

# ¿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

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

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

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

 Powered by 	Hosting Patrocinado por <b>enREDados.com</b> 
---	--

[Home](#) | [Quienes Somos](#) | [Empleo](#) | [Tutoriales](#) | [Contacte](#)

## Lanzamiento TNTConcept

**Autentia** da un paso más en su evolución: Lanzamiento de software propio. Ponemos a vuestra disposición el software que hemos construido para nuestra gestión interna, llamado TNTConcept (**autentia**).

Construida con las últimas tecnologías de desarrollo Java/J2EE (Spring, JSF, Hibernate, Maven, Subversion, etc.) y disponible en licencia GPL, seguro que a muchos profesionales independientes y PYMES os ayudará a organizar mejor vuestra operativa.

Las cosas grandes empiezan siendo algo pequeño ..... Saber más en: <http://tntconcept.sourceforge.net/>

	<p><b>Autor del tutorial:</b> <i>Cristian Kirs Herrera Basurto</i></p> <ul style="list-style-type: none"> <li><b>Lugar de residencia:</b> Quito - Ecuador</li> </ul> <p><i>Cuento con experiencia en el área de desarrollo de software y en la docencia académica. Dentro de la construcción de software he manejado las etapas de: análisis, diseño, personalización e implementación de aplicaciones bajo ambientes Cliente / Servidor e Internet.</i></p> <p style="text-align: right;"> <a href="mailto:Cristian.Herrera@gmail.com">Cristian.Herrera@gmail.com</a> /  <a href="mailto:cherrera@kruger.com.ec">cherrera@kruger.com.ec</a> </p>	<p><a href="http://www.adictosaltrabajo.com">www.adictosaltrabajo.com</a> es el Web de difusión de conocimiento de <a href="http://www.autentia.com">www.autentia.com</a></p>  <p style="text-align: center;"><b>autentia</b> real business solutions</p> <p style="text-align: center;"><a href="#">Catálogo de cursos</a></p>
---	---	--

Descargar este documento en formato PDF [JMSJBOSS.pdf](#)

Firma en nuestro libro de Visitas <-----> [Asociarme al grupo AdictosAlTrabajo en eConozco](#)

### WebSphere Portal Experts

IWWCM and WebSphere Portal development and consulting  
[www.wpexperts.com](http://www.wpexperts.com)

### Real Estate Applications

Software Development, Maintenance & Support for Real Estate Enterprises  
[www.annetsite.com](http://www.annetsite.com)

### BPM and Business Rules

Discover how to improve business agility, save time and cut costs  
[www.ilog.com/solutions//bpm\\_brms/](http://www.ilog.com/solutions//bpm_brms/)

Anuncios Goooooogle

[Anunciarse en este sitio](#)

### Manejo de JMS en JBOSS

[Introducción a JMS](#)  
[Arquitectura de JMS](#)  
[Objetos administrados](#)  
[Mensajes](#)  
[Clientes JMS](#)  
[Configuración de colas en JBOSS](#)  
[Un ejemplo de JMS](#)  
[Session Bean para envío de mensajes](#)  
[Message Driver Bean consumidor de una cola](#)  
[Una clase de servicio para acceder al EJB de sesión](#)  
[Un cliente del EJB de sesión y las salidas del MDB en la consola del servidor](#)  
[Conclusiones](#)  
[Bibliografía](#)

### Manejo de JMS en JBOSS

[1]

