JMS 2.0 – An Introduction

Matthew B White (whitemat@uk.ibm.com)
WebSphere MQ Platform Integration Lead

Please Note

- IBM's statements regarding its plans, directions, and intent are subject to change or withdrawal without notice at IBM's sole discretion.
- Information regarding potential future products is intended to outline our general product direction and it should not be relied on in making a purchasing decision.
- The information mentioned regarding potential future products is not a commitment, promise, or legal obligation to deliver any material, code or functionality. Information about potential future products may not be incorporated into any contract. The development, release, and timing of any future features or functionality described for our products remains at our sole discretion.
- Performance is based on measurements and projections using standard IBM benchmarks in a controlled environment. The actual throughput or performance that any user will experience will vary depending upon many factors, including considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve results similar to those stated here.

Introduction & Aims

- Aims for you:
  - Get some education on what's new and different in JMS 2.0
- Aims for us:
  - How much interest is there in JMS 2.0?
  - What business use cases for the new features can you think of?

References:

- JMS 2.0 – JSR 343 Java Message Service (JMS 2.0)
  - http://jcp.org/en/jsr/detail?id=343
- Final release on 21 May 2013.
  - https://java.net/projects/jms-spec/pages/JMS20FinalRelease

Headlines – What's new? [§1.2]

- *Specification updates and clarifications*
  - Point to Point *AND* Publish/Subscribe domains required
- *New Messaging Features*
  - Delivery Delay
  - Asynchronous Send
  - Subscriptions can be shared across a messaging provider
- *API Changes*
  - Use of `java.lang.AutoCloseable`
  - Simplified API
  - Session doesn't need parameters (for JavaEE)
- *JavaEE Updates*
  - *Recommendation* on provision of a Resource Adapter, and clarifications
  - Part of Java EE 7 (and uses Java 7 Runtime class libraries)

Simplified API

Messaging Features

JavaEE

Updates

Simplified API

Messaging Features

JavaEE

Updates

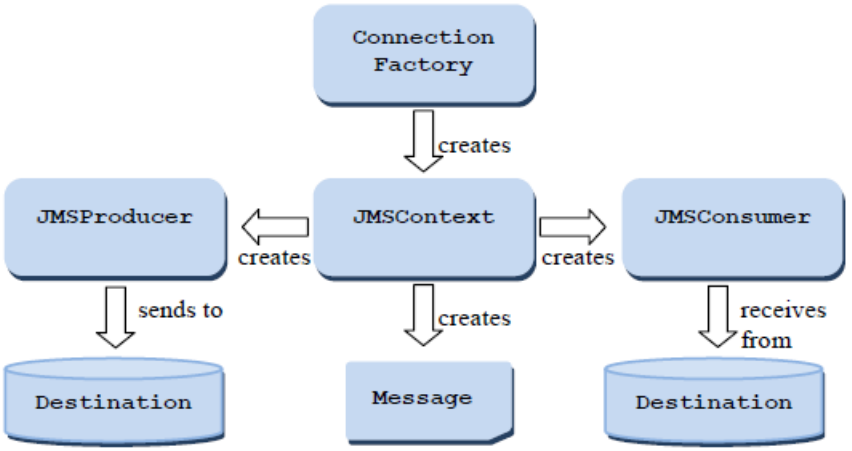
Architecture & Simplified API [§2]

- Overall architecture still the same
- Best practices all the same
- Still a specification for a messaging and and state model

- *Simplified API*
  - New set of interface to reduce number of objects needed
  - Make easier to use within both JavaEE and JavaSE
  - Faster access to data for creating and manipulating message objects
  - Error handling changes

Simplified API [§2.8]

- **ConnectionFactory**
  an administered object to create a Connection. As used by the classic API.
- **JMSContext**
  an active connection to a JMS provider and a single-threaded context for sending and receiving messages
- **JMSProducer**
  created by a **JMSContext,** used for sending messages to a queue or topic
- **JMSConsumer**
  created by a **JMSContext,** used for receiving messages sent to a queue or topic



