ORACLE

JMS.Next(): JMS 2.0 and Beyond

Reza Rahman
Java EE/GlassFish Evangelist
Reza.Rahman@Oracle.com
@reza_rahman

JMS

- Small, successful Java API for Message Oriented Middleware (MOM)
- JMS 1.1 – 2002
  - J2EE era!
- JMS 2 – 2013
  - Included in Java EE 7
- Overdue and well received
- JMS 2.1 already started
  - Included in Java EE 8
  - Projected delivery 2016
  - Time to get involved is now!

JMS 2 Goals

- API Modernization
- Some New Features
- Java EE Alignment
- EJB3/MDB Alignment
- Minor Corrections and Clarifications

Modernizing the API

JMS 1.1

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

@Resource(lookup = "java:global/jms/demoQueue")
private 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);
    }
}
```

Simplifying the API

Strategy

- Maintain backwards compatibility!
- Simplify existing JMS API where possible
- Define new abstractions where needed
  - JMSContext, JMSProducer, JMSConsumer
- Take advantage of modern Java EE paradigms
  - Annotations
  - Convention-over-configuration
  - Injection/scoping

Streamlining Session Creation

- Methods in 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()`

Auto Closing JMS Objects

- Make JMS objects implement java.jang.AutoCloseable
  - Connection
  - Session
  - MessageProducer
  - MessageConsumer
  - QueueBrowser

JMS Auto Closable Example

```
@Resource(lookup = "jms/connFactory")
private ConnectionFactory cf;
@Resource(lookup="jms/inboundQueue")
Destination dest;

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

A First Look at the New API

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

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

- An abstraction that encapsulates Connection, Session, Producer and Consumer
  - Created from ConnectionFactory
    - `JMSContext context = connectionFactory.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
- Geared towards common use cases while retaining flexibility
- Fluent API

JMSProducer Example

Setting message delivery options

- JMS 1.1

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

- JMS 2

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

JMSProducer Example

Setting message properties and headers

- JMS 1.1

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

- JMS 2

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

Sending Message Payloads Directly

- Sending just the payload
  - `send(Destination dest, String payload)`
  - `send(Destination dest, Serializable payload)`
  - `send(Destination dest, byte[] payload)`
  - `send(Destination dest, Map<String, Object> payload)`
  - Use methods on JMSProducer to set delivery options, message headers and message properties
- Sending JMS Message Object
  - `send(Destination dest, Message message)`

JMSConsumer

- Messages are consumed by creating a JMSConsumer object
- Geared towards common use cases while retaining flexibility
  - Underlying connection is automatically started (configurable)

Receiving Message Payloads Directly

- 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);`
- You can still return a JMS Message Object:
  - `Message receive();`
  - `Message receive(long timeout);`
  - `Message receiveNoWait();`

JMSConsumer Example

```
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);
    }
}
```

Enter the Injection Dragon

```
@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

- 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 username and password (may be aliased)

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

Injection Under the Hood

- 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

The New Features

Delivery Delay

- Allows future timed delivery of a message
  - Deferred processing, e.g. end of day
- Method on JMSProducer

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

- Sets minimum time in ms from that a message should be retained by the messaging system before delivery to a consumer
- Also available in the old API

Asynchronous Send

- Send a message and return immediately without blocking until an acknowledgement has been received from the JMS server
  - Allows thread to do other work whilst waiting for acknowledgement
- When the acknowledgement is received an asynchronous callback will be invoked
- Method on JMSProducer:

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

- Also available in the old API

Asynchronous Send Listener

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

Better Handling of Poison Messages

JMSXDeliveryCount mandatory

- JMS 1.1 defines optional JMS message property JMSXDeliveryCount
  - Number of times message has been delivered
- JMS 2 makes this mandatory
- Allows applications (and the JMS provider) to handle poison messages better
  - Not a replacement to dead letter queues, etc

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:

```
JMSConsumer consumer =
   context.createSharedConsumer(topic, sharedSubscriptionName);
```

- Existing methods used for durable subscriptions:

```
JMSConsumer consumer =
   context.createDurableConsumer(topic, durableSubscriptionName);
```

- Also available in old API

Java EE Alignment

Platform Default Connection Factory

- Ships preconfigured with the runtime
- Useful for majority of cases

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

Defining JMS Resources

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

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

Supplanted/Overridden in XML…

```
<jms-destination>
   <name>java:global/jms/demoQueue</name>
   <interface-name>javax.jms.Queue</interface-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>
```

EJB3/MDB Alignment

Better Standard Configuration for MDBs

- Configuration of MDBs surprisingly non-standard
  - JNDI name of queue or topic
  - JNDI name of connection factory
  - Client ID
  - Durable subscription name

Specifying Queue or Topic

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

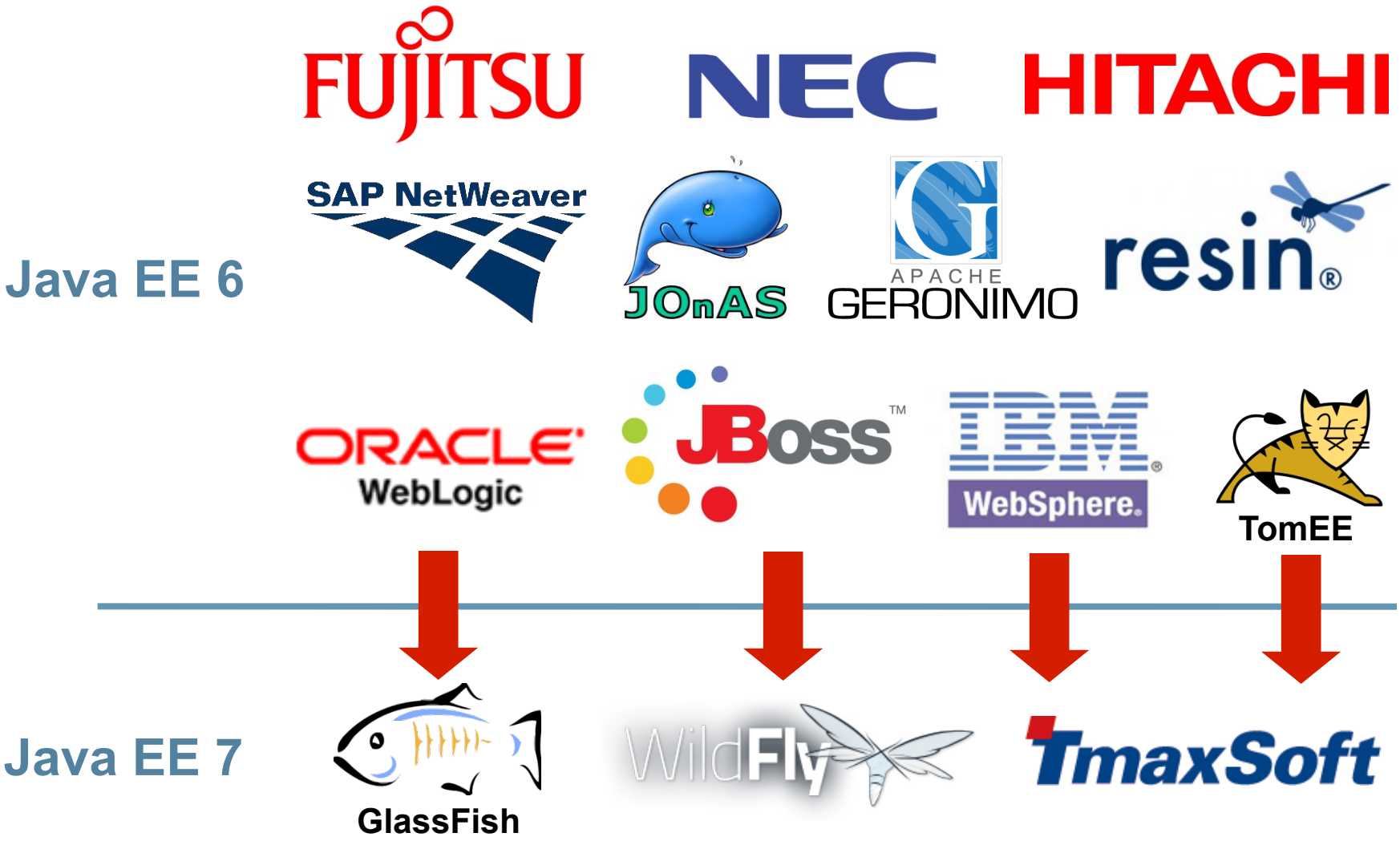
Specifying Connection Factory

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

Specifying Durable Subscriptions

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

JMS 2 Implementations



JMS 2.1

- Declarative JMS Listeners
  - https://java.net/jira/browse/JMS_SPEC-134
  - https://java.net/jira/browse/JMS_SPEC-154
- Cloud/Multitenancy
  - https://java.net/jira/browse/JMS_SPEC-57
- Java SE 8 Alignment
  - https://java.net/jira/browse/JMS_SPEC-151
- JMS/Java EE Pluggability
  - https://java.net/jira/browse/JMS_SPEC-25
- Java SE Bootstrap
  - https://java.net/jira/browse/JMS_SPEC-89
- Management/Monitoring
  - https://java.net/jira/browse/JMS_SPEC-18

Summary

- Long awaited and well received
- API modernization
- Some new features
- Java EE alignment
- EJB3/MDB alignment
- Minor corrections and clarifications
- Get involved in JMS 2.1!

Learning More

- Java EE/JMS Tutorials
  - http://docs.oracle.com/javaee/7/tutorial/doc/partmessaging.htm
- JMS 2 Hands-on-Lab
  - https://github.com/m-reza-rahman/jms2-lab
- JMS Transparent Expert Group
  - http://jms-spec.java.net
- Java EE 7 Reference Implementation
  - http://glassfish.org
- JMS Reference Implementation
  - http://mq.java.net
- The Aquarium
  - http://blogs.oracle.com/theaquarium


Java™