# JMS:

Queue = POP

Topic : Subscribe/publish

1. [Queue\_Vs\_topic](#Queue_Vs_topic)
2. [Active MQ jms example](#ActiveMQExample)

What is JMS,MQ Queue,

JMS is the specification provided by Sun for messaging. MQ Queue is the IBM's implementation of JMS. Similary [JBoss](http://www.coderanch.com/forums/f-63/JBoss) has its own implementation. JMS Queue is the generic term. MQ Queue is the concrete implementation provided by IBM.

**ASYNC**: In asynchronous communication, the requester of a service does not wait for a response from the service, it can continue processing once it sends the request (message).Asynchronous communication deals better with increase number of requests and traffic than synchronous communication, because of no waiting for response feature. Moreover, messages in asynchronous communication are not lost rather delayed in case of intensive traffic.  
**SYNC**: Whereas the requester in synchronous communication must wait for a response from the service, the requester is blocked till an acknowledgment is received. In synchronous communication, both parties must be active, otherwise the transaction won't be completed.

Very good architecture diagram at below link

<http://docs.oracle.com/javaee/1.4/tutorial/doc/JMS3.html#wp78649>

[Figure 33-2](http://docs.oracle.com/javaee/1.4/tutorial/doc/JMS3.html" \l "wp78711) illustrates the way these parts interact. Administrative tools allow you to bind destinations and connection factories into a JNDI namespace. A JMS client can then look up the administered objects in the namespace and then establish a logical connection to the same objects through the JMS provider.

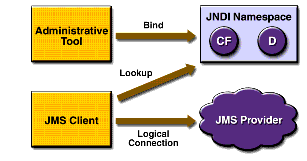


Figure 33-2 JMS API Architecture

[Figure 33-2](http://docs.oracle.com/javaee/1.4/tutorial/doc/JMS3.html#wp78711) illustrates the way these parts interact. Administrative tools allow you to bind destinations and connection factories into a JNDI namespace. A JMS client can then look up the administered objects in the namespace and then establish a logical connection to the same objects through the JMS provider.

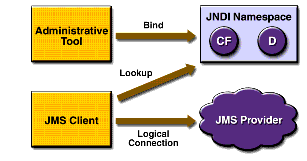
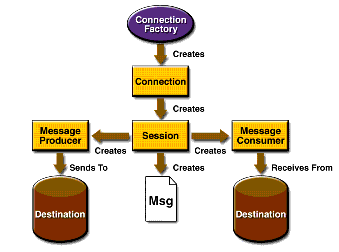


Figure 33-2 JMS API Architecture

[Figure 33-5](http://docs.oracle.com/javaee/1.4/tutorial/doc/JMS4.html#wp78861) shows how all these objects fit together in a JMS client application.



The **Java Message Service** (**JMS**) [API](http://en.wikipedia.org/wiki/Application_Programming_Interface) is a [Java](http://en.wikipedia.org/wiki/Java_(programming_language)) [Message Oriented Middleware](http://en.wikipedia.org/wiki/Message_Oriented_Middleware) (MOM) API[[1]](http://en.wikipedia.org/wiki/Java_Message_Service#cite_note-1) for sending messages between two or more [clients](http://en.wikipedia.org/wiki/Client_(computing)). JMS is a part of the [Java Platform, Enterprise Edition](http://en.wikipedia.org/wiki/Java_Platform,_Enterprise_Edition), and is defined by a specification developed under the [Java Community Process](http://en.wikipedia.org/wiki/Java_Community_Process) as JSR 914.[[2]](http://en.wikipedia.org/wiki/Java_Message_Service#cite_note-JSR914-2) It is a messaging standard that allows application components based on the Java Enterprise Edition (Java EE) to create, send, receive, and read messages. It allows the communication between different components of a [*distributed application*](http://en.wikipedia.org/wiki/Distributed_computing) to be [loosely coupled](http://en.wikipedia.org/wiki/Loose_coupling), reliable, and asynchronous.[[3]](http://en.wikipedia.org/wiki/Java_Message_Service#cite_note-oraclejms-3)

activeMQ, easy to use open source message oriented middleware (MOM)

