



What's ^{probably} coming in Java Message Service 2.0

Nigel Deakin

Oracle

JMS 2.0 Specification Lead

DEVOXX™
the java™ community conference





The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.

Agenda

- JSR 343 Update
- What's coming in the JMS 2.0 Early Draft
 - Simplifying the JMS API
 - Improving integration with application servers
 - New API features
- Q&A





- Java Message Service (JMS) specification
 - Part of Java EE but also stands alone
 - Last maintenance release (1.1) was in 2003
- Does not mean JMS is moribund!
 - Multiple active commercial and open source implementations
 - Shows strength of existing spec
- Meanwhile
 - Java EE has moved on since, and now Java EE 7 is planned
 - Time for JMS 2.0

JMS 2.0

- March 2011: JSR 343 launched to develop JMS 2.0
- Target: to be part of Java EE 7 in Q3 2012
- Expert group now in operation and working towards the "early draft" for public review
- Community involvement invited
 - Visit jms-spec.java.net and get involved
 - Join the mailing list
 - Submit suggestions to the issue tracker



JSR 343 Expert Group



- Oracle (lead)
- Caucho
- IBM
- Red Hat
- TIBCO
- Red Hat
- Pramati Technologies
- VMware
- ...and 6 individual members
- FuseSource (soon)

Initial goals of JMS 2.0



- Simpler and easier to use
 - simplify the API
 - make use of CDI (Contexts and Dependency Injection)
 - clarify any ambiguities in the spec
- Support new themes of Java EE 7
 - PaaS
 - Multi-tenancy
- Standardise interface with application servers
- Clarify relationship with other Java EE specs
 - some JMS behaviour defined in other specs
- New messaging features
 - standardize some existing vendor extensions (or will retrospective standardisation be difficult?)

JMS 2.0 Timeline



What's coming in the Early Draft



- Here are some items expected in the JMS 2.0 Early Draft
 - Based on Expert Group members' priorities
 - All items in JIRA at jms-spec.java.net
 - Subject to final approval by the Expert Group**
- Things are still changing
- It's not too late
 - to give us your views on these items
 - to propose additional items for a later draft or revision



Simplifying the JMS API



What's wrong with the JMS API?

Not a lot...

Receiving messages in Java EE

```
@MessageDriven(mappedName = "jms/inboundQueue")
public class MyMDB implements MessageListener {

    public void onMessage(Message message) {
        String payload = (TextMessage) textMessage.getText();
        // do something with payload
    }
}
```

Sending messages in Java EE

```
@Resource(lookup = "jms/connFactory")
ConnectionFactory cf;

@Resource(lookup="jms/inboundQueue")
Destination dest;

public void sendMessage (String payload) throws JMSEException {
    Connection conn = cf.createConnection();
    Session sess =
        conn.createSession(false, Session.AUTO_ACKNOWLEDGE);
    MessageProducer producer = sess.createProducer(dest);
    TextMessage textMessage = sess.createTextMessage(payload);
    producer.send(textMessage);
    connection.close();
}
```

Sending messages in Java EE

```
@Resource(lookup = "jms/connFactory")
ConnectionFactory cf;

@Resource(lookup="jms/inboundQueue")
Destination dest;

public void sendMessage (String payload) throws JMSEException {
    Connection conn = cf.createConnection();
    Session sess =
        conn.createSession(false, Session.AUTO_ACKNOWLEDGE);
    MessageProducer producer = sess.createProducer(dest);
    TextMessage textMessage = sess.createTextMessage(payload);
    messageProducer.send(textMessage);
    connection.close();
}
```

Need to create intermediate objects just to satisfy the API

Sending messages in Java EE

```
@Resource(lookup = "jms/connFactory")
ConnectionFactory cf;

@Resource(lookup="jms/inboundQueue")
Destination dest;

public void sendMessage (String payload) throws JMSException {
    Connection conn = cf.createConnection();
    Session sess =
        conn.createSession(false, Session.AUTO_ACKNOWLEDGE);
    MessageProducer producer = sess.createProducer(dest);
    TextMessage textMessage = sess.createTextMessage(payload);
    producer.send(textMessage);
    connection.close();
}
```



Redundant arguments

Sending messages in Java EE

```
@Resource(lookup = "jms/connFactory")
ConnectionFactory cf;

@Resource(lookup="jms/inboundQueue")
Destination dest;

public void sendMessage (String payload) throws JMSException {
    Connection conn = cf.createConnection();
    Session sess =
        conn.createSession(false, Session.AUTO_ACKNOWLEDGE);
    MessageProducer producer = sess.createProducer(dest);
    TextMessage textMessage = sess.createTextMessage(payload);
    producer.send(textMessage);
    connection.close();
}
```



Boilerplate
code

Sending messages in Java EE

```
public void sendMessage (String payload) throws JMSEException {  
    try {  
        Connection conn = null;  
        con = cf.createConnection();  
        Session sess =  
            conn.createSession(false, Session.AUTO_ACKNOWLEDGE);  
        MessageProducer producer = sess.createProducer(dest);  
        TextMessage textMessage=sess.createTextMessage(payload);  
        messageProducer.send(textMessage);  
    } finally {  
        connection.close();  
    }  
}
```

