Java Message Service

Java Message Service (JMS) es un middleware de mensajería desarrollado por Sun Microsystems para permitir a los programas Java acceder a sistemas de mensajería empresariales.

Los sistemas de mensajería empresariales, también conocidos como *Message Oriented Middleware (MOM)*, proporcionan un mecanismo para integrar las aplicaciones de forma flexible y débilmente acoplada. En concreto, estos sistemas proporcionan una entrega asíncrona de los datos entre las aplicaciones usando el concepto "almacenaje y reenvío" (*store and forward*). Es decir, las aplicaciones no se comunican directamente entre sí, sino que se comunican con el MOM, quien actúa como intermediario.

Por ejemplo, una aplicación A se comunica con una aplicación B enviando un mensaje a través de la API (*Application programming interface*) del MOM. El MOM redirige el mensaje a la aplicación B, que puede estar en un ordenador diferente. El MOM se ocupa de la red de comunicaciones. Si la conexión de red no está disponible, el MOM almacenará el mensaje hasta que la conexión esté disponible y entonces lo reenviará a la aplicación. Además, podría ocurrir que la aplicación B no estuviera en ejecución cuando la aplicación A envía su mensaje. El MOM retendrá dicho mensaje hasta que la aplicación B comience su ejecución y solicite recibir sus mensajes. De este modo, se evita que la aplicación A tenga que bloquearse esperando a que la aplicación B reciba el mensaje.

La principal ventaja de los sistemas de mensajería empresarial consiste en la **comunicación débilmente acoplada**. En el ejemplo anterior, supongamos que la aplicación A ha enviado sus mensajes a un determinado destino, por ejemplo "order processing". Y supongamos que la aplicación B se encarga de recibir mensajes de dicho destino y procesar las órdenes que contienen. Se podría sustituir la aplicación B con otro programa diferente, también encargado de procesar órdenes, y dicho cambio no afectaría en absoluto a la aplicación A, quien continuaría enviando sus mensajes a "order processing" y estos mensajes seguirían siendo procesados (aunque ahora, por otra aplicación B'). Del mismo modo, podríamos también reemplazar la aplicación A por otra aplicación A' que enviara mensajes a "order processing", de modo que la aplicación B (o B') no tendría por qué conocer que hay una nueva aplicación A' enviando órdenes.

1. Modelos de Mensajería

Existen dos modelos o dominios de mensajería:

- **Punto a Punto** (Point-To-Point, PTP), en la que un mensaje se consume por un único consumidor.
- **Publicación/Suscripción** (Pub/Sub), en la que un mensaje se consume por muchos consumidores.

Originalmente, los sistemas de mensajería empresarial fueron desarrollados para implementar un modelo ***PTP***, en el que un mensaje se consume por un único consumidor, pero puede haber varios emisores. El destino del mensaje es una cola FIFO definida y con un nombre.

En los últimos años, ha emergido un nuevo modelo, llamado ***Publicación y Suscripción*** (Publish and Subscribe, o Pub/Sub), que reemplaza el destino simple del modelo PTP con una jerarquía de contenidos, conocida como temas o tópicos (*topics*). Las aplicaciones que envían mensajes los publicitan, indicando que el mensaje representa información sobre un tema de la jerarquía. Las aplicaciones que desean recibir dichos mensajes se suscriben al tema en el que están interesadas. Suscribirse a temas en la jerarquía que contienen subtemas permite al subscriptor recibir todos los mensajes publicitados en el tema y en sus subtemas.

Múltiples aplicaciones pueden tanto suscribirse como publicitar mensajes en un mismo tema, sin necesidad de conocerse entre sí. El MOM actúa como un *broker*, redirigiendo los mensajes publicitados para un tema a todos los suscriptores de dicho tema.

2. Qué proporciona Java Message

La especificación de Java Message Service 1.0.2 indica que: *JMS es un conjunto de interfaces y semánticas asociadas que define cómo un cliente JMS accede a las facilidades de un producto de mensajería empresarial.*

Antes de JMS, cada suministrador MOM proporcionaba acceso a las aplicaciones a sus productos a través de una API propietaria, normalmente disponible en múltiples lenguajes, incluyendo el lenguaje Java. Con JMS se proporciona una forma estándar y portable para los programadores Java para enviar y recibir mensajes a través de cualquier producto MOM. Los programas escritos con JMS podrán ser ejecutados en cualquier MOM que implemente el estándar JMS.