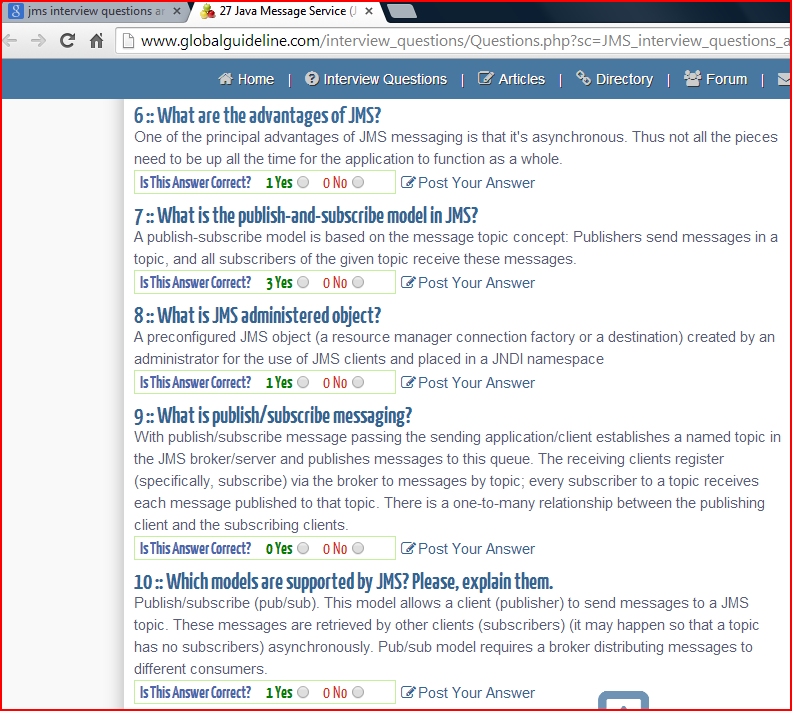
ActiveMQ is message oriented middleware (MOM), useful for receiving and processing asynchronous messages (queues or topics).

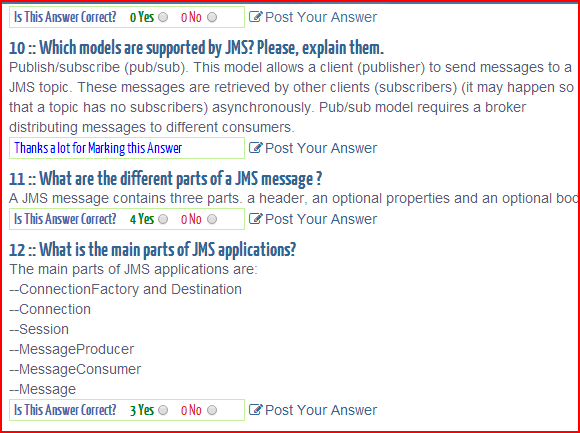
It is easily integrated with Java based applications (and others) via  the JMS API.

ActiveQ offers enterprise scale features which may be required when implementing complex distributred systems such as

* High availability with failover.
* Scalability and clustering through a distributed Network of Brokers.
* Pluggable persistence stores (JDBC, BDB, JDBM, file system).
* Distributed destinations and XA support.
* Caching.
* Connection pooling.
* Support for cross language clients such as .NET, C++.
* Support for scripting environments such as Perl, Python, and Ruby.
* Multiple Transport layers (TCP, UDP, multicast, NIO, SSL, Zeroconf, JXTA, JGroups).















<http://javadecodedquestions.blogspot.com/2012/02/jms-interview-questions.html>

11. **Whats is a subtle difference between a "durable" JMS message and a "persistent" JMS message ?**  
A "durable" JMS message is applicable only to publish/subscribe paradigm and the "persistent" JMS message is applicable to both "Point-to-Point" & "publish/subcribe" paradigms.   
Let's look at both:

A " **durable message** " is a message where the JMS server will hold on to a message if the subscriber is temporarily unavaliable. So the durability is defined by the relationship between a "Topicc Subscriber" and the "JMS Server". Durability is applicable only to publish/Subscribe paradigm. For this to happen subscribers need to register themselves with a unique " **client id** ".