Need to close
connections
after use

Sending messages in Java EE

```
public void sendMessage (String payload) {
    Connection conn = null;
    try {
        con = cf.createConnection();
        Session sess =
            conn.createSession(false, Session.AUTO_ACKNOWLEDGE);
        MessageProducer producer = sess.createProducer(dest);
        TextMessage textMessage = sess.createTextMessage(payload);
        messageProducer.send(textMessage);
    } catch (JMSEException e1) {
        // do something
    } finally {
        try {
            if (conn != null) connection.close();
        } catch (JMSEException e2) {
            // do something else
        }
    }
}
```

And there's
always
exception
handling to add

Approaches to simplification

- Simplify the existing API
- Define new simplified API
- Use CDI annotations to hide the boilerplate code

Simplify the existing API



- Need to maintain backwards compatibility limits scope for change
 - New methods on `javax.jms.Connection`:
 - Keep existing method
`connection.createSession(transacted, deliveryMode)`
 - New method for Java SE
`connection.createSession(sessionMode)`
 - New method for Java EE
`connection.createSession()`
 - Make `javax.jms.Connection` implement `java.lang.AutoCloseable`

Possible new API

```
@Resource (mappedName="jms/contextFactory")
ContextFactory contextFactory;

@Resource (mappedName="jms/orderQueue")
Queue orderQueue;

public void sendMessage(String payload) {
    try (MessagingContext mCtx = contextFactory.createContext();) {
        TextMessage textMessage = mCtx.createTextMessage(payload);
        mCtx.send(orderQueue, textMessage);
    }
}
```

Annotations for the new API

```
@Resource (mappedName="jms/orderQueue")
Queue orderQueue;

@Inject
@MessagingContext (lookup="jms/contextFactory")
MessagingContext mCtx;

@Inject
TextMessage textMessage;

public void sendMessage (String payload) {
    textMessage.setText (payload);
    mCtx.send (orderQueue, textMessage);
}
```

Annotations for the old API

```
@Inject
@JMSConnection(lookup="jms/connFactory")
@JMSDestination(lookup="jms/inboundQueue")
MessageProducer producer;

@Inject
TextMessage textMessage;

public void sendMessage (String payload) {
    try {
        textMessage.setText(payload);
        producer.send(textMessage);
    } catch {JMSException e}
        // do something
    }
}
```



**Some other
simplifications**

Sending object payloads directly



- No need to create a message
 - `producer.send(String text);`
 - `producer.send(Serializable object);`
- But:
 - wouldn't allow message properties to be set
 - may not be appropriate for `BytesMessage` etc
 - less obvious how to offer this for `receive()` methods
- How useful is this in practice?

Making durable subscriptions easier to use



- Durable subscriptions are identified by {clientId, subscriptionName}
- ClientId will no longer be mandatory when using durable subscriptions
- For a MDB, container will generate default subscription name (EJB 3.2)



New features for PaaS

Annotations to create resources in Java EE

- Currently no standard way for an application to define what JMS resources should be created in the application server and registered in JNDI
- No equivalent to DataSourceDefinition:

```
@DataSourceDefinition(name="java:global/MyApp/MyDataSource",  
    className="com.foobar.MyDataSource",  
    portNumber=6689,  
    serverName="myserver.com",  
    user="lance",  
    password="secret" )
```

Annotations to create resources in Java EE

- JSR 342 (Java EE 7) will define new annotations

```
@JMSConnectionFactoryDefinition(  
    name="java:app/MyJMSFactory",  
    resourceType="javax.jms.QueueConnectionFactory",  
    clientId="foo",  
    resourceAdapter="jmsra",  
    initialPoolSize=5,  
    maxPoolSize=15 )
```

```
@JMSDestinationDefinition(  
    name="java:app/orderQueue",  
    resourceType="javax.jms.Queue",  
    resourceAdapter="jmsra",  
    destinationName="orderQueue")
```

- Possible new SPI to create the physical destinations as well



Improving integration with application servers

JMS 1.1 Chapter 8

JMS Application Server Facilities



API for JTA transactions		API for concurrent processing of messages
XAQueueConnectionFactory	XAConnection	ServerSession
XAQueueConnection	XASession	ServerSessionPool
XAQueueSession	XAConnectionFactory	ConnectionConsumer
XATopicConnectionFactory	XAConnection	Session.setMessageListener
XATopicConnection	XASession	Session.getMessageListener
XATopicSession		Session.run
XAConnectionFactory		

JMS 1.1 Chapter 8

JMS Application Server Facilities



- Interfaces all optional, so not all vendors implement them
- No requirement for application servers to support them
- Some omissions
 - no support for pooled connections
- Meanwhile....

