





MOVING JAVA FORWARD

ORACLE®

JSR 343: What's Coming in Java Message Service 2.0

Nigel Deakin, Oracle (GlassFish MQ)

Clebert Suconic, Red Hat (HornetQ)

Reza Rahman, Caucho (Resin)

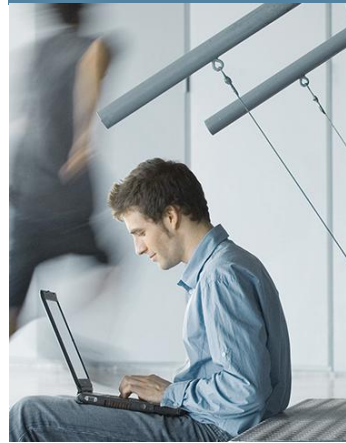
Presenting with



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
- Agenda for the JMS 2.0 Early Draft
 - Tidying up the API
 - Improving integration with application servers
 - Simplifying the JMS API
 - New API features
- Q&A



This session assumes some familiarity with the JMS API

JMS

- 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
- Expert group now in operation
- Community involvement invited
 - Visit jms-spec.java.net and get involved
 - Join the mailing list and submit suggestions to the issue tracker

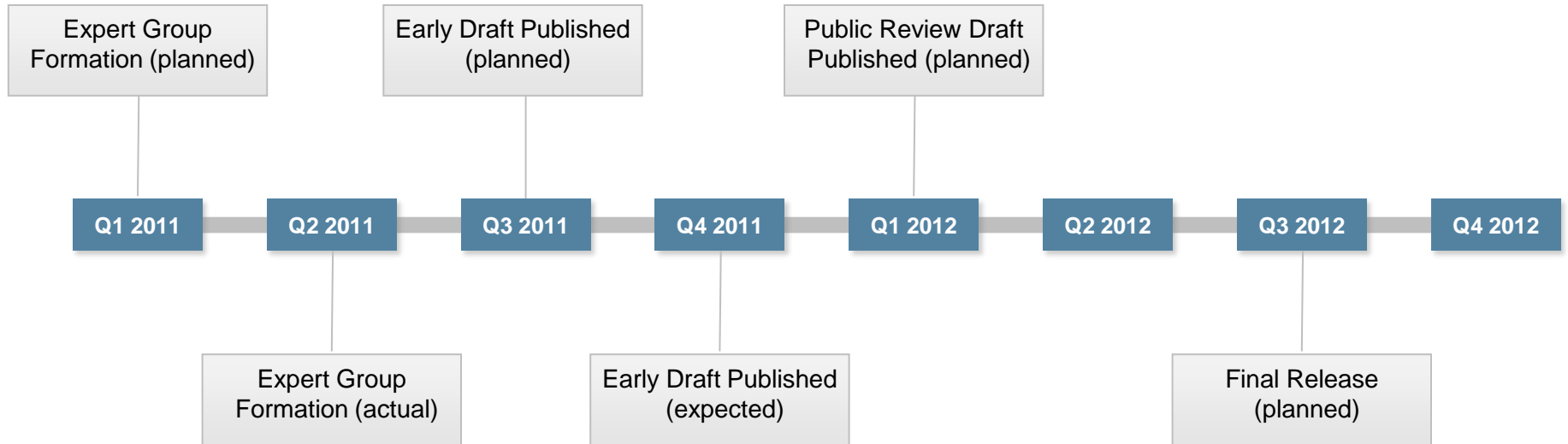
JSR 343 Expert Group

- Oracle
 - Caucho Technology, Inc
 - IBM
 - TIBCO Software Inc
 - Red Hat
 - Pramati Technologies
 - VMWare
 - ... and 6 individual members
- Not too late to join

Initial goals of JMS 2.0

- Changes to ease development
 - catch up with ease of use changes in Java EE 5 and 6
 - improved programming model
 - make use of JSR 299 (Contexts and Dependency Injection)
 - clarify any ambiguities in the spec
- Support new themes of Java EE 7
- Changes to support “the cloud” (e.g. multi-tenancy)
- Mandatory API to integrate with application servers
- Clarify relationship with other Java EE specs
 - some JMS behavior defined in other specs
- Other enhancements
 - standardize some existing vendor extensions (or will retrospective standardisation be difficult?)

JMS 2.0 Timeline



Agenda for the Early Draft

- Here are some items being considered for the JMS 2.0 Early Draft
 - Based on Expert Group members' priorities
 - All items in JIRA at jms-spec.java.net
 - **Many items not yet formally approved by the Expert Group**
- It's not too late
 - to give us your views on these items
 - to propose additional items for a later draft

TIDYING UP THE JMS API

Deprecate domain-specific interfaces

- JMS 1.0 had separate interfaces for queues and topics
 - QueueConnectionFactory, QueueConnection, etc
 - TopicConnectionFactory, TopicConnection etc
- JMS 1.1 introduced unified interfaces
 - ConnectionFactory, Connection etc
- Time to deprecate the domain-specific interfaces and propose for removal from a future version
- Would reduce interfaces from 44 to 28

Deprecate domain-specific interfaces

Domain-specific interface to be deprecated	Superseded by existing unified interface
QueueConnectionFactory	ConnectionFactory
QueueConnection	Connection
QueueSession	Session
QueueReceiver	MessageConsumer
QueueSender	MessageProducer
TopicConnectionFactory	ConnectionFactory
TopicConnection	Connection
TopicSession	Session
TopicSubscriber	MessageConsumer
TopicPublisher	MessageProducer