La clave de la portabilidad de JMS es el hecho de que la API JMS se proporciona a través de un conjunto de interfaces. Los productos que deseen proporcionar la funcionalidad JMS deben proporcionar un *Proveedor* que implemente dichas interfaces. Por su parte, para construir una aplicación JMS basta con definir un conjunto de mensajes y un conjunto de aplicaciones cliente que intercambie dichos mensajes, haciendo uso de las interfaces JMS.

3. Componentes de JMS

A partir de la versión 1.3 de JavaEE, el API JMS forma parte de la especificación *enterprise* y los desarrolladores la pueden utilizar dentro de componentes JavaEE.

Una aplicación JMS está compuesta de los siguientes elementos:

- ***Clientes JMS***: programas Java que envían o reciben mensajes usando el API JMS. Cualquier componente JavaEE puede actuar como cliente JMS.
- ***Clientes no-JMS***: una aplicación escrita en un lenguaje que no es Java que envía y recibe mensajes. Es importante tener en cuenta que este tipo de programas podrán ser parte de una aplicación JMS y que su inclusión debe ser prevista durante el desarrollo de la aplicación.
- ***Mensajes***: el formato y contenido de los mensajes a intercambiar por clientes JMS y no-JMS es fundamental en el diseño de una aplicación JMS.
- ***Proveedor JMS***: Java Message Service define un conjunto de interfaces para las que un proveedor debe proporcionar una implementación específica para su producto MOM.
- ***Objetos administrados***: El proveedor crea objetos para facilitar la mensajería, pero que son independientes de las tecnologías propietarias del proveedor del sistema de mensajería.

3.1. Objetos administrados

Los proveedores de productos MOM difieren significativamente en los mecanismos y técnicas que utilizan para implementar la mensajería. Para permitir la portabilidad de los clientes JMS, los objetos que implementan las interfaces JMS deben ser independientes de las tecnologías propietarias del proveedor.

Esto se consigue con los *objetos administrados.* Estos objetos, que implementan las interfaces JMS, son creados por un administrador del sistema de mensajería del proveedor y se posicionan en el espacio de nombres JNDI (*Java Naming and Directory Interface*)[1].

Los programas JMS pueden obtener dichos objetos y accederlos a través de las interfaces JMS que implementan. El proveedor JMS deben proporcionar herramientas para la creación y administración de objetos y su posicionamiento en el espacio de nombres JNDI.

Existen dos tipos de objetos administrados:

- `ConnectionFactory` (Factoría de conexiones): utilizados para crear una conexión al sistema de mensajería soportado por el proveedor.
- `Destination` (Destinos): utilizados por los clientes JMS para especificar el destino de los mensajes a enviar, o bien la fuente de los mensajes a recibir.

Aunque los objetos administrados son instancias de clases específicas de una implementación del proveedor, se obtienen o recuperan utilizando mecanismos portables (en concreto, haciendo uso de JNDI) y se acceden a través de interfaces

[1] *Java Naming Directory Interface* (JNDI) es una interfaz de Java de servicios de directorio que permite a los clientes descubrir y buscar objetos/datos a través de un nombre.

portables (las proporcionadas por JMS). Por tanto, un programa JMS solamente necesita conocer el nombre JNDI y el tipo de interfaz JMS del objeto administrado, de modo que no se requiere ningún conocimiento más específico del proveedor.

La ***factoría de conexión*** es el objeto que utiliza el cliente para crear una conexión con el proveedor, encapsulando un conjunto de parámetros de configuración de la conexión que han sido previamente definidos por el administrador del servidor de mensajes. Cada factoría de conexión es una instancia de `ConnectionFactory`,. Al inicio de un cliente JMS, normalmente se inyecta un recurso de factoría de conexión en un objeto `ConnectionFactory`.

Por ejemplo, el siguiente fragmento de código muestra cómo se inyecta el recurso cuyo nombre es `DefaultJMSConnectionFactory` y se asigna a un objeto `ConnectionFactory`:

```
@Resource(lookup = "java:comp/DefaultJMSConnectionFactory")
private static ConnectionFactory connectionFactory;
```

Un ***destino*** (`javax.jms.Destination`) es el objeto que utiliza el cliente para especificar el destino de los mensajes que produce y el origen de los mensajes que consume. En **PTP** los destinos son las colas (`javax.jms.Queue`), mientras que en **Pub/Sub** son los temas (`javax.jms.Topic`). Una aplicación JMS puede utilizar múltiples colas o temas (o ambos).

