
JavaOne™

JavaOne™

What's new in Java Message Service 2.0?

Nigel Deakin
JSR 343 Specification Lead
Oracle

MAKE THE FUTURE JAVA

ORACLE

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.

BOF

Meet the experts:
JMS 2.0 expert group

Tuesday 2 Oct 2012
1630 - 1715
Parc 55: Cyril Magnin II/III

JMS

- A Java API for sending and receiving messages
- Many competing implementations
- Two distinct API variants
  - Java SE applications
  - Java EE applications (web, EJB, application client)
    - adds support for JTA transactions, MDBs
    - removes features considered inappropriate in a managed application server environment

What JMS is and isn't

- A standard API
  - Not a messaging system in itself
  - Not a wire protocol
- Defines Java API only
  - Doesn't define API for non-Java clients (e.g. C++, HTTP) - but many implementations do)
- An application API
  - Not (currently) an admin, management or monitoring API

JMS 2.0

- JMS 1.1 last updated in 2002
- JMS 2.0 launched in 2011 as JSR 343
  - Expert group contains 20 members,
    including FuseSource, IBM, Oracle, Pramati, Red Hat, TIBCO
  - Early Draft published Feb 2012
  - NOT FINISHED!
  - Final release planned Q2 2013 (aligned with Java EE 7)
  - More information at http://jms-spec.java.net
  - Join the user email alias and get involved

Goals of JMS 2.0

- Simplicity and ease of use
- New messaging features
- Better Java EE integration
  - define the slightly different JMS API more clearly
  - simpler resource configuration
  - standardized configuration of JMS MDBs
  - better application server pluggability
- Minor corrections and clarifications
- Cloud / PaaS features deferred to Java EE 8

JMS 2.0:
Simplifying the JMS API

What's wrong with the JMS 1.1 API?

JMS 1.1: Sending a message

```
@Resource(lookup = "java:global/jms/demoConnectionFactory")
ConnectionFactory connectionFactory;

@Resource(lookup = "java:global/jms/demoQueue")
Queue demoQueue;

public void sendMessage(String payload) {
   try {
      Connection connection = connectionFactory.createConnection();
      try {
         Session session = connection.createSession(false,Session.AUTO_ACKNOWLEDGE);
         MessageProducer messageProducer = session.createProducer(demoQueue);
         TextMessage textMessage = session.createTextMessage(payload);
         messageProducer.send(textMessage);
      } finally {
         connection.close();
      }
   } catch (JMSException ex) {
      Logger.getLogger(getClass().getName()).log(Level.SEVERE, null, ex);
   }
}
```

13 lines of code just to send a message

JMS 1.1: Sending a message

```
@Resource(lookup = "java:global/jms/demoConnectionFactory")
ConnectionFactory connectionFactory;

@Resource(lookup = "java:global/jms/demoQueue")
Queue demoQueue;

public void sendMessage(String payload) {
   try {
      Connection connection = connectionFactory.createConnection();
      try {
         Session session = connection.createSession(false,Session.AUTO_ACKNOWLEDGE);
         MessageProducer messageProducer = session.createProducer(demoQueue);
         TextMessage textMessage = session.createTextMessage(payload);
         messageProducer.send(textMessage);
      } finally {
         connection.close();
      }
   } catch (JMSException ex) {
      Logger.getLogger(getClass().getName()).log(Level.SEVERE, null, ex);
   }
}
```

must create several intermediate objects

JMS 1.1: Sending a message

```
@Resource(lookup = "java:global/jms/demoConnectionFactory")
ConnectionFactory connectionFactory;

@Resource(lookup = "java:global/jms/demoQueue")
Queue demoQueue;

public void sendMessage(String payload) {
   try {
      Connection connection = connectionFactory.createConnection();
      try {
         Session session = connection.createSession(false,Session.AUTO_ACKNOWLEDGE);
         MessageProducer messageProducer = session.createProducer(demoQueue);
         TextMessage textMessage = session.createTextMessage(payload);
         messageProducer.send(textMessage);
      } finally {
         connection.close();
      }
   } catch (JMSException ex) {
      Logger.getLogger(getClass().getName()).log(Level.SEVERE, null, ex);
   }
}
```

