02

Transformação com XSLT e integração com serviço SOAP

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-cap5.zip). Baixe este arquivo, caso não tenha feito os exercícios dos capítulos anteriores.

Revisão do capítulo anterior

Vimos, anteriormente, como separar as rotas em sub-rotas usando o `direct`. Isso aumenta a legibilidade e facilita a manutenção no futuro. Apresentamos também como configurar que cada sub-rota receba a mensagem original, usando o `multicast`:

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

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

from("direct:http").
    routeId("rota:http").
    //código omitido
to("http4://.....");
```

Como não implementamos a chamada SOAP, usamos o componente `mock`. Agora vamos completar essa rota e realmente executar a requisição.

Testando o serviço SOAP

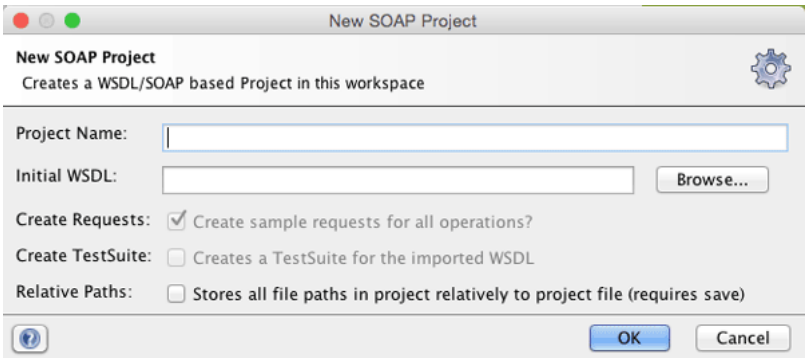
Quando importamos a aplicação web para se comunicar com o serviço http, também já importaremos o serviço SOAP. Veremos o contrato do serviço acessando a URL: http://localhost:8080/webservices/financeiro?wsdl (http://localhost:8080/webservices/financeiro?wsdl)

Vamos testar rapidamente o serviço com a ferramenta SoapUI (http://www.soapui.org/) e submeter uma requisição SOAP.

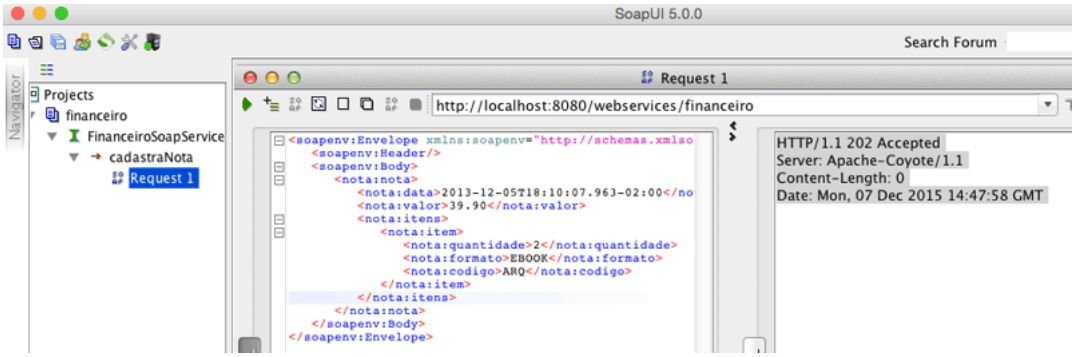
O SoapUI é uma ferramenta para testar serviços web. Com ela podemos ler o WSDL e criar uma mensagem SOAP automaticamente para depois invocar o serviço.

Você pode baixar o SoapUI aqui: http://www.soapui.org/downloads/soapui/open-source.html (http://www.soapui.org/downloads/soapui/open-source.html)

Vamos iniciar o SoapUI e criar um novo projeto SOAP. O nome do projeto será o `financeiro`, e no campo `Initial WSDL`, vamos colar o endereço do WSDL: http://localhost:8080/webservices/financeiro?wsdl (http://localhost:8080/webservices/financeiro?wsdl)



Pronto, vamos testar o serviço. Abriremos o `Request 1` e depois, enviamos a requisição SOAP:



Você poderá se aprofundar sobre SOAP, WSDL e SoapUI no curso da Alura:

- JAX-WS : Domine a criação de webservices SOAP (https://www.alura.com.br/course/soap)

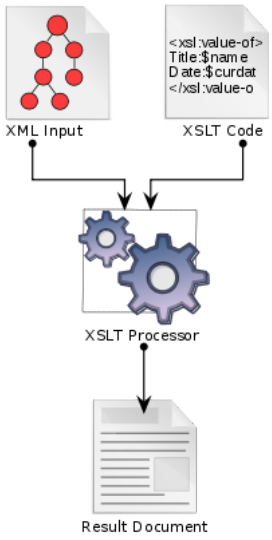
O XML SOAP não tem nada de especial e manda um documento para o *financeiro* que irá gerar a nota fiscal. Repare que todas as informações no XML SOAP estão inclusas no XML do pedido - como a data, valor e dados sobre o item:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:nota="http://
   <soapenv:Header/>
   <soapenv:Body>
      <nota:nota>
         <nota:data>2013-12-05T18:10:07.963-02:00</nota:data>
         <nota:valor>39.90</nota:valor>
         <nota:itens>
            <nota:item>
               <nota:quantidade>2</nota:quantidade>
               <nota:formato>EBOOK</nota:formato>
               <nota:codigo>ARQ</nota:codigo>
            </nota:item>
         </nota:itens>
      </nota:nota>
   </soapenv:Body>
</soapenv:Envelope>
```