Para crear un destino mediante el servidor de aplicaciones, hay que crear un recurso JMS que especifique un nombre JNDI para el destino. Dentro de la implementación del servidor de aplicaciones, cada destino referencia a un destino físico. Del mismo modo que con la factoría de conexiones, los destinos también se inyectan, pero en este caso son específicos a un dominio u otro.

Por ejemplo, el siguiente fragmento especifica dos recursos: una cola y un tema (o tópico). Los nombre de los recursos se mapean con destinos creados vía JNDI.

```
@Resource(mappedName=”jms/Queue”) private static Queue queue;

@Resource(mappedName=”jms/Topic”) private static Topic topic;
```

4. Interfaces

JMS define un conjunto de interfaces de alto nivel que encapsulan varios conceptos de mensajería. En la versión de JMS 2.0, estas interfaces son[2]:

- `ConnectionFactory`: Es un objeto administrado que permite crear un contexto de conexión, es decir, un `JMSContext`.
- `JMSContext`: Permite crear una conexión activa a un proveedor y un contexto o sesión para enviar y recibir mensajes, así como controlar las transacciones de dichos mensajes.
- `JMSProducer`: Utilizado para enviar mensajes
- `JMSConsumer`: Utilizado para recibir mensajes
- `Destination`: Es un objeto administrado que encapsula la identidad del destino de un mensaje, es decir, es el lugar al que los mensajes se envían o desde el que se reciben los mensajes.
- `Message`: Permite especificar el tipo de contenido del mensaje.

4.1 Mensajes

JMS proporciona diferentes tipos de mensajes para diferentes tipos de contenidos, pero todos los mensajes derivan de la interfaz *Message*.

Un mensaje se divide en tres partes:

- **Cabecera**: es un conjunto de campos fijos, definidos por JMS, utilizados por los clientes y proveedores para identificar y redirigir (en su caso) los mensajes.
- **Propiedades**: es una extensión de la cabecera. Consiste en un conjunto de campos, definidos por el programador, que permiten añadir campos opcionales al mensaje.
- **Cuerpo:** contiene el contenido propio del mensaje. Existen distintos tipos de cuerpos de mensaje y cada uno está definido por una interfaz que extiende la interfaz *`Message`*. Estas interfaces son:
  - `BytesMessages`: contiene bytes.
  - `ObjectMessage`: contiene un objeto Java `Serializable`.
  - `TextMessage`: contiene un `String`.
  - `MapMessage`: contiene un conjunto de pares nombre-valor, siendo los nombres de tipo `String` y los valores son primitivas Java.
  - `StreamMessage`: contiene un flujo de valores primitivos Java que se rellenan y leen secuencialmente usando operaciones estándar.

Cada proveedor proporciona clases específicas para su producto MOM que implementan estas interfaces.

---

[2] En versiones previas de JMS, las interfaces se denominan: *ConnectionFactory*, *Connection*, *Session* (estas dos interfaces están subsumidas en la versión 2.0 dentro de la interfaz *JMSContext*), *MessageProducer*, *MessageConsumer*, *Destination* y *Message*.

Las **propiedades** de un mensaje permiten añadir campos adicionales al mensaje. Por ejemplo, si una aplicación necesita categorizar o clasificar un mensaje de un modo no proporcionado en los campos fijos de la cabecera, se puede añadir una propiedad al mensaje que refleje dicha categorización.

JMS proporciona los métodos `set<Type>Property(...)` y `Property(...)` para fijar y obtener propiedades de una gran variedad de tipos Java, incluyendo `Object`. Además, JMS define un conjunto estándar de propiedades que pueden proporcionar los proveedores de forma opcional.