redundant and misleading arguments

JMS 1.1: Sending a message

```
@Resource(lookup = "java:global/jms/demoConnectionFactory")
ConnectionFactory connectionFactory;

@Resource(lookup = "java:global/jms/demoQueue")
Queue demoQueue;

public void sendMessage(String payload) {
   try {
      Connection connection = connectionFactory.createConnection();
      try {
         Session session = connection.createSession(false,Session.AUTO_ACKNOWLEDGE);
         MessageProducer messageProducer = session.createProducer(demoQueue);
         TextMessage textMessage = session.createTextMessage(payload);
         messageProducer.send(textMessage);
      } finally {
         connection.close();
      }
   } catch (JMSException ex) {
      Logger.getLogger(getClass().getName()).log(Level.SEVERE, null, ex);
   }
}
```

boilerplate code

JMS 1.1: Sending a message

```
@Resource(lookup = "java:global/jms/demoConnectionFactory")
ConnectionFactory connectionFactory;

@Resource(lookup = "java:global/jms/demoQueue")
Queue demoQueue;

public void sendMessage(String payload) {
   try {
      Connection connection = connectionFactory.createConnection();
      try {
         Session session = connection.createSession(false,Session.AUTO_ACKNOWLEDGE);
         MessageProducer messageProducer = session.createProducer(demoQueue);
         TextMessage textMessage = session.createTextMessage(payload);
         messageProducer.send(textMessage);
      } finally {
         connection.close();
      }
   } catch (JMSException ex) {
      Logger.getLogger(getClass().getName()).log(Level.SEVERE, null, ex);
   }
}
```

must close resources after use!

JMS 1.1: Sending a message

```
@Resource(lookup = "java:global/jms/demoConnectionFactory")
ConnectionFactory connectionFactory;

@Resource(lookup = "java:global/jms/demoQueue")
Queue demoQueue;

public void sendMessage(String payload) {
   try {
      Connection connection = connectionFactory.createConnection();
      try {
         Session session = connection.createSession(false,Session.AUTO_ACKNOWLEDGE);
         MessageProducer messageProducer = session.createProducer(demoQueue);
         TextMessage textMessage = session.createTextMessage(payload);
         messageProducer.send(textMessage);
      } finally {
         connection.close();
      }
   } catch (JMSException ex) {
      Logger.getLogger(getClass().getName()).log(Level.SEVERE, null, ex);
   }
}
```

all methods throw checked exceptions

Simplifying the JMS API

Strategy

- Simplify existing JMS 1.1 API where it won't break compatibility
- Define new simplified API requiring fewer objects
  - JMSContext, JMSProducer, JMSConsumer
- In Java EE, allow JMSContext to be injected and managed by the container

Simplifying the existing JMS 1.1 API

Simpler API to create a Session

- Need to maintain backwards compatibility limits scope for change
- New methods on **`javax.jms.Connection`** to create a Session:
  - Existing method (will remain)

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

  - New method mainly for Java SE

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

  - New method mainly for Java EE

```
connection.createSession()
```

Simplifying the existing JMS 1.1 API

Simpler API to close JMS objects

- Make JMS objects implement **`java.jang.AutoCloseable`**
  - **`Connection`**
  - **`Session`**
  - **`MessageProducer`**
  - **`MessageConsumer`**
  - **`QueueBrowser`**
- Requires Java SE 7

Simplifying the existing JMS 1.1 API

Simpler API to close JMS objects

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

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

public void sendMessage (String payload) throws JMSException {
   try ( Connection conn = connectionFactory.createConnection();
         Session session = conn.createSession();
         MessageProducer producer = session.createProducer(dest);
   ){
      Message mess = sess.createTextMessage(payload);
      producer.send(mess);
   } catch(JMSException e){
      // exception handling
   }
}
```

Create closeable resources in a try-with-resources block

close() is called automatically at end of block

New simplified API for JMS 2.0

Introducing JMSContext and JMSProducer

```
@Resource(lookup = "java:global/jms/demoConnectionFactory")
ConnectionFactory connectionFactory;

@Resource(lookup = "java:global/jms/demoQueue")
Queue demoQueue;

public void sendMessageNew(String payload) {
   try (JMSContext context = connectionFactory.createContext();){
      context.createProducer().send(demoQueue, payload);
   } catch (JMSRuntimeException ex) {
      Logger.getLogger(getClass().getName()).log(Level.SEVERE, null, ex);
   }
}
```

13 lines reduced to 5

New simplified API for JMS 2.0

Introducing JMSContext and JMSProducer

```
@Resource(lookup = "java:global/jms/demoConnectionFactory")
ConnectionFactory connectionFactory;

@Resource(lookup = "java:global/jms/demoQueue")
Queue demoQueue;

public void sendMessageNew(String payload) {
    try (JMSContext context = connectionFactory.createContext();){
        context.createProducer().send(demoQueue, payload);
    } catch (JMSRuntimeException ex) {
        Logger.getLogger(getClass().getName()).log(Level.SEVERE, null, ex);
    }
}
```

JMSContext combines Connection and Session

Payload can be sent directly

close() is called automatically at end of block

No checked exceptions thrown

JMSContext (1/2)

- A new object which encapsulates a Connection, a Session and an anonymous MessageProducer
- Created from a ConnectionFactory

```
JMSContext context = connectionFactory.createContext(sessionMode);
```

- Call close() after use, or create in a try-with-resources block
- Can also be injected (into a Java EE web or EJB application)

JMSContext (2/2)

- Can also create from an existing JMSContext (to reuse its connection – Java SE only)

```
JMSContext context2 = context1.createContext(sessionMode);
```

- Used to create JMSProducer objects for sending messages
- Used to create JMSConsumer objects for receiving messages
- Methods on JMSContext, JMSProducer and JMSConsumer throw only unchecked exceptions

JMSProducer

- Messages are sent by creating a JMSProducer object
  - does *not* encapsulate a MessageProducer so is lightweight
  - supports method chaining for a fluid style
- JMS 1.1

```
MessageProducer producer = session.createProducer();
producer.send(destination,message);
```

- JMS 2.0

```
JMSProducer producer = context.createProducer();
producer.send(destination,message);
```

JMSProducer

Setting message delivery options using method chaining

- JMS 1.1

```
MessageProducer producer = session.createProducer();
producer.setDeliveryMode(DeliveryMode.NON_PERSISTENT);
producer.setPriority(1);
producer.setTimeToLive(1000);
producer.send(destination,message);
```

- JMS 2.0

```
context.createProducer().setDeliveryMode(DeliveryMode.NON_PERSISTENT).
   setPriority(1).setTimeToLive(1000).send(destination,message);
```

JMSProducer

Setting message properties and headers

- JMS 1.1 (need to set on the message)

```
MessageProducer producer = session.createProducer();
TextMessage textMessage = session.createTextMessage("Hello);
textMessage.setStringProperty("foo","bar");
producer.send(destination,message);
```

- JMS 2.0 (can also set on the JMSProducer)

```
context.createProducer().setProperty("foo","bar").send(destination,"Hello");
```

JMSProducer

Sending message payloads directly

- Methods on JMSProducer to send a Message
  - `send(Destination dest, Message message)`
- No need to create a Message
  - `send(Destination dest, Map<String,Object> payload)`
  - `send(Destination dest, Serializable payload)`
  - `send(Destination dest, String payload)`
  - `send(Destination dest, byte[] payload)`
- Use methods on JMSProducer to set delivery options, message headers and message properties

JMSConsumer

- Messages are consumed by creating a JMSConsumer object
  - encapsulates a MessageConsumer
  - similar functionality and API to MessageConsumer
- Synchronous

```
JMSConsumer consumer = context.createConsumer(destination);
Message message = consumer.receive(1000);
```

- Asynchronous

```
JMSConsumer consumer = context.createConsumer(destination);
consumer.setMessageListener(messageListener);
```

- Connection is automatically started (configurable)

JMSConsumer
Receiving message payloads directly

- Methods on JMSConsumer that return a Message
  - `Message receive();`
  - `Message receive(long timeout);`
  - `Message receiveNoWait();`
- Methods on JMSConsumer that return message payload directly
  - `<T> T receivePayload(Class<T>  c);`
  - `<T> T receivePayload(Class<T> c, long timeout);`
  - `<T> T receivePayloadNoWait(Class<T> c);`

JMSConsumer
Receiving message payloads directly

