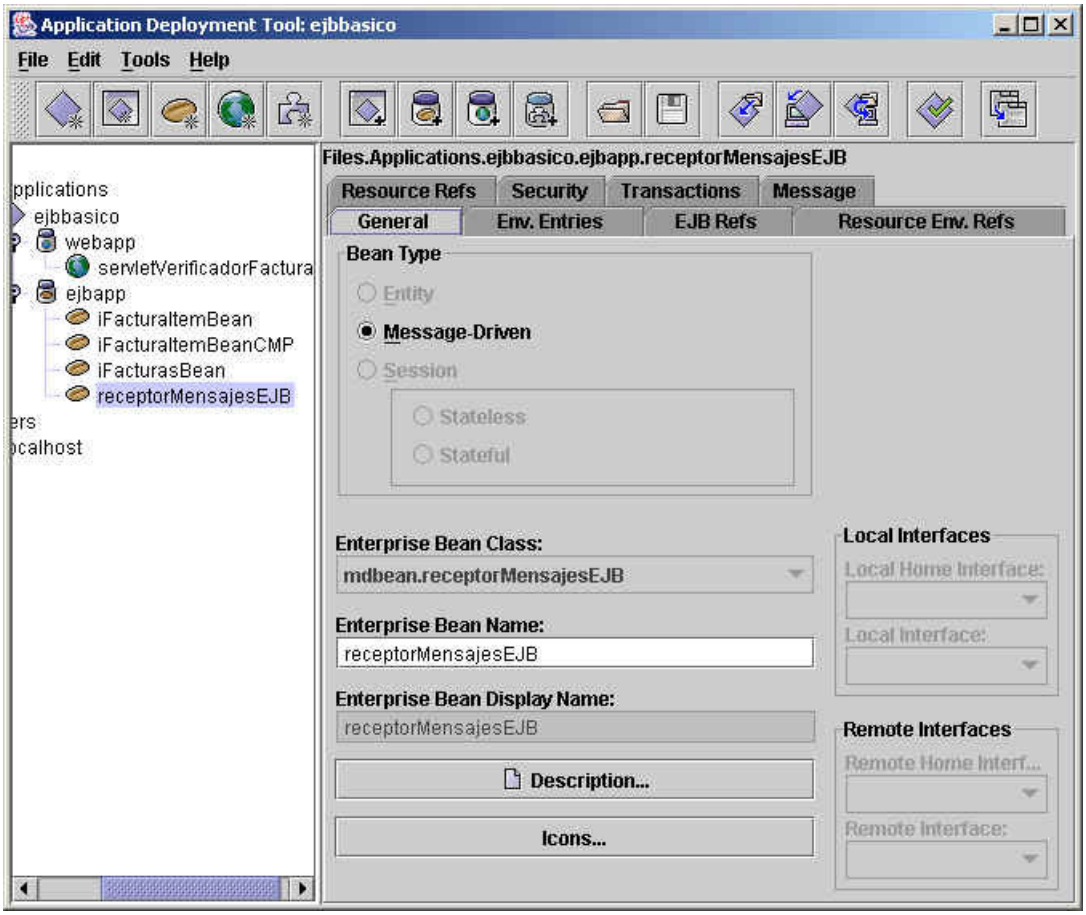
El código es tan sencillo como el mecanismo de un chupete .... hemos copiado el que viene en el tutorial de EJBs de Sun (aunque es que tampoco hay muchos mas modos de hacerlo ;-)), adaptándolo un poco (hemos dejado algunas llamadas relaccionadas con la transaccionabilidad para el futuro)