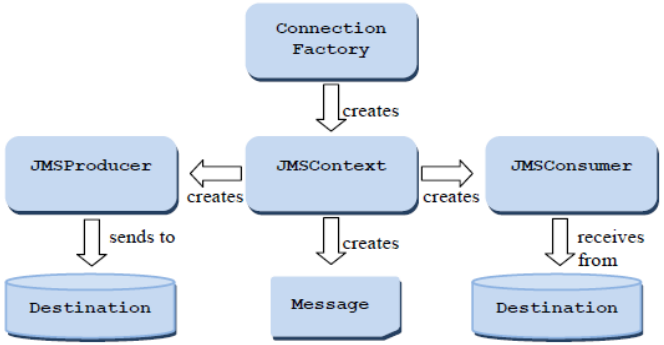
- All based on JMS1.1 *Unified Domain* concepts

JMSContext Features [§6]

- Connection and Session abstraction
- eg. Temporary destinations – for JMSContext scoped by concept of underlying connection

- AutoStart of underlying connection – i.e. can forget to forget starting the connection
- Can create duplicate JMSContext, but using same underlying connection.

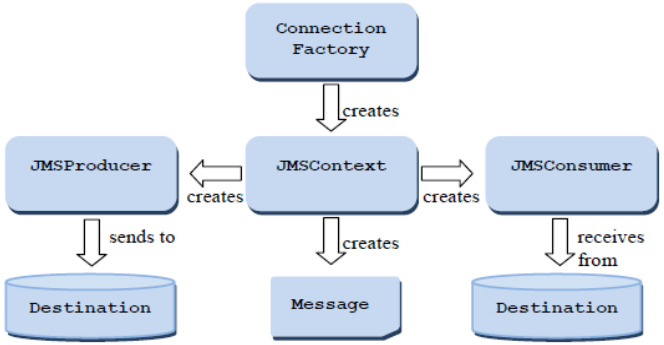
- Application Managed – from `createContext()` on ConnectionFactory
- Container Managed - @Inject annotation



Producing Messages [§7]

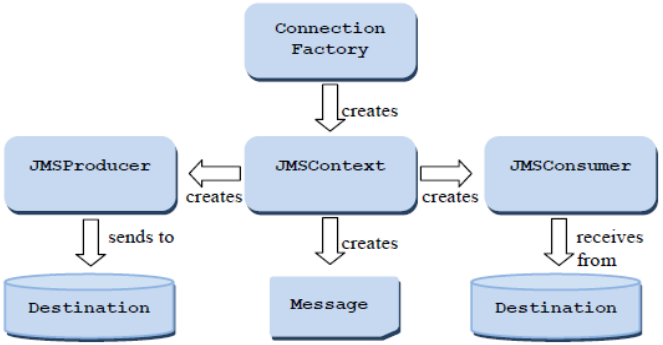
- **JMSProducer**
  - Can take on role of being 'proxy' message object.
  - Message properties set on the producer object prior to sending a 'body'
  - Existing MessageProducers can't do that; though have been extended for new messaging styles
  - Method chaining
  - 'lightweight object' therefore no close

```
11      producer.setProperty("MyProperty", "JMS2.0").send(destination, "SimplePTP: ");
12
13      context.createProducer().setTimeToLive(1000).setDeliveryMode(NON_PERSISTENT).send(dataQueue, body);
```



Consuming messages [§8]

- Async and Sync consumption as before
- **JMSConsumer**
  - `ReceiveBody(...)` methods - message bodies only.
  - Not completely symmetrical with sending – i.e. if want properties still need a message object
- Can closed from another thread
  - Waits on in-progress consumption to finish

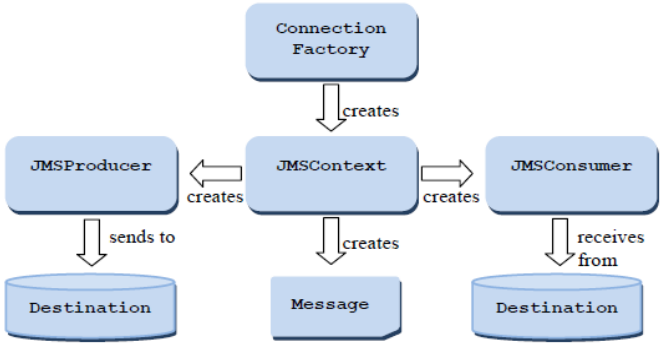


Messages [§3]

- Message Body only
  - JMSProducer/JMSConsumers can now work with the message bodies without message objects
  - Message's getBody method