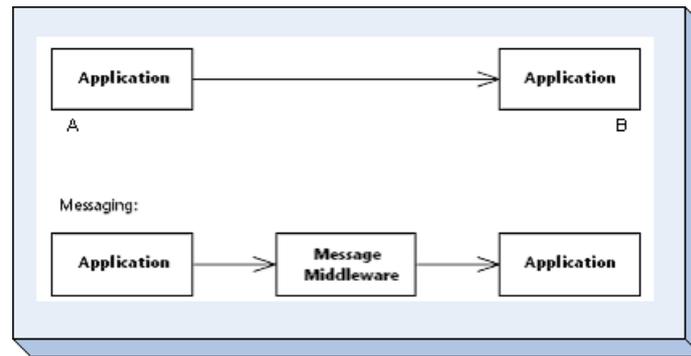
#### Introducción a JMS

JMS es la solución de Sun para los sistemas de mensajes, empecemos por saber que es un sistema de mensajes empresarial:

En las comunicaciones cliente servidor, los datos que se intercambian entre las dos partes necesitan de una comunicación síncrona, es decir, que las dos partes estén presentes en el momento de la comunicación. Los sistemas de mensajes aportan una serie de mejoras a la comunicación entre aplicaciones que no tienen por que residir en la misma máquina.

JMS se sitúa como middleware (capa media) de la comunicación de dos aplicaciones. En los entornos cliente servidor, cuando la aplicación A quiere comunicarse con la Aplicación B, necesita saber donde esta B (su IP por ejemplo) y que B esté

escuchando en ese momento. Cuando se usa JMS (o cualquier otro sistema de mensajes), la aplicación A envía un mensaje, el sistema de mensajes lo recibe y se lo envía a B cuando se conecte al servicio. De esta manera se consigue una comunicación asíncrona entre A y B, es decir no hace falta que B este presente en el momento del envío del mensaje, y no por ello va a dejar de recibirlo.



La anterior es una de las ventajas de JMS, pero no la única. La comunicación anterior tiene dos extremos, el productor (A) y el consumidor (B). JMS soporta otro tipo de comunicaciones que Sun denomina Publisher/Subscriber, traducido sería Publicador (Publicante)/Suscriptor. En este tipo de comunicación, la aplicación A publica su mensaje en el servicio JMS y lo reciben todas las aplicaciones que estén suscritas al servicio JMS al que se envió el mensaje, esta forma de comunicación es exactamente igual que la que se produce en los chats del IRC.

Otra de las ventajas de usar JMS (o cualquier otro sistema de mensajes) es que las aplicaciones se pueden cambiar simplemente asegurándose que la nueva aplicación entiende los mensajes que se intercambian.

#### Arquitectura de JMS

Una aplicación JMS consta de los siguientes elementos:

- Cientes JMS**                      Aplicaciones que envían o reciben mensajes a través de JMS
- Mensajes**                              Los mensajes que se intercambian
- Objetos administrados**      Los objetos JMS a los que se dirigen las comunicaciones

#### Objetos administrados

Los objetos administrados son el punto al que se comunican los clientes JMS para enviar o recibir mensajes, se denominan objetos administrados por que los crea el administrador (en la implementación de referencia mediante `j2eeadmin`). Implementan las interfaces JMS y se sitúan en el espacio de nombres de JNDI (Java Naming and Directory Interface) para que los clientes puedan solicitarlos.

Hay dos tipos de objetos administrados en JMS:

- **ConnectionFactory:** Se usa para crear una conexión al proveedor del sistema de mensajes.
- **Destination:** Son los destinos de los mensajes que se envían y el recipiente de los mensajes que se reciben.

#### Mensajes

Es el corazón del sistema de mensajes. Están formados por tres elementos:

- **Header:** Es la cabecera del mensaje, contiene una serie de campos que le sirven a los clientes y proveedores a identificar a los mensajes. Aquí se presenta la lista de campos completa :