Transformando XML com XSLT

A primeira ideia para chamar o serviço SOAP é usar uma biblioteca JAX-WS. O Apache Camel se integra bem com CXF e com certeza seria útil. No entanto, não tem a necessidade de enviar o SOAP a partir de objetos Java. O mais fácil seria extrair as informações do XML pedido e incluir em um novo XML SOAP. Como isso, é comum na integração já existe um padrão com exatamente esse propósito, o **XSLT** (*EXtensible Stylesheet Language Transformation*).

Então a ideia é parecida com linguagens de template como o JSP ou Velocity. Criaremos uma *amostra* (*template*) com algumas lacunas. Essas lacunas serão preenchidas dinamicamente. Vamos ver um pequeno exemplo antes de atacar o nosso SOAP. Para usar XSLT devemos criar um arquivo *stylesheet*, que define um ou mais templates.



Os templates definem as regras da transformação. Veja um primeiro exemplo:

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

    <xsl:template match="/pedido">

        <html>
            <body><xsl:value-of select="id"/></body>
        </html>

    </xsl:template>
</xsl:stylesheet>
```

Repare que o template (`xsl:template`) possui um atributo `match` . Assumindo que o nosso XML de entrada é o arquivo `pedido.xml` , o template procura (*match*) um elemento `/pedido` no XML. Se você está achando que `/pedido` parece com XPath, saiba que você está no caminho certo. As expressões no template usam o XPath.

O XLS também é utilizado no mundo HTML para aplicar o CSS.

Dentro do template, encontraremos o XML que queremos produzir. É bem simples: basta criar uma *lacuna* indicado pelo elemento `<xsl:value-of select="id" />` . O `id` representa o elemento `id` no XML do pedido. Aplicando esse XSLT usando o arquivo `1_pedido.xml` , como entrada recebemos:

```
<?xml version="1.0" encoding="UTF-8"?>
<html>
    <body>2451256</body>
</html>
```

Mas como executar isso com o Apache Camel?

O componente XSLT

Você deve imaginar que o Camel vem preparado para isso. Usar XSLT é simples com Camel (não poderia ser diferente) pois possui um componente para chamar o template. O componente se chama `xslt`.

Vamos aplicar esse componente no método `to()` dentro da rota `soap`:

```
from("direct:soap").
    routeId("rota-soap").
to("xslt:exemplo.xslt"). //aqui chamamos o template que deve estar na pasta src/main/resources
    log("Resultado do Template: ${body}").
to("mock:soap");
```

Ao executar, veremos o seguinte no console:

```
Resultado do template: <html>
<body>2451256</body>
</html>
```

Agora vamos realmente criar a mensagem SOAP usando o XSLT. O template não fica muito mais complicado:

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:output omit-xml-declaration="no" indent="yes"/>

    <xsl:template match="/pedido">
        <soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/" xmlns:nota="http:/
           <soap:Body>
                <nota:nota>
                    <nota:data><xsl:value-of select="dataCompra"/></nota:data>
                    <nota:valor><xsl:value-of select="pagamento/valor"/></nota:valor>
                    <nota:itens>
                        <xsl:for-each select="itens/item">
                            <nota:item>
                                <nota:quantidade><xsl:value-of select="quantidade"/></nota:quant
                                <nota:formato><xsl:value-of select="formato"/></nota:formato>
                                <nota:codigo><xsl:value-of select="livro/codigo"/></nota:codigo
                            </nota:item>
                        </xsl:for-each>
                    </nota:itens>
                </nota:nota>
            </soap:Body>
        </soap:Envelope>
    </xsl:template>

</xsl:stylesheet>
```

No início do *stylesheet*, temos mais duas declarações para formatar o XML produzido. O template cria um mensagem SOAP com os dados de pagamento e todos os itens do pedido. Realmente novo no template é o `<xsl:for-each select="itens/item">` para iterar pelos itens do pedido. Vamos salvar esse template no arquivo `pedido-para-soap.xslt` dentro da pasta `src/main/resources` .

A rota fica:

```
from("direct:soap").
    routeId("rota-soap").
to("xslt:pedido-para-soap.xslt").
    log("Resultado do Template: ${body}").
to("mock:soap");
```

Ao executar, teremos o XML SOAP pronto:

```
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:nota="http://financeiro.com.br/nota">
    <soap:Body>
        <nota:nota>
            <nota:data>2013-12-05T18:21:07.529-02:00</nota:data>
            <nota:valor>29.90</nota:valor>
            <nota:itens>
                <nota:item>
                    <nota:quantidade>1</nota:quantidade>
                    <nota:formato>EBOOK</nota:formato>
                    <nota:codigo>ARQ</nota:codigo>
                </nota:item>
            </nota:itens>
        </nota:nota>
    </soap:Body>
</soap:Envelope>
```

Executando o SOAP

Falta enviar uma requisição HTTP POST com o XML SOAP. Sabemos que basta usar o componente `http4` :

```
from("direct:soap").
    routeId("rota-soap").
to("xslt:pedido-para-soap.xslt").
    log("Resultado do template: ${body}").
to("http4://localhost:8080/webservices/financeiro");
```

Ao executar, recebemos um erro indicando um código HTTP de erro 415. Observando o console do servidor, ficará mais claro o que aconteceu:

```
Tipo de Conteúdo Não Suportado: text/plain Os tipos suportados são: [text/xml]
```

Devemos enviar junto com a mensagem SOAP um cabeçalho HTTP (`content-type`) que indica que o conteúdo é XML. Faremos isso com prazer por meio do método `setHeader` :

```
from("direct:soap").
    routeId("rota-soap").
to("xslt:pedido-para-soap.xslt").
    log("Resultado do template: ${body}").
    setHeader(Exchange.CONTENT_TYPE, constant("text/xml")).
to("http4://localhost:8080/webservices/financeiro");
```

A execução funcionou perfeitamente, segue a rota completa para que você possa comparar:

```
from("file:pedidos?delay=5s&noop=true").
    routeId("rota-pedidos").
multicast().
    to("direct:soap").
        log("Chamando soap com ${body}").
    to("direct:http");

from("direct:soap").
    routeId("rota-soap").
to("xslt:pedido-para-soap.xslt").
    log("Resultado do template: ${body}").
    setHeader(Exchange.CONTENT_TYPE,constant("text/xml")).
to("http4://localhost:8080/webservices/financeiro");

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']").
    setHeader("CamelFileName", simple("${file:name}.json")).
    setProperty("ebookId", xpath("/item/livro/codigo/text()")).
    setProperty("ebookId", xpath("/item/livro/codigo/text()")).
    setHeader(Exchange.HTTP_QUERY,
                simple("clienteId=${property.email}&pedidoId=${property.pedidoId}&ebookId=${pro
to("http4://localhost:8080/webservices/ebook/item");
```

O que aprendemos?

- o que é XSLT;
- como transformar um XML por meio do XSLT com Camel;
- como definir o cabeçalho HTTP `content-type` ;
- chamar o serviço SOAP com o componente `http4` .