Asynchronous Messaging in the Java Stack

Mike Heath
• This is a training **NOT** a presentation
• Please ask questions
Outline

• Asynchronous Messaging
• Java Message Service (JMS) API
• Oracle Streams AQ
• Usage in the Java Stack
• Lab
• Use Cases
Asynchronous Messaging

• A form of loosely coupled distributed communication
• Allows software components to communicate indirectly
• Senders do not need to know who the receivers are
Message-oriented middleware (MOM) provides a distributed asynchronous message queue between a sender and receiver.

Examples:
- ActiveMQ
- Rabbit MQ
- Tibco
- Websphere MQ (Previously MQSeries)
Advantages of Messaging

• Asynchronous – Sender does not need to wait until receiver is done
• Reliable – Messages are typically persisted
• Transactional – Message sending can be done atomically
Java Message Service (JMS)

- JMS is the standard API for asynchronous messaging on the Java platform.
- JMS is similar to JDBC, it is an API not an implementation.
JMS messaging models

• Point-to-point
  – May have multiple senders and receivers
  – Each message gets delivered once

• Publish/subscribe
  – May have multiple senders and receivers
  – Each message gets delivered to each receiver
JMS Terminology

- Provider – Implementation of JMS
- Client – A JMS sender or receiver
- Producer – A point-to-point sender
- Consumer – A point-to-point receiver
- Publisher – A publish/subscribe sender
- Subscriber – A publish/subscribe receiver
- Message – The data being sent and received
- Queue – Distribution mechanism for point-to-point
- Topic – Distribution mechanism for topic/subscribe
- Listener – A consumer or a subscriber
- Destination – A queue or a topic
Streams AQ

• Oracle Streams AQ is MOM provider included in Oracle 10g+.
• There are no additional licensing costs to use Streams AQ.
• Oracle provides a JMS adapter for accessing Streams AQ.
Why Streams AQ

- No additional infrastructure is required.
- Database updates and messaging can be done in a single transaction.
- Configuration can be managed using existing tools.
Configuring Streams AQ

• The schema needs to have sufficient rights to use Streams AQ
  – Or simply Google 'lds stack streams aq configure'
Queue tables

• To use Streams AQ, you must first create a queue table.
• The queue table determines the messaging model (point-to-point or publish/subscribe).
begin
    dbms_aqadm.create_queue_table(
        queue_table => 'QUEUE_TABLE_NAME',
        queue_payload_type => 'SYS.AQ$_JMS_MESSAGE',
        multiple_consumers => false,
        comment => 'Put a descriptive comment here.'
    );
end;
/

Create queue table SQL
Queues

- Each Streams AQ queue must be backed by a queue table.
- One queue table may host multiple queues.
begin
    dbms_aqadm.create_queue(
        queue_name => 'QUEUE_NAME',
        queue_table => 'QUEUE_TABLE_NAME',
        comment => 'Put a descriptive comment here.'
    );
    dbms_aqadm.start_queue(queue_name => 'QUEUE_NAME');
end;
/

Create queue SQL
Using JMS in the Java Stack

- Add the stack-streams-aq-spring dependency to your Maven pom.xml file.
- Add the stack-jms namespace handler to your Spring applicationContext.xml file.
- For details, see the Stack documentation.
Creating a JMS ConnectionFactory

• A JMS ConnectionFactory is used for connecting to the JMS provider
• In your Spring applicationContext.xml do:

  `<stack-jms:jms />`

• This will use the default JDBC data source to create a Streams AQ JMS Connection Factory
Spring JmsTemplate

• Just like JDBC, JMS is a difficult API to use directly
• Similar to JdbcTemplate, Spring provides JmsTemplate to simplify things
• Use the Stack `<stack-jms:destination />` element to add JmsTemplate instance to your app
Creating a Spring JmsTemplate

• The following is an example of configuring a Spring JmsTemplate:

```xml
<stack-jms:jms>
  <stack-jms:destination destination=“QUEUE” />
</stack-jms:jms>
```

• Where QUEUE is the name of your AQ queue
Enqueuing a message

• Once you have configured a JmsTemplate instance, you can use it to enqueue a message.

```java
final String data = "Hello JMS World!";
jmsTemplate.send(new MessageCreator() {
    public Message createMessage(Session session)
        throws JMSException {
        return session.createTextMessage(data);
    }
});
```
• **To listen for messages, we need:**
  – A message listener Spring bean to handle incoming messages. This bean must implement the JMS interface `javax.jms.MessageListener`
  – A Spring JMS listener container

• **To use the Spring JMS listener container, you must have a** `java.util.concurrent.ExecutorService` configured.
An example Spring JMS listener

<!-- Executor service -->
<concurrency:executor />

<!-- Implementation of javax.jms.MessageListener -->
<bean
  id="myMessageListener"
  class="org.lds.foo.MessageListenerImpl" />

<stack-jms:jms>
  <stack-jms:listener-container>
    <stack-jms:listener
      destination="QUEUE"
      ref="myMessageListener" />
  </stack-jms:listener-container>
</stack-jms:jms>
Follow the lab instructions at:

http://tech.lds.org/wiki/Streams_AQ
Use cases for using Streams AQ

• Background task execution
• Event-driven application integration
• Database to application communication
• Making non-transactional resources transactional
You can use a `java.util.concurrent.ExecutorService` for executing background tasks.

However, an `ExecutorService` has a number of drawbacks:

- Non-transactional
- Non-persistent
- Queue is in-memory so if the queue gets too big, it can cause an `OutOfMemory` exception
- Currently, there is no support for monitoring an `ExecutorService`
• Most of the Church application integration is batch oriented
  – MDM
  – LDS Account
  – etc.
• Streams AQ enables near real-time application integration.
Database to application integration

- Streams AQ messages can be queue via PL/SQL
- This enables the database to notify the application of certain events
- This can be useful in cases where multiple applications share a single database
- This can also be used for running background “cron” jobs.
Making non-transactional resources transactional

- Examples of non-transactional resources
  - E-mail
  - File system
- Streams AQ can be employed to access these resources in a transaction safe manner
  - Instead of accessing the resource directly, queue a message indicating how the resource should be used.
  - If transaction fails, message enqueuing will be rolled back.
• Questions?