Campo	Tipo de Dato	Descripción
JMSMessageID	String	Un numero que identifica de manera única al mensaje. Solo se puede consultar una vez que esta enviado el mensaje.
JMSDestination	Destination	El destino a donde se envía el mensaje.
JMSDeliveryMode	int	Puede ser de tipo PERSISTENT, entonces se envía una única vez, o de tipo NON_PERSISTENT, de manera que se envía como mucho una vez, lo cual incluye también que no sea enviado nunca. PERSISTENT y NON_PERSISTENT están definidas como constantes.
JMSTimestamp	long	Pues eso, la hora a la que se envió el mensaje.
JMSExpiration	long	La hora hasta la cual el mensaje es valido, si es 0 quiere decir que no caduca nunca.
JMSPriority	int	Prioridad del mensaje de 0 a 9, siendo 0 la mas baja.
JMSCorrelationID	String	Este campo se usa para relacionar una respuesta a un mensaje, se copia aquí el ID de mensaje del mensaje al que se esta respondiendo.
JMSReplyTo	Destination	Especifica el lugar a donde se deben enviar las respuestas al mensaje actual.
JMSType	String	Este campo lo puede usar el programa de mensajería para almacenar el tipo del mensaje.
JMSRedelivered	boolean	Indica que el mensaje ha sido enviado con anterioridad pero el destino no lo ha procesado, por lo que se reenvía.

- **Properties:** Son propiedades personalizadas para un mensaje en particular. Aquí se indica la lista completa:

Propiedad	Tipo de Dato	Descripción
JMSXUserID	String	El usuario que envía el mensaje.
JMSXAppID	String	La aplicación que envía el mensaje.
JMSXDeliveryCount	int	El numero de veces que se ha intentado enviar el mensaje
JMSXGroupID	String	Identificador del grupo al que pertenece el mensaje.
JMSXGroupSeq	int	Numero de secuencia en el grupo de mensajes.
JMSXRcvTimestamp	long	La hora a la que JMS le entrego el mensaje al/los destinatario/s.
JMSXState	int	Para uso del proveedor de mensajería.
JMSX_<nombre_del_proveedor> -		Reservado para propiedades particulares del proveedor.

- **Body:** Es el mensaje en si, hay distintos tipos:
  - StreamMessage: Contiene un stream de datos que se escriben y leen de manera secuencial.
  - MapMessage: Contiene pares nombre-valor.
  - TextMessage: Contiene un String.
  - ObjectMessage: Contiene un objeto que implemente la interfaz Serializable.
  - BytesMessage: Contiene un stream de bytes.

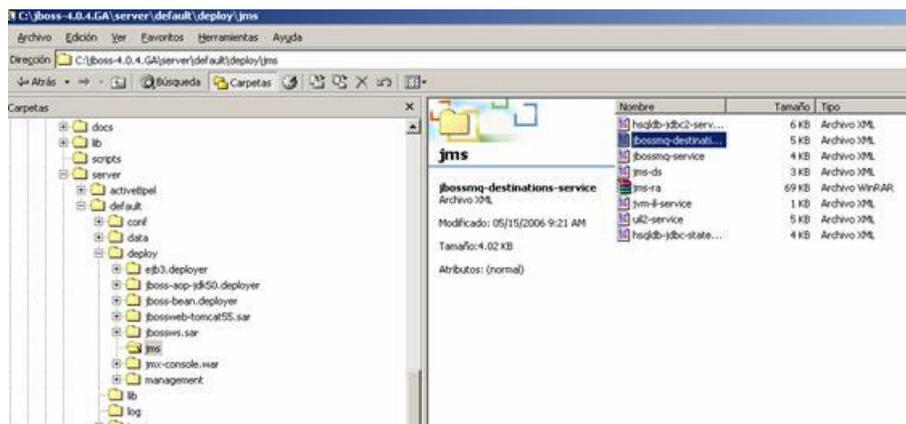
### Cientes JMS

Son clientes de JMS tanto el que suministra mensajes como el que los recibe. Todos tienen una serie de pasos en común antes de lograr enviar o recibir un mensaje:

- Conseguir un objeto ConnectionFactory a través de JNDI.
- Conseguir un destino, mediante el objeto Destination a través de JNDI.
- Usar ConnectionFactory para conseguir un objeto Connection
- Usar Destination para crear un objeto Session.

