

JMS Tutorial – Java Message Service Tutorial

📅 Last Updated: December 18, 2021 👤 By: Lokesh Gupta 📁 JMS 🔖 JMS

[Java Message Service](#) is an API that supports the formal communication called **messaging between computers** on a network. JMS provides a common interface for standard message protocols and message services in support of the Java programs.

JMS provides the **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 also provides the features that support the messaging applications.

JMS helps in building the communication between two or more applications in a loosely coupled manner. It means that the applications which have to communicate are not connected directly they are connected through a common destination.

Table Of Contents ▼

1. Why We Need JMS?

When one application wants to send a message to another application in such a way that both applications do not know anything about each other; and even they may NOT be deployed on the same server.

For example, one application A is running in India and another application B is running in the USA, and A is interested in sending some updates/messages to B – whenever something unique happens on A. There may be N number of such applications that are interested in such updates to B.

In this scenario, Java provides one of the solutions in form of JMS – and solves the exact same problem discussed above.

JMS is also useful when we are writing any event-based application like a [chat server](#) where it needs a publish event mechanism to send messages between the server and the clients.

Note that JMS is different from RMI so there is no need for the destination object to be available online while sending a message. The publisher publishes the message and forgets it, whenever the receiver comes online, it will fetch the message. It's a very powerful solution for very common problems in today's world.

2. Benefits of JMS

2.2. Reliable

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

3. JMS Messaging Domains

Even before the JMS API existed, most messaging products supported either the **point-to-point** or the **publish/subscribe** approach to messaging. The JMS also provides a separate domain for each of both approaches 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 enable us to use the JMS API in such a way that it is not specific to either domain.

Let's see both types of messaging domains in more detail to understand **how JMS works**.

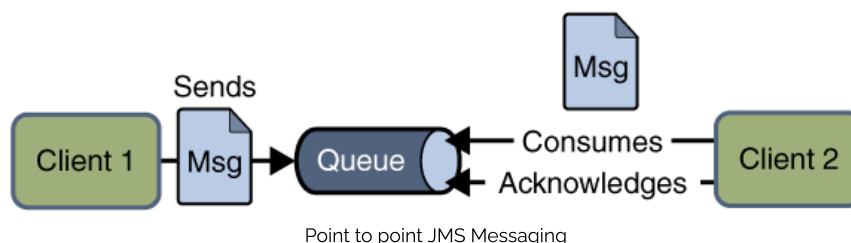
3.1. Point-to-Point Messaging

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.



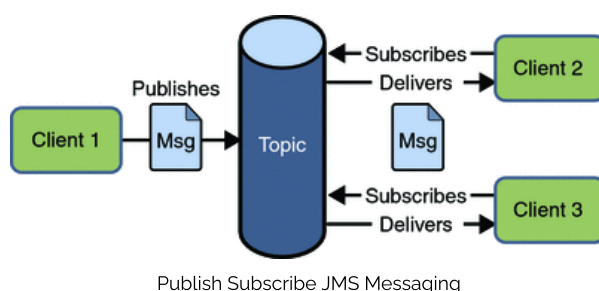
3.2. Publish/Subscribe Messaging

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.



Read More: [HornetQ Basic Example](#)

4. Receiving a Message

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

4.1. Synchronous

In synchronous message consumption, the subscriber/receiver requests the message from the destination by calling the `receive()` method.

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.

4.2. Asynchronous

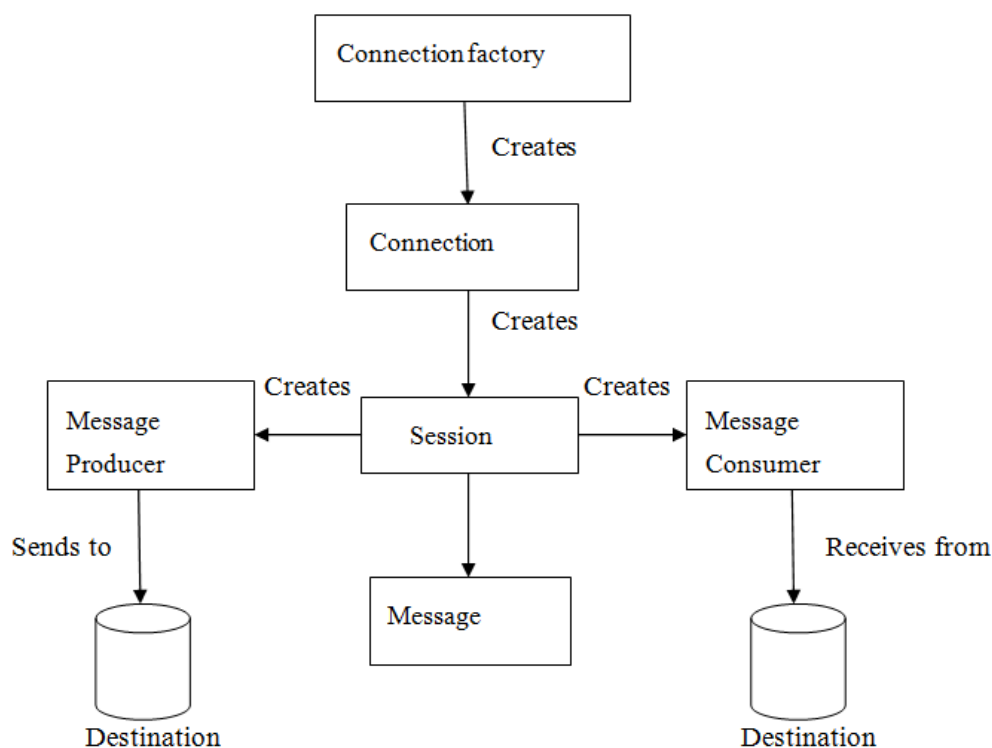
In asynchronous message consumption, a subscriber can register (or subscribe) as a *message listener* with the consumer.

