01

Selectores e propriedades da Mensagem JMS

Transcrição

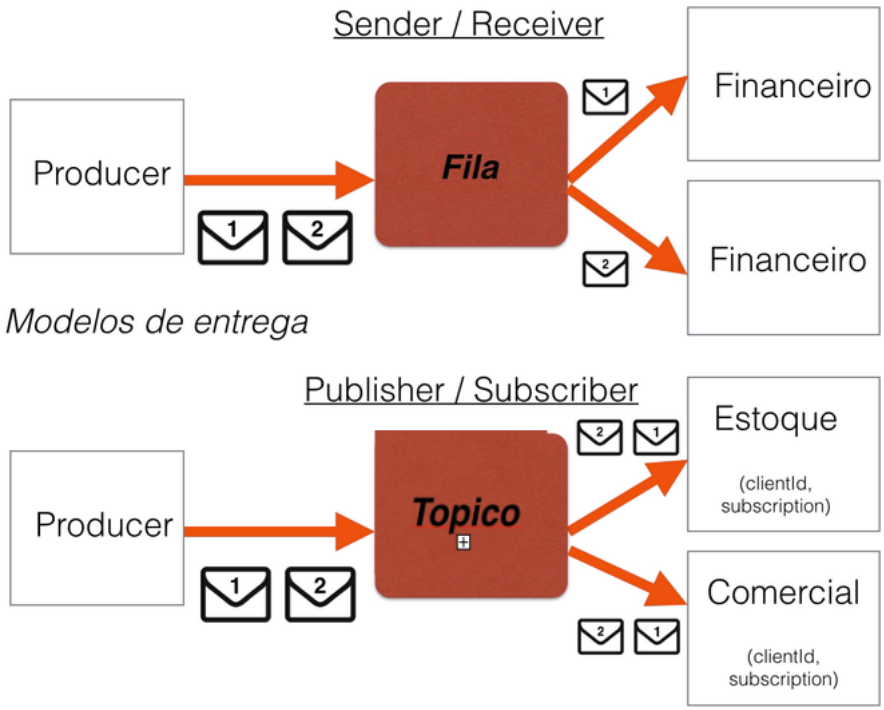
Começando deste ponto? Você pode fazer o DOWNLOAD (https://s3.amazonaws.com/caelum-online-public/jms/stages/jms-stage-cap6.zip) completo do projeto do capítulo anterior e continuar seus estudos a partir deste capítulo.

No capítulo anterior, vimos outro modelo de entrega, o topico. Aprendemos que não há diferença da fila no que diz respeito à criação e envio de mensagens. **A diferença mora na entrega da mensagem**.

Vimos que nossos producers enviam suas mensagem para a fila e que ela se encarrega de balancear sua entrega, ou seja, apenas um consumer receberá a mensagem. Porém, há situações em que faz sentido avisar mais de um sistema que a mensagem foi recebida através de um broadcast. Neste caso não usamos uma fila, mas um topico.

Aprendemos também que o topico por padrão não garante a entrega, por exemplo, se não ouver nenhum consumidor online, perderemos a mensagem enviada.

Um outro ponto importante, quando estamos falando de **fila**, é que não é incomum ouvirmos o termo sender para nossos producers e receiver para nossos consumers, caracterizando o modelo **sender/receiver**. Já no **topico**, os producers são chamados de publisher e os consumers de subscriber. Nesta aula, continuaremos a focar no modelo **publish/subscriber**.



Consumo condicional de mensagens

Vamos voltar para nosso estoque, organizando a baixa de produtos. Um detalhe é que muitos produtos de uma loja virtual não precisam de baixa no estoque, como no caso dos livros digitais. Dentro desse contexto, precisamos definir algum critério na mensagem para que o estoque saiba qual delas é de seu interesse, em outras palavras: o consumer quer consumir todas as mensagem, mas pode ter algum critério de rejeição de mensagem.

