

Introdução ao Mensageria com ActiveMQ

Transcrição

Bem-vindo ao treinamento de mensageria, JMS e ActiveMQ.

Downloads desse capítulo

- [Download do ActiveMQ \(<https://s3.amazonaws.com/caelum-online-public/jms/apache-activemq-5.12.0-bin.zip>\)](https://s3.amazonaws.com/caelum-online-public/jms/apache-activemq-5.12.0-bin.zip)
- [Download do JAR \(<https://s3.amazonaws.com/caelum-online-public/jms/aula-jms.jar>\)](https://s3.amazonaws.com/caelum-online-public/jms/aula-jms.jar) (para testar a primeira fila)

Loja virtual e geração de nota fiscal

Bom, temos uma loja virtual que já está no ar há algum tempo. Produtos são colocados em um carrinho de compra e no final o pedido é finalizado. Quando a compra é finalizada, enviamos os dados da compra para outro sistema responsável pela geração da nota. Até aí tudo bem.



Problemas de comunicação

Sabemos que eventos promocionais como *black friday* aumentam o número de acesso vertiginosamente. Queremos que nossa aplicação funcione, mas o problema é que o outro sistema pode falhar, inclusive pode haver algum problema de comunicação na rede e também ele pode não aguentar processar um grande número de informações enviadas. Não podemos perder um pedido só porque o sistema de geração de notas não funciona.



Para evitarmos os problemas que foram citados, precisamos desacoplar os dois sistemas através de um bloco arquitetural que ficará entre ambos. É uma espécie de servidor entre dois sistemas. Esse servidor é chamado de **middleware** e com ele o a loja virtual não conversa mais diretamente com o sistema de geração de notas fiscais. O objetivo do **middleware** é desacoplar os dois lados, isto é, as duas aplicações.



Agora, a loja enviará o pedido empacotando-o dentro de um envelope para o middleware que guardará a mensagem recebida e algum momento depois a entregará para o sistema de nota fiscal (**assíncrono**). Esse processo é orientado à mensagens, por isso que é chamado de MOM (*Message Oriented Middleware*).

Então, dentro do que vimos, percebemos o desacoplamento arquitetural entre a loja e o sistema de notas (um sistema deixar de conhecer o outro) e todo comunicação é via mensagem. A mensagem é recebida pelo middleware e algum momento posterior, é entregue para o sistema destinatário. Justamente por ser “um momento posterior” tudo ocorre assincronamente, pois não sabemos quando a mensagem será entregue.

ActiveMQ, o MOM da Apache Foundation

Para implementarmos essa solução, precisamos de um middleware e o mais famoso no mundo Java se chama ActiveMQ da Apache Foundation. Usaremos o ActiveMQ ao longo deste treinamento, um MOM da Apache.

Primeiro passo é instalar o ActiveMQ.

Instalando o ActiveMQ

Baixamos o ActiveMQ 5.12.x (ou mais recente) em <http://activemq.apache.org/download.html>
<http://activemq.apache.org/download.html>.

Observação: No Windows é preciso executar o script `InstallService.bat` da pasta `win32` ou `win64` dependendo da arquitetura do computador.

Depois de baixado, só precisamos descompactá-lo. Como todo servidor, precisamos rodá-lo e fazemos através do terminal entrando na pasta `apache-activemq-5.12.2/bin`. Lá uma série de scripts e precisamos rodar aquele que condiz com o sistema operacional que estamos usando. Por exemplo, no OSX usamos no terminal o comando

```
sh activemq
```

Só que não foi dessa vez. A razão é que precisamos passar parâmetros para esse script. Usarmos a opção console:

```
sh activemq console
```

Agora sim. Além de subir nosso servidor, ele nos apresenta uma série de logs. O terminal também indica o endereço do nosso servidor: <http://localhost:8161> (<http://localhost:8161/admin>)

Visualizamos a página principal do ActiveMQ. Há um link (<http://localhost:8161/admin> (<http://localhost:8161/admin>)) para o console de administrar. Ao ser clicado, precisamos um usuário e uma senha. O login e senha padrão são `admin`.

A primeira fila

É bem simples, porém há informações que já são úteis como o nome da máquina que está rodando, versão do servidor, etc. Podemos até enviar já mensagens através do link `send`. Há também duas opções, `queues` e `topics`. Vamos focar no primeiro, pois nossa loja quer enviar uma mensagem com o pedido empacotado a esse MOM. O MOM poderia ter vários outros clientes, isto é, aplicações. Não simplesmente entregamos a mensagem para o MOM, nós indicamos também qual o destino desta mensagem. A mensagem do pedido enviado para o MOM fica cadastro dentro de uma fila (queue) para o ActiveMQ organizar, inclusive poderíamos ter várias filas. O topic, que não vamos utilizar agora, é um outro destino.

The screenshot shows the ActiveMQ management console. At the top, there's a navigation bar with links for Home, Queues, Topics, Subscribers, and Connections. Below the navigation, a search bar has 'fila.financeiro' typed into it, and a 'Create' button is visible. The main area displays a table of queues. The first row, representing 'fila.financeiro', has columns showing 0 pending messages, 0 consumers, 0 enqueued messages, and 0 dequeued messages. To the right of this row is a 'Views' section with links for Browse Active Consumers, Active Producers, atom, and RSS. Below these are buttons for Send To, Purge, and Delete.

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

Clicando em queue, é perguntada o nome da fila que chamaremos de **financeiro**. Podemos acessar a fila criada e seus consumidores (quem gostaria de receber) e os *active producers*, quem está enviando. Há uma opção de enviar uma mensagem para a fila e é exatamente o que faremos.

Queues

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

Enviando a mensagem

Na tela de envio de mensagem, precisamos informar qual é o destino, mas não apenas isso. Há um monte de cabeçalhos da mensagem que podemos preencher quando necessário.

Send a JMS Message

Message Header

Destination	fila.financeiro	Queue or Topic	Queue
Correlation ID		Persistent Delivery	<input type="checkbox"/>
Reply To		Priority	
Type		Time to live	
Message Group		Message Group Sequence Number	
delay(ms)		Time(ms) to wait before scheduling again	
Number of repeats		Use a CRON string for scheduling	
Number of messages to send	1	Header to store the counter	JMSXMessageCounter

[Send](#) [Redefinir](#)

Message body

Enter some text here for the message body...

Na caixa “Message Body” vamos escrever: *Oi mundo mensageria!*

Uma mensagem foi enviada, ela está *enqueued*. Isso significa que a fila recebeu e guardou, salva. Que tal enviarmos mais uma mensagem? Dessa vez será “Oi! Mensageria segunda mensagem”. As mensagens ficam *enqueued*, mas nenhum foi entregue pois ainda não temos um consumidor.

Simulando a entrega

Vamos simular essa entrega para nossa aplicação de nota fiscal que tem interesse em receber essas mensagens. Para isso, preparamos um jar com código Java que consumirá a mensagem, mas ele não possui todo o protocolo possível do ActiveMQ,

é por isso que da pasta do ActiveMQ que baixamos, vamos mover o `activemq-all-5.12.0.jar`. Resumindo: nosso `aula-jms.jar` depende de `activemq-all-5.12.0.jar`.

Sem fechar o terminal do ActiveMQ que está rodando, vamos abrir um novo terminal e executar:

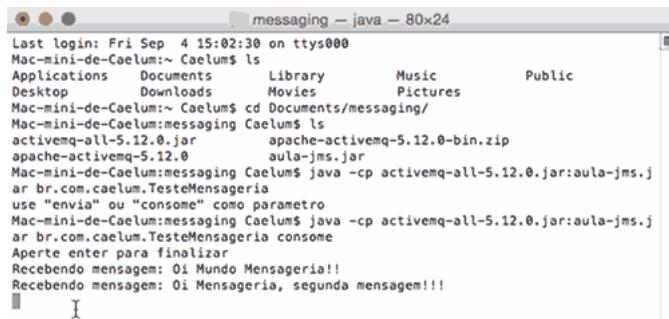
No Linux e Mac:

```
java -cp activemq-all-5.12.0.jar:aula-jms.jar br.com.caelum.TesteMensageria consome
```