- After sending message, application free to modify message

- JMSXDeliveryCount mandatory [§3.5.11]
  - Doens't have to be exactly correct – i.e. no persistence of value required
  - If JMSRedelivered=true, then JMSXDevlieryCount>=2



Receiving Message Bodies Synchronously [§8.6]

- `<T> T receiveBody( Class<T> c)`
  - Receives the next message produced for this JMSConsumer and returns its body as an object of the specified type
- `<T> T receiveBody( Class<T> c, long timeout)`
  - Receives the next message produced for this JMSConsumer that arrives within the specified timeout period, and returns its body as an object of the specified type
- `<T> T receiveBodyNoWait( Class<T> c)`
  - Receives the next message produced for this JMSConsumer if one is immediately available and returns its body as an object of the specified type
- Can throw a MessageFormatRuntimeException if wrong class used
  - If AUTO_ACK, will be as if method never called
  - ..else will be as if method failed, app must force rollback

Body Conversions

```
15      Message receivedMessage = consumer.receive(15000);
16      System.out.println("Received message:\n" + receivedMessage.getBody(String.class));
17      System.out.println(receivedMessage.getStringProperty("MyProperty"));
18
19      System.out.println("Message Body"+consumer.receiveBody(String.class));
```

Message Type	Parameter
TextMessage	String.class
ObjectMessage	Java.io.Serializable
MapMessage	Java.util.Map or java.lang.Object
BytesMessage	byte[].class
Message	*Always returns null*
<nobody>	*Always returns null*

JMS 1.1 Interfaces Updates [§6.1]

- **Connections**
  - New methods for creating session with no transaction arguments
  - Shared Connection Consumers
- **Sessions**
  - Creating share consumers
  - Creating JMS1.1 MessageConsumer for durable subscriptions
- **Message Producer**
  - New set/get for Delivery Delay
  - Send methods extended to supply Completion Listener
- **Message Consumer**
  - No change
- **Message**
  - For delivery delay – getJMSDeliveryTime()
  - New methods to directly access body data, getBody() isBodyAssignableTo()

Java 7 - Auto-Closeable

- Java 7 gives large variety of
  - new functions – class libraries
  - Java Language enhancements
  - New class file version
- JMS2.0 drives Java 7 specifically because of the try-with-resources enhancement

```
2   ConnectionFactory cf = createConnectionFactory();
3   try {
4     // Create JMS objects
5     connection = cf.createConnection();
6   } catch (JMSException jmsex) {
7     recordFailure(jmsex);
8   } finally {
9     if (connection != null) {
10      try {
11        connection.close();
12      } catch (JMSException jmsex) {
13        System.out.println("Connection could not be closed.");
14        recordFailure(jmsex);
15      }
16    }
17  }
```

```
2   ConnectionFactory cf = createConnectionFactory();
3
4   try (JMSContext jmsctx = cf.createContext();) {
5     producer = jmsctx.createProducer();
6   } catch (Exception jmsex) {
7     recordFailure(jmsex);
8   }
```

Exceptions and Exception Listeners [§10]

- New exception **`JMSRuntimeException`** thrown from new API calls
- JavaDoc shows mandatory cases
  - ...but... provider may thrown these for other cases as well

- Exceptions thrown on a JMS calls, <u>must not</u> be delivered to an ExceptionListener

- Compiler will not force you to do error checking
- Must take responsibility for doing it at suitable points

Simplified API

Messaging Features

JavaEE

Updates

New messaging features - recap

- **Asynchronous Send**
  - Application sends messages via API that returns before server has processed messages
  - Confirmation via listener
- **Delayed Delivery**
  - Allow delivery at a later point in time
  - (not a database)
- **Shared Subscriptions**
  - Subscription which can opened by multiple consumers
  - Messages shared amongst consumers (no fairness rules provided)

Subscriptions
[§8.3 §4.2]

- **Unshared non-durable subscriptions**
  - As per JMS 1.1. non-durable subscription
- **Shared non-durable subscription**
  - Identified by 'sharedSubscriptionName' and 'clientId' if set
  - If clientId is set, all consumers must share the same clientId
  - Subscription/undelivered messages deleted when last consumer is closed
- **Unshared Durable Subscriptions**
  - As per JMS 1.1 Durable subscriptions
- **Shared Durable Subscriptions**
  - Has the features of a durable subscription but pulls in multiple consumers aspect
  - ClientId is optional
- Same 'sharedSubscriptionName' can be used for durable and non-durable