### Configuración de colas en JBOSS

Para declarar una cola en JBOSS es necesario modificar el archivo jbossmq-destinations-service.xml ubicado en JBOSS\_HOME/Server/default/deploy/jms



Y adicionar una sección como la siguiente

```
<?xml version="1.0" encoding="UTF-8" ?>
<bean code="org.jboss.mq.server.jmx.Queue"
  name="jboss.mq.destination.service.Queue.name=myQueue">
  <depends optional-attribute-name="DestinationManager">jboss.mq.service=DestinationManager</depends>
  <depends optional-attribute-name="SecurityManager">jboss.mq.service=SecurityManager</depends>
  <attribute name="MessageCounterHistoryDayLimit">1</attribute>
  <attribute name="SecurityConf">
    <security>
      <role name="guest" read="true" write="true"/>
      <role name="publisher" read="true" write="true" create="false"/>
      <role name="noacc" read="false" write="false" create="false"/>
    </security>
  </attribute>
</bean>
```

```
<mbean code="org.jboss.mq.server.jmx.Queue"
  name="jboss.mq.destination.service.Queue.name=myQueue">
  <depends optional-attribute-name="DestinationManager">jboss.mq.service=DestinationManager</depends>
  <depends optional-attribute-name="SecurityManager">jboss.mq.service=SecurityManager</depends>
  <attribute name="MessageCounterHistoryDayLimit">1</attribute>
  <attribute name="SecurityConf">
    <security>
      <role name="guest" read="true" write="true"/>
      <role name="publisher" read="true" write="true" create="false"/>
      <role name="noacc" read="false" write="false" create="false"/>
    </security>
  </attribute>
</mbean>
```

**Nota:** Si se llega a modificar este archivo en caliente, es decir sin bajar el servidor, los servicios de JBOSS detectan automáticamente la nueva configuración y realizan el cambio.

### Un ejemplo de JMS

La siguiente sección ilustrará algo del código necesario para crear un EJB de sesión con posibilidades de enviar mensajes a una cola en un servidor JBOSS

#### Session Bean para envío de mensajes

Declaramos una interfaz remota y local del EJB con los siguientes métodos

```
/**
 * Método para inicilizar las propiedades de la cola de mensajes
 *
 * @param properties
 */
public void initQueueDefinitions(Properties properties);

/**
 * Send text to a queue and returns the ID assigned by the JMS provider.
 *
 * @ejb.interface-method
 * @param message
 *         The message to publish.
 * @param queue
 *         The queue to send to.
 * @return the JMSMessageID assigned to this message.
 */
public String send(Properties properties, String message, Queue queue, Date date,
Object object) throws JMSEException;

/**
 * Setear la cola de mensajes
 *
 * @param properties
 * @return
 */
public Queue returnQueue(Properties properties)
```

Veamos ahora un poco de la implementación

```
/**
 * Método para inicilizar las propiedades de la cola de mensajes
 *
 * @param fisaUserData
 * @param properties
 */
public void initQueueDefinitions(
    Properties properties) {
    try {
        logger.info("Dentro de initQueue");
        InitialContext ctx = new InitialContext(properties);
        queue = (Queue) ctx.lookup("queue/myQueue");
        queueConnectionFactory = (QueueConnectionFactory) ctx
            .lookup("UIL2ConnectionFactory");
    } catch (NamingException e) {
        String msg = "An error was encountered trying to lookup an object from
JNDI";
        logger.error(msg);
    }
}
```

Este método initQueueDefinitions servirá para inicializar los objetos factory y queue que emplearemos para el envío y recepción de mensajes.

```
/**
 * Setear la cola de mensajes
 *
 * @param fisaUserData
 * @param properties
 * @return
 */
public Queue returnQueue(
    Properties properties) throws JMSEException {
    logger.info("Estoy en returnQueueSender ");
    initQueueDefinitions( properties);

    return getQueue();
    // return queue;
}
```

El método returnQueue nos permite tomar una cola del servidor para emplearla en el envío de mensajes.

```
/**
 * Send a Message to a queue with a scheduled delivery and returns the ID
```

```

    * assigned by the JMS provider.
    *
    * @ejb.interface-method
    * @param messageCreator
    *     MessageCreator to create the message with.
    * @param queue
    *     The queue to send to.
    * @param deliveryDate
    *     the date at which the message should be delivered. If null, no
    *     delivery date is set
    * @return the JMSMessageID assigned to this message.
    */
public String send(Properties properties, MessageCreator messageCreator, Queue queue, Date
deliveryDate, Object object) throws JMSException {
    if (logger.isDebugEnabled()) {
        logger.debug("Sending message [" + messageCreator + "] to queue [" +
            queue.getQueueName() + "]);
    }

    QueueConnection connection = null;
    QueueSession session = null;
    QueueSender sender = null;

    try {
        initQueueDefinitions( properties);

        connection = queueConnectionFactory.createQueueConnection(); //
        session = connection.createQueueSession(false,
            QueueSession.AUTO_ACKNOWLEDGE);
        sender = session.createSender(queue);
        Message msg = messageCreator.getMessage(session);

        if (deliveryDate != null) {
            logger.debug("Message will be delivered on [" + deliveryDate + "]);
            msg.setLongProperty("JMS_JBOSS_SCHEDULED_DELIVERY",
                deliveryDate.getTime());
        }

        msg.setObjectProperty("MyObject", object);

        sender.send(msg);
        return msg.getJMSMessageID();
    } finally {
        closeQueueSender(sender);
        closeSession(session);
        closeConnection(connection);
    }
}

```

Como se observa en el método send se ha seguido los pasos indicados en la sección clientes JMS, es decir:

- Conseguir un objeto ConnectionFactory a través de JNDI.
- Conseguir un destino, mediante el objeto Destination a través de JNDI.
- Usar ConnectionFactory para conseguir un objeto Connection
- Usar Destination para crear un objeto Session.

Algo que también es interesante en este ejemplo es que al método send le he dado un parámetro tipo Object y en el mensaje se settea este objeto, una utilidad podría ser el envío de adjuntos en un mensaje de texto o el envío de cualquier clase de objeto para que luego sea procesado por la lógica de negocio.

#### Message Driver Bean consumidor de una cola

Bien en la sección anterior definimos un EJB común de session y lo hemos configurado para poder acceder a una cola y enviar mensajes a la misma, ahora definiremos un MDB que será un cliente consumidor de la misma cola.

Para la implementación usaremos las anotaciones de EJB 3.0

```

import javax.ejb.ActivationConfigProperty;
import javax.ejb.EJBException;
import javax.ejb.MessageDriven;

import javax.jms.Message;
import javax.jms.MessageListener;
import javax.jms.TextMessage;
import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;

/**
 *
 * Message Driver Bean implementado como cliente de la cola de mensajes
 *
 * @author Kirs Herrera
 * @version ${Revision} 1.0$

```

```

*/
@MessageDriven(mappedName = "jms/JmsSampleListener", activationConfig = {
    @ActivationConfigProperty(propertyName = "acknowledgeMode", propertyValue =
"Auto-acknowledge"),

    @ActivationConfigProperty(propertyName = "destinationType", propertyValue =
"javax.jms.Queue"),

    @ActivationConfigProperty(propertyName = "destination", propertyValue =
"queue/myQueue")
})
public class JmsSampleListener implements MessageListener
{
    @PersistenceContext
    EntityManager manager;

    public JmsSampleListener()
    {

    }

    /**
     * Método que toma los mensajes de la cola
     * @param msg
     */
    public void onMessage(Message msg) {
        TextMessage tm = (TextMessage) msg;
        try {
            String texto= tm.getText();
            System.out.println("En Mensaje " + tm.getJMSMessageID());
            System.out.println("Con texto " + texto);
            //Recupero el objeto adjuntado al mensaje
            Object obj = msg.getObjectProperty("MyObject");

            if(obj instanceof String)
            {
                System.out.println("Tengo la cadena " + obj);
            }

            System.out.println("Luego de obtener objeto");

        } catch (Exception e) {
            e.printStackTrace();
            throw new EJBException(e);
        }
    }

    public void persist(Object object) {
        // TODO:
        // manager.persist(object);
    }

    public void save(Object object) {
        // TODO:
        manager.persist(object);
    }
}