No Windows:

```
java -cp activemq-all-5.12.0.jar;aula-jms.jar br.com.caelum.TesteMensageria consome
```

Nosso programa se conectou ao nosso MOM e recebeu as mensagens que cadastramos.



```
messaging - java - 80x24
Last login: Fri Sep  4 15:02:30 on ttys000
Mac-mini-de-Caelum:~ Caelum$ ls
Applications  Documents  Library  Music  Public
Desktop      Downloads   Movies    Pictures
Mac-mini-de-Caelum:~ Caelum$ cd Documents/messaging/
Mac-mini-de-Caelum:messaging Caelum$ ls
activemq-all-5.12.0.jar          apache-activemq-5.12.0-bin.zip
apache-activemq-5.12.0           aula-jms.jar
Mac-mini-de-Caelum:messaging Caelum$ java -cp activemq-all-5.12.0.jar:aula-jms.j
ar br.com.caelum.TesteMensageria
use "envia" ou "consome" como parametro
Mac-mini-de-Caelum:messaging Caelum$ java -cp activemq-all-5.12.0.jar:aula-jms.j
ar br.com.caelum.TesteMensageria consome
Aperte enter para finalizar
Recebendo mensagem: Ol Mundo Mensageria!
Recebendo mensagem: Ol Mensageria, segunda mensagem!!!
```

Podemos enviar mensagens na linha de comando também usando o parâmetro `envia` seguido com a quantidade de números :

No Linux e Mac:

```
java -cp activemq-all-5.12.0.jar:aula-jms.jar br.com.caelum.TesteMensageria envia 10
```

No Windows:

```
java -cp activemq-all-5.12.0.jar;aula-jms.jar br.com.caelum.TesteMensageria envia 10
```

Neste exemplo, estamos enviando 10 mensagens. Agora em nosso MOM, temos 10 mensagens. Ao testar a consumidor (recebimento das mensagens) aparece no console:

```
messaging - java - 80x24
use "envia" ou "consome" como parametro
Mac-mini-de-Caelum:messaging Caelum$ java -cp activemq-all-5.12.0.jar:aula-jms.j
ar br.com.caelum.TesteMensageria consome
Aperte enter para finalizar
Recebendo mensagem: Oi Mundo Mensageria!!
Recebendo mensagem: Oi Mensageria, segunda mensagem!!!
Mac-mini-de-Caelum:messaging Caelum$ java -cp activemq-all-5.12.0.jar:aula-jms.j
ar br.com.caelum.TesteMensageria envia 10
Enviando 10 mensagens(ns)
Mac-mini-de-Caelum:messaging Caelum$ java -cp activemq-all-5.12.0.jar:aula-jms.j
ar br.com.caelum.TesteMensageria consome
Aperte enter para finalizar
Recebendo mensagem: Mensagem 1 (2015-09-04T15:12:11.125)
Recebendo mensagem: Mensagem 2 (2015-09-04T15:12:11.167)
Recebendo mensagem: Mensagem 3 (2015-09-04T15:12:11.169)
Recebendo mensagem: Mensagem 4 (2015-09-04T15:12:11.171)
Recebendo mensagem: Mensagem 5 (2015-09-04T15:12:11.173)
Recebendo mensagem: Mensagem 6 (2015-09-04T15:12:11.175)
Recebendo mensagem: Mensagem 7 (2015-09-04T15:12:11.176)
Recebendo mensagem: Mensagem 8 (2015-09-04T15:12:11.178)
Recebendo mensagem: Mensagem 9 (2015-09-04T15:12:11.179)
Recebendo mensagem: Mensagem 10 (2015-09-04T15:12:11.181)
```

Agora vamos praticar com os exercícios do capítulo.