A " **persistent message** " is a message that defines the relationship between a "Message Producer" and the "JMS Server". This can be established for both   
point-to-point and publish/subscribe. This has to do with the guaranteed once only delivery of the message by persisting the message after it has been received from the message producer.

<http://integrationspot.blogspot.com/2011/03/jms-queue-difference-between-queue-and.html>

**The comparison: Queue VS Topic**  
  
**Queue:**

* Point-to-point model
* Only one consumer gets the message
* Messages have to be delivered in the order sent
* A JMS queue only guarantees that each message is processed only once.
* The Queue knows who the consumer or the JMS client is. The destination is known.
* The JMS client (the consumer) does not have to be  active or connected to the queue all the time to receive or read the message.
* Every message successfully processed is acknowledged by the consumer.

Descriptive example: A JMS queue is a channel through which users "pull" messages they want to receive using the p2p model, instead of automatically receiving messages on a particular topic. The producer submits messages to the queue, and recipients can browse the queue and decide which messages they wish to receive. In the p2p model, users can see the contents of the messages held in the queue before deciding whether or not to accept their delivery.

Read more: [JMS Queue vs. JMS Topic | eHow.com](http://www.ehow.com/info_8035572_jms-queue-vs-jms-topic.html#ixzz1I1rcTSdS)

    
**Topic:** 

Publish/subscribe model

* Multiple clients subscribe to the message
* There is no guarantee messages have to be delivered in the order sent
* There is no guarantees that each message is processed only once. -- As this can be sensed from the model
* The Topic, have multiple subscribers and there is a chance that the topic does not know all the subscribers. The destination is unknown.
* The subscriber / JMS client needs to the active when the messages are produced by the producer, unless the subscription was a durable subscription.
* No, Every message successfully processed is not acknowledged by the consumer/subscriber.

Descriptive Example: A JMS topic is the channel through which users subscribe to receive specific messages from a producer in the publish-and-subscribe model of JMS messaging. The model can be compared to subscribing to a newspaper; for example, if John Doe subscribed to "The New York Times," he would receive the paper every day from the newspaper producer. Similarly, if John Doe used JMS messaging to subscribe to a particular topic, he would receive all sent messages from a producer regarding that topic.

Example: JMS Example with Active MQ

<http://www.javablogging.com/simple-guide-to-java-message-service-jms-using-activemq/>

Now that we have a JMS provider running, let’s write our message producer and consumer programs. For that, you will need to put the ActiveMQ’s JAR file on the class path. The file you need is called (for version 5.3.0) ‘activemq-all-5.3.0.jar’ or something similar and is in the extracted ActiveMQ directory. In Eclipse you could click right mouse button on your project and choose Properties->Java Build Path->Libraries->Add External Library.

Here is the code of the program sending (producing) the messages:

|  |  |
| --- | --- |
| 1:  2:  3:  4:  5:  6:  7:  8:  9:  10:  11:  12:  13:  14:  15:  16:  17:  18:  19:  20:  21:  22:  23:  24:  25:  26:  27:  28:  29:  30:  31:  32:  33:  34:  35:  36:  37:  38:  39:  40:  41:  42:  43:  44:  45: | import javax.jms.\*;  import org.apache.activemq.ActiveMQConnection;  import org.apache.activemq.ActiveMQConnectionFactory;  **public** **class** Producer {  // URL of the JMS server. DEFAULT\_BROKER\_URL will just mean  // that JMS server is on localhost  **private** **static** String url = ActiveMQConnection.DEFAULT\_BROKER\_URL;  // Name of the queue we will be sending messages to  **private** **static** String subject = "TESTQUEUE";  **public** **static** **void** main(String[] args) **throws** JMSException {  // Getting JMS connection from the server and starting it  ConnectionFactory connectionFactory =  **new** ActiveMQConnectionFactory(url);  Connection connection = connectionFactory.createConnection();  connection.start();  // JMS messages are sent and received using a Session. We will  // create here a non-transactional session object. If you want  // to use transactions you should set the first parameter to 'true'  Session session = connection.createSession(**false**,  Session.AUTO\_ACKNOWLEDGE);  // Destination represents here our queue 'TESTQUEUE' on the  // JMS server. You don't have to do anything special on the  // server to create it, it will be created automatically.  Destination destination = session.createQueue(subject);  // MessageProducer is used for sending messages (as opposed  // to MessageConsumer which is used for receiving them)  MessageProducer producer = session.createProducer(destination);  // We will send a small text message saying 'Hello' in Japanese  TextMessage message = session.createTextMessage("こんにちは");  // Here we are sending the message!  producer.send(message);  System.out.println("Sent message '" + message.getText() + "'");  connection.close();  }  } |

<http://java.dzone.com/articles/jms-activemq>

01.package com.eviac.blog.jms;

02.

