

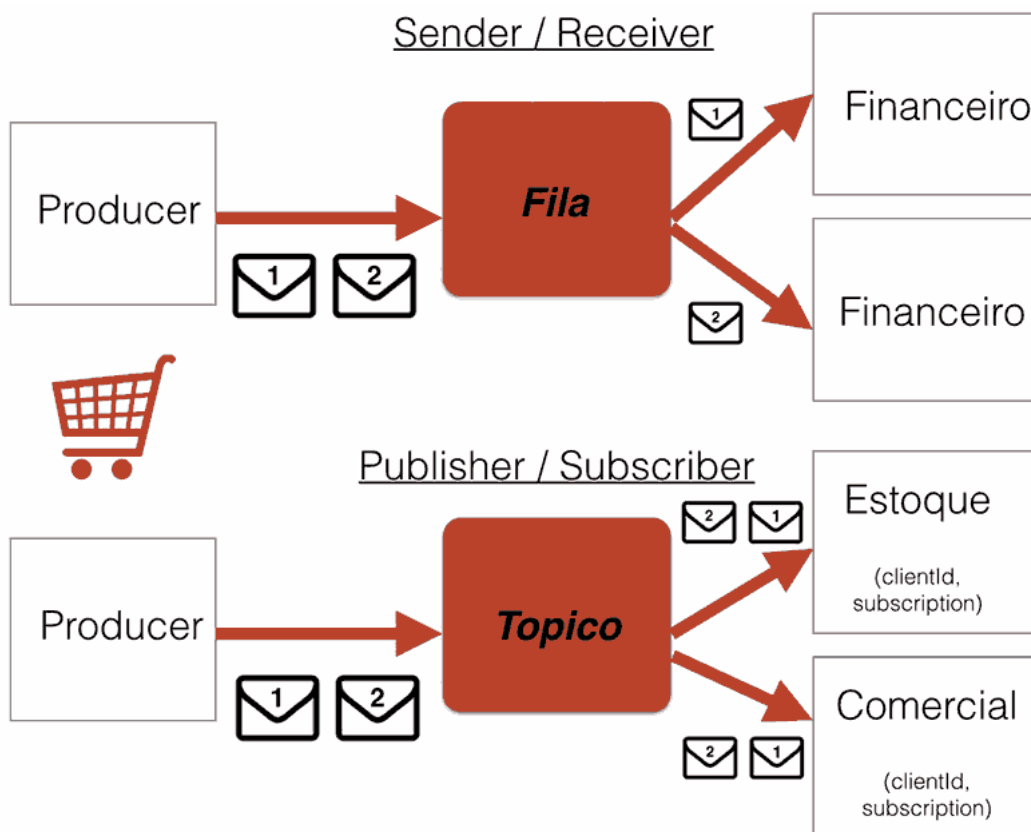
Prioridade e Tempo de vida da mensagem

Transcrição

A motivação inicial do uso do JMS era a nossa loja virtual onde é criado um pedido a partir de uma compra. Com o pedido em mãos, precisávamos nos integrar com outros sistemas e utilizamos um MOM para desacoplar nossos sistemas, sendo assim, a geração de um e-book e o processamento no sistema financeiro podem acontecer assíncronamente.



Dentro desse contexto, vimos dois modelos de entrega: a fila e o tópico. Na hora que estamos enviando a mensagem para o MOM, não há diferença entre fila e tópico, porém quando nosso MOM entrega nossas mensagens a coisa muda de figura. Enquanto a fila envia exatamente para apenas um, o tópico espalha a mensagem. Neste capítulo, vamos mudar um pouco o nosso caso de uso.



Um novo problema: vários logs

Arquivos de logs são comuns em sistema e é muito provável que todos aqui já tenham visto um. Alguns fornecem informações sobre o sistema outros o que está sendo executado e por aí vai. Cada aplicação de cada sistema gera seu arquivo de log e se quisermos saber o que está acontecendo com cada um deles precisaríamos olhar diversos arquivos. A ideia agora é centrar o acesso aos arquivos de log em um único lugar.