Deprecate domain-specific interfaces

Domain-specific XA interface to be deprecated (already optional)	Superseded by existing unified XA interface (remain optional)
XAQueueConnectionFactory	XAConnectionFactory
XAQueueConnection	XAConnection
XAQueueSession	XASession
XATopicConnectionFactory	XAConnectionFactory
XATopicConnection	XAConnection
XATopicSession	XASession

Deprecate domain-specific interfaces

Problem

```
session.createDurableSubscriber(topic, name)
```

```
session.createDurableSubscriber(topic, name, selector, noLocal)
```

return a TopicSession, which is being deprecated

Solution

```
session.createDurableConsumer(topic, name)
```

```
session.createDurableConsumer(topic, name, selector, noLocal)
```

Which return a MessageConsumer

IMPROVING INTEGRATION WITH APPLICATION SERVERS

Defining the interface between JMS provider and an application server

- Requirement: allowing any JMS provider to work in any Java EE application server
- Current solution: JMS 1.1 Chapter 8 *JMS Application Server Facilities*

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

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

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

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

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

All names
provisional

Defining the clientId and durable subscription name used by a MDB

```
<ejb-jar>
<enterprise-beans>
<message-driven>
<ejb-name>test.MyMDB</ejb-name>
<activation-config>
  <activation-config-property>
    <activation-config-property-name>subscriptionDurability</activation-config-property-name>
    <activation-config-property-value>Durable</activation-config-property-value>
  </activation-config-property>
  <activation-config-property>
    <activation-config-property-name>clientId</activation-config-property-name>
    <activation-config-property-value>MyMDB</activation-config-property-value>
  </activation-config-property>
  <activation-config-property>
    <activation-config-property-name>subscriptionName</activation-config-property-name>
    <activation-config-property-value>MySub</activation-config-property-value>
  </activation-config-property>
  . . .
```

SIMPLIFYING THE JMS API

Issues with the JMS API

```
@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);
    messageProducer.send(textMessage);
    connection.close();
}
```

Issues with the JMS API

Need to create intermediate objects just to satisfy the API

```
@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);
    messageProducer.send(textMessage);
    connection.close();
}
```

Issues with the JMS API

```
@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);
    messageProducer.send(textMessage);
    connection.close();
}
```

Redundant
arguments

Issues with the JMS API

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

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

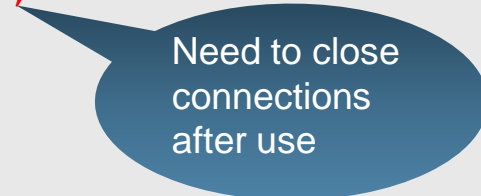
public void sendMessage (String payload) throws JMSException {
    Connection conn = cf.createConnection();
    Session sess =
        conn.createSession(false, Session.AUTO_ACKNOWLEDGE);
    MessageProducer producer = sess.createProducer(dest);
    TextMessage textMessage = sess.createTextMessage(payload);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

Boilerplate
code



Issues with the JMS API

```
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

Issues with the JMS API

```
public void sendMessage (String payload) {  
    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);  
    } catch (JMSEException e) {  
        // do something  
    } finally {  
        connection.close();  
    }  
}
```

And there's
always exception
handling to add

Arguments to createSession()

- Existing method has two args when it should have one:

```
connection.createSession(transacted, deliveryMode)
```

- New method for Java SE and Java EE

```
connection.createSession(sessionMode)
```

- New method for Java EE

```
connection.createSession()
```

JMSException

- Almost all JMS method signatures declare JMSException
 - even methods like `ObjectMessage.getObject()`,
`Queue.getQueueName()`
- Calling method must catch or declare the exception
- Calling method doesn't know what to do
 - javadocs say "some internal error" – how do we handle that?
 - best policy often to declare or rethrow the exception so it can be handled at a higher level in the stack

JMSException

- JMSException becomes an unchecked exception

If a client can reasonably be expected to recover from an exception, make it a checked exception. If a client cannot do anything to recover from the exception, make it an unchecked exception. *JMS Tutorial*

- Existing applications still work
- Makes it easier to define frameworks or annotation

Annotations

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

@Inject
TextMessage textMessage;

public void sendMessageNew(String payload) {
    textMessage.setText(payload);
    producer.send(textMessage);
}
```

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
- Still unresolved

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 Platform) will define new annotations

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

- XML equivalent as well
- Exact details still to be determined

Define standard connection factory properties

Property Name	Type	Description
description	String	description
user	String	user name
password	String	password
networkProtocol	String	network protocol
serverName	String	server name of the JMS provider
portNumber	String	port number of the JMS provider
networkProtocols	String[]	array of network protocols
serverNames	String[]	array of server names
portNumbers	String[]	array of port numbers
url	String	Opaque string which defines how to connect to the JMS provider
clientId	String	JMS client identifier

Making shared subscriptions easier to use

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

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

Delivery delay

- Allows a JMS client to specify a delivery delay when sending 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.

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  
    }  
});
```

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
- Allows app servers and applications to handle "poisonous" messages better

Topic hierarchies

- JMS 1.1 specification states that a topic is a "well-known node in a content-based hierarchy"
 - but it does not specify how this hierarchy is used, how topics are named, or define any special functionality to operate on more than one node of the hierarchy at the same time.
- It is proposed that new features are added to JMS to explicitly support the use of topic hierarchies
 - No detail yet
- Already supported by many JMS vendors using non-standard extensions

Multiple consumers on the same topic subscription

- Allows scalable consumption of messages from a topic subscription
- No further change to API for durable subscriptions
- New API for non-durable subscriptions

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

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)
```

Agenda for the Early Draft

- Those were some items being considered for the JMS 2.0 Early Draft
- **Many items not yet formally approved 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

Q&A

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 Oct/Nov 2011
- Contact the spec lead
 - nigel.deakin@oracle.com