```

Al actuar como cliente este MDB se pega a la cola del servidor a escuchar todos los mensajes que llegan, el método onMessage se disparará con cada nuevo mensaje.

Es importante notar que el mapeo de anotaciones al inicio de la clase nos evita el tener que ir a definir el archivo xml con los MDB que estamos manejando.

#### Una clase de servicio para acceder al EJB de sesión

Será necesario acceder al EJB Sender por medio de JNDI para esto necesitamos de una clase de servicio que haga un lookup del EJB y pueda acceder a los métodos expuestos.

Básicamente declaramos una de las interfaces del EJB y la instanciamos en el constructor de la clase de servicios

```

FacadeJmsSampleSender sender = null;

public JMSSampleService()
{
    try {
        ServiceLocator locator = ServiceLocator.getInstance();
        sender = (FacadeJmsSampleSender) locator
            .lookup("FacadeJmsSampleSender/remote");
    } catch (Throwable ex) {

```

```

        ex.printStackTrace();
    }
}

/**
 * Método para inicializar las propiedades de la cola de mensajes
 *
 * @param properties
 */
public void initQueueDefinitionsSender(Properties properties) {
    sender.initQueueDefinitions( properties);
}

```

Como se observa para ir a cualquiera de los métodos lo hacemos a través de la propiedad que declaramos en la clase de servicios.

#### Un cliente del EJB de sesión y las salidas del MDB en la consola del servidor

El siguiente código muestra de manera muy sencilla un cliente de consola para los EJB que se han definido en este ejemplo:

```

import java.util.Date;
import java.util.Properties;

import javax.jms.Queue;

import ec.com.kirs.jms.service.JMSSampleService;

public class ClientJMS {

    /**
     * @param args
     */
    public static void main(String[] args) throws Exception
    {
        // TODO Auto-generated method stub
        JMSSampleService service = new JMSSampleService();
        Date date;
        date = new Date();
        Properties properties = new Properties();

        properties.put("java.naming.factory.initial",
            "org.jnp.interfaces.NamingContextFactory");
        properties.put("java.naming.factory.url.pkgs",
            "org.jboss.naming:org.jnp.interfaces");
        properties.put("java.naming.provider.url", "localhost:1099");
        System.out.println("Antes de obtener la cola del servidor ");
        Queue queue = service.returnQueueSender( properties);
        String mensaje = "Vamos a ir a myQueue";
        String obj = "Objeto tipo string a adjuntar la mensaje";

        String id = service.send( properties, mensaje, queue,
            date, obj);

        System.out.println("Id de mensaje en la cola es " + id);

    }
}

```

Para probar el desarrollo levantamos el servidor

```

C:\WINNT\system32\cmd.exe - run
11:03:16.436 INFO [StdSchedulerFactory] Quartz scheduler version: 1.5.2
11:03:16.436 INFO [QuartzScheduler] Scheduler DefaultQuartzScheduler_5_NON_CLUS
TERED started.
11:03:17.030 INFO [ConnectionFactoryBindingService] Bound ConnectionManager 'jb
oss.jca:service=DataSourceBinding,name=fisaDS' to JNDI name 'java:fisaDS'
11:03:17.905 INFO [ConnectionFactoryBindingService] Bound ConnectionManager 'jb
oss.jca:service=DataSourceBinding,name=DefaultDS' to JNDI name 'java:DefaultDS'
11:03:18.358 INFO [A] Bound to JNDI name: queue/A
11:03:18.358 INFO [B] Bound to JNDI name: queue/B
11:03:18.358 INFO [C] Bound to JNDI name: queue/C
11:03:18.374 INFO [D] Bound to JNDI name: queue/D
11:03:18.374 INFO [ex] Bound to JNDI name: queue/ex
11:03:18.421 INFO [testTopic] Bound to JNDI name: topic/testTopic
11:03:18.436 INFO [securedTopic] Bound to JNDI name: topic/securedTopic
11:03:18.436 INFO [testDurableTopic] Bound to JNDI name: topic/testDurableTopic
11:03:18.452 INFO [testQueue] Bound to JNDI name: queue/testQueue
11:03:18.452 INFO [myQueue] Bound to JNDI name: queue/myQueue
11:03:18.483 INFO [UILServerILService] JBossMQ UIL service available at : /0.0.
0.0:8093
11:03:18.577 INFO [DLQ] Bound to JNDI name: queue/DLQ
11:03:18.811 INFO [ConnectionFactoryBindingService] Bound ConnectionManager 'jb
oss.jca:service=ConnectionFactoryBinding,name=JmsXA' to JNDI name 'java:JmsXA'

```

Notamos que la cola myQueue que definimos en el archivo de configuración se encuentra ahora instanciada



## Recuerda

que el personal de [Autentia](#) te regala la mayoría del conocimiento aquí compartido ([Ver todos los tutoriales](#))

¿Nos vas a tener en cuenta cuando necesites consultoría o formación en tu empresa?  
¿Vas a ser tan generoso con nosotros como lo tratamos de ser con vosotros?

[info@autentia.com](mailto:info@autentia.com)

Somos pocos, somos buenos, estamos motivados y nos gusta lo que hacemos .....  
**Autentia = Soporte a Desarrollo & Formación**

Formación en nuevas tecnologías

[Autentia S.L.](#) Somos expertos en:

**J2EE, Struts, JSF, C++, OOP, UML, UP, Patrones de diseño ..**  
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	<input type="text"/>
	<input type="button" value="Enviar"/>

## Otros Tutoriales Recomendados ([También ver todos](#))

Nombre Corto	Descripción
<a href="#">Escritura log con Fichero UDP y JMS</a>	Os mostramos ejemplos para cuantificar el coste de escritura de Logs por pantalla, fichero, UDP y JMS (describiendo como configurar el entorno)
<a href="#">mod_jk en Linux / Apache2-JBoss</a>	El conector mod_jk se encarga de enviar las peticiones dinámicas de Apache2 a un servidor de aplicaciones JBoss
<a href="#">mod_jk en WindowsXP / Apache2-JBoss</a>	Os mostramos como instalar el conector mod_jk sobre WindowsXP utilizando Apache2 y JBoss
<a href="#">Instalar JBoss</a>	Os mostramos como instalar en servidor gratuito de aplicaciones JBOSS así como a automatizar su arranque y parada.

Nota: Los tutoriales mostrados en este Web tienen como objetivo la difusión del conocimiento.

Los contenidos y comentarios de los tutoriales son responsabilidad de sus respectivos autores.

En algún caso se puede hacer referencia a marcas o nombres cuya propiedad y derechos es de sus respectivos dueños. Si algún afectado desea que incorporemos alguna reseña específica, no tiene más que solicitarlo.

Si alguien encuentra algún problema con la información publicada en este Web, rogamos que informe al administrador [rcanales@adictosaltrabajo.com](mailto:rcanales@adictosaltrabajo.com) para su resolución.

[Patrocinados por enredados.com .... Hosting en Castellano con soporte Java/J2EE](#)