Message Selectors

Vamos criar uma cópia do arquivo `TesteComsumidorTopicoEstoque.java`, mas com o nome `TesteComsumidorTopicoEstoqueSelector.java`. O sufixo Selector não é por acaso, pois no mundo JMS chamamos os critério de consumo das mensagens de **Selector**.

O código é muito parecido com o que já vimos, por isso fizemos uma cópia. Vamos alterar apenas a linha

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

O método `createDurableSubscriber(..)` é sobrecarregado e um método que nos interessa é aquele que recebe uma `string` de selector e um booleano. Vamos entender um pouquinho mais desse *Message Selector* a partir do Javadoc.

https://docs.oracle.com/javaee/6/api/javax/jms/Message.html (https://docs.oracle.com/javaee/6/api/javax/jms/Message.html)

A interface principal do JMS que se chama `Message` é bem esclarecedora. Nela, há uma documentação sobre os *Message Selector*. Duas coisas merecem atenção.

Considerações sobre o Message Selector

Não podemos usar messages selectors para buscar algum valor dentro do corpo da mensagem, ou seja, nosso XML, porém ele permite buscar dentro do cabeçalho e propriedades da mensagem. A consequência disso para nós é que precisaremos de alguma forma alterar o cabeçalho e propriedades da mensagem para que o message selector execute seu papel.

O segundo ponto é que os messages selectors possuem uma sintaxe parecida com SQL para busca de informações.

Vamos alterar nosso código, indicando que se houver a propriedade ebook com valor false nosso estoque está interessando na mensagem. É uma regra simples, poderia ser outra:

```
MessageConsumer consumer = session.createDurableSubscriber(topico, "assinatura", "ebook= false", fa
```

Alterando propriedades da mensagem JMS

É claro, nossas mensagens precisam ser enviadas com essa informação. Vamos alterar a classe `TesteProdutoTopico`. Uma mensagem (`Message`) tem uma série de métodos e estamos interessados no `setBooleanProperty` que recebe como primeiro parâmetro o nome da propriedade e o segundo seu valor.

```
message.setBooleanProperty("ebook", false);
```

Antes de rodarmos, em nossa classe `TesteConsumidorTopicoEstoqueSelector`, vamos alterar a assinatura para “assinatura-selector” para deixar ainda mais claro.

Testando o selector

Rodando nosso teste, lá no ActiveMQ, vamos que nosso "assinatura-seletor" já está registrado. Inclusive ele mostra o seletor “ebook=false” para sabermos qual critério ele utilizará.

Agora, vamos rodar nosso `TesteProdutorTopic`. Tudo funciona, passou no critério do nosso seletor, mas também vamos testar o contrário, vamos enviar a mensagem com o seletor "ebook=true". Não deve aparecer nenhum mensagem de log do nosso estoque, pois ele deve ignorar essa mensagem.

Excelente, aprendemos a criar um critério para indicar o interesse ou não de uma mensagem em nosso tópico. Agora, vamos fazer o seguinte: se a propriedade não existir, isto é, "ebook", o estoque deverá sempre receber. Vamos testar. Bem, não recebeu a mensagem, então, por padrão, se não há a propriedade na mensagem ela é ignorada. Porém, queremos que ele processe, caso ela não exista. Para isso, na string do critério do selector, vamos adicionar "ebook is null OR ebook=false". Um teste rapidamente mostra que o resultado é o esperado.

Mensagens noLocal

Agora, vamos voltar para aquele parâmetro, logo após do nosso selector que é false. Ela se chama "noLocal" e diz respeito à conexão. Uma conexão pode servir para criar várias assinaturas, mas também produtores. Poderíamos usar a mesma conexão para também enviar a mensagem. A questão é se estamos interessados em mensagens enviadas pela nossa própria conexão. Com `false` indicamos que não queremos receber essas mensagens. Para nós não faz diferença, por que usamos sempre uma nova conexão para enviar ou receber uma mensagem.