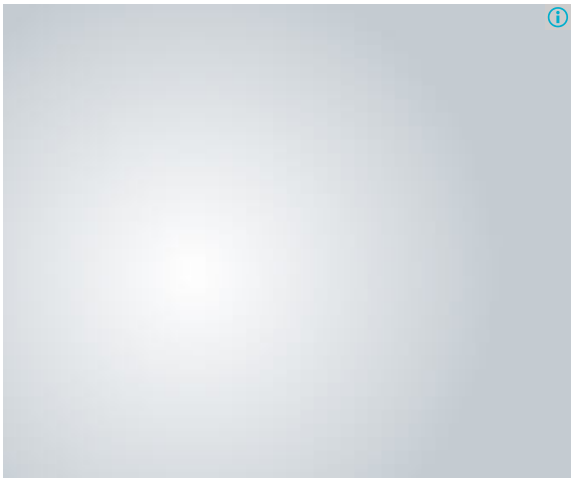
JMS (Java Message Service) Tutorial

April 13, 2016 by Lokesh Gupta



Java Message Service is an API which supports the formal communication called as **messaging between computers** on a network. It provides a common interface for standard message protocols and message services in support to the Java programs. It provides facility to create, send and read messages. The JMS API reduces the concepts that a programmer must learn to use the messaging services/products and it also provides the features that support the messaging applications. It is a technique which is used in J2EE technology for an application to communicate with the other application in loosely coupled manner. It means that the applications which have to communicate are not connected directly they are connected through a common destination. We will go in detail later in this tutorial.

Table of Contents

Need of JMS

In Java, if a person wants to send a message from one application to another in such a way that both application do not know anything about each other, even they may be deployed in separate continents with no dependency at all. For example, one application A is running in India and another application is running in USA, and B is interested in getting some updates/messages from A – whenever something unique happen on A. There may be N number of such applications who are interested in such updates from A.

In this scenario, java provides it's best solution in form of JMS – and solve the exactly same problem discussed above.

The JMS is also useful when we are writing any event based application like chat server where it needs a publish event mechanism to send messages between the server to the clients who are connected with the server. As the JMS is different from RMI so there is no need of the destination object to be available online while sending a message from the client to the server. Server publish the message and forget it, whenever client comes online, it will fetch the message. It's very powerful solution for very common problems in today's world.

Advantages of JMS

1. ### Asynchronous

   JMS is asynchronous by default. So to receive a message, the client is not required to send the request. The message will arrive automatically to the client as they become available.

2. ### Reliable

   JMS provides the facility of assurance that the message will delivered once and only once. You know that duplicate messages create problems. JMS helps you avoiding such problems.

JMS Messaging Domains

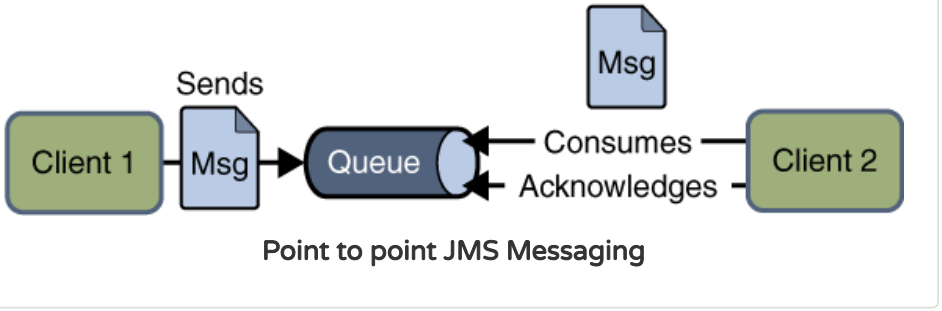
Before the JMS API existed, most messaging products supported either the **point-to-point** or the **publish/subscribe** approach to messaging. The JMS provides a separate domain for each approach and defines the compliance for each domain. Any JMS provider can implement both or one domain, it's his own choice. The JMS provides the common interfaces which enables us to use the JMS API in such a way that it is not specific to the either domain.

Let's see both type of messaging domains in more detail:

Point-to-Point Messaging Domain

In the point-to-point messaging domain the application is built on the basis of message queues, senders and receivers. Each and every message is addressed to a particular **queue**. Queues retain all messages sent to them until the messages are consumed or expired. There are some characteristics of PTP messaging:

1. There is only one client for each message.
2. There is no timing dependency for sender and receiver of a message.
3. The receiver can fetch message whether it is running or not when the sender sends the message.
4. The receiver sends the acknowledgement after receiving the message.



