02

Melhor legibilidade com sub-rotas

Downloads

Caso queira começar o treinamento a partir dessa aula, pode baixar o projeto aqui (https://s3.amazonaws.com/caelum-online-public/camel/camel-stage-cap4.zip). Baixe este arquivo, caso não tenha feito os exercícios anteriores.

Revisão do capítulo anterior

No último capítulo, vimos como enviar requisições HTTP com os métodos POST e GET. Além disso, usamos propriedades para guardar dados na rota. Vamos preparar a nova rota para chamar um serviço SOAP a seguir. Mãos à obra!

Separando a rota já existente

A nossa rota atual já possui um bom tamanho e, para dificultar mais ainda, vamos incluir uma outra chamada a um serviço SOAP. O problema é que mesmo trabalhando do alto nível com a Camel DSL, não devemos esquecer que é necessário dar uma manutenção nesse código. Adicionar uma nova chamada na rota não simplificará o trabalho.

Felizmente, o Camel também possui uma solução para este problema já que é possível decompor uma rota baseando em outras rotas menores. Criaremos duas sub-rotas para as chamadas dos serviços SOAP e HTTP chamarem essas sub-rotas por meio de uma rota inicial:

```
//lendo arquivos e delegando para outras rotas
from("file:pedidos?delay=5s").
to("rota:http"). //delegando
to("rota:soap"); //delegando

//aqui vem toda nossa rota já existente
from("rota:http").
//codigo omitido
to("http4://.....");

//aqui vem a nova rota para chamar serviço soap
from("rota:soap").
//codigo omitido
to("chamando o serviço soap");
```

Repare que a rota inicial apenas delega para as sub-rotas. É uma boa ideia, mas será que funciona? Claro, basta substituir a palavra `rota` por `direct`, simples assim...

```
from("file:pedidos?delay=5s").
to("direct:http").
to("direct:soap");

from("direct:http").
//codigo omitido
to("http4://.....");
```

```
from("direct:soap").
//codigo omitido
to("chamando o serviço soap");
```

Mas para testar isso, colocaremos algo no *endpoint* da rota SOAP. Ainda não sabemos qual será o serviço concreto, mas será que não podemos simular um *endpoint*, apenas para testar as sub-rotas?

Usando serviços mock

O Apache Camel integra um framework de testes (*mock*) sofisticado para simular *endpoints* e testar o resultado, inclusive bastante útil para testes automatizados.

Se quisermos *mockear* um serviço devemos usar a palavra `mock` no método `to()`:

```
to("mock:soap")
```

Com isso temos:

```
from("file:pedidos?delay=5s").
to("direct:http").
to("direct:soap");

from("direct:http").
//codigo omitido
to("http4://.....");

from("direct:soap").
log("chamando servico soap")
to("mock:soap");
```

Ao testarmos, veremos que a rota `http` é a primeira a ser executada, depois, a rota `soap`.

Multicast de mensagens

Tudo parece funcionar perfeitamente, no entanto, não implementamos o serviço SOAP. Antes de realmente implementá-lo, faremos mais um teste. Vamos inverter a ordem das sub-rotas, primeiro chamando a rota `soap` e, depois, `http`:

```
from("file:pedidos?delay=5s").
to("direct:soap"). //primeiro chamando soap
to("direct:http");  //http por último

from("direct:http").
//codigo omitido
to("http4://.....");

from("direct:soap").
log("chamando servico soap")
to("mock:soap");
```

Porém, para nossa surpresa, as coisas deixaram de funcionar. Vamos analisar a nossa rota inicial:

```
from("file:pedidos?delay=5s").
to("direct:soap"). //primeiro chamando soap
to("direct:http");  //http por último
```

Por padrão, o resultado da rota `soap` é passado para a próxima rota, ou seja, a rota `http` . Qual é o resultado? Podemos testar isso facilmente usando o componente `log` :

```
from("file:pedidos?delay=5s").
to("direct:soap"). //primeiro chamando soap
log("Chamando soap com ${body}")
to("direct:http");  //http por último
```

Para simplificar a leitura no console, usaremos apenas um arquivo XML, por exemplo `1_pedido.xml` .

Executando podemos ver que o `body` está vazio. Isso está relacionado ao nosso componente `mock` , mas de qualquer forma não queremos depender desse resultado.

Queremos enviar o nosso XML de pedido independente para cada sub-rota, ou seja, a sub-rota `soap` deve receber o XML do pedido inteiro. A rota `http` deve receber o XML do pedido inteirinho. A solução para isso recebe o nome de `multicast` pelo Apache Camel:

```
from("file:pedidos?delay=5s&noop=true").
    multicast().
        to("direct:soap").//sem log agora
        to("direct:http");

//sub-rotas omitidas
```

Testando novamente, agora sim, tudo funcionou perfeitamente. O Camel não deixa a gente na mão.

Nomeando rotas

Nosso código está mais organizado, mas quando recebemos uma exceção, fica fácil se perder. Como podemos saber em qual rota o problema foi gerado? O Apache Camel sempre mostra o nome da rota junto com a exceção, porém, nesse caso foi gerado um nome padrão pois não usamos um nome explícito para identificar a rota. Por isso, é uma boa prática a identificação de uma rota usando o método `routeId()` :

```
from("file:pedidos?delay=5s&noop=true").
    routeId("rota-pedidos").
    multicast().
        to("direct:soap").
        to("direct:http");
```

Faremos isso para todas rotas e sub-rotas! No próximo capítulo, vamos chamar o serviço SOAP. Mas antes, faça os exercícios. Segue o código completo:

```
from("file:pedidos?delay=5s&noop=true").
    routeId("rota-pedidos").
    multicast().
        to("direct:soap").
        to("direct:http");

from("direct:soap").
    routeId("rota-soap").
    log("chamando servico soap ${body}").
to("mock:soap");

from("direct:http").
    routeId("rota-http").
    setProperty("pedidoId", xpath("/pedido/id/text()")).
    setProperty("email", xpath("/pedido/pagamento/email-titular/text()")).
    split().
        xpath("/pedido/itens/item").
    filter().
        xpath("/item/formato[text()='EBOOK']").
    setProperty("ebookId", xpath("/item/livro/codigo/text()")).
    setHeader(Exchange.HTTP_QUERY,
            simple("clienteId=${property.email}&pedidoId=${property.pedidoId}&ebookId=${property
to("http4://localhost:8080/webservices/ebook/item");
```

O que aprendemos?

- dividir a rota em sub-rotas;
- chamar uma sub-rota com `direct`;
- enviar a mesma mensagem pelo `multicast`;
- identificar a rota através da `routeId`.