Java EE Connector Architecture (JCA)



- Designed for integrating pooled, transactional resources in an application server
- Designed to support async processing of messages by MDBs
- JCA support already mandatory in Java EE
- Many JMS vendors already provide JCA adapters

Defining the interface between JMS provider and an application server

- **JMS 2.0 will make provision of a JCA adaptor mandatory**
- JMS 1.1 Chapter 8 API remains optional, under review

Improvements to MDBs

- Proposals being sent to JSR 342 (EJB 3.2)
- Fill "gaps" in MDB configuration
- Surprisingly, no standard way to specify
 - JNDI name of queue or topic (using annotation)
 - JNDI name of connection
 - clientID
 - durableSubscriptionName

Defining the destination used by a MDB



- annotation...

```
MessageDriven (messageDestinationLookup="jms/inboundQueue")
public class MyMDB implements MessageListener {
    ...
}
```

- ejb-jar.xml...

```
<ejb-jar>
  <enterprise-beans>
    <message-driven>
      <ejb-name>MessageBean</ejb-name>
      <message-destination-lookup-name>
        jms/inboundQueue
      </message-destination-lookup-name>
    </message-driven>
  </enterprise-beans>
</ejb-jar>
```

- ...all names provisional

Defining the connection factory used by a MDB



- annotation...

```
MessageDriven (connectionFactoryLookup="jms/myCF")
public class MyMDB implements MessageListener {
    ...
}
```

- ejb-jar.xml...

```
<ejb-jar>
  <enterprise-beans>
    <message-driven>
      <ejb-name>MessageBean</ejb-name>
      <connection-factory-lookup-name>
        jms/myCF
      <connection-factory-lookup-name>

```

- ...all names provisional

Defining the clientId and durable subscription name used by a MDB



- Define as standard activation config properties

```
@MessageDriven(activationConfig = {  
    @ActivationConfigProperty(  
  
        propertyName="subscriptionDurability",propertyValue="Durable"),  
    @ActivationConfigProperty(  
        propertyName="clientId",propertyValue="MyMDB"),  
    @ActivationConfigProperty(  
        propertyName="subscriptionName",propertyValue="MySub")  
})
```

- Many app servers support these already



New API Features

New API features



- Delivery delay
- Send a message with async acknowledgement from server
- JMSXDeliveryCount becomes mandatory
- Topic hierarchies
- Multiple consumers on the same topic subscription (both durable and non-durable)
- Batch delivery

Some products implement some of these already

Delivery delay



- Allows a JMS client to schedule the future delivery of a message
- New method on MessageProducer

```
public void setDeliveryDelay(long deliveryDelay)
```

- Sets the minimum length of time in milliseconds from its dispatch time that a produced message should be retained by the messaging system before delivery to a consumer.
- *Why?* If the business requires deferred processing, e.g. end of day

Send a message with async acknowledgement from server



- Send a message and return immediately without blocking until an acknowledgement has been received from the server.
- Instead, when the acknowledgement is received, an asynchronous callback will be invoked

```
producer.send(message, new AcknowledgeListener() {  
    public void onAcknowledge(Message message) {  
        // process ack  
    }  
});
```

- *Why?* Allows thread to do other work whilst waiting for the acknowledgement

Make JMSXDeliveryCount mandatory



- JMS 1.1 defines an optional JMS defined message property JMSXDeliveryCount.
 - When used, this is set by the JMS provider when a message is received, and is set to the number of times this message has been delivered (including the first time). The first time is 1, the second time 2, etc
- JMS 2.0 will make this mandatory
- *Why?* Allows app servers and applications to handle "poisonous" messages better

Topic hierarchies



- Topics can be arranged in a hierarchy
 - STOCK.NASDAQ.TECH.ORCL
 - STOCK.NASDAQ.TECH.GOOG
 - STOCK.NASDAQ.TECH.ADBE
 - STOCK.NYSE.TECH.HPQ
- Consumers can subscribe using wildcards
 - STOCK.*.TECH.*
 - STOCK.NASDAQ.TECH.*
- Most vendors support this already
- Details TBD

Multiple consumers on a topic subscription

- Allows scalable consumption of messages from a topic subscription
 - multiple threads
 - multiple JVMs
- No further change to API for durable subscriptions (clientId not used)
- New API for non-durable subscriptions

```
MessageConsumer messageConsumer=  
    session.createSharedConsumer(topic, sharedSubscriptionName);
```

- *Why?* Scalability

Batch delivery



- Will allow messages to be delivered asynchronously in batches
- New method on MessageConsumer

```
void setBatchMessageListener(  
    BatchMessageListener listener,  
    int batchSize,  
    long batchTimeout)
```

- New listener interface BatchMessageListener

```
void onMessages(Message[] messages)
```

- Acks also sent in a batch
- *Why?* May be more efficient for JMS provider or application

Agenda for the Early Draft

- Those were some items being considered for the JMS 2.0 Early Draft
- **Many items still being discussed by the Expert Group**
- A more detailed description can be found in JIRA at jms-spec.java.net
- It's not too late
 - to give us your views on these items
 - to propose additional items for a later draft (or for JMS 2.1)



Questions?

Get involved!

- Mailing lists, issue tracker and wiki:
 - jms-spec.java.net
- Applications to join the expert group
 - <http://jcp.org/en/jsr/summary?id=343>
- Look out for the Early Draft in Jan 2012
- Contact the spec lead
 - nigel.deakin@oracle.com





DEVOXXTM
the java™ community conference