03.import javax.jms.\*;

04.

05.import org.apache.activemq.ActiveMQConnection;

06.import org.apache.activemq.ActiveMQConnectionFactory;

07.import org.apache.log4j.BasicConfigurator;

08.

09.public class Consumer {

10. // URL of the JMS server

11. private static String url = ActiveMQConnection.DEFAULT\_BROKER\_URL;

12.

13. // Name of the queue we will receive messages from

14. private static String subject = "MYQUEUE";

15.

16. public static void main(String[] args) throws JMSException {

17.  BasicConfigurator.configure();

18.  // Getting JMS connection from the server

19.  ConnectionFactory connectionFactory = new ActiveMQConnectionFactory(url);

20.  Connection connection = connectionFactory.createConnection();

21.  connection.start();

22.

23.  // Creating session for seding messages

24.  Session session = connection.createSession(false,

25.    Session.AUTO\_ACKNOWLEDGE);

26.

27.  // Getting the queue

28.  Destination destination = session.createQueue(subject);

29.

30.  // MessageConsumer is used for receiving (consuming) messages

31.  MessageConsumer consumer = session.createConsumer(destination);

32.

33.  // Here we receive the message.

34.  // By default this call is blocking, which means it will wait

35.  // for a message to arrive on the queue.

36.  Message message = consumer.receive();

37.

38.  // There are many types of Message and TextMessage

39.  // is just one of them. Producer sent us a TextMessage

40.  // so we must cast to it to get access to its .getText()

41.  // method.

42.  if (message instanceof TextMessage) {

43.   TextMessage textMessage = (TextMessage) message;

44.   System.out.println("Received message '" + textMessage.getText()

45.     + "'");

46.  }

47.  connection.close();

48. }

49.}

able 30–2 JMS Message Types

| **Message Type** | **Body Contains** |
| --- | --- |
| TextMessage | A java.lang.String object (for example, the contents of an XML file). |
| MapMessage | A set of name-value pairs, with names as String objects and values as primitive types in the Java programming language. The entries can be accessed sequentially by enumerator or randomly by name. The order of the entries is undefined. |
| BytesMessage | A stream of uninterpreted bytes. This message type is for literally encoding a body to match an existing message format. |
| StreamMessage | A stream of primitive values in the Java programming language, filled and read sequentially. |
| ObjectMessage | A Serializable object in the Java programming language. |
| Message | Nothing. Composed of header fields and properties only. This message type is useful when a message body is not required. |

# Difference between SOAP over HTTP and SOAP over JMS

SOAP over HTTP: (webservice)

* Firewall friendly is that all services support this transport
* Supported on all platforms (easiest connectivity in b2b scenario)
* Clients can be simple and lightweight
* It is not always reliable. No guaranteed delivery.

SOAP over JMS: (JMS)

* Assured delivery/only once delivery using either persistent queues or durable topics.
* Asynchronous support
* Publish/subscribe
* Queuing if better for achieving larger scalability and reliability
* Better handles temporary high load
* Large volume of messages (EDA)
* Better support in middleware software
* Transaction boundary

In SOA architecture best practice is to use JMS internally (for clients/providers that can easily connect to ESB) and HTTP for connecting to outside partners (over internet).

Using SOAP over JMS gives  some advantages compared to HTTP,

specially related to reliability as you may use the persistence and acknowledgment features built in the standard.

The same applies if you need to establish asynchronous communication or need to use the load balancing features provided by JMS servers.

We can achieve this using http but the implementation would be much more complicated.  
  
If we do SOAP over JMS, in fact we can do load balancing..  whereas with SOAP over HTTP requires additional hardware like IP Sprayers.