```
public String receiveMessage() throws NamingException {
   InitialContext initialContext = getInitialContext();
   ConnectionFactory connectionFactory =
      (ConnectionFactory) initialContext.lookup("jms/connectionFactory");
   Queue inboundQueue = (Queue)initialContext.lookup("jms/inboundQueue");

   try (JMSContext context = connectionFactory.createContext();) {
      JMSConsumer consumer = context.createConsumer(inboundQueue);
      return consumer.receivePayload(String.class);
   }
}
```

Injection of JMSContext objects

into a Java EE web or EJB container

```
@Inject
@JMSConnectionFactory("jms/connectionFactory")
private JMSContext context;

@Resource(mappedName = "jms/inboundQueue")
private Queue inboundQueue;

public void sendMessage (String payload) {
   context.createProducer().send(inboundQueue, payload);
}
```

Injection of JMSContext objects

into a Java EE web or EJB container

```
@Inject
@JMSConnectionFactory("jms/connectionFactory")
private JMSContext context;

@Resource(mappedName = "jms/inboundQueue")
private Queue inboundQueue;

public void sendMessage (String payload) {
   context.createProducer().send(inboundQueue, payload);
}
```

Use @Inject to inject the JMSContext, specifying connection factory to use

Container will close JMSContext automatically at end of transaction

Injection of JMSContext objects

into a Java EE web or EJB container

- Connection factory will default to platform default JMS

```
@Inject private JMSContext context;
```

- Specifying session mode

```
@Inject
@JMSConnectionFactory("jms/connectionFactory")
@JMSSessionMode(JMSContext.AUTO_ACKNOWLEDGE)
private JMSContext context;
```

- Specifying user and password (may be aliased)

```
@Inject
@JMSConnectionFactory("jms/connectionFactory")
@JMSPasswordCredential(userName="admin",password="mypassword")
private JMSContext context;
```

Injection of JMSContext objects

into a Java EE web or EJB container

- Injected JMSContext objects have a scope
  - In a JTA transaction, scope is the transaction
  - If no JTA transaction, scope is the request
- JMSContext is automatically closed when scope ends
- Inject two JMSContext objects within the same scope and you get the same object
  - if @JMSConnectionFactory, @JMSPasswordCredential and @JMSSessionMode annotations match
  - Makes it easier to use same session within a transaction

JMS 2.0:
New API features

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)

Delivery delay

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

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

- New method on JMSProducer

```
public JMSProducer setDeliveryDelay(long deliveryDelay)
```

- Sets minimum time in ms from that a 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

Async send

- 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
- New methods on MessageProducer

```
messageProducer.send(message,completionListener)
```

- Feature also available on JMSProducer
- *Why?* Allows thread to do other work whilst waiting for the acknowledgement

Async send

- Application specifies a CompletionListener instance

```
public interface CompletionListener {
   void onCompletion(Message message);
   void onException(Message message, Exception exception);
}
```

Better handling of "poison" messages

Make JMSMXDeliveryCount 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

Multiple consumers on a topic subscription

- Allows scalable consumption of messages from a topic subscription
  - multiple threads, multiple JVMs
- New methods needed for non-durable subscriptions

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

- Existing methods used for durable subscriptions

```
MessageConsumer messageConsumer=
    session.createDurableConsumer(topic,durableSubscriptionName);
```

- Also available on JMSContext

Easier definition of JMS resources in Java EE

Joint effort with
JSR 342 (Java EE 7 platform)

Easier definition of JMS resources in Java EE

The problem

- Java EE and JMS recommend applications should obtain JMS ConnectionFactory and Destination resources by lookup from JNDI

```
@Resource(lookupName = "jms/inboundQueue")
private Queue inboundQueue;
```

- Keeps application code portable
- Creating these resources is a burden on the deployer, and is non-standard

Platform default connection factory

Making the simple case simple

- if you simply want to use the application server's built in JMS

```
@Resource(lookup="java:comp/defaultJMSConnectionFactory")
ConnectionFactory myJMScf;
```

Easier definition of JMS resources in Java EE

New optional feature in Java EE 7

- Application may specify the JMS connection factories and JMS destinations that it needs using annotations
- Deployer can further define requirements using deployment descriptor elements
- Application server can use this information to create resources automatically when application is deployed
- The JMS equivalent to @DataSourceDefinition annotations
- Supporting these automatically is *optional*

Easier definition of JMS resources in Java EE

Application defines required resources using annotations