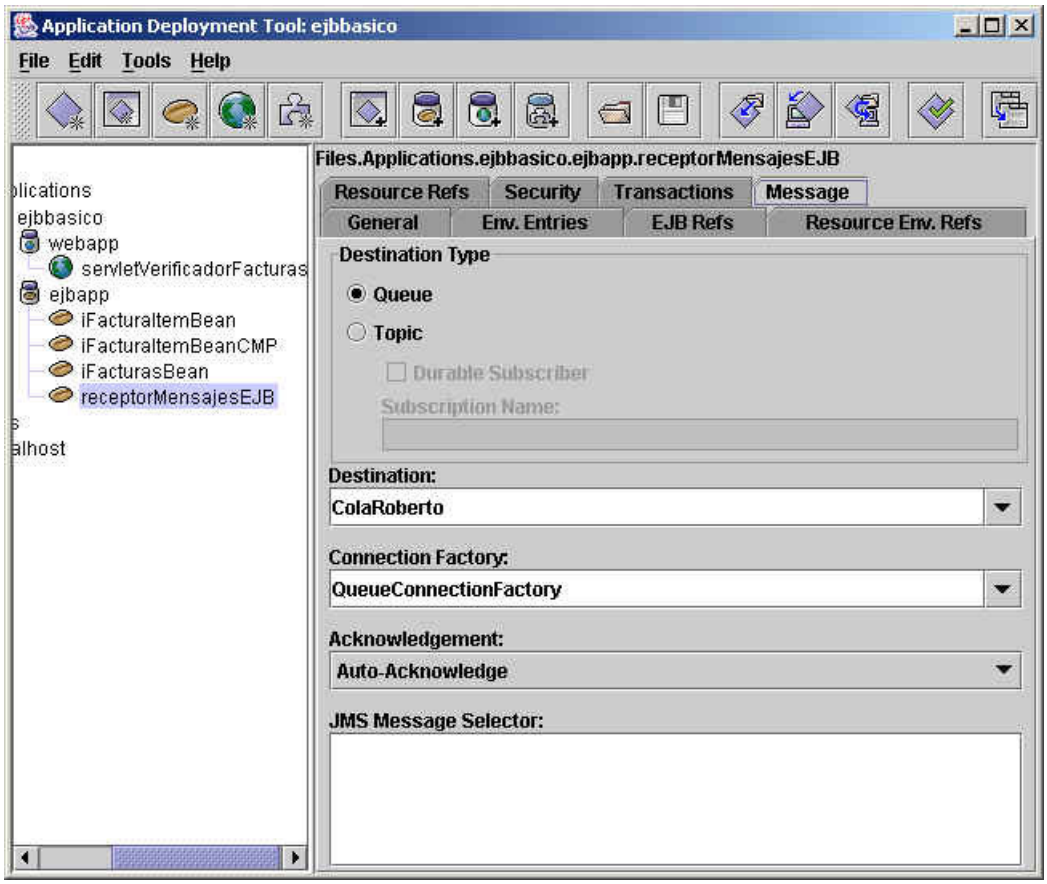
```
/*
 * receptorMensajesEJB.java
 *
 * Created on 3 de octubre de 2003, 16:11
 */

package mdbean;

/**
 *
 * @author Roberto Canales
 */
import java.io.Serializable;
import java.rmi.RemoteException;
import javax.ejb.EJBException;
import javax.ejb.MessageDrivenBean;
import javax.ejb.MessageDrivenContext;
import javax.ejb.CreateException;
import javax.naming.*;
import javax.jms.*;

public class receptorMensajesEJB implements MessageDrivenBean,  MessageListener
{
    private transient MessageDrivenContext mdc = null;
    private Context context;

    public receptorMensajesEJB() {
        depura("Constructor");
    }

    public void setMessageDrivenContext(MessageDrivenContext mdc) {
```

```
        depura("Establecemos el contexto");
                this.mdc = mdc;
    }

    public void ejbCreate() {
                depura("ejbCreate");
    }

    public void onMessage(Message inMessage) {
        depura("Recibido mensaje");

        TextMessage msg = null;

        try
        {
            if (inMessage instanceof TextMessage)
            {
                msg = (TextMessage) inMessage;
                depura("Mensaje recibido: " + msg.getText());
            } else {
                depura("Mensaje inadecuado: "+ inMessage.getClass().getName());
            }
        } catch (JMSException e)
        {
            e.printStackTrace();
            mdc.setRollbackOnly();
        }
        catch (Throwable te)
        {
            te.printStackTrace();
        }
    }  // fin del procesamiento del mensaje

    public void ejbRemove() {
        System.out.println("remove");
    }

    private void depura(String cadena)
    {
        System.out.println("EJB receptorMensajesEJB: " + cadena);
    }
} // fin de la clase
```

Basicamente iremos consumiendo los mensajes a medida que lleguen...

1. Solamente tenemos ahora que desplegarlo:
2. Añadimos un nuevo bean
3. Incluimos en el Path el fichero .class de nuestro nuevo Bean
4. Le asignamos tipo y clase
5. Asignamos la cola y filtro de escucha