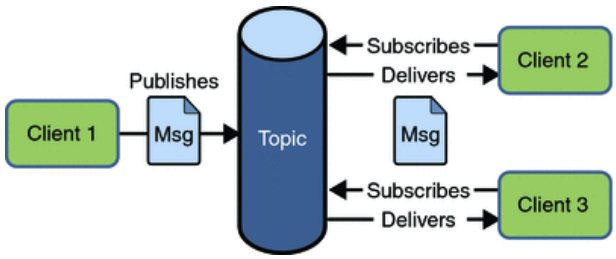
Point to point JMS Messaging

Publish/Subscribe Messaging Domain

In publish/subscribe messaging domain, only one message is published which is delivered to all clients through Topic which acts as a bulletin board. Publishers and subscribers are generally anonymous and can dynamically publish or subscribe to the topic. The Topic is responsible to hold and deliver messages. The topic retains messages as long as it takes to distribute to the present clients.

Some of the characteristics are:

1. There can be multiple subscribers for a message.
2. The publisher and subscribe have a timing dependency. A client that subscribes to a topic can consume only messages published after the client has created a subscription, and the subscriber must continue to be active in order for it to consume messages.



Publish Subscribe JMS Messaging

Read More: HornetQ Basic Example

Message Consumption

In JMS, the message consumption can be done in two ways:

Synchronous

In synchronous message consumption, the subscriber/receiver request the message from the destination by calling the `receive()` method. In the `receive()` method will block till the message arrives or time out if the message does not arrive within a given time. Just like normal java method calls with some return value.

Asynchronous

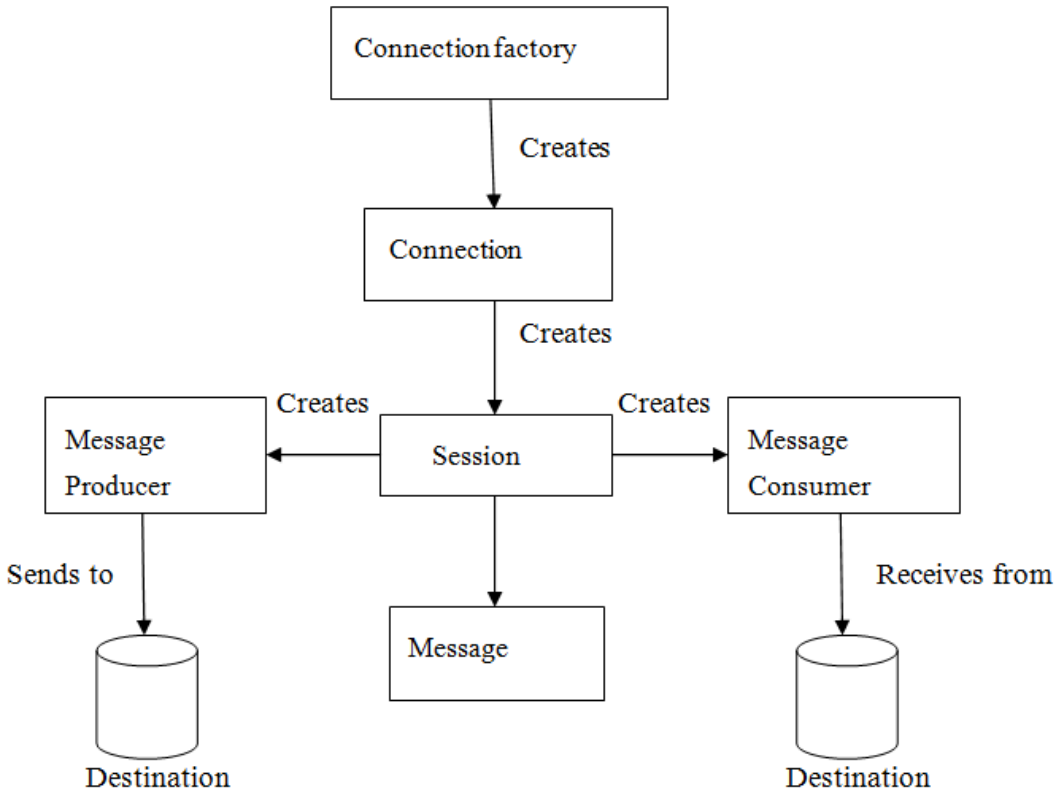
In asynchronous message consumption, a subscriber can register (or subscribe) as message listener with the consumer. The message listener is same as event listener, whenever the message arrives at the destination the JMS provider will deliver the message by calling the listener's `onMessage()` method which will act on the content of the message.

JMS participating objects

JMS application has some basic building blocks, which are:

1. Administered objects – Connection Factories and Destination

2. Connections
3. Sessions
4. Message Producers
5. Message Consumers
6. Message Listeners



JMS API Programming Model

JMS Administered Objects

JMS application provides two types of administered objects:

- Connection Factories
- Destinations

These two administered objects are created by JMS system administrator in JMS Provider by using the Application Server the admin console. These both objects are stored in Application server JNDI Directory or JNDI Registry.