```
@JMSConnectionFactoryDefinition(
    name="java:global/jms/demoConnectionFactory",
    className= "javax.jms.ConnectionFactory",
    description="ConnectionFactory to use in demonstration")
```

```
@JMSDestinationDefinition(
    name = "java:global/jms/demoQueue",
    description = "Queue to use in demonstration",
    className = "javax.jms.Queue",
    destinationName="demoQueue")
```

- Can also specify deployment-specific properties via annotations, but these are best added at deployment time

Easier definition of JMS resources in Java EE

Deployer adds further requirements using deployment descriptor

```
<jms-destination>
   <name>"java:global/jms/demoQueue</name>
   <class-name>javax.jms.Queue</class-name>
   <resource-adapter-name>jmsra</resource-adapter-name>
   <destination-name>demoQueue</destination-name>
</jms-destination>
```

```
 <jms-connection-factory>
   <name>java:global/jms/demoConnectionFactory</name>
   <property>
      <name>addressList</name>
      <value>mq://localhost:7676</value>
   </property>
   <max-pool-size>30</max-pool-size>
   <min-pool-size>20</min-pool-size>
   <max-idle-time>5</max-idle-time>
</jms-connection-factory>
```

non-standard properties may be specified if needed

More standardized configuration of JMS MDBs

Joint effort with JSR 345 (EJB 3.2)

More standardized configuration of JMS MDBs

- Configuration of JMS MDBs is surprisingly non-standard
- EJB 3.1 does not define how to specify
  - JNDI name of queue or topic (using annotation)
  - JNDI name of connection factory
  - clientID
  - durableSubscriptionName
- EJB 3.1 does not define how topic messages delivered to clustered MDBs

More standardized configuration of JMS MDBs

New activation property to specify the queue or topic

```
@MessageDriven(activationConfig =  {
   @ActivationConfigProperty(
      propertyName = "destinationLookup",
      propertyValue = "jms/myTopic"),
   . . .
})
```

- Can also be configured in **`ejb-jar.xml`**

More standardized configuration of JMS MDBs

New activation property to specify the connection factory

```
@MessageDriven(activationConfig =  {
   @ActivationConfigProperty(
      propertyName = "connectionFactoryLookup",
      propertyValue = "jms/myCF"),
   . . .
})
```

- Can also be configured in **`ejb-jar.xml`**

More standardized configuration of JMS MDBs

New activation properties to specify durable subscriptions

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

- Surprisingly, these have never been standardized before

Easier configuration of durable subscriptions

No need to specify clientId and subscription name

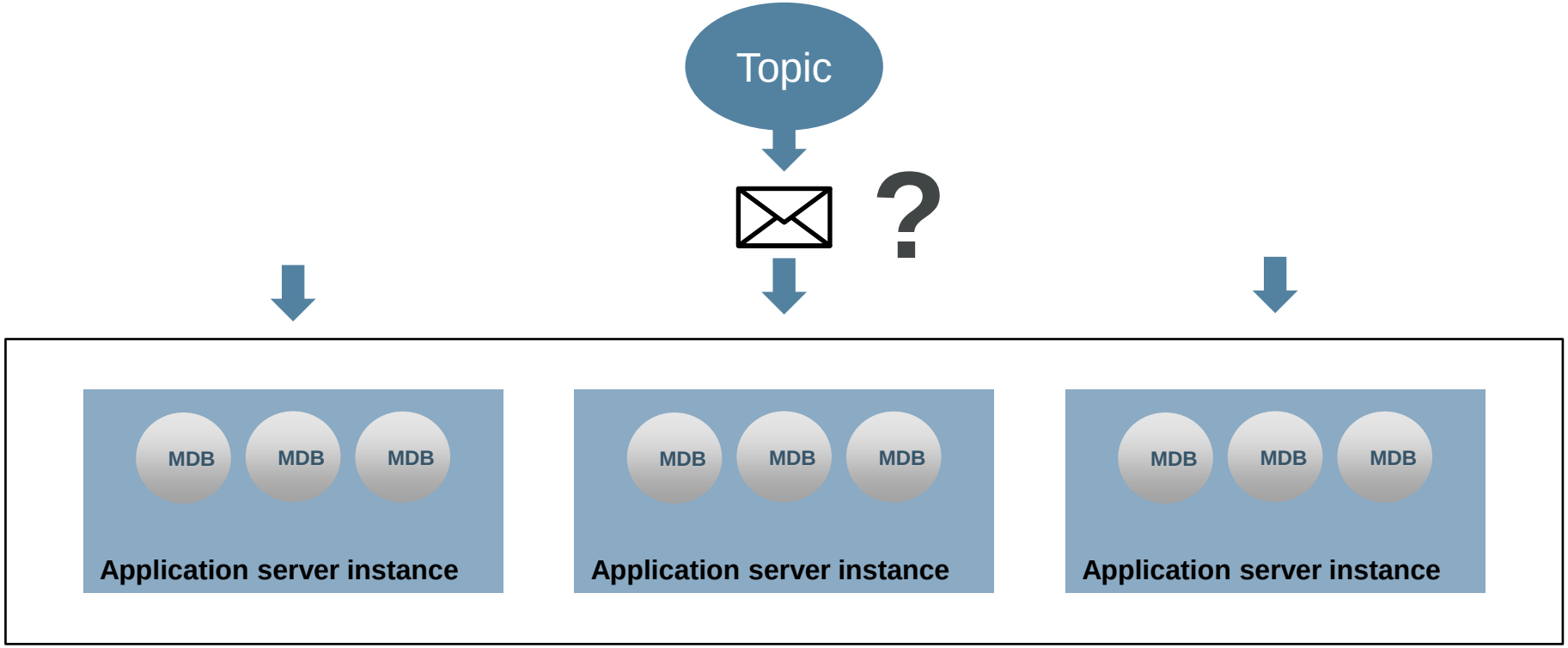
```
@MessageDriven(activationConfig =  {
   @ActivationConfigProperty(
      propertyName = "subscriptionDurability",
      propertyValue = "Durable"),
   @ActivationConfigProperty(
      propertyName = "clientId",
      propertyValue = "myClientID"),
   @ActivationConfigProperty(
      propertyName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