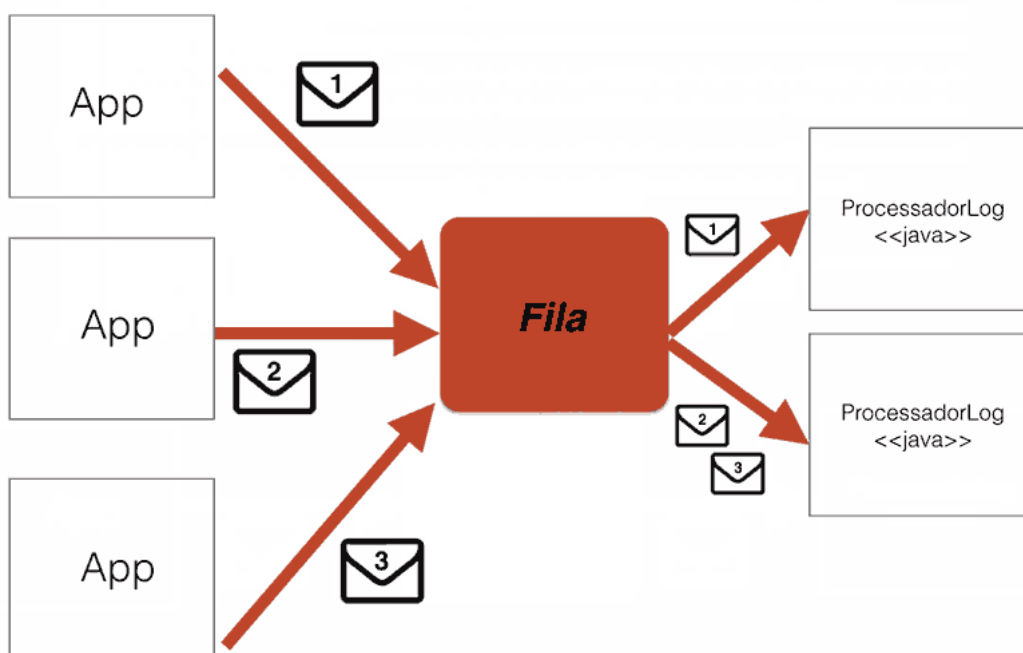
```

bin — java — 136x24
INFO | Recovery replayed 7 operations from the journal in 0.038 seconds.
INFO | Apache ActiveMQ 5.12.0 (localhost, ID:Mac-mini-de-Caelum.local-49707-1443131774307-0:1) is starting
INFO | Listening for connections at: tcp://Mac-mini-de-Caelum.local:61616?maximumConnections=1000&wireFormat.maxFram
INFO | Connector openwire started
INFO | Listening for connections at: amqp://Mac-mini-de-Caelum.local:5672?maximumConnections=1000&wireFormat.maxFram
INFO | Connector amqp started
INFO | Listening for connections at: stomp://Mac-mini-de-Caelum.local:61613?maximumConnections=1000&wireFormat.maxFram
INFO | Connector stomp started
INFO | Listening for connections at: mqtt://Mac-mini-de-Caelum.local:1883?maximumConnections=1000&wireFormat.maxFram
INFO | Connector mqtt started
INFO | {}
INFO | Listening for connections at ws://Mac-mini-de-Caelum.local:61614?maximumConnections=1000&wireFormat.maxFrames5
INFO | Connector ws started
INFO | Apache ActiveMQ 5.12.0 (localhost, ID:Mac-mini-de-Caelum.local-49707-1443131774307-0:1) started
INFO | For help or more information please see: http://activemq.apache.org
INFO | ActiveMQ WebConsole available at http://0.0.0.0:8161/
INFO | ActiveMQ Jolokia REST API available at http://0.0.0.0:8161/api/jolokia/
INFO | Initializing Spring FrameworkServlet 'dispatcher'
INFO | jolokia-agent: No access restrictor found at classpath:/jolokia-access.xml, access to all MBeans is allowed
INFO | jolokia-agent: Cannot join multicast group on NIF bwdl0: Can't assign requested address
INFO | jolokia-agent: Cannot join multicast group on NIF bwdl0: Can't assign requested address
WARN | Transport Connection to: tcp://192.168.0.208:49710 failed: java.io.EOFException
WARN | Transport Connection to: tcp://192.168.0.208:49775 failed: java.io.EOFException
  