The message listener is the same as the 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.

5. 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

5.1. JMS Administered Objects

JMS application provides two types of administered objects:

- Connection Factories
- Destinations

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

5.2. Connection Factories

The client uses an object which is a [connection factory](#) used to create a connection to a provider. It creates a 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, the 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 ("primaryQueue");
Queue purchaseQueue = (Queue) initialCtx.lookup ("Purchase_Queue");
Queue returnQueue = (Queue) initialCtx.lookup ("Return_Queue");
```

5.3. Destination

The client uses an object known as a 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 destinations:

- Queue
- Topic

The code examples to create a queue or a topic:

Create queue session

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

5.4. 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();
```

5.5. JMS Session

The *session* is a single-threaded context that 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);
```

5.6. Message Producer

A 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);
```

5.7. Message Consumer

A 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);
```

5.8. Message Listener

A message listener is an object which acts as an asynchronous event handler for messages. The message listener implements the `MessageListener` interface which contains the one method `onMessage()`.

In `onMessage()` method, we define the actions to be performed when the message arrives. By using `setMessageListener()` we define the message listener with a specific `MessageConsumer`.

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

6. Message Components

The JMS Messages are used by the JMS Clients to have communication between systems. The JMS messages have a simple format but are highly flexible, which allows creating messages matching different formats.

The JMS message is divided into three parts:

6.1. 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`

6.2. 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.

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

That's all for the **JMS Introduction Tutorial and it's related terminologies**. In the next set of posts, we will see some examples of JMS.

Happy Learning !!

Was this post helpful?

Let us know if you liked the post. That's the only way we can improve.

Yes

No

Recommended Reading:

1. [JMS Point-To-Point Message Example](#)
2. [JMS Publish/Subscribe Message Example](#)
3. [HornetQ Stand Alone – Basic JMS Messaging Example](#)
4. [How to Install/Uninstall/Execute MySQL as Windows Service](#)
5. [What is AWS \(Amazon Web Service\)](#)
6. [Spring Cloud Service Discovery with Netflix Eureka](#)
7. [Consul Service Registration and Discovery Example](#)
8. [Service Monitoring – Hystrix, Eureka admin and Spring boot admin](#)
9. [Angular Service Example](#)
0. [Retrofit 2 Service Generator](#)

Join 7000+ Awesome Developers


Get the latest updates from industry, awesome resources, blog updates and much more.

Email Address

Subscribe

** We do not spam !!*



career  girls

**WHICH COLLEGE MAJOR
IS RIGHT FOR YOU**

TAKE THE QUIZ

9 thoughts on “JMS Tutorial – Java Message Service Tutorial”

Rafael

July 8, 2020 at 8:28 pm

Very good explained, but if I want to send messages to a remote destination, where it is supposed to configure the connection for sending messages to a remote destination. As you mentioned in the introductory part, the clients may be in different locations but you didn't say anything about how to do it?. I beg you to be clearer, thanks.

[Reply](#)

Lokesh Gupta

July 8, 2020 at 11:18 pm

Look at [Hornetq Spring Integration](#)

[Reply](#)**Sandy**[February 28, 2020 at 7:33 am](#)

Nice blog. Appreciate your effort.

[Reply](#)**mike**[April 2, 2019 at 3:32 am](#)

well explained, very helpful

[Reply](#)**prateek shah**[February 19, 2019 at 11:36 pm](#)

very good

[Reply](#)**Gannoj Sreenivasulu**[December 24, 2019 at 1:22 pm](#)

Very Good Article and also very helpful.

[Reply](#)**Shivkant**[July 29, 2018 at 1:46 am](#)

Thanks a lot for details info on this. Helpful contents.

[Reply](#)

Mnajeet Rana

[July 20, 2018 at 1:49 pm](#)

I am working on integration of a software with other software's where I am supposed to develop a middle layer that will expose the webservices to other software's (To make sure that there is no impact on the performance of the software as every webservice can be called with limit/min therefore not allowing direct access to webservices of software).

So here we trying to achieve is this middle layer holds the load of webservices by other software (queues, batches, database etc.) then this middle layer send the required load to the software.

[Reply](#)

MAULID BILLIE

[February 6, 2017 at 5:49 am](#)

Am a java beginner,I kindly therefore asking how to develop JMS(messaging) and how to create space for mobile number

[Reply](#)

Leave a Comment

Website

☐ Add me to your newsletter and keep me updated whenever you publish new blog posts

Post Comment

Search ...





HowToDoInJava

A blog about Java and related technologies, the best practices, algorithms, and interview questions.

Meta Links

- [About Me](#)
- [Contact Us](#)
- [Privacy policy](#)
- [Advertise](#)
- [Guest Posts](#)

Blogs

[REST API Tutorial](#)

