Using Message Driven Beans to Consume Messages from WebSphere MQ

10 December, 2013
Tim Elwood (telwood@us.ibm.com)
Ty Shrame (tyshrame@us.ibm.com)
Agenda:

- General Overview
- What is a Message Driven Bean?
- What is a Resource Adapter?
- The WebSphere MQ RA
- Activation Specifications
- Message Delivery to MDBs
- Common MDB Problems
- Tracing MDB Behavior and Performance
- Tuning the MQ RA
- Troubleshooting (Javacores)
- Question and Answer Session
## Presenters:

<table>
<thead>
<tr>
<th>Panelist</th>
<th>Role at IBM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tim Elwood</td>
<td>Staff Software Engineer</td>
</tr>
<tr>
<td>Ty Shrake</td>
<td>Advisory Software Engineer</td>
</tr>
</tbody>
</table>
General Overview

The key question is: How do we configure our system so that a Message Driven Bean (MDB) deployed to WAS can consume messages that are on a real WebSphere MQ queue?
General Overview – Cont'd

There are 3 basic components needed:

1) A Message Driven Bean running in WebSphere Application server.

2) An Activation Specification that will configure the MQ Resource Adapter to connect to MQ and attach to a specific queue. This is configured in WebSphere Application Server.

3) A defined queue in MQ and an MQ Listener that is running.
What is a Message Driven Bean (MDB)?

An application whose purpose is to receive (consume) messages from a JMS messaging system.

MDBs are specifically designed to take the load off of larger applications that are busy with business logic, database updates, order confirmation, etc... And to simplify the coding of consumer applications.

In most cases the MDB will consume a message and perform a small amount of work on the message and then pass the message data to a larger application (like a Session bean that performs the heavy duty work, such as storing the message data in a database, etc...)

Important Features of MDBs:

1. MDBs do not need any connection code to consume messages.
2. All MDBs have a method named `onMessage()`.
3. MDBs can only run in a J2EE JVM, not a J2SE JVM
4. MDBs run inside an EJB Container
The most important feature of an MDB is the `onMessage()` method inside the bean code. This method is used to accept messages that are passed to it. The `onMessage()` method takes a message object as a parameter, as follows:

```java
onMessage(message_object)
```

It should be noted that the `onMessage()` method is, by default, unimplemented. This means that when you create an MDB this method is included in the bean code by default but it doesn't actually DO anything with the message object that is passed to it. There is no code in the method. The DO part is the responsibility of the code owner (the customer). The code owner must write the code that processes the message. Once this message processing starts the message is completely owned by the MDB and is outside the control of any WAS or MQ code.
What is a Message Driven Bean? - Cont'd

Every MDB includes a **Deployment Descriptor**, which is a configuration file in XML format usually named **ejb-jar.xml**. This file specifies a number of things including which **Activation Specification the MDB will use**. When working with MDBs it is often helpful to know which Activation Specification the MDB is using. This information can be found in the WAS Admin Console by navigating as follows:

**Applications > Enterprise Applications > YOUR_MBD > Message Driven Bean listener bindings**

The Listener Bindings column on the right will show which Activation Specification your MDB is using.
What is a Resource Adapter?

Some History: Before J2EE, when a customer needed to access information in another system (such as another enterprise system) they or the vendor had to create a custom interface to connect to and access the information. With so many different enterprise systems in existence this became a real problem.

One way to think if this is to imagine connecting 2 systems that were never intended to connect to one another, much like having a round connector and a square connector that must be joined together. Each vendor would create their own way of connecting these together. The creation of J2EE Resource Adapters solves this problem by creating standardized interfaces that everyone uses.

A Resource Adapter (RA) is a component of the J2EE Connector Architecture (JCA).

The architecture defines a uniform way to integrate J2EE application servers with enterprise information systems and applications. Vendors no longer have to customize their products to interact with application servers.

In a nutshell, Resource Adapters allow various application servers and enterprise systems to connect to each other in a standardized way. All vendors will use the same architecture to connect to the application servers.
What is a Resource Adapter? - Cont'd

In simplest terms, *Resource Adapters are programs* that are used to connect 2 different technologies that often times weren't originally designed to connect to each other.

WebSphere Application Server comes with 2 JMS Resource Adapters that connect JMS applications to messaging environments:

**Default Messaging (SIB) Resource Adapter** (to connect to Service Integration Bus inside of WAS) The name of the file is *sib.ra.rar*

**WMQ Resource Adapter** (to connect to WMQ). The name of the file is *wmq.jmsra.rar*

Updates to all Resource Adapters, including the WMQ RA, are included in WAS fixpacks. Problems with Resource Adapters should be addressed by applying a more recent fixpack level.
The WebSphere MQ RA

The WebSphere MQ Resource Adapter connects applications running in WebSphere Application Server to queues in Websphere MQ. You can think of it as 'the middle man' between WAS and WMD:

The MQ RA is a program that connects these 2 environments. **It has a thread that connects to the queue in MQ and 'listens' for messages arriving on that queue. When a message arrives on the queue it passes ('delivers') that message to the WAS side.**
Activation Specifications

An **Activation Specification** (AS) is a *definition* or object stored in JNDI (it’s configuration parameters are also stored in *resources.xml* in WAS). It stores connection parameters (much like a Connection Factory) that are passed to the Resource Adapter in WAS so that the RA can create a connection to a messaging engine or MQ queue manager. *In other words, an Activation Specification is an object or definition that configures the RA so that the RA knows WHAT to connect to and WHERE it is on the network.*

An Activation Specification is NOT a program. It is simply a configuration object that is stored JNDI. When the RA starts it calls into JNDI to retrieve the Activation Specification object and then reads the configuration information inside the object and configures itself accordingly.
You create the **Activation Specification** (AS) in the WAS Admin Console by navigating as follows:

**Resources > JMS > Activation Specifications**

Then click the New button to create a new AS. You will be given an option to choose either “Default messaging provider” or “WebSphere MQ messaging provider”. Be sure to select **WebSphere MQ messaging provider**.
Activation Specifications – Cont'd

Once you provide a name and JNDI name for the AS you will be asked for the Destination JNDI name. This is actually a JMS Queue definition. This is the name of the queue in WMQ that the MQ Resource Adapter will attach to and the MDB will receive message from. **You should configure this in WAS before creating the AS.**

Instructions for creating the destination can be found here:

http://www14.software.ibm.com/webapp/wsbroker/redirect?version=phil&product=was-nd-dist&topic=tj00001
Activation Specifications – Cont'd

The next part of the configuration much like configuring a normal WMQ Client application and this is where you will enter the **name of the MQ Queue Manager** the Resource Adapter will connect to, the **transport** (client or bindings), the **hostname** (or IP address) of the machine where WMQ is running, the **port number the MQ Listener is running on** and the **Server Connection Channel** the MQ Resource Adapter will connect to.

![Diagram showing configuration details](image)

- Queue manager: QM1
- Transport: Client
- Hostname: 111.222.333.444
- Port: 1414
- Server connection channel: SVRCONN1
The next step allows you to test the connection. Make sure this works! If it fails make sure the values you entered are correct and that MQ has an MQ Listener running on the port number you entered.
Activation Specifications – Cont'd

If the connection test passes then save this Activation Specification definition. This definition is now stored in JNDI and you will need to restart the servers in WAS in order for this new definition to get picked up.

When the MQ resource Adapter starts up it will read this AS definition and use the values you entered to configure a connection to the WMQ queue manager. It will then connect to WMQ, attach to the queue and wait for messages to arrive on the queue. If messages already exist on the queue then message delivery to the MDB will begin immediately.

If message delivery fails you should have a look at the following WebSphere Technical Exchange Presentation:

Service Integration Bus and Stuck Messages
Message Delivery to MDBs

Now let’s put all of this together. Here are the steps that occur when a message arrives on the queue in WMQ:

1) Message arrives on queue in WMQ.

2) MQ informs the connected MQ RA thread that a message has arrived.

3) MQ RA thread receives a copy of the message from the queue.

4) The MQ RA now calls the onMessage() method of the MDB that is using the MQ RA. When it calls the onMessage() method it passes the message object to the method (and to the bean itself). This is called **delivering the message to the MDB**.

5) Once the message has been delivered to the MDB the bean must process the message. In a normal system this should not take very long (perhaps a few milliseconds). If processing the message takes a long time this indicates a possible problem with the processing.

6) When message processing completes (returns) the EJB container the bean is running in is notified and the MQ RA is then allowed to get the next message from the queue. The previously delivered message is deleted from the queue.

... Back to Step 1...
Message Delivery to MDBs – Cont'd

The diagram below illustrates the basic message delivery process:

onMessage(msg_object) is called and message is delivered to the MDB

OnMessage() method completes and returns control back to the MQ RA to get the next message.
Common MDB Problems

- The most common problems encountered when using MDBs are:

  1) The MDB is consuming messages but at a slow rate. As a result, messages are sent to the queue faster than they are consumed and messages begin to stack up on the queue.

  2) Messages are not consumed. The MDB is running but is hung or is otherwise unresponsive.

  3) The MDB is not running (and may not be able to start).
Tracing MDB Behavior and Performance

You can get a good understanding of how your MDB is performing by setting the following trace string on the server where your MDB is running:

*=all

You will want to enable this trace anytime message consumption by the MDB seems slow or you see messages accumulating on the queue the MDB is consuming from. This trace string will reveal how rapidly messages are consumed from the queue.

If messages are stuck on a queue you may want to gather javacores and view them using the Javacore Analyzer tool, here:

IBM Thread and Monitor Dump Analyzer for Java
Tracing MDB Behavior and Performance – Cont'd

When an MDB is connected to WMQ you should check for trace entries similar to this:


You should always look for the Entry and Exit points of the onMessage() method. The Entry point (first line) shows when onMessage() was called (when a message was delivered to the MDB) and the Exit point (second line) shows when onMessage() returned (completed). Between the Enter and Exit times the MDB will usually be running customer code. The more time spent between the Enter and Exit times the slower the performance of the MDB. In the example above we can see that only 15 milliseconds were spent in onMessage() so there is no performance problem here.
Tuning the MQ RA

Message consumption is generally slower than message production and since MDBs are message consumers we often see performance problems with them. Activation Specifications have a feature to help with this called Maximum server sessions. The default value is 10. Navigate to this property as follows:

Resources > JMS > Activation Specifications > YOUR_Act Spec > Advanced Properties

<table>
<thead>
<tr>
<th>Maximum server sessions</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
</tr>
</tbody>
</table>

Note: If strict message order must be maintained this value should be set to 1.

This means that up to 10 instances (copies) of the same MDB can run simultaneously. While 1 instance of the MDB is busy processing a message one of the other instances can consume the next message. This can significantly increase the rate of message consumption. Each MDB instance runs on a different thread. Using more than 1 MDB instance is sometimes called MDB Throttling.

Unfortunately there is no simple way in WAS to see how exactly many MDB instances are running. One way to see is with trace or Javacores. However, in the PMI tool that comes with WAS the LiveCount counter under Enterprise Beans will show the total number of beans running in a particular JVM.
Troubleshooting (Javacores)

In situations where the onMessage() method is not returning (completing) in a timely manner it is a good idea to gather javacores against the JVM where the MDB runs. Ideally at least 3 javacores should be taken every 30-60 seconds until a total of at least 3 or 4 javacores have been gathered. The javacores should be taken against the JVM the bean runs on.

In general javacores can be gathered using the kill command, as follows:

```
kill -3 PID
```

(where PID is the Process ID of the JVM where the MDB runs)

Javacores will reveal where the application is spending so much time inside the onMessage() method. Once onMessage() is called we enter non WAS code. Code that executes from this point forward is the responsibility of the owner of this code. Many times this code will stop executing for any number of reasons, but in most cases execution stops because the code is waiting for something to happen. A good example would be when the code is waiting on a response from some other component such as another EJB or a database. In these situations the javacores will show the MDB thread is ‘stuck’ in the same part of code for a long period of time.

The example shown on the next page shows what a typical MDB hang looks like in a javacore.
Troubleshooting (Javacores) – Cont’d

"Default : o" (TID:0x000000001A774400, sys_thread_t:0x0000000019544630, state:B, native ID:0x0000000003566) prio=5
22 at com/company3/common/jdbc/SimpleDataSource.popConnection(SimpleDataSource.java:566(Compiled Code)) < HANG
21 at com/company3/common/jdbc/SimpleDataSource.getConnection(SimpleDataSource.java:222)
20 at com/company3/sqlmap/engine/transaction/jdbc/JdbcTransaction.init(JdbcTransaction.java:48)
19 at com/company3/sqlmap/engine/transaction/jdbc/JdbcTransaction.getConnection(JdbcTransaction.java:89(Compiled Code))
18 at com/company3/sqlmap/engine/mapping/statement/MappedStatement.executeQueryForObject(MappedStatement.java:120)
17 at com/company3/sqlmap/engine/impl/SqlMapExecutorDelegate.queryForObject(SqlMapExecutorDelegate.java:518)
16 at com/company3/sqlmap/engine/impl/SqlMapExecutorDelegate.queryForObject(SqlMapExecutorDelegate.java:493)
15 at com/company3/sqlmap/engine/impl/SqlMapSessionImpl.queryForObject(SqlMapSessionImpl.java:106)
14 at com/company3/sqlmap/engine/impl/SqlMapClientImpl.queryForObject(SqlMapClientImpl.java:82)
13 at com/company1/cos/data/service/impl/company3/company2DataServiceImpl.upsertServiceRequestStatus(Bytecode PC:120)
12 at com/company1/cos/services/messaging/tasks/company2/company2SrTask.refreshSR(Bytecode PC:127)
11 at com/company1/cos/services/messaging/tasks/company2/company2SrTask.doWork(Bytecode PC:292)
10 at com/company1/cos/services/messaging/service/XmITextMessageService.onMessage(Bytecode PC:121)
 9 at com/company1/cos/services/messaging/ejb/QueueConsumerBean.onMessage(Bytecode PC:108) <<< Enter onMessage() !!!
 8 at com/ibm/ejs/container/MessageEndpointHandler.invokeMdbMethod(MessageEndpointHandler.java:1014(Compiled Code))
 7 at com/ibm/ejs/container/MessageEndpointHandler.invoke(MessageEndpointHandler.java:747(Compiled Code))
 6 at $Proxy7.onMessage(Bytecode PC:18)
 5 at com/ibm/ws/wmq/api/jmsr/impl/JmsJcaEndpointInvokerImpl.invokeEndpoint(JmsJcaEndpointInvokerImpl.java:201)
 4 at com/ibm/ws/wmq/ra/inbound/impl/WMQRaDispatcher.dispatch(SibRaDispatcher.java:768(Compiled Code))
 3 at com/ibm/ws/wmq/ra/inbound/impl/WMQRaSingleProcessListener$SibRaWork.run(SibRaSingleProcessListener.java:584)
 2 at com/ibm/ejs/j2c/work/WorkProxy.run(WorkProxy.java:419(Compiled Code))
 1 at com/ibm/ws/util/ThreadPool$Worker.run(ThreadPool.java:1473(Compiled Code))

Read from bottom to top!
Troubleshooting (Javacores) – Cont'd

Javacore Analysis:

We start from the bottom and read up:

On line 4 notice the **WMQRaDispatcher**. This is the WMQ Resource Adapter.
On line 6 we see the first mention of **onMessage()**.
On lines 7 and 8 we see the EJB Container prepare to call onMessage() of the customer MDB.
On line 9 we see that **onMessage()** is called. The method is in a customer class (MDB) named **QueueConsumerBean**. We are now in customer code!
Notice that from lines 10 to 13 the package name is “com/company1” and we are no longer in “com/ibm” code.
This means we are now in code from the company1 company (company1.com).
Notice from lines 14-22 the package name is “com/company3”. This again is not IBM code.
Notice that on line 19 “jdbc” appears in the call stack. JDBC stands for Java Database Connectivity. This means the customer code is now connecting to a database.
In line 22 the code is now in **popConnection()**, which is customer code. This is where the hang occurs.

If at least one other javacore shows this same thread in the exact same code (if they show exactly the same stack with popConnection() at the top), and the customer complains that this MDB (**QueueConsumerBean**) isn’t consuming messages, then this MDB is hung in **popConnection()** (at the top of the call stack). The MDB is probably waiting for a response from the database, so the database, not the MDB, is the real problem.
Summary

In this presentation we have discussed the basic operation of Message Driven Bean and how to configure Activation Specifications in WAS to bind an MDB to a WebSphere MQ Queue for message consumption. We also reviewed problem determination using trace and what to look for in the trace. It is our hope that this material will guide you to success but also help you analyze and make sense of any problems that may arise.
Additional WebSphere Product Resources

- Learn about upcoming WebSphere Support Technical Exchange webcasts, and access previously recorded presentations at: http://www.ibm.com/software/websphere/support/supp_tech.html

- Discover the latest trends in WebSphere Technology and implementation, participate in technically-focused briefings, webcasts and podcasts at: http://www.ibm.com/developerworks/websphere/community/

- Join the Global WebSphere Community: http://www.websphereusergroup.org

- Access key product show-me demos and tutorials by visiting IBM® Education Assistant: http://www.ibm.com/software/info/education/assistant


- Sign up to receive weekly technical My Notifications emails: http://www.ibm.com/software/support/einfo.html
Connect with us!

1. Get notified on upcoming webcasts
   Send an e-mail to wsehelp@us.ibm.com with subject line “wste subscribe” to get a list of mailing lists and to subscribe

2. Tell us what you want to learn
   Send us suggestions for future topics or improvements about our webcasts to wsehelp@us.ibm.com

3. Be connected!
   Connect with us on Facebook
   Connect with us on Twitter
Question and Answer