```



Em nossa infraestrutura teremos uma aplicação com a responsabilidade de analisar os logs. Todas as aplicações enviarão uma mensagem JMS para uma fila log, deixaremos de trabalhar com pedidos aqui. Os processadores agora poderão analisar os logs, tudo centralizado.

Sender / Receiver



O problema é quando os processadores estão offline. A fila continuará a receber as mensagens de log de todas as aplicações gerando um volume muito grande de informações. Precisamos pensar como a fila lidará com esse grande volume de

informações.

Definindo o que é importante

Nossos log's possuem informações mais importantes e menos importantes. Mensagens menos importantes são as `INFO`. Vamos começar por elas.

Podemos informar para nosso ActiveMQ que mensagens de erro `INFO` podem ter um tratamento diferente.

Vamos alterar nosso `TesteProdutoFila.java`. No método `message.send`, vamos passar mais algumas parâmetros:

```
// TesteProdutorFila
// código anterior omitido
Message message = session.createTextMessage("INFO | ....");
producer.send(message, DeliveryMode.NON_PERSISTENT);
// código posterior omitido
```

Persistente vs não persistente

O padrão é `DeliveryMode.PERSISTENT` fazendo o ActiveMQ guardar a mensagem em um banco de dados. Caso ele tenha sido reiniciado, não haverá problema, pois ele recuperará a mensagem que foi salva. Mas no caso dos nossos log's do tipo `INFO`, não precisamos dessa persistência. Se o ActiveMQ reiniciar e perdermos essas informações, não será de grande importância.

Prioridade da mensagem

O próximo parâmetro é a prioridade. Podemos usar um número inteiro de 0 ate 9, na menor prioridade para a maior. Quando temos várias mensagens na fila, o ActiveMQ consegue reorganizá-la em termos de prioridade.

```
// código anterior omitido
producer.send(message, DeliveryMode.NON_PERSISTENT, 3);
// código posterior omitido
```

E o último parâmetro é o tempo de vida. Se ninguém consegue consumir a mensagem, podemos deixá-la viva por algum tempo e se esse tempo expirar, a mensagem é descartada. Vamos usar 5 segundos, mas em milissegundos:

```
// código anterior omitido
producer.send(message, DeliveryMode.NON_PERSISTENT, 3, 5000);
// código posterior omitido
```

Habilitando prioridades em activemq.xml

Podemos saber ainda mais sobre a parte de prioridade na própria documentação online do ActiveMQ (<http://activemq.apache.org/how-can-i-support-priority-queues.html> (<http://activemq.apache.org/how-can-i-support-priority-queues.html>)). Lá, vemos que por padrão o ActiveMQ não respeita a prioridade das mensagens, precisamos ativar esse recurso em sua configuração.

Precisamos colocar no XML do ActiveMQ a configuração:

```
<destinationPolicy>
  <policyMap>
    <policyEntries>
      <policyEntry queue=">" prioritizedMessages="true"/>
    ...
  
```

Vamos editar o arquivo `apache-activemq-5.12.0/conf/activemq.xml` e adicionar a configuração:

```
<policyEntry queue=">" prioritizedMessages="true"/>
```

Com a alteração feita, basta enviarmos uma mensagem e verificarmos no admin web do ActiveMQ se ela está na fila.

Agora, vamos **parar o ActiveMQ** subindo-o logo em seguida para que leve em consideração a nova configuração. Se olharmos a mensagem ela não está mais lá, porque não é persistente.

Agora, vamos enviar algumas mensagens com diferentes prioridades e vê-las no admin o ActiveMQ (*INFO*, *DEBUG*, *ERROR*).

Basta executar nosso consumidor e ver a ordem na qual ele consumirá as mensagens.