

Consumindo mensagens com JMS

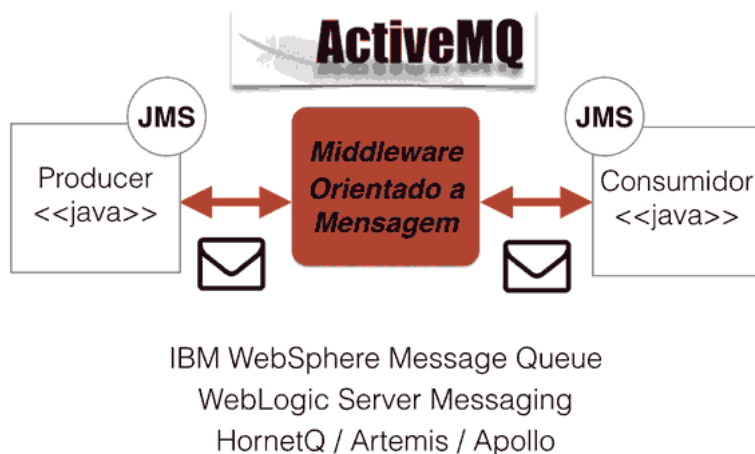
Transcrição

No primeiro capítulo, tivemos uma introdução de o que é um MOM, como ele desacopla o produtor do consumidor da mensagem, e que a entrega das mesmas é feita de forma assíncrona. Neste segundo capítulo do curso, queremos entrar nos detalhes do código, criar um consumidor e dar início a nossa aplicação.

JMS - A API de mensageria

Vimos um dos MOM's mais famosos do mundo Java, que é o ActiveMQ. Porém ele não é o único, e a própria ideia de MOM's é bem antiga. Existe uma série de servidores e middlewares no mercado, como: IBM WebSphere, WebLogic Server Messaging, HornetQ, Artemis, Apollo ...

Imagine agora, que para produzir ou consumir mensagens a partir do código java, devemos aprender sempre a API específica desses MOM's. Porém, no final, a gente sempre cria mensagem e recebe mensagem. Para facilitar o trabalho do desenvolvedor, foi criado um padrão JavaEE em cima dessa ideia de mensageria, que é o **JMS (Java Messaging Service)**.



Continuaremos usando o ActiveMQ, porém o código que estaremos utilizando, serve também para um IBM WebSphere ou para um WebLogic Server Messaging. Muda muito pouco o código Java necessário para consumir ou escrever mensagens, e por debaixo dos panos, continuamos a usar o ActiveMQ.

Criação do projeto

Agora, vamos rapidamente subir o ActiveMQ, do mesmo jeito que foi feito na aula anterior (com o parâmetro 'console' na linha de comando) e acessar com "User Name: admin / Password: admin".

Para começar com os códigos, precisamos abrir também o eclipse JavaEE, e criar um projeto Java padrão. Usaremos o Java 1.8 como JRE e o nome será *jms*. Precisamos do arquivo do ActiveMQ no nosso projeto. Para isso, clique com o botão direito no projeto, **New -> folder**. Chame essa pasta de **lib** e então, arraste o arquivo do ActiveMQ **activemq-all-5.12.0.jar** para dentro dela. Após isso, dentro do eclipse, clique com o botão direito no JAR copiado e **Build Path -> Add to Build Path**.

Criando uma Conexão com o ActiveMQ

Com a importação do .jar feita, podemos começar com o nosso código. Crie uma classe `TesteConsumidor` no pacote `br.com.caelum.jms`. Usaremos o tempo todo importações da biblioteca `javax.jms`.

Criaremos uma conexão. Porém, de onde vem a nossa `ConnectionFactory`? O MOM vai te fornecer! A ideia é que quando o MOM é inicializado, que ele já disponibilize essa conexão dentro de um registro. Com isso, precisamos apenas pegar essa conexão dentro de um registro, e esse é o JNDI. O nome utilizado no `lookup` é apresentado na documentação do MOM.

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

        InitialContext context = new InitialContext();

        //imports do package javax.jms
        ConnectionFactory factory = (ConnectionFactory) context.lookup("ConnectionFactory");
        Connection connection = factory.createConnection();
        connection.start();

        new Scanner(System.in).nextLine(); //parar o programa para testar a conexao

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

Além disso, devemos criar um arquivo `jndi.properties` na pasta `src` que copiamos do site do ActiveMQ:

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

# use the following property to configure the default connector
java.naming.provider.url = vm://localhost

# use the following property to specify the JNDI name the connection factory
# should appear as.
#connectionFactoryNames = connectionFactory, queueConnectionFactory, topicConnectionFactory

# register some queues in JNDI using the form
# queue.[jndiName] = [physicalName]
queue.MyQueue = example.MyQueue

# register some topics in JNDI using the form
# topic.[jndiName] = [physicalName]
topic.MyTopic = example.MyTopic
```

Subindo o ActiveMQ em memória

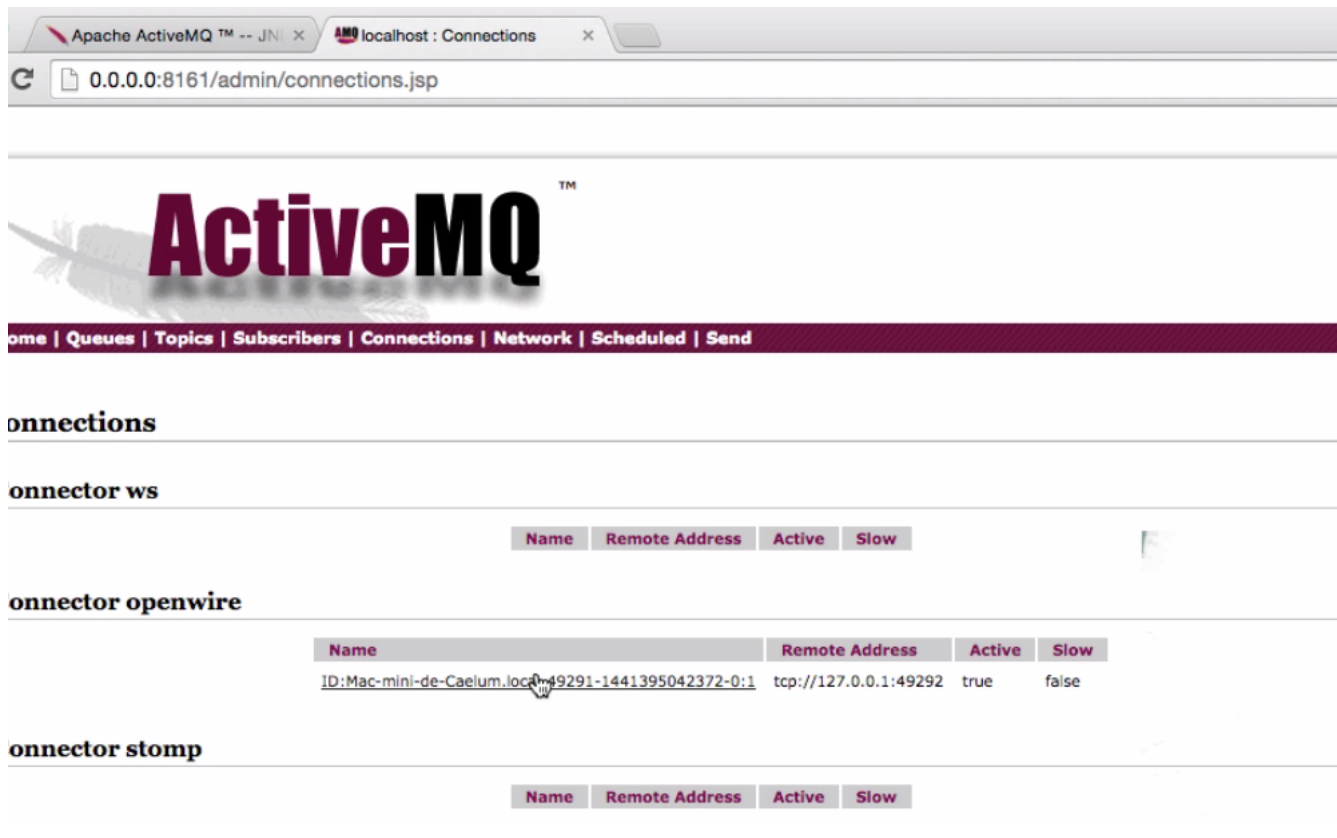
Ao rodar o código aparentemente foi estabelecida uma conexão com ActiveMQ como previsto. Vamos verificar isso no console de administração do ActiveMQ. Para nossa surpresa não aparece nenhuma conexão. O que aconteceu? A resposta está no arquivo `jndi.properties`. Repare a linha:

```
vm://localhost
```

O `vm` indica que o ActiveMQ subiu em memória. O ActiveMQ não precisa rodar em sua JVM dedicada. O nosso código Java não só estabeleceu uma conexão como também criou uma nova instancia do ActiveMQ! Mas com certeza isso não foi a nossa intenção pois já subimos o ActiveMQ antes separadamente. Vamos mudar isso e alterar a linha no `jndi.properties` :

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

Rodando nossa classe de teste, vemos que não imprime nada no console, mas ao entrarmos na console de administração do (<http://localhost:8161/admin/connections.jsp> (<http://localhost:8161/admin/connections.jsp>)) ActiveMQ -> Connections , observamos que há uma nova conexão com o ID da sua máquina local.



Name	Remote Address	Active	Slow
ID:Mac-mini-de-Caelum.local-49291-1441395042372-0:1	tcp://127.0.0.1:49292	true	false

Criando um MessageConsumer

Agora queremos criar um Consumer, que irá receber nossas mensagens.

```
public class TesteConsumidor {  
    //cria context, factory, connection  
  
    Session session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);  
    Destination fila = (Destination) context.lookup("financeiro");  
    MessageConsumer consumer = session.createConsumer(fila);  
  
    Message message = consumer.receive();  
    System.out.println("Recebendo msg: " + message);  
  
    session.close();  
    //fecha conexões  
}
```

Já tínhamos nossa fila de consumo, que foi criada na primeira aula. Sendo assim, o ActiveMQ disponibiliza no `jndi.properties`, e acessamos pelo `lookup`. Porém, precisamos configurar o arquivo de *properties* para ele saber o que procurar.

Dentro do JNDI, temos a linha:

```
# queue.[jndiName] = [physicalName]
Queue.MyQueue = example.MyQueue
```

Na direita, fica o nome da fila que foi criada no ActiveMQ, que no nosso caso é `fila.financieiro`. Na esquerda, fica o nome na qual usaremos para referenciar essa fila no código Java. Substituindo:

```
queue.financieiro = fila.financieiro
```

Testando o consumidor

Assim, já podemos acessar o nosso consumer do ActiveMQ pelo nosso código Java. Criamos também uma `Message` que irá receber uma mensagem do `consumer` e printá-la no nosso console. Rodamos a classe para criar o nosso `MessageConsumer`:

ActiveMQ

Home | Queues | Topics | Subscribers | Connections | Network | Scheduled | Send

Active Consumers for fila.financieiro

Client ID	Connection ID	SessionId	Selector	Enqueues	Dequeues	Dispatched	Dispatched Queue	Prefetch Max pending	Exclusive Retroactive
ID:Mac-mini-de-Caelum.local-49303-1441395483674-0:1	ID:Mac-mini-de-Caelum.local-49303-1441395483674-1:1	1		0	0	0	0	10000	false

Agora para testar, enviamos uma mensagem através da console de administração.

ActiveMQ

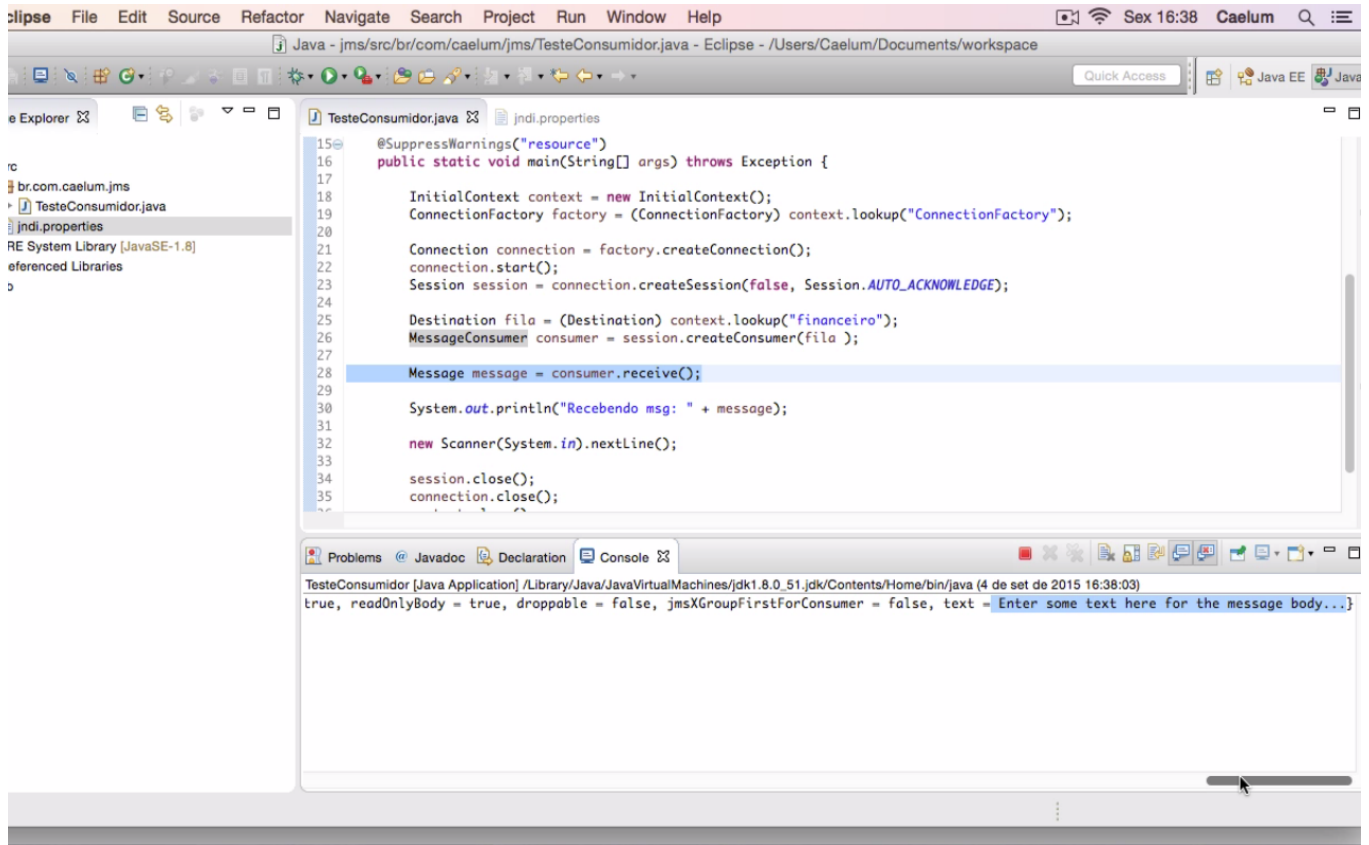
Home | Queues | Topics | Subscribers | Connections | Network | Scheduled | Send

Queue Name: Create

Queues

Name	Number Of Pending Messages	Number Of Consumers	Messages Enqueued	Messages Dequeued	Views	Operations
fila.financieiro	0	1	0	0	Browse Active Consumers Active Producers atom rss	Send To Purge Delete

E no console...



The screenshot shows the Eclipse IDE interface. The top menu bar includes File, Edit, Source, Refactor, Navigate, Search, Project, Run, Window, and Help. The toolbar contains various icons for file operations and development tools. The left sidebar shows the Project Explorer with the following structure:

- br.com.caelum.jms
 - TesteConsumidor.java
 - jndi.properties
- RE System Library [JavaSE-1.8]
- Referenced Libraries

The main editor displays the code for `TesteConsumidor.java`. The code is as follows:

```
15 @SuppressWarnings("resource")
16 public static void main(String[] args) throws Exception {
17
18     InitialContext context = new InitialContext();
19     ConnectionFactory factory = (ConnectionFactory) context.lookup("ConnectionFactory");
20
21     Connection connection = factory.createConnection();
22     connection.start();
23     Session session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);
24
25     Destination fila = (Destination) context.lookup("financeiro");
26     MessageConsumer consumer = session.createConsumer(fila);
27
28     Message message = consumer.receive();
29
30     System.out.println("Recebendo msg: " + message);
31
32     new Scanner(System.in).nextLine();
33
34     session.close();
35     connection.close();
36 }
```

The bottom of the IDE shows the Console view. It displays the output of the application:

```
TesteConsumidor [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_51.jdk/Contents/Home/bin/java (4 de set de 2015 16:38:03)
true, readOnlyBody = true, droppable = false, jmsXGroupFirstForConsumer = false, text = Enter some text here for the message body...}
```

Queremos agora que nossa aplicação fique online, que o `MessageConsumer` consiga receber essas mensagens o tempo todo e não só uma mensagem. Essa configuração, iremos aprender mais a frente. Agora é a hora dos exercícios.