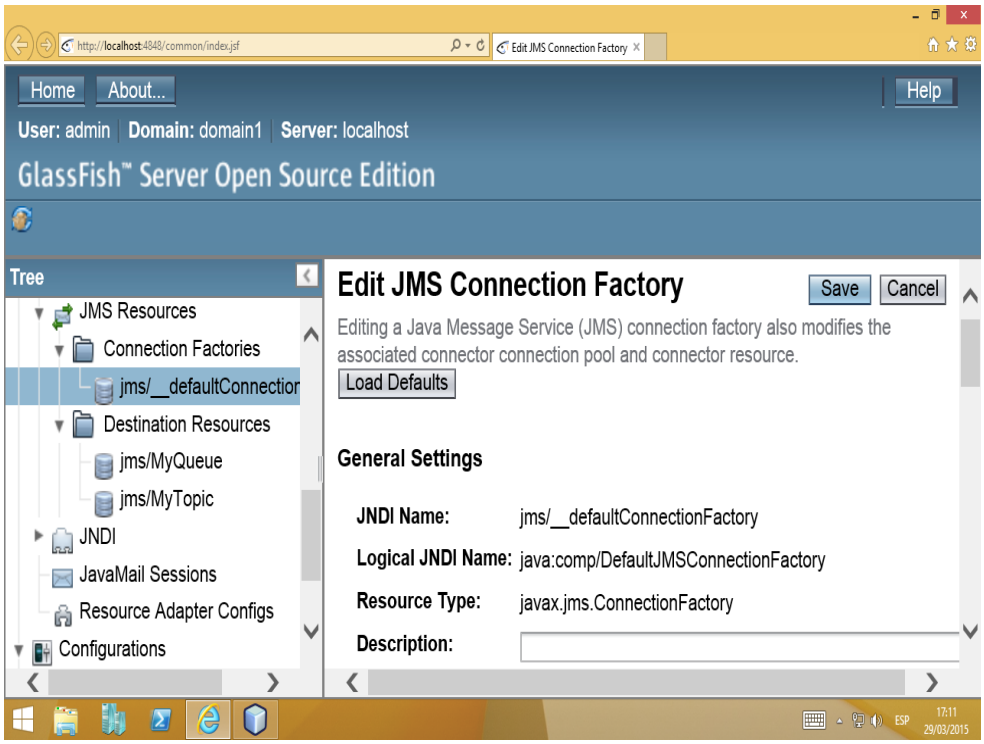
5. Cómo desarrollar un programa JMS

Para desarrollar un programa JMS típico se precisa seguir los siguientes pasos:

1) Buscar una `ConnectionFactory` mediante el JNDI.
2) Buscar una o más `Destination` mediante el JNDI.
3) Utilizar la `ConnectionFactory` para crear un `JMSContext`. De este modo se crea una conexión con el proveedor y una sesión para el envío y recepción de mensajes.
4) Utilizar el `JMSContext` y la `Destination` para crear los `JMSProducer` y `JMSConsumer` necesarios.
5) A partir de ahí, los productores y consumidores podrán enviar y recibir mensajes, respectivamente, utilizando la conexión (contexto) y el destino (cola o tema) creados.

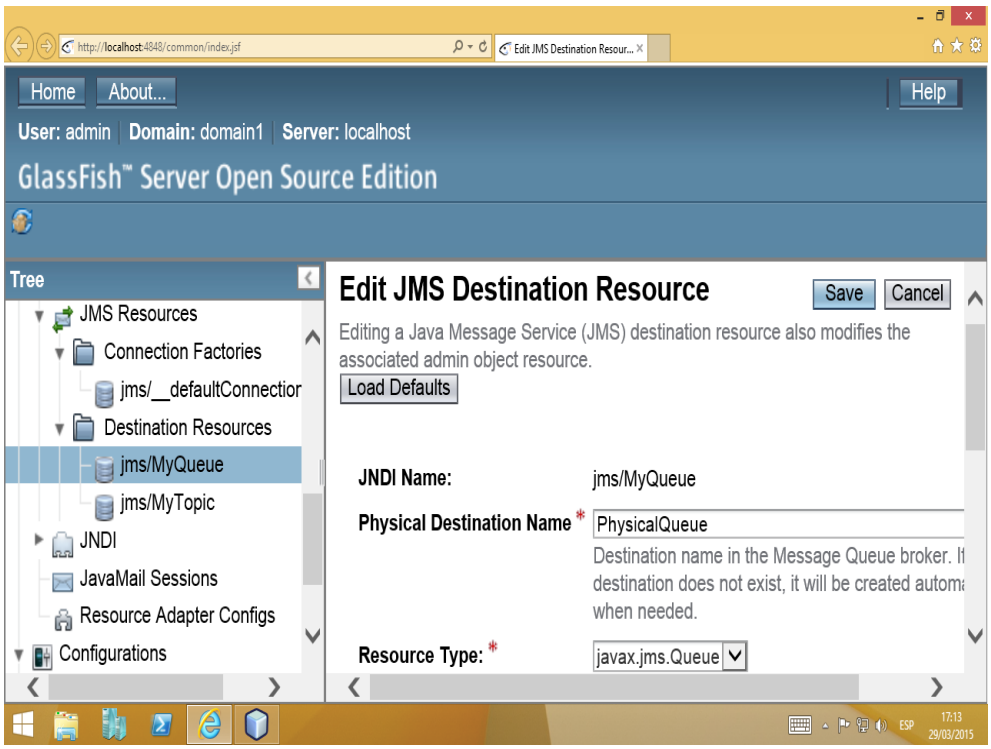
A continuación veremos estos pasos con más detalle, a través de un ejemplo. En este ejemplo, se desea crear un ***productor*** que envíe un mensaje de texto a una determinada cola, llamada "*jms/MyQueue*"; así como un ***consumidor*** que reciba mensajes de esa cola, muestre el mensaje y finalice si no ha recibido mensajes durante 10 segundos. Como proveedor de JMS, utilizaremos GlassFish 4.

