02

Separando e filtrando mensagens

Downloads

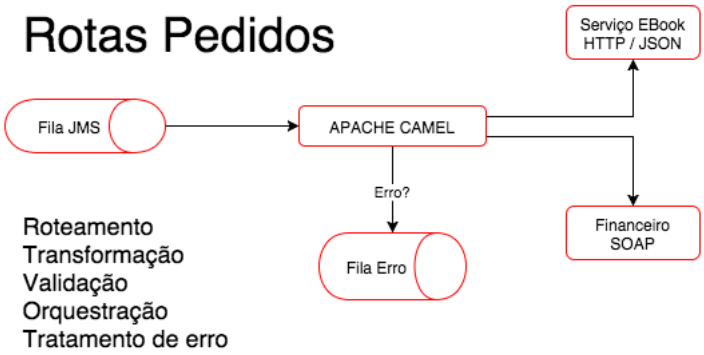
Caso queira começar o treinamento a partir dessa aula, pode baixar o projeto aqui (https://s3.amazonaws.com/caelum-online-public/camel/camel-stage-cap2.zip). Só baixe este arquivo se não tiver feito os exercícios do capítulo anterior.

Revisão do capitulo anterior

Vimos como inicializar o Camel e configurar a primeira rota por meio do Camel DSL. Vimos os métodos principais como `from()`, `to()` e `log()`. Aprendemos que o Camel trabalha com mensagens que internamente possuem uma ID. Para cada mensagem é criada uma identificação que podemos acessar usando a *Expression Language* (EL ${ ... }).

```
public void configure() throws Exception {
    from("file:pedidos?delay=5s&noop=true").
    log("${id} - ${body}").
    to("file:saida");
}
```

Neste capítulo, vamos preparar a chamada HTTP do nosso primeiro serviço externo, lembrando que nosso objetivo é criar a rota seguinte:



Repare que é preciso se conectar com o serviço de ebooks. O pedido de um livro ebook precisa ser enviado para este serviço para pedir a geração dos ebooks para o comprador.

##Trabalhando com JSON

Nosso foco é a preparação do serviço HTTP que recebe um JSON, no entanto, todos os pedidos são apresentados como XML. Como podemos transformar o XML para JSON? Isso é uma tarefa muito rotineira na integração e claro que o Apache Camel já vem preparado para isso.

O Camel oferece o método `xmljson()` com exatamente este propósito. Para podermos chamá-lo devemos primeiro deixar claro a nossa intenção, isso é, configurar que queremos pegar a mensagem da rota e transformá-la em um outro formato. Esse processo é chamado de **Marshalling**!

A nossa rota ficará assim:

```
    from("file:pedidos?delay=5s&noop=true").
    marshal(). //queremos transformar a mensagem em outro formato
        xmljson(). //de xml para json
    log("${id} - ${body}").
    to("file:saida");
```

É importante mencionar que a biblioteca principal do Camel (camel-core) não possui essa funcionalidade. O código até compila mas falta um JAR (dependência) para realmente executar o código. No `pom.xml` temos as dependências já configuradas:

```
<dependency>
    <groupId>org.apache.camel</groupId>
    <artifactId>camel-xmljson</artifactId>
    <version>${camel-version}</version>
</dependency>
```

Ao executar já recebemos um JSON, depois, cada elemento do Xml foi transformado em um elemento JSON. Segue o JSON do primeiro XML:

```
INFO 10:28:36.140 - ID-MacBook-Pro-de-Nico-local-54633-1447936114146-0-1 - {"id":"2451256","data
```

Em geral, o processo de transformação da representação de um objeto em memória em um outro formato para transmissão é chamado de **marshal**. O processo inverso é chamado de **unmarshal**.*

##Filtrando mensagens com XPath

Já temos o JSON, mas analisando profundamente o XML existente, perceberemos que nem todo pedido realmente é um pedido de ebook. Pode ser que o cliente apenas tenha escolhido um livro impresso. Neste caso, não é preciso notificar o serviço ebook nem gerar o JSON. Para resolver isso, vamos filtrar as mensagens e verificar se realmente existe um ebook no pedido.

Como trabalhamos com XML, no início da rota, podemos aproveitar um padrão do mundo XML que facilita a busca de um elemento específico. Esse padrão é conhecido como XPath e claro que o Camel possui uma forma simples de usá-lo.

Para filtrar com Camel usamos os método `filter()` e `xpath()`:

```
 from("file:pedidos?delay=5s&noop=true").
    filter().
        xpath("aqui devemos definir o criterio do filter").
    marshal().
        xmljson().
    log("${id} - ${body}").
to("file:saida");
```

Agora só falta definir o critério usando o XPath e, como o nome já diz, devemos definir um caminho para encontrar o elemento no XML. Para entender melhor, vamos analisar o arquivo `1_pedido.xml`:

```
<pedido>
    <id>2451256</id>
    <dataCompra>2013-12-05T18:21:07.529-02:00</dataCompra>
    <itens>
         <item>
            <formato>EBOOK</formato>

        <!-- resto do xml omitido -->
```

O nosso filtro deve se basear no elemento `<formato>`. Devemos gerar o JSON (a mensagem passa pelo filtro), se possui o texto EBOOK. No entanto, para extrair essa informação do XML, devemos definir o *path*. Observe que o XML começa com `/pedido`, abaixo dele teremos `/itens/item/formato`. Nosso caminho ficou assim: `/pedido/itens/item/formato`. Isso é **XPath**.