clientId no longer required for durable subscriptions

if subscription name is omitted, the container generates a suitable name

Topic delivery to clustered application servers

One message per instance or one message per cluster?



Application server cluster

Topic delivery to clustered application servers

- Defined behavior if **subscriptionName** and **clientId** not set
  - each message will be delivered once per cluster
  - clustered app server instances will share the same subscription
- To disable, set **sharedSubscriptions** activation property to **false**
  - each message will be delivered once per instance
  - each app server instance will have a separate subscription
- Applies to both durable and non-durable subscriptions

Improved Java EE pluggability

Improved Java EE pluggability

- The goal
  - To make it easier to use a particular JMS provider in different Java EE application servers
  - e.g. GlassFish application sending messages to WebLogic JMS
- The solution
  - To require JMS providers to supply a JCA resource adapter

Improved Java EE pluggability

using the Java Connector Architecture (JCA)

- Java Connector Architecture is designed for this:
  - for integrating pooled, transactional resources in an application server
  - for async processing of messages by MDBs
- JCA support already mandatory in Java EE
- Many JMS vendors already provide JCA adapters
- **JMS 2.0 will make provision of a JCA adaptor mandatory**
- Should be invisible to applications!

What's new in JMS 2.0

- Simplicity and ease of use
- New messaging features
  - multi-threaded topic subscribers
  - delivery delay
  - async send
- Better Java EE integration
  - simpler resource configuration
  - standardized configuration of JMS MDBs
  - better application server pluggability
- Minor corrections and clarifications

Where to find out more

- See the demo at the Java EE booth in the DemoGrounds
- Come to the BOF tomorrow
- Read the draft spec and API docs at jms-spec.java.net
- Join users@jms-spec.java.net and send questions and comments
- Try the latest GlassFish and Open Message Queue builds

Try the new features as they are added

JMS 2.0, EJB 3.2 and Java EE

- GlassFish 4.0
  - http://glassfish.java.net/
  - http://dlc.sun.com.edgesuite.net/glassfish/4.0/promoted/
- Open Message Queue 5.0
  - http://mq.java.net/
  - http://mq.java.net/5.0.html
- These are not finished!

Any questions?

More time for questions and comments at our BOF tomorrow

Meet the JMS 2.0 expert group
Tuesday 2 Oct 2012
1630 - 1715
Parc 55: Cyril Magnin II/III

MAKE THE FUTURE JAVA

ORACLE


JavaOne™