Y ahora, si mandamos un mensaje desce una aplicación (os recordamos el código)

```
/**
 *
 * @author Roberto Canales
 */
```

```
package mdbean;

import javax.naming.*;
import javax.jms.*;
import java.util.*;


public class envioMensajeSimple {
   /** Punto de entrada a la aplicacion*/
   public static void main(String[] args) {
      envioMensajeSimple programa = new envioMensajeSimple();
      programa.arranca();
   }

   /** Funcion para centralizar mensajes*/
   void depura(String cadena) {
      System.out.println("Mensaje: " + cadena);
   }

   private static InitialContext getInitialContext() throws NamingException
   {
      Hashtable env = new Hashtable();
      env.put(Context.INITIAL_CONTEXT_FACTORY,"com.sun.enterprise.naming.SerialInitContextFactory");
      env.put("java.naming.factory.url.pkgs", "com.sun.enterprise.naming");
      env.put(Context.PROVIDER_URL, "iiop://127.0.0.1:1050");
      //env.put("org.omg.CORBA.ORBInitialHost","127.0.0.1");

      return new InitialContext(env);
   }

   /** Metodo encargado de ejecutar la tarea*/
   void arranca() {
      // inicializamos las variables
      Context jndiContext = null;
      Queue destino = null;
      QueueConnection  conexion = null;
      QueueSession  sesionActual = null;
      QueueSender  enviadorMensajes = null;
      TextMessage mensaje = null;


      // identificamos en nombre de la cola
      String colaDestino = "ColaRoberto";

      try {
         depura("Creamos un contexto");
          //jndiContext = new InitialContext();
         jndiContext = getInitialContext();
      }
      catch (NamingException e) {
         System.out.println("Imposible crear el contexto  " + "contexto: " + e.toString());
         System.exit(1);
      }

      try {
         // creamos el objeto para construir la conexion
         depura("Buscamos la factoria");
         QueueConnectionFactory queueConnectionFactory = (QueueConnectionFactory)jndiContext.lookup("QueueConnectionFactory");

         // localizamos la cola destino
         depura("Buscamos la cola");
         destino = (Queue) jndiContext.lookup(colaDestino);

         // creamos una conexión
         depura("Creamos la conexion");
         conexion = queueConnectionFactory.createQueueConnection();

         depura("Creamos la sesion");
         sesionActual = conexion.createQueueSession(false, Session.AUTO_ACKNOWLEDGE);

         // Creamos una factoria de mensaje y un mensaje
         depura("Creamos el productor de mensajes");

         enviadorMensajes = sesionActual.createSender(destino);

         mensaje = sesionActual.createTextMessage();
         mensaje.setStringProperty("TipoMensaje", "1");


        long inicio = System.currentTimeMillis();

        //for(int i=0;i<10;i++)
        //{
           // Asociamos texto al mensaje
           mensaje.setText("Nuevo mensaje " + new Date() );

           //depura("Enviamos el mensaje");
           enviadorMensajes.send(mensaje);
        //}
         enviadorMensajes.close();
         //productorMensajes.send(sesionActual.createMessage());


         long fin = System.currentTimeMillis() - inicio;
         System.out.println("El tiempo transcurrido es " + fin + " milisegundos");
```

```
        }
        catch (Exception e) {
            depura("Error en la aplicación " + e.toString());
            e.printStackTrace();
        }
        finally {
            if (conexion != null) {
                try {
                    conexion.close();
                }
                catch (JMSException e)
                {}
            }
        }

        depura("Salimos del programa");
    }
}
```

Y solo tenemos que comprobar en el log del servidor de aplicaciones, como responde adecuadamente nuestro EJB.

Mostramos el detalle del fichero **C:\j2sdkee1.3.1\logs\rautentia\j2ee\j2ee\system.out**

```
rmic ejbfacturas.iFacturaItemHome...
Deploying message driven bean receptorMensajesEJB, consuming from ColaRoberto
Binding name:`java:comp/env/ejb/entidadfactura`
Binding name:`java:comp/env/ejb/entidadfacturacmp`
EJB receptorMensajesEJB: Mensaje recibido: Nuevo mensaje Fri Oct 03 16:38:49 CEST 2003
remove
```

Bueno ... sencillo verdad ...

Esto del java no es complicado .... el problema es que es la suma de 100 millones de cosas simples .... jejejeje

Sobre el Autor ..

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
Desarrollo de Entity Beans	Os mostramos como construir un Entity Bean básico y desplegarlo en el servidor J2EE de referencia. Lo usaremos como base de buenas prácticas J2EE
Aplicación básica con RMI	Gracias a este tutorial, podreis aprender paso a paso como crear una aplicación cliente-servidor con RMI
Despliegue gráfico de EJBs	Os mostramos como crear y desplegar de un modo gráfico un EJB de sesión el el servidor de aplicaciones de referencia de Sun
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)
CMP Entity Beans y MySql	Os mostramos como crear un Entity Bean con persistencia controlada por el servidor, configurado para usar MySql