Asynchronous Send [§7.3]

- *Synchronous Send* means
  - Send message and wait for a response before returning from send call
- *Asynchronous Send* means
  - Send message and return from the send call before response from server

- Closing must wait for failure or completion of async sends
- Callbacks made in the same order the messages where sent
- Message objects not multi-threaded – don't modify until onCompletion
- `CompletionListener` called when response has been received.
  - Does not work like message listener in terms of thread of control
- Quality of Service
  - After `onCompletion` equal to sync send(...)

Message Delivery Delay [§7.9]

- Earliest Time JMS provider may give message to consumer
- Sets the minimum length of time (in ms) that must elapse after a message is sent before the JMS provider may deliver the message to a consumer
- For transacted sends, this time starts when the client sends the message, not when the transaction is committed.
- Delivery Delay set longer than expiry is an error!

Simplified API

Messaging Features

JavaEE

Updates

JMS and JavaEE
[§12 §13]

- JMS one of the constituent specifications of JavaEE
- JavaEE 7 brings in JMS 2.0

- ASF Mode still present – extended for Shared Subscriptions
- Context APIs also include XA variants
- MDB Activation Specifications main way of driving messages into JavaEE

- Strongly *Recommended* to provide a JCA Resource Adapter

JMS and JavaEE – API updates [§12 §13]

- Prescriptive list of APIs that can not be called in the containers
  - No message listeners
  - Single session / connection
  - No connection consumers
  - No Asynchronous Send
  - Transaction control etc.

- Session/Context APIs - No local transactions or client acknowledgement
  - `javax.jms.Connection.createSession()`
  - `javax.jms.ConnectionFactory.createContext()`

Injection of JMSContext [§12.4.3]

- Can declare a JMSContext and annotate it with javax.inject.Inject annotation

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

- Specifying the Connection Factory to use

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

- With optional credentials

```
@JMSPasswordCredential(userName="admin",password="mypass
word")
@JMSPasswordCredential(username="admin",password="$
{ALIAS=myAdminPassword}")
```

- With container-applicable session mode

```
@JMSSessionMode(JMSContext.AUTO_ACKNOWLEDGE)
```

Scope of inject JMSContext

- Different depending if a JTA Transaction in progress or not.

- JTA transaction (*cf.* JTA Specification)
  - the scope will be @TransactionScoped.
  - automatically created the first time a method called within the transaction.
  - JMSContext will be automatically closed when the transaction is committed or rolled back.
  - Within the same JTA transaction, JMSContext injected using identical annotations will refer to the same JMSContext object.
- no JTA transaction (*cf.* CDI Specification)
  - the scope will be @RequestScoped.
  - automatically created the first time a method called within a request.
  - JMSContext will be automatically closed when the request ends.
  - Within the same request, JMSContext injected using identical annotations will refer to the same JMSContext object.

Resource Adapter [§13]

- Assumption that one will be provided

- Two additional properties for an activation specification
- *destinationLookup*
  - the JNDI name of javax.jms.Queue or javax.jms.Topic that defines the JMS queue or topic for the MDB
- *connectionFactoryLookup*
  - the JNDI name of javax.jms.ConnectionFactory, used to connect to the JMS provider

Simplified API

Messaging Features

JavaEE

Updates

Notable specification Clarifications

- [§A.1.19 / §6.1.5]

  Stop blocks until message listeners complete. Listener must not attempt to stop it's own connection → IllegalStateException

- [§A.1.17]

  Message may be sent from any session

- [§A.1.18 / §6.1.7]

  Exception Listener clarifications
  - Exceptions from API should be thrown to listener
  - Stop/close must wait for exceptions to be delivered
  - Exception Listener can use API

Notable specification Clarifications

- [§A.1.20]

  noLocal and Durable Subscriptions
  - –Messages sent on the same connection or ClientID are subject to noLocal
  - –ClientIDs can only be used by one active connection at a time
- [§A.1.21]
  Defines properties that will be ignored if set by the application
- [§A.1.16]
  Minimum set of characters valid for a subscription name, must permit 128 length