01

Recebendo mensagens com MessageListener

Transcrição

Começando deste ponto? Você pode fazer o DOWNLOAD (https://s3.amazonaws.com/caelum-online-public/jms/stages/jms-stage-cap3.zip) completo do projeto do capítulo anterior e continuar seus estudos a partir deste capítulo.

No capítulo anterior conseguimos estabelecer uma conexão utilizando a fábrica disponibilizada pelo ActiveMQ, através do `InitialContext`. E a partir dessa conexão que inicializamos, criamos a nossa sessão, que faz o recebimento e a confirmação das mensagens.

Da nossa `Session`, criamos um `MessageConsumer`, que fica escutando mensagens da fila (`Destination`) que pegamos do nosso `InitialContext`. E conseguimos obter nossa mensagem através do método `consumer.receive()`.

Tratador de mensagem

O problema que encontramos no último capítulo, é que nosso sistema recebia apenas uma mensagem e encerrava o seu funcionamento, e não é isso que queremos. Nos esperamos que ele fique escutando o tempo todo por mensagens, e para isso, devemos cadastrar um novo objeto no nosso consumer, com o responsabilidade de tratar as mensagens que recebemos.

Para manter a separação de responsabilidades, vamos fazer com o que `Consumer` delegue o tratamento das mensagens para um objeto da interface `MessageListener`:

```
consumer.setMessageListener();
```

E para usar esta interface, vamos utilizar uma classe anônima `MessageListener` que a implementa:

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

});
```

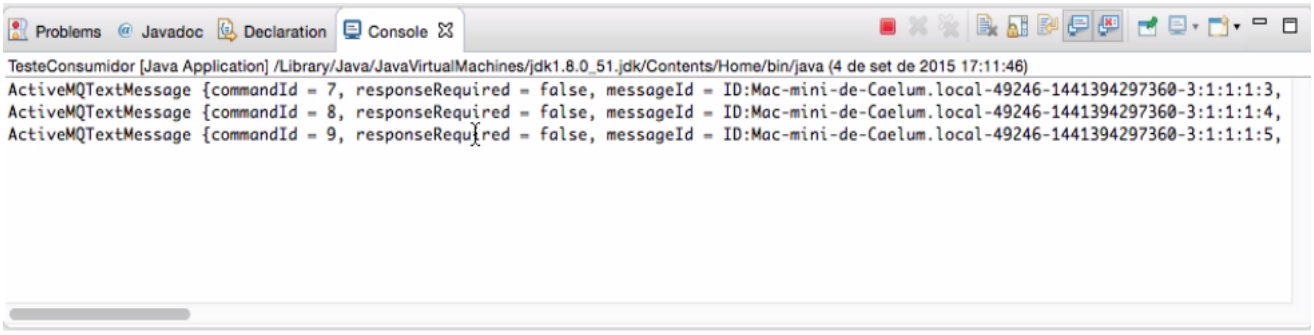
E nesta classe devemos implementar o método chamado `onMessage`, que recebe a mensagem e a faz algum processamento com ela. Por enquanto vamos apenas exibi-la no console:

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

    @Override
    public void onMessage(Message message){
        System.out.println(message);
    }

});
```

Removendo a linha `System.out.println("Recebendo msg: " + message);`, podemos ir na página de envio de mensagens do ActiveMQ testar que o nosso método `onMessage` funciona, e enviar diversas mensagens sem que o nosso sistema se encerre.



Repare que apesar de enviarmos apenas mensagens de texto, no nosso console aparece algo do tipo:

```
ActiveMQTextMessage {commandId = 7,responseRequired = false, messageId = ID, message="Sua mensagem a
```

Subinterfaces da Message

Isto ocorre pois estamos usando a interface Message, que possui subinterfaces mais específicas, que podemos utilizar para pegar o texto da mensagem mais diretamente.

Primeiro vamos pegar a mensagem que estamos recebendo e fazer um cast dela para uma das suas subinterfaces, a `TextMessage`, já que estamos trabalhando com mensagens de texto:

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

    @Override
    public void onMessage(Message message){
        TextMessage textMessage  = (TextMessage)message;
        System.out.println(message);
    }

});
```

Agora que temos uma `TextMessage` podemos utilizar o método `getText()` para pegar apenas o texto da mensagem, deixando assim o resultado mais elegante:

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

    @Override
    public void onMessage(Message message){
        TextMessage textMessage  = (TextMessage)message;
        System.out.println(textMessage.getText());
    }

});
```

Como esta API é um pouco antiga, o Eclipse deve reclamar dizendo que o método `getText` joga uma exceção, então devemos envolve-lo com um `try-catch`:

```
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();
        }
    }

});
```

Testando agora vemos que as nossas mensagens aparecem muito mais limpas no console: