Table of Contents

JMS Enhancements Lab

Goals

- Understand JMS enhancements implemented in JBoss A-MQ
- Complete lab exercises on exclusive consumer, hierarchical chat, message groups, prioritized consumer, and virtual destinations

Lab Assets

The lab exercises are available in the following zip archive:

- GitHub GPE MW Training : Messaging Labs Repository

1. Create an Exclusive Consumer

Introduction

In this lab exercise, you create a MessageProducer queue and an exclusive consumer, using the JBoss A-MQ broker and Apache Maven.

The exclusive consumer strategy is to avoid load-balancing messages between the *n* consumers connected to a queue and instead consume messages with the client by defining the `consumer.exclusive=true` property in its URI connection definition.

You start two consumers with the same URL definition, and the broker uses the exclusive mechanism to assign messages to the first client connected. If that client looses its connection with the broker, then the second client receives the messages and processes them.

The project has two parts:

- The consumer project starts a JMS MessageConsumer that reads messages as long as it can find one.

- The producer project starts a JMS MessageProducer that creates the specified number of JMS messages and then stops and exits. The sending of messages is artificially slowed with 100ms "sleep" between messages to enable the user to interrupt them when desired.

Procedure

1. If it is not already running, starts the message broker as described in the Module 1 lab.
   - By default, the connection details are provided for connecting to the broker on localhost.
   - If you are using an OpenShift environment, use the correct connection details (URL, `admin` user ID/password) to connect to the OpenShift Enterprise JBoss Fuse cartridge.
2. Start the first consumer:
   a. Open a command line terminal (Windows or UNIX) and navigate to `3_JMS_Enhancements/exclusive-consumer`.
   b. Run this Maven command:

      ```
      mvn -P consumer
      ```

3. Start the second consumer:
   a. Open another terminal window and navigate to `3_JMS_Enhancements/exclusive-consumer`.
   b. Run this Maven command:

      ```
      mvn -P consumer
      ```

4. Start the producer:
   a. Open another terminal window and navigate to `3_JMS_Enhancements/exclusive-consumer`.
   b. Run this Maven command:

      ```
      mvn -P producer
      ```

5. Observe that the messages are consumed by the first consumer connected to the queue.
   - If the first consumer is killed (Ctrl+C), then the second consumer becomes the exclusive consumer and consumes from the queue.
6. (Optional) There is no limit on how many consumers can be created.
   a. Experiment with shutting down clients while the message processing is underway

and watch what happens.

b. Try to restart a client and see what happens.

> The consumer project is designed to continue listening for messages.

7. To exit the consumer project, press **Ctrl+C** for each consumer.

2. Create a Hierarchical Chat

Introduction

In this JBoss A-MQ exercise, you apply policies in a chat that uses a JMS broker and Apache Maven to run JMS clients. The project uses hierarchical topics, where `sales` represents all sales, `sales.usa` represents US sales, and `sales.usa.ca` represents California sales.

```
World:   sales
USA:     sales.usa
Cal:     sales.usa.cal
```

The hierarchicalClient project starts a chat client that catches command line input, listens for incoming messages on the chat room topic, and publishes the input to a topic with the same topic name that has a specified `ptype` extension.

Because the hierarchy is created by defining the topic name with dot separators (`x.y.z`), the client accesses more or fewer published messages depending on the level of the hierarchy from which it requests messages. It is possible for the client to receive all messages or only part of a level’s messages by using the `>` and `*` symbols.

Therefore `mvn -P chatter -Dtype=usa -Dptype=".>"`:

- Publishes to: `sales.usa`
- Listens to: `sales.usa.>`

Whereas `mvn -P chatter -Dtype=usa -Dptype=“.*”`:

- Publishes to: `sales.usa`
- Listens to: `sales.usa.`

> The sending of messages is artificially slowed with 100ms "sleep" between messages to enable the user to interrupt them when desired.

Procedure

1. If it is not already running, start the message broker as described in the Module 1 lab.
   - By default, the connection details are provided for connecting to the broker on localhost.
   - If you are using an OpenShift environment, use the correct connection details (URL, `admin` user ID/password) to connect to the OpenShift Enterprise JBoss Fuse cartridge.

   > This procedure copies files to implement the chat clients (World, USA, Cal). The process can fail if the clients are started concurrently. Be sure that one chat client is up and running before you start the next one.

2. Start the first chat client:
   a. Open a command line terminal (Windows or UNIX) and navigate to the `3_JMS_Enhancements/hierarchical-chat` directory.
   b. Run this Maven command:

```
mvn -P chatter -Dtype=world -Dptype="<Policy Type which can be '.*' or '.>'
without the quotes>"
```

3. Start the second chat client:
   a. Open another terminal window and navigate to `3_JMS_Enhancements/hierarchical-chat`.
   b. Run this Maven command:

```
mvn -P chatter -Dtype=usa -Dptype="<Policy Type which can be '.*' or '.>'
without the quotes>"
```

4. Start the third chat client:
   a. Open another terminal window and navigate to `3_JMS_Enhancements/hierarchical-chat`.
   b. Run this Maven command:

```
mvn -P chatter -Dtype=cal -Dptype="<Policy Type which can be '.*' or '.>'
without the quotes>"
```

5. (Optional) Experiment with different `ptype` settings and see what happens.

   > The consumer project is designed to continue listening for messages.

6. To exit the consumer project, press **Ctrl+C** for each consumer.

3. Create Message Groups

Introduction

In this exercise, you create a JMS MessageProducer queue that sets the `JMSXGroupID` property, and you create queue consumers that use the JBoss A-MQ broker and Apache Maven.

The `JMSXGroupID` message group property tells the JBoss A-MQ broker that messages with the `JMSXGroupID` property should be delivered to a connected consumer based on the value of `JMSXGroupID`.

The project has two parts:

- The consumer project starts a JMS MessageConsumer that reads messages as long as it can find one.
- The producer project starts a JMS MessageProducer that creates the specified number of JMS messages and then stops and exits. The sending of messages is artificially slowed with 100ms "sleep" between messages to enable the user to interrupt them when desired.

Procedure

1. If it is not already running, start the message broker as described in the Module 1 lab.
   - By default, the connection details are provided for connecting to the broker on localhost.
   - If you are using an OpenShift environment, use the correct connection details (URL, `admin` user ID/password) to connect to the OpenShift Enterprise JBoss Fuse cartridge.
2. Start the first consumer:
   a. Open a command line terminal (Windows or UNIX) and navigate to the `3_JMS_Enhancements/message-groups` directory.
   b. Run this Maven command:

   ```
   mvn -P consumer
   ```

3. Start the second consumer:
   a. Open another terminal window and navigate to

`3_JMS_Enhancements/message-groups`.

   b. Run this Maven command:

```
mvn -P consumer
```

4. Start the producer:

   a. Open another terminal window and navigate to `3_JMS_Enhancements/message-groups`.

   b. Run this Maven command:

```
mvn -P producer
```

5. Verify that the messages created with the `JMSXGroupID` property `MSG_1` are consumed by the first consumer and that the `MSG_0` group messages are consumed by the second consumer.

6. (Optional) There is no limit on how many consumers can be created. Experiment with shutting down clients while the message processing is underway and watch what happens.

   The consumer project is designed to continue listening for messages.

7. To exit the consumer project, press **Ctrl+C** for each consumer.

4. Create Prioritized Exclusive Consumer

Introduction

In this exercise, you create a MessageProducer queue and a prioritized exclusive consumer. By combining the exclusive consumer with a priority level, messages with the highest `consumer.priority` value are delivered to the consumer. If this consumer is removed, then the consumer with the next highest priority gets the messages.

The project has two parts:

- The consumer project starts a JMS MessageConsumer that reads messages as long as it can find one.
- The producer project starts a JMS MessageProducer that creates the specified number of JMS messages and then stops and exits. The sending of messages is artificially slowed with 100ms "sleep" between messages to enable the user to interrupt them when desired.

Procedure

1. If it is not already running, starts the message broker as described in the Module 1 lab.
   - By default, the connection details are provided for connecting to the broker on localhost.
   - If you are using an OpenShift environment, use the correct connection details (URL, `admin` user ID/password) to connect to the OpenShift Enterprise JBoss Fuse cartridge.
2. Start the three different consumers with different priorities:
   a. Open a command line terminal (Windows or UNIX) and navigate to the `/3_JMS_Enhancements/prioritized-exclusive-consumer` directory.
   b. Run this Maven command:

      ```
      mvn -P consumer -Dpriority=1
      ```
3. Start the producer:
   a. Open another terminal window and navigate to `/3_JMS_Enhancements/prioritized-exclusive-consumer`.
   b. Run this Maven command:

      ```
      mvn -P producer
      ```

      - The messages are consumed by the consumer with priority 1.
4. Change the consumer priorities:
   a. Open another terminal window and navigate to `/3_JMS_Enhancements/prioritized-exclusive-consumer`.
   b. Run this Maven command:

      ```
      mvn -P consumer -Dpriority=4
      ```
   c. Observe that the messages are now processed by the consumer with the highest priority.
   d. Open another terminal window and navigate to `/3_JMS_Enhancements/prioritized-exclusive-consumer`.
   e. Run this Maven command:

      ```
      mvn -P consumer -Dpriority=2
      ```

   f. Observe that the messages are still processed by the priority 4 consumer. Priority 4 supercedes the other consumers.

5. (Optional) There is no limit on how many consumers can be created.
   a. Experiment with shutting down clients while the message processing is underway and watch what happens.
   b. Use the same priority for several consumers and see what happens.

   The consumer project is designed to continue listening for messages.

6. To exit the consumer project, press **Ctrl+C** for each consumer.

5. Create Virtual Destinations

This lab exercise works only in a localhost environment. If you are using the OpenShift environment, skip this exercise and go to the next section.

Introduction

In this exercise, you use JBoss A-MQ and Apache Maven to create virtual destinations.

This exercise requires a different JBoss A-MQ broker configuration because the virtual destinations are defined on the broker side. You must shut down the running broker and use the local one packaged with this exercise.

The XML configuration file of this local broker contains a `CompositeQueue` definition to forward `test.virtual.queue` messages to `test.queue.one` only if the message contains `id=0`, and to `test.queue.two` and `test.topic.three`.

```
<compositeQueue name="test.virtual.queue">
    <forwardTo>
        <filteredDestination selector="i = 0" queue="test.queue.one"/>
        <queue physicalName="test.queue.two"/>
        <topic physicalName="test.topic.three"/>
    </forwardTo>
</compositeQueue>
```

The project has four parts:

- The root project starts the ActiveMQ message broker.
- The consumer project starts a JMS MessageConsumer that reads messages as long as it can find one.

- The producer project starts a JMS MessageProducer that creates the specified number of JMS messages and then stops and exits. The sending of messages is artificially slowed with 100ms "sleep" between messages to enable the user to interrupt them when desired.
- The listener project starts a JMS MessageListener that reads messages as long no `SHUTDOWN` command message is sent. It also creates a MessageConsumer that listens for control reply messages like reports and the shutdown message.

Procedure

1. Start the message broker:
   a. Open a command line terminal (Windows or UNIX) and navigate to the `3_JMS_Enhancements/virtual-destination` directory.
   b. Run this Maven command:

      ```
      mvn -P broker
      ```

      - This starts the local ActiveMQ message broker and displays a rolling log file.
2. Start the first consumer:
   a. Open another terminal window and navigate to `3_JMS_Enhancements/virtual-destination`.
   b. Run this Maven command:

      ```
       mvn -P consumer -DdestExt=one
      ```

3. Start the second consumer:
   a. Open another terminal window and navigate to `3_JMS_Enhancements/virtual-destination`.
   b. Run this Maven command:

      ```
      mvn -P consumer -DdestExt=two
      ```

4. Start the listener:
   a. Open another terminal window and navigate to `3_JMS_Enhancements/virtual-destination`.
   b. Run this Maven command:

      ```
      mvn -P consumer -DdestExt=three
      ```

5. Start the producer:

   a. Open another terminal window and navigate to the exercise root directory.

   b. Run this Maven command:

   ```
   mvn -P producer
   ```

6. Verify that the consumer connected to `test.queue.one` and the consumer connected to `test.queue.three` receive all 100 messages, and that the client connected to `test.queue.one` receives only the `odd` messages.

   The consumer project is designed to continue listening for messages.

7. To exit the consumer project, press **Ctrl+C** for each consumer.

Last updated 2015-10-27 21:13:22 EDT