JBoss World 2006 LAS VEGAS

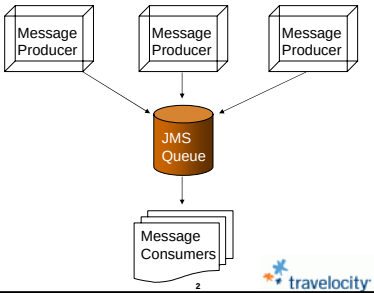
The Internals of JBoss Messaging

How to Customize JMS

travelocity © JBoss Inc. 2006

Messaging & JMS

- JMS is a standard used to pass messages around.



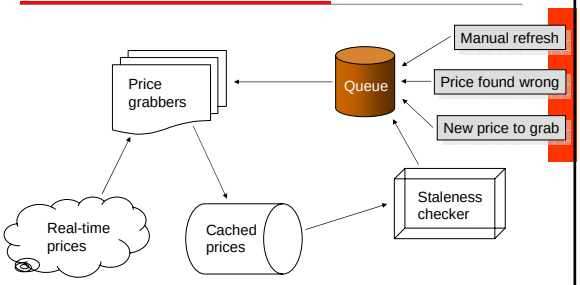
travelocity

Advantages of using messaging

- Common interface
- Good way to throttle applications
- Resistant to crashes

travelocity

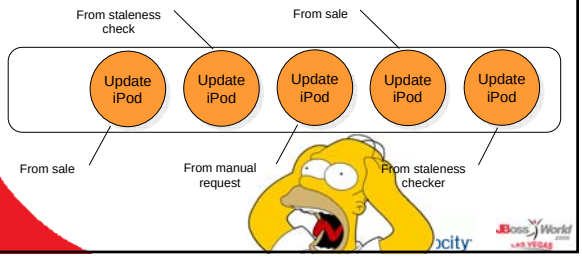
Imagine an app…



travelocity

Problem:

How do you prevent processing duplicate messages?



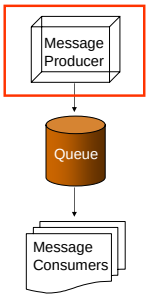
What are some different approaches?

- Checking the queue to make sure no duplicates are inserted.
- Not processing messages that have been recently processed.

travelocity

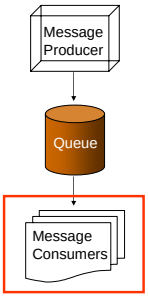
Checking the queue beforehand

- Logic is in message producer
- A lot of network traffic.
- Gets expensive with large queues.
- Unfeasible with many small, individual inserts.

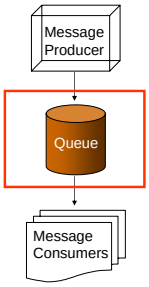


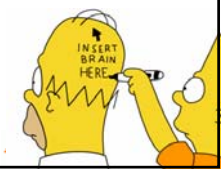
Not processing recently seen messages

- Logic is in message consumer
- Often good enough
- Means prices can only be updated so often.
- Still causes extra load



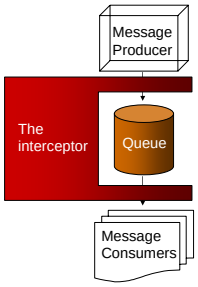
Making the queue smarter





The interceptor approach

- addMessage
- receive
- acknowledge
- transact (?)
- browse
- create/close conn
- subscriptions
- temp destinations



Pros/cons of the interceptor approach

Pros	Cons
• Non-intrusive. • Simple to code. • Resistant to JBoss upgrades.	• Will not track messages internally. • No support for persistent messages. • Applies to all JMS destinations.

Note: Interceptors go into jboss/server/…/lib

Adding an interceptor

```
<!-- Here we hook it into the call chain -->
<mbean code="org.jboss.mq.security.SecurityManager"
    name="jboss.mq:service=SecurityManager">
  <attribute name="DefaultSecurityConfig">
    <security>
      <role name="guest" read="true" write="true" create="true"/>
    </security>
  </attribute>
  <attribute name="SecurityDomain">java:/jaas/jbossmq</attribute>
  <!--depends optional-attribute-name="NextInterceptor">
    jboss.mq:service=DestinationManager</depends-->
  <depends optional-attribute-name="NextInterceptor">
    jboss.mq:service=UniqueMessageInterceptor</depends>
</mbean>

<!-- Here we define our interceptor -->
<mbean code="org.jboss.mq.server.jmx.InterceptorLoader"
    name="jboss.mq:service=UniqueMessageInterceptor">
  <attribute name="InterceptorClass">
    com.gt.jboss.interceptor.UniqueMessageInterceptor</attribute>
  <depends optional-attribute-name="NextInterceptor">
    jboss.mq:service=DestinationManager</depends>
</mbean>
```

The interceptor (addMessage method)

```
public class UniqueMessageInterceptor extends JMSServerInterceptorSupport {

 private static Log log=LogFactory.getLog(UniqueMessageInterceptor.class);
 private HashSet messageSet = new HashSet();
 private String destinationName = "QUEUE.testQueue";

 public void addMessage(ConnectionToken conn, SpyMessage msg) throws
    JMSException {
  if (destinationName.equals(msg.getJMSDestination().toString())) {
   if (msg instanceof TextMessage) {
    TextMessage textMessage = (TextMessage) msg;
    boolean addMessage = false;
    synchronized ( messageSet ) {
     if (messageSet.contains(textMessage.getText())) {
      log.info("Kicking out dupe: [“ + textMessage.getText() + "]");
     } else {
      log.info("Accepting new: [" + textMessage.getText() + "]");
      messageSet.add(textMessage.getText());
      addMessage = true;
    }}
    if ( addMessage ) { super.addMessage(conn, msg); }
   }
  }
 }
```


The interceptor (receive method)

```
//private Set messageSet = Collections.synchronizedSet(new HashSet());
//private String destinationName = "QUEUE.testQueue";

public SpyMessage receive(ConnectionToken conn, int subId, long wait)
   throws JMSException {
 SpyMessage msg = super.receive(conn, subId, wait);
 if (destinationName.equals(msg.getJMSDestination().toString())) {
  if (msg == null) {
   log.info("No more messages.  Clearing set.");
   synchronized (messageSet) {
    messageSet.clear();
   }
  } else if (msg instanceof TextMessage) {
   TextMessage textMessage = (TextMessage) msg;
   log.info("Removing from set: [" + textMessage.getText() + "]");
   synchronized (messageSet) {
    messageSet.remove(textMessage.getText());
   }
  }
 }
 return msg;
}
```

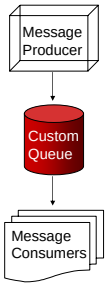

JBoss World 2006 LAS VEGAS

Getting off the beaten path

Implementing a custom queue

The custom queue approach

- addMessage
- receive
- acknowledge
- restoreMessage
- browse
- create/close conn
- subscriptions
- temp destinations
- Access to PersistenceManager

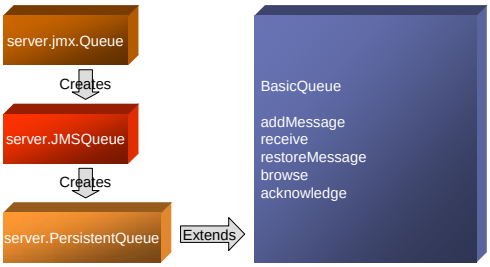


Pros/cons of the custom queue approach

Pros	Cons
• Greatly improved flexibility. • Relatively little coding involved. • Destination-specific.	• Restricts ability to swap out JBossMQ • Not at all supported. • Some knowledge of JbossMQ necessary

The JBossMQ call-chain

```
<mbean code="org.jboss.mq.server.jmx.Queue“ name="…">
  <depends optional-attribute-name="DestinationManager">
    jboss.mq:service=DestinationManager</depends>
</mbean>
```



The new framework

Old	New
Queue.java: `destination = new JMSQueue(…);`	**CustomQueue.java:** `destination = new JMSCustomQueue(getQueueClass(), …);`
JMSQueue.java: `queue = new PersistentQueue(…);`	**JMSCustomQueue.java:** `queue = QueueFactory.createQueue(queueClass, …);`
	QueueFactory.java (85 lines): Uses reflection to find the appropriate constructor, and then creates the queue object.

Creating the UniqueAckQueue

public class UniqueAckQueue **extends** PersistentQueue

- **void** addMessage(MessageReference, Tx)
- **void** acknowledge(AcknowledgementRequest, Tx)
- **void** restoreMessage(MessageReference)
- **void** removeAllMessages()
- **void** dropMessage(MessageReference)
- **void** setupMessageAcknowledgement(Subscription, MessageReference)
- 160 lines of code

Notes:

- Need to keep track of outstanding acknowledgement requests via setupMessageAcknowledgement and a private Map of messageId -> message
- Make sure that the ack response is positive before removing the message

Performance Data

	PersistentQueue	UniqueAckQueue
Insert 10,000	9,911 sec	10,044 sec
Read 10,000 with 1 Thread	9,812 sec	9,875 sec
Read 10,000 with 10 Threads	10,412 sec	10,505 sec

Memory Concerns

- Depends on the key used to determine duplicate messages.
- If an element of the message is used, minimal extra memory is needed.
- All messages are stored in memory.
  - ✓ BasicQueue.messages (SortedSet) > MessageReference > Message > Payload

The UniqueReplaceQueue

- Removes similar messages on the queue if a new one is added.
- Uses a HashMap to map keys to Messages.
- Removes messages both from memory and from persistent storage.
- 194 lines of code

	PersistentQueue	UniqueReplaceQueue
Insert 10,000	9,911 sec	13,112 sec

About Topics

- A topic is just a collection of queues.
- One queue is created for each subscriber.
- Topics can be customized just like queues once the custom loader is built.

Looking into the future

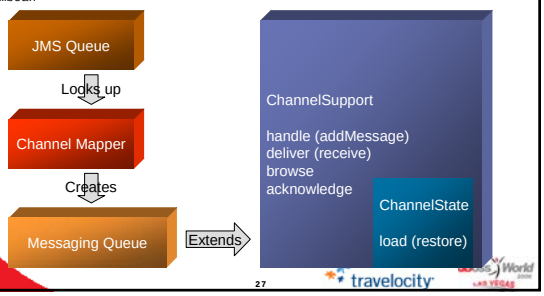
The JBoss Messaging product

What is JBoss Messaging?

- Replacement for JBoss MQ
- Separation of JMS and messaging
- Undergoing constant development

The JBoss Messaging call-chain

```
<mbean code="org.jboss.jms.server.destination.Queue"
      name="jboss.messaging.destination:service=Queue,name=testQueue"
      xmbean-dd="xmdesc/Queue-xmbean.xml">
      <depends optional-attribute-
    name="ServerPeer">jboss.messaging:service=ServerPeer</depends>
</mbean>
```



The interceptor approach

- Completely AOP-based.
- Divided into client-side and server-side interceptors.

```
<aspect class="org.jboss.jms.server.container.CustomAspect"
    scope="PER_INSTANCE"/>
<bind pointcut="execution(*
    org.jboss.jms.server.endpoint.advised.SessionAdvised->send(..))">
 <advice name="handleSend"
    aspect="org.jboss.jms.server.container.CustomAspect"/>
</bind>
```


The custom queue approach

- Currently not feasible because of the speed of JBoss Messaging development.
- Some pieces critical for dynamic queue loading buried deep within some very large files.

Questions?

gary.tong@travelocity.com

(Source code and presentation available)