Connection Factories

The client uses an object which is a connection factory used to create a connection to a provider. It creates connection between JMS Provider and JMS Client. When JMS Client such as sender or receiver search out for this object in JNDI Registry, then the JMS Client receives one connection object which is nothing just a physical connection between JMS Provider and JMS Client. Using this connection, client can have communication with the destination object to send or receive messages into Queue or Topic. Let us have an example to understand it to send the message:

```
QueueConnectionFactory queueConnFactory = (QueueConnectionFactory) initialCtx.lookup
("primaryQCF");
Queue purchaseQueue = (Queue) initialCtx.lookup ("Purchase_Queue");
Queue returnQueue = (Queue) initialCtx.lookup ("Return_Queue");
```

Destination

Client uses an object known as destination which is used to specify the target of messages it produces and the source of message who consumes it. The JMS application uses two types of destination Queue or Topic. The code specifies queue and a topic.

Create queue session

```
QueueSession ses = con.createQueueSession (false, Session.AUTO_ACKNOWLEDGE);  //get the Queue
object
Queue t = (Queue) ctx.lookup ("myQueue");  //create QueueReceiver
QueueReceiver receiver = ses.createReceiver(t);
```

Create topic session

```
TopicSession ses = con.createTopicSession (false, Session.AUTO_ACKNOWLEDGE); // get the Topic
object
Topic t = (Topic) ctx.lookup ("myTopic");  //create TopicSubscriber
TopicSubscriber receiver = ses.createSubscriber(t);
```

JMS Connection

The connection encapsulates the virtual connection with a JMS Provider. The connection implements the Connection interface, when it will have a ConnectionFactory object then we can use this to create a connection.

```
Connection connection = connectionFactory.createConnection();
```

After creating any connection they should be closed before the application completes using:

```
connection.close();
```

JMS Session

The session is a single threaded context which is used for producing and consuming messages. The sessions are used to create the following:

- Message Producers
- Message Consumers

The session implements the Session interface and after creating a Connection object we use this to create a Session.

```
Session session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);
```

JMS Message Producer

The message producer is an object which is created by a session and is used for sending messages to the destination. This implements the `MessageProducer` interface. We use Session to create a `MessageProducer` for the destination, queue or topic object.

```
MessageProducer producer = session.createProducer(dest);
MessageProducer producer = session.createProducer(queue);
MessageProducer producer = session.createProducer(topic);
```

After creating the message producer is used to send the messages by using the send method.

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

JMS Message Consumer

The message consumer is an object which is created by a session and is used to receive messages sent at a destination. It will implement the `MessageConsumer` interface. We use a session to create a MessageConsumer for a destination, queue or topic object.

```
MessageConsumer consumer = session.createConsumer(dest);
MessageConsumer consumer = session.createConsumer(queue);
MessageConsumer consumer = session.createConsumer(topic);
```

JMS Message Listeners

The message listener is an object which acts as asynchronous event handler for messages. The message listener implements the `MessageListener` interface which contains the one method `onMessage()`. In this method we define the actions to be performed when message arrives. By using `setMessageListener()` we define the message listener with a specific `MessageConsumer`.

```
Listener myListener = new Listener();
consumer.setMessageListener(myListener);
```

JMS Message Components

The JMS Messages are used by the JMS Clients to have communication between systems. The JMS messages have simple format but highly flexible, which allows to create messages which match the formats. The JMS message is divided into three parts. They are:

1. ### Message Header

   The JMS message header contains the number of predefined fields which contain those values which are used by the clients and providers to identify and send messages. The predefined headers are:

- JMSDestination
- JMSDeliveryMode
- JMSMessageID
- JMSTimestamp
- JMSCorrelationID
- JMSReplyTo
- JMSRedelivered
- JMSType
- JMSExpiration
- JMSPriority

2. ## Message Properties

   In message properties we can create and set properties for messages. The message properties are custom name value pairs which are set or read by applications. The message properties are useful for supporting filtering messages. The JMS API provides some predefined property that a provider can support. The message property is optional.

3. ## Message Body

   In message bodies the JMS API defines five message body formats which are also called as message types which allow us to send and receive data in many different forms and also provides compatibility with the existing messaging formats. It basically consists of the actual message sent from JMS sender to receiver. The different message types are:

   Text message : Represented by javax.jms.TextMessage. It is used to represent a block of text.
   Object message : Represented by javax.jms.ObjectMessage. It is used to represent a java object.
   Bytes message : Represented by javax.jms.BytesMessage. It is used to represent the binary data.
   Stream message : Represented by javax.jms.StreamMessage. It is used to represent a list of java primitive values.
   Map message : Represented by javax.jms.MapMessage. It is used to represent a set of keyword or value pairs.

That's all for **JMS introduction tutorial** and it's related terminologies. In next set of posts. I will give some examples of JMS.

Happy Learning !!

References(s):

JSR 914: JavaTM Message Service (JMS) API
JMS Reference