01

Tópicos e assinaturas duráveis

Transcrição

Começando deste ponto? Você pode fazer o DOWNLOAD (https://s3.amazonaws.com/caelum-online-public/jms/stages/jms-stage-cap5.zip) completo do projeto do capítulo anterior e continuar seus estudos a partir deste capítulo.

Introdução

Nos últimos capítulos estudamos como enviar uma mensagem JMS a partir de um código JAVA para o nosso MOM e como depois funciona a entrega da mensagem para um consumidor.

Vimos que dentro do MOM existe um destino que chamamos de Fila e ela depois entrega para um consumidor. Até percebemos que podemos ter mais de um consumidor, só que quando a FILA recebe a mensagem, essa mensagem vai apenas para um dos consumidores. Nunca temos consumidores recebendo a mesma mensagem.

Esse cenário faz todo sentido para sistemas que geram uma Nota Fiscal. Quando um pedido chega na nossa FILA e esse pedido vai para um consumidor que fica gerando Nota Fiscal, eles se torna ocupado, e nesse tempo outro consumidor está disponível para pegar outro pedido. Assim a gente melhora a disponibilidade do nosso sistema e até aumentamos o desempenho porque os dois conseguem fazer as tarefas em paralelo e até mesmo distribuídos em máquinas diferentes.

Modelo Publish-Subscriber

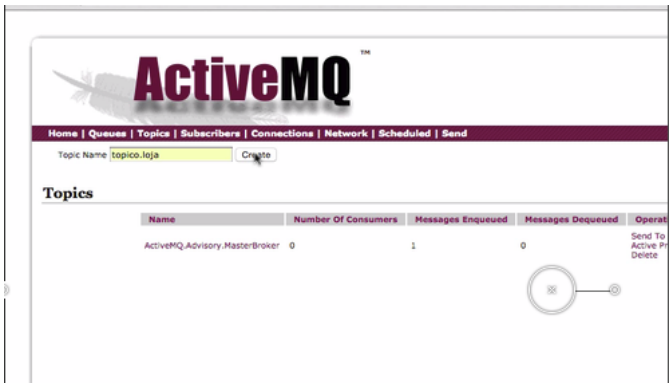
Em outros cenários poderíamos querer avisar ao sistema financeiro e também ao meu estoque que um produto foi comprado. Ou seja, nesse caso não basta apenas enviar apenas a mensagem para um consumidor. Essa mensagem deveria chegar nos dois consumidores(sistemas). Neste caso queremos **espalhar a mensagem para os nossos consumidores** e isso chamamos de Broadcast.

O Producer fica apenas enviando a mensagem para o MOM e ele fica encarregado de entregá-las para vários outros sistemas. Portanto a tarefa do MOM é avisar aos sistemas que estão interessados em escutar ou receber essa mensagem. Para o Producer isso não importa quantos sistemas serão notificados.

Nesse capítulo iremos focar nesse modelo de entrega, na hora de enviar teremos poucas alterações o que muda realmente é o modelo de entrega.

Criação do Topic e Producer

Vamos no console de administração do ActiveMQ e entraremos agora em *Topics* não mais em *Queues*. Criaremos um tópico chamado de `topico.loja` e depois de criado o tópico está disponível para nosso uso.



Agora vamos enviar uma mensagem JMS via Java, para isso criamos uma classe `TestaProdutorTopico` que será semelhante ao `TestaProdutor` que já tínhamos, inclusive para diferenciar o envio da Fila para o Tópico renomeie a classe `TestaProdutor` para `TestaProdutorFila`.

Como sempre Vamos precisar de `InitialContext`, `ConnectionFactory` e uma `Connection` inicializada e também de uma `Session` e por último de um destino, só que agora o destino não será a fila `financeiro`. Lembrando que o nome vem do `jndi.properties`. Vamos modificar esse arquivo para que ele tenha um tópico com o mesmo nome que configuramos no ActiveMQ. O formato de cadastro é:

```
topic.[jndiName] = [physicalName]
```

Onde o *physicalName* é o nome que cadastramos no ActiveMQ e o `jndiName` o nome para conseguirmos fazer *lookup*.

Sabendo disso, vamos alterar o nosso *properties* para:

```
java.naming.factory.initial = org.apache.activemq.jndi.ActiveMQInitialContextFactory

java.naming.provider.url = tcp://localhost:61616
queue.financeiro = fila.financeiro
```

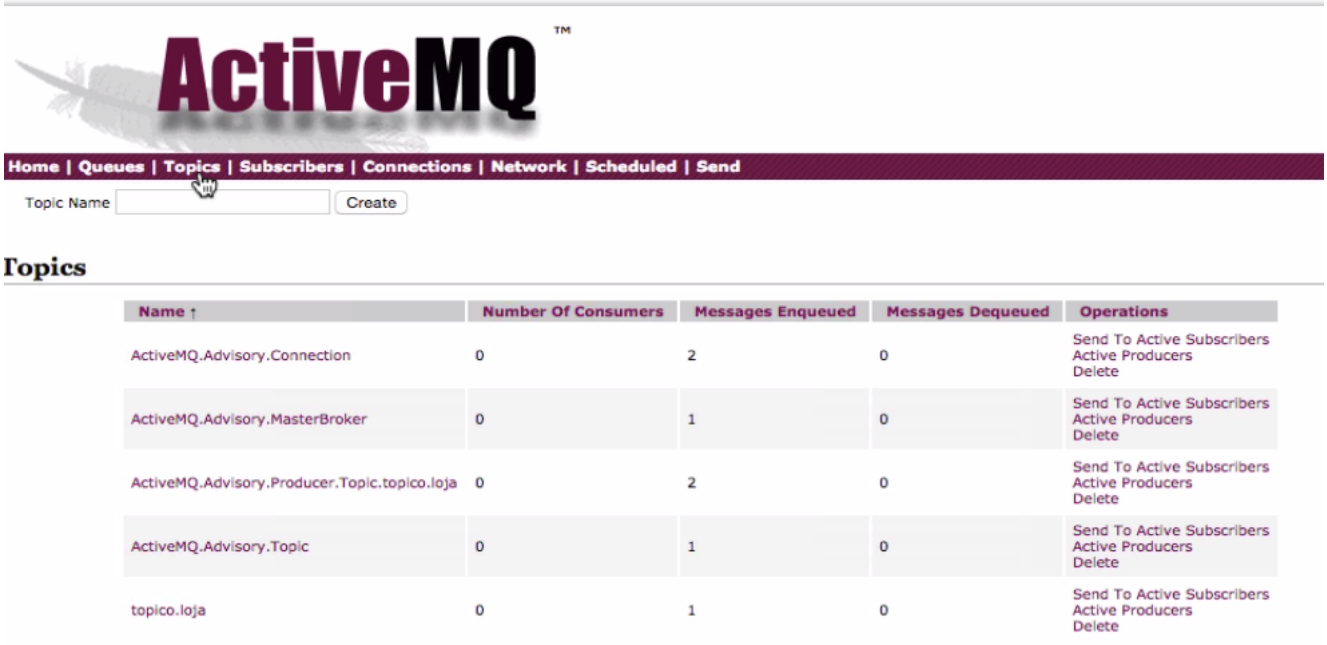
```
# novidade aqui
topic.loja = topico.loja
```

Vamos alterar o código do `TesteProdutorTopico` para que o tópico seja "loja" e que enviemos apenas uma mensagem.

```
// criação da fabrica, conexão e sessão
Destination loja = (Destination) context.lookup("loja");
```

Vamos rodar essa classe para enviar uma mensagem, não deve ocorrer nenhum erro. E sendo assim, iremos verificar via ActiveMQ no tópico `topico.loja` que uma mensagem foi recebida. Segue uma vez o código da classe `TesteProdutorTopico` :

```
public class TesteProdutorTopico {

    public static void main(String[] args) throws Exception {

        InitialContext context = new InitialContext();
        ConnectionFactory factory = (ConnectionFactory) context.lookup("ConnectionFactory");

        Connection connection = factory.createConnection();
        connection.start();
        Session session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);

        Destination topico = (Destination) context.lookup("loja");
        MessageProducer producer = session.createProducer(topico);

        Message message = session.createTextMessage("<pedido><id>222</id></pedido>");
        producer.send(message);

        session.close();
        connection.close();
        context.close();
    }
}
```



Perceba que o Produtor para Tópicos não é muito diferente do para Fila. O que vai **mudar é o modelo de entrega do MOM**.

Criaremos agora nosso `TesteConsumidorTopico`, que será de forma análoga ao já criado para testar o consumo da Fila. A única coisa que muda é o nome do *lookup* do destino:

```
public class TesteConsumidorTopico {

    @SuppressWarnings("resource")
    public static void main(String[] args) throws Exception {

        InitialContext context = new InitialContext();
        ConnectionFactory factory = (ConnectionFactory) context.lookup("ConnectionFactory");

        Connection connection = factory.createConnection();
        connection.start();
        Session session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);

        Destination topico = (Destination) context.lookup("loja");
        MessageConsumer consumer = session.createConsumer(topico );

        consumer.setMessageListener(new MessageListener() {

            @Override
            public void onMessage(Message message) {

                TextMessage textMessage = (TextMessage)message;

                try {
                    System.out.println(textMessage.getText());
                } catch (JMSException e) {
                    e.printStackTrace();
                }
            }

        });


        new Scanner(System.in).nextLine();

        session.close();
        connection.close();
        context.close();
    }
}
```

Obs: Renomeie o seu arquivo `TestaConsumidor` para `TestaConsumidorFila` para ficar mais claro no projeto.

Se executamos nesse instante o código aparentemente nenhuma mensagem foi recebida. Execute uma vez e confira. Mesmo resultado, nenhuma mensagem!

Assinaturas duráveis

O problema é que o tópico não sabe quantos consumidores iremos ter. Isso é diferente da Fila que define que tem de entregar a mensagem para um e tanto faz onde o consumidor esteja ela vai entregar para UM. Um tópico não sabe se vai ter um sistema interessado na mensagem ou vários outros. Em algum lugar o sistema financeiro, ou estoque, deverá avisar ao tópico que está interessado em receber as mensagens desse tópico. Se a gente não faz essa primeira identificação o tópico não saberá quem são os consumidores para entregar a mensagem. Logo precisamos dizer ao tópico.

Em no nosso primeiro cenário isso não ocorreu, ou seja quando estávamos offline a mensagem foi perdida. Imagine a aplicação de estoque offline por qualquer motivo e sem dar baixa de um produto quando ela voltasse ao seu estado de ativa, ou online, complicado não é? Portanto é muito importante identificarmos no tópico os consumidores interessados nas mensagens para que eles possam receber sem perdas.

Vamos adicionar algumas identificações pela classe `TestaConsumidorTopico` . A primeira é através da `Connection` passar um identificador, fazemos isso via método `setClientID` .

```
//codigo omitido
Connection connection = factory.createConnection();
connection.setClientId("estoque"); //identificar a conexão
//codigo omitido
```

Além de identificarmos uma conexão devemos identificar o nosso consumidor, porque com um conexão poderíamos ter vários consumidores. Fazemos essa segunda identificação através do método `createDurableSubscriber(...)` que recebe dois parâmetros: um tópico e um nome (a identificação).

```
//codigo omitido
MessageConsumer consumer = session.createDurableSubscriber(topico, "assinatura");
//codigo omitido
```

Execute uma vez o seu `TestaConsumidorTopico` para que o Tópico recebe esse cadastro do consumer. Deixe ele offline após isso. Segue uma vez o código completo desse consumidor:

```
public class TesteConsumidorTopico {

    @SuppressWarnings("resource")
    public static void main(String[] args) throws Exception {

        InitialContext context = new InitialContext();
        ConnectionFactory factory = (ConnectionFactory) context.lookup("ConnectionFactory");

        Connection connection = factory.createConnection();
        connection.setClientID("estoque");

        connection.start();
        Session session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);

        Topic topico = (Topic) context.lookup("loja");

        MessageConsumer consumer = session.createDurableSubscriber(topico, "assinatura");

        consumer.setMessageListener(new MessageListener() {

            @Override
            public void onMessage(Message message) {
```

```
                TextMessage textMessage = (TextMessage)message;

                try {
                    System.out.println(textMessage.getText());
                } catch (JMSException e) {
                    e.printStackTrace();
                }
            }

        });


        new Scanner(System.in).nextLine();

        session.close();
        connection.close();
        context.close();
    }
}
```

Entrega garantida

Para verificarmos se realmente a mensagem fica guardada para o nosso consumer estoque na assinatura execute o `TestaProdutorTopico` uma vez e depois o `TestaConsumidorTopico que estava offline quando a mensagem foi enviada.

Perceba que realmente conseguimos recebê-la!

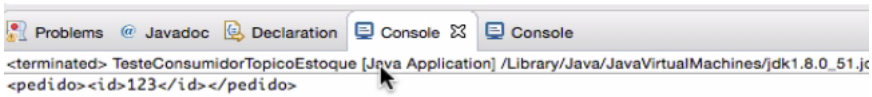
Testando o broadcast

Uma das funcionalidade que queríamos com o tópico era o de envio via Broadcast, ou seja vários consumidores deveriam receber a mensagem enviada. Para testar essa recebimento vamos criar outro consumidor `TestaConsumidorTopicoComercial`, renomeie o seu `TestaConsumidorTopico` para `TestaConsumidorTopicoEstoque` para identificarmos de maneira clara. A nossa classe nova terá o código similar mas com identificação diferente:

```
public class TesteConsumidorTopicoComercial {

    @SuppressWarnings("resource")
    public static void main(String[] args) throws Exception {

        InitialContext context = new InitialContext();
        ConnectionFactory factory = (ConnectionFactory) context.lookup("ConnectionFactory");

        Connection connection = factory.createConnection();
        connection.setClientID("comercial");

        connection.start();
        Session session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);

        Topic topico = (Topic) context.lookup("loja");

        MessageConsumer consumer = session.createDurableSubscriber(topico, "assinatura");
```

```
        consumer.setMessageListener(new MessageListener() {

            @Override
            public void onMessage(Message message) {

                TextMessage textMessage = (TextMessage)message;

                try {
                    System.out.println(textMessage.getText());
                } catch (JMSException e) {
                    e.printStackTrace();
                }
            }

        });


        new Scanner(System.in).nextLine();

        session.close();
        connection.close();
        context.close();
    }
}
```

Execute os dois consumidores, fazendo que eles representem um sistema online. Mantenha os dois console abertos. E após isso execute uma vez o seu `TestaProdutorTopico` . Veja que agora os dois consumidores receberam a mensagem:



Era exatamente isso que queríamos, vários sistemas recebendo a mesma mensagem. Diferentemente da Fila que apenas um sistema recebia. Além disso, vimos que o tópico guarda uma mensagem desde que algum consumidor tenha se identificado previamente e ele não tenha conseguido entregá-la.

Vamos aos exercícios, praticar um pouco?