Já sabemos navegar no XML pelo Xpath, agora só falta extrair o texto do elemento `formato` e verificar se é igual ao EBOOK:

```
/pedido/itens/item/formato[text()='EBOOK']
```

Aplicando isso na rota:

```
from("file:pedidos?delay=5s&noop=true").
    filter().
        xpath("/pedido/itens/item/formato[text()='EBOOK']").
    marshal().
        xmljson().
    log("${id} - ${body}").
to("file:saida");
```

Perfeito, mas ao executar a rota, você perceberá que todos os pedidos possuem no mínimo um EBOOK, ou seja, todos os pedidos passam pelo filtro.

##Divisão do conteúdo

Temos um novo problema: o serviço de ebook só deveria receber dados sobre o ebook em questão. Jamais devemos enviar informações sobre livros impressos ou dados de pagamento juntos, porque os dados não tem nada a ver com a geração de ebooks.

Vamos imaginar que você olhou na documentação do serviço ebook e encontrou um exemplo de como a chamada HTTP deveria ser executada. Você chegou a um código parecido no envio dos dados:

```
POST /ebook/item
Content-type: application/json

{"formato":"EBOOK","quantidade":"1","livro":{"codigo":"ARQ","titulo":"Introdução à Arquitetura e
```

Ou seja, devemos usar um POST com um JSON, que só possui os dados de um único item. É preciso então dividir cada pedido pelos itens. Claro que o Camel oferece esse recurso, já que faz parte dos padrões de integração. Usaremos o método `split()` e uma expressão XPath (`/pedido/itens/item`):

```
from("file:pedidos?delay=5s&noop=true").
    split().
        xpath("/pedido/itens/item").
    filter().
        xpath("/pedido/itens/item/formato[text()='EBOOK']").
    marshal().
        xmljson().
    log("${id} - ${body}").
to("file:saida");
```

##Ajustando o filter

Ao executar, não receberemos o arquivo JSON. O que aconteceu? O problema é que alteramos a mensagem na rota usando o `split()`. Agora temos mais mensagens na rota do que arquivos XML, pois cada elemento `item` no pedido será representado por meio de uma nova mensagem. Por isso, o nosso filtro não funciona. Após o `split()`, teremos um XML novo na rota que começa com o elemento `item`. É preciso ajustar a expressão XPath do `filter()`:

```
from("file:pedidos?delay=5s&noop=true").
    split().
        xpath("/pedido/itens/item").
    filter().
        xpath("/item/formato[text()='EBOOK']"). //ajuste aqui
    marshal().
        xmljson().
    log("${id} - ${body}").
to("file:saida");
```

Quando executarmos novamente, veremos que o JSON, com as informações, sobre um item é exibidas no console, por exemplo:

```
{"formato":"EBOOK","quantidade":"1","livro":{"codigo":"ARQ","titulo":"Introdução à Arquitetura
```

O serviço HTTP aceitará o arquivo JSON.

Alterando a extensão do arquivo

Ao executar, você já deve ter percebido que os arquivos na pasta `saida` ainda possuem a extensão `.xml` apesar do conteúdo ser um JSON. Claro que essa pasta só existe, porque estamos testando e conhecendo o Camel (no final queremos chamar um serviço HTTP) mas mostraremos como é possível alterar o nome do arquivo gravado.

Para isso, podemos consultar a documentação do Camel (http://camel.apache.org/file2.html). Nela, estão listadas os parâmetros do componente `file`. Para renomear o arquivo - alterar o nome do arquivo original -, devemos definir um cabeçalho da nossa mensagem. Isso é feito usando o método `setHeader(...)`:

```
setHeader("CamelFileName", simple("${file:name.noext}.json"))
```

Isso faz com que seja utilizado o nome da arquivo, por exemplo, `1_pedido` (sem extensão), adicionando a extensão `.json`. Executando a rota, todos os arquivos na pasta `saida` tem a extensão `.json`.

Segue uma vez o código completo:

```
package br.com.caelum.camel;

import org.apache.camel.CamelContext;
import org.apache.camel.builder.RouteBuilder;
import org.apache.camel.impl.DefaultCamelContext;

public class RotaPedidos {

    public static void main(String[] args) throws Exception {

        CamelContext context = new DefaultCamelContext();
        context.addRoutes(new RouteBuilder() {

            @Override
            public void configure() throws Exception {
                from("file:pedidos?delay=5s&noop=true").
                    split().
                        xpath("/pedido/itens/item").
                    filter().
                        xpath("/item/formato[text()='EBOOK']").
                    log("${id} \n ${body}").
                    marshal().xmljson().
                    setHeader("CamelFileName", simple("${file:name.noext}.json")).
                to("file:saida");
            }
        });

        context.start();

        Thread.sleep(20000);
    }
}
```

##O que aprendemos?

- Marshalling - trata-se da transformação de um objeto em outro formato (XML ou JSON, normalmente);
- trabalhar com JSON;
- filtrar e dividir conteúdo;
- aplicar expressões XPath;
- renomear o nome de um arquivo na rota;