Lo primero que debemos realizar es editar una `ConnectionFactory` a través del proveedor de JMS. Para ello, en la interfaz gráfica de GlassFish 4, editamos la *`jms/_defaultConnectionFactory`*, pudiéndole asignar una descripción.



A continuación creamos tantos recursos `Destination` como necesitemos. Por ejemplo, podemos crear una cola llamada `jms/MyQueue`, o bien un tema (*topic*) llamado `jms/MyTopic`.

En la siguiente figura podemos ver la edición de una cola, a la que asignamos como nombre físico "`PhysicalQueue`", y cuyo nombre en el JNDI es `jms/MyQueue`.



Creación del Productor

En el editor de Java, al crear el código para el productor del mensaje, debemos seguir los siguientes pasos:

a) Importar las interfaces JMS

```
import javax.jms.ConnectionFactory;

import javax.jms.Destination;
```

b) En la clase del código del productor, debemos acceder a los objetos administrados, es decir, a la `ConnectionFactory` y al `Destination` (que en nuestro ejemplo es una cola, i.e. `Queue`).

```
@Resource(lookup = "java:comp/DefaultJMSConnectionFactory")
private static ConnectionFactory connectionFactory;

@Resource(lookup = "jms/MyQueue”) private static Queue
queue;
```

El acceso a los objetos administrados se ha realizado mediante la inyección de recursos vía anotaciones, utilizando para ello `@Resource(lookup=....)`

c) En el método *main*, que puede lanzar la excepción JMSException, debemos:

- crear el contexto a partir de la factoría de conexiones

```
public static void main(String[] args) throws JMSException
{ /// Crea contexto a partir de la factoría de conexiones
try (JMSContext context =
        connectionFactory.createContext();) {
```

- crear el productor de mensajes a partir del contexto

```
JMSProducer producer = context.createProducer();
```

- crear el mensaje a partir del contexto

```
TextMessage message = context.createTextMessage();
```

- construir el cuerpo del mensaje (en este caso, de tipo texto)

```
message.setText("This is a message");
```

- y añadimos una propiedad al mensaje, para indicar que es muy importante.

```
// Crea y asocia al mensaje una propiedad
message.setBooleanProperty("Important", true);
```

d) Finalmente, se envía el mensaje a la cola destino, mediante el productor de mensajes. Dicho productor esperará a que el proveedor JMS lo guarde en el middleware de mensajería, pero no a que lo reciba el cliente.

```
    producer.send(queue, message);

} catch (JMSRuntimeException e) { ...
```

Creación del Consumidor

De forma similar, en el editor de Java, al crear el código para el consumidor del mensaje, debemos seguir los siguientes pasos:
a) Importar las interfaces JMS

```
import javax.jms.ConnectionFactory;

import javax.jms.Destination;
```

b) En la clase del código del consumidor, debemos acceder a los objetos administrados, es decir, a la `ConnectionFactory` y al `Destination` (que en nuestro ejemplo es una cola, i.e. `Queue`).

```
@Resource(lookup = "java:comp/DefaultJMSConnectionFactory")
private static ConnectionFactory connectionFactory;

@Resource(lookup = "jms/MyQueue") private static Queue
queue;
```

c) En el método *main*, que puede lanzar la excepción JMSException, debemos:

- crear el contexto a partir de la factoría de conexiones

```
public static void main(String[] args) throws JMSException
{ //// Crea contexto a partir de la factoría de conexiones
try (JMSContext context =
        connectionFactory.createContext();) {
```

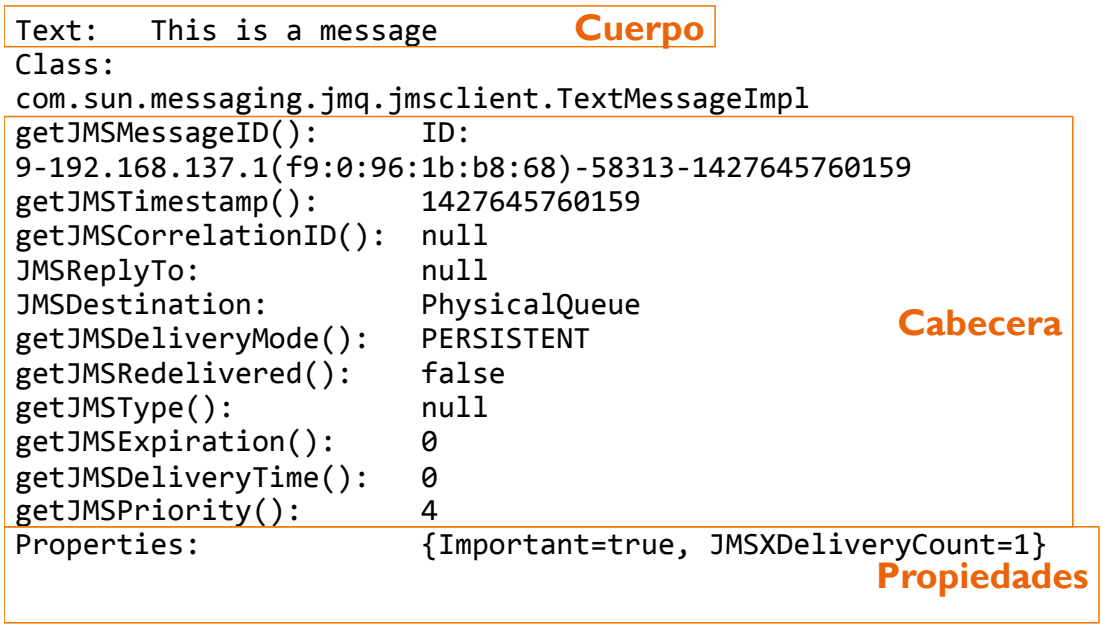
- crear el consumidor de mensajes a partir del contexto. Importante: el consumidor estará asociado a un destino particular (en este caso, a la cola que hemos obtenido anteriormente).

```
JMSConsumer consumer = context.createConsumer(queue);
```

- esperar mensaje de la cola y mostrarlo. En este caso, esperaremos durante 10 segundos como máximo, terminando si pasado ese tiempo no se ha recibido ningún mensaje.

```
while (true) {
   // Espera un mensaje durante 10 segundos máximo
   Message m = consumer.receive(10000);
   if (m != null) { // Mensaje recibido
       System.out.println(m);
   } else { // Espera máxima alcanzada
            break;
   }
}
```

Finalmente, mostramos a continuación un ejemplo de mensaje que se habría enviado (y recibido) en la aplicación que hemos creado. Podemos ver las distintas partes de este mensaje, así como la propiedad "Important" que hemos añadido en la zona de "propiedades".



REFERENCIAS:

Este documento se ha elaborado a partir de la información contenida en las siguientes referencias:

- ***Introducción a JMS (Java Message Service)***. Departamento de Ciencia de la Computación e Inteligencia Artificial. Universidad de Alicante.
http://expertojava.ua.es/j2ee/publico/mens-2010-11/sesion01-apuntes.html

- *Introducing the Java Message Service*. Willy Farrel. ibm.com/developerWorks
http://www.ibm.com/developerworks/java/tutorials/j-jms/j-jms-updated.html

- *Java Message Service Concepts*. Java Platform, Enterprise Edition: The Java EE Tutorial. Chapter 45. Oracle Java Documentation.
http://docs.oracle.com/javaee/7/tutorial/partmessaging.htm#GFIRP3

- *Package javax.jms*
http://docs.oracle.com/javaee/7/api/javax/jms/package-summary.html