02

Entendendo os estilos Document e RPC

Aluno, você está começando nesse capitulo? Não tem problema, você pode baixar o projeto para importar no Eclipse aqui (https://s3.amazonaws.com/caelum-online-public/soap/stage/estoquews-cap5.zip). **Você só precisar baixar o arquivo se você não fez os exercícios do capítulo anterior.**

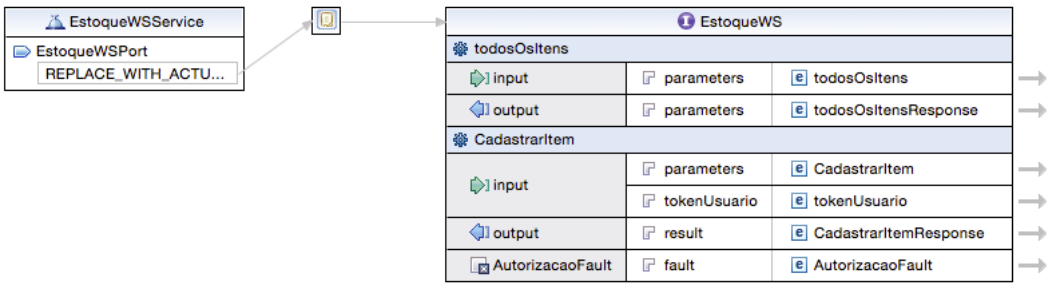
Revisão

Vimos no capítulo anterior que o WSDL está dividido em duas partes, uma abstrata, outra concreta. A abstrata é parecida com uma interface, define os tipos, mensagens e operações que se compõem no elemento `portType`. A parte concreta é para definir o protocolo e endereço que é o grande foco desse capítulo.

A parte concreta é importante para o servidor subir o serviço corretamente pois ele precisa saber o protocolo, encoding etc. Mas para fazer a implementação do serviço basta a parte abstrata.

O elemento binding

Vamos abrir o WSDL no Eclipse para visualizar os elementos. Repare da ligação entre as duas partes do WSDL. Não por acaso esse elemento se chama de *binding* pois ele referencia o `<portType>`:



Vamos mudar a visualização para ver o XML. Logo após do elemento `<binding>` podemos ver uma configuração importante:

```
<soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"/>
```

Aqui temos a configuração que realmente usamos SOAP com HTTP, pois isso fica declarado na URI:

```
http://schemas.xmlsoap.org/soap/http
```

O protocolo HTTP é utilizado por baixo dos panos como um protocolo de transporte (`/soap/http`). Isso parece estranho pois deve ser o padrão usar HTTP quando falamos de um **Web**service, porém o SOAP não depende do HTTP e poderia ser transportado através de outros protocolos.

O outro atributo nessa mesmo linha define o estilo da mensagem. Aqui o estilo se chama de `Document` mas existe também o `RPC` .

RPC e Document

Serviços web podem ser utilizados de maneira diferente. No nosso caso publicamos o serviço para o cliente chamar alguns métodos remotamente. O cliente envia uma requisição SOAP para executar o método ou procedimento no servidor. Para atender essa forma de chamada foi criado o estilo **RPC** que significa *R*emote *P*rocedure *C*all (Chamada remota de um procedimento) um estilo de integração muito antigo que foi criado muito antes do mundo SOAP. Para usar RPC com SOAP devemos enviar primeiro o nome do método ou procedimento e, logo abaixo, os parâmetros. Algo assim:

```
<soapenv:Envelope ...>
   <soapenv:Body>
      <ws:CadastrarItem>
         <item>
            <codigo>MEA</codigo>
            <nome>MEAN</nome>
            <tipo>Livro</tipo>
            <quantidade>5</quantidade>
         </item>
      </ws:CadastrarItem>
   </soapenv:Body>
</soapenv:Envelope>
```

Você pode testar o estilo `RPC` , basta anotar a classe `EstoqueWS` com a anotação `@SOAPBinding` :

```
@WebService
@SOAPBinding(style=Style.RPC)
public class EstoqueWS {
```

Para ver a diferença é preciso republicar o serviço. O WSDL mudou um pouco já que agora estamos usando o estilo `RPC` :

```
<soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="rpc"/>
```

Também é preciso atualizar o SoapUI e gerar o novo request. Logo abaixo do elemento `Body` fica um elemento com o nome do método/operation:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:ws="http://ws.estc
   <soapenv:Body>
      <ws:TodosOsItens>
         <filtros>
            <!--Zero or more repetitions:-->
            <filtro>
```

```
            <!--Optional:-->
            <nome>?</nome>
            <!--Optional:-->
            <tipo>?</tipo>
         </filtro>
      </filtros>
   </ws:TodosOsItens>
</soapenv:Body>
</soapenv:Envelope>
```

Isso parece muito familiar, não? Repare que isso é a mesma coisa que fizemos com o estilo `Document` ! Então para que existe o estilo `Document` ?

O estilo Document

Novamente, existem formas diferentes de se comunicar no mundo de serviços web. Por exemplo: quando uma loja recebe uma compra de um produto é gerado um pedido. Imagine que a partir desse pedido devemos notificar um sistema de notas fiscais. Queremos apenas entregar o pedido e o que esse sistema de notas fiscais realmente fará com esse pedido não interessa para o loja. Ou seja, não estamos interessados em chamar algum método ou procedimento do outro sistema. Apenas queremos notificar e entregar o pedido. Nesse caso faz sentido enviar apenas os dados do pedido na mensagem SOAP, por exemplo:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:ws="http://ws.estc
   <soapenv:Body>
         <pedido>
            <numero>123</numero>
            <data>22/07/2015</data>
            <!-- outras infos omitidas -->
         </pedido>
   </soapenv:Body>
</soapenv:Envelope>
```

A mensagem SOAP representa apenas um documento! Mas porque estávamos usando `Document` invés de `RPC` já que estamos chamando um método?

Problemas do RPC

http://mangstacular.blogspot.com.br/2011/05/wsdl-soap-bindings-confusion-rpc-vs.html (http://mangstacular.blogspot.com.br/2011/05/wsdl-soap-bindings-confusion-rpc-vs.html)

Realmente o nosso serviço usa o estilo de integração RPC. No entanto, ao expor serviços dessa maneira muitas vezes o XML fica muito amarrado a aplicação. Isso pode gerar uma acoplamento forte e criar problemas de compatibilidade que dificulta a integração heterogênea. Na realidade, isso significava que muitos vezes um cliente não conseguia se comunicar com um serviço por causa do RPC.

Document/Wrapped

Para não gerar problemas de compatibilidade, a grande maioria dos serviços usa hoje em dia o estilo `Document` . O grande problema do `Document` é que não havia uma forma padrão para fazer RPC! Felizmente isso mudou, como vocês já viram

podemos usar o estilo `Document` para fazer uma chamada remota de um método. Basta embrulhar o documento em um elemento XML como mesmo nome do método! Esse forma se chama de `Document/Wrapped` . Ou seja, usamos o tempo todo `Document/Wrapped` para fazer RPC, ok?

Podemos deixar essa configuração explícita, usando a mesma anotação `@SOAPBinding` mas não é necessário já que é o padrão:

```
@WebService
@SOAPBinding(style=Style.DOCUMENT,parameterStyle=ParameterStyle.WRAPPED)
public class EstoqueWS {
```

No SOAP temos um elemento *Wrapped* (com o nome do método) como vimos antes:

```
<soapenv:Envelope ...>
   <!-- header omitido -->
   <soapenv:Body>
      <ws:CadastrarItem><!-- document/wrapped -->
         <item>
            <codigo>?</codigo>
            <nome>?</nome>
            <tipo>?</tipo>
            <quantidade>?</quantidade>
         </item>
      </ws:CadastrarItem>
   </soapenv:Body>
</soapenv:Envelope>
```

Document/Bare

Será que existem serviço do tipo `document` que não são wrapped? Existem, claro! E já discutimos isso, quando queremos entregar apenas o item sem ter conhecimento de qual método/procedimento é chamado no lado do servidor. Podemos testar isso facilmente usando a mesma anotação `@SOAPBinding`

```
@WebService
@SOAPBinding(style=Style.DOCUMENT,parameterStyle=ParameterStyle.BARE)
public class EstoqueWS {
```

Como sempre, devemos republicar e atualizar o cliente.

Como resultado disso vemos que, a mensagem SOAP gerada não possui mais o elemento *wrapped*, apenas o item:

```
<soapenv:Envelope ...>
    <!-- header omitido -->
   <soapenv:Body>
      <ws:item>
         <codigo>MEA</codigo>
         <nome>MEAN</nome>
         <tipo>Livro</tipo>
         <quantidade>5</quantidade>
      </ws:item>
```

```
    </soapenv:Body>
</soapenv:Envelope>
```

Será que ainda podemos chamar o nosso serviço? Teste e você vai ver que funciona! Mas como o JAX-WS sabe resolver isso já que não tem o nome do método no SOAP? Bom nesse caso foi fácil. Pois só há um método que recebe um item. Mas se houvessem mais um método o JAX-WS já reclamaria na hora de subir o serviço (mas sobe), no entanto ao executar a mesma mensgam SOAP recebemos como resposta:

```
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
   <S:Body>
      <S:Fault xmlns:ns4="http://www.w3.org/2003/05/soap-envelope">
         <faultcode>S:Client</faultcode>
         <faultstring>Não é possível localizar o método de despacho para Request=[SOAPAction="",Pay
      </S:Fault>
   </S:Body>
</S:Envelope>
```

Ou seja, se a assinatura das *operation* não for clara, o JAX-WS vai gerar um `fault`.

Usando SOAPAction

Repare que na resposta aparece um elemento `SOAPAction` sendo uma String vazia. Esse `SOAPAction` foi criado para mensagens do tipo `Document` que querem definir o método a ser chamado fora do XML. A configuração do `SOAPAction` fica no WSDL que se baseia na nossa classe `EstoqueWS`. Nela podemos aproveitar a anotação `@WebMethod` para definir a *action*:

```
@WebService()
@SOAPBinding(style=Style.DOCUMENT,parameterStyle=ParameterStyle.BARE)

public class EstoqueWS {

   //novidade atributo action
   @WebMethod(action="CadastrarItem", operationName="CadastrarItem")
   @WebResult(name="item")
   public Item cadstrarItem(@WebParam(name="tokenUsuario", header=true) TokenUsuario token, @WebPara
```

Republicando percebemos uma pequena mudança no WSDL:

```
<operation name="CadastrarItem">
   <soap:operation soapAction="CadastrarItem"/>
</operation>
```

O `soapAction` já existia antes, mas agora está preenchido com o valor da anotação `@WebMethod`. Ao atualizar o cliente e recriar o request não há nenhuma diferença na mensagem SOAP. A diferença está no protocolo HTTP que ganhou um novo cabeçalho:

```
SOAPAction: "CadastrarItem"
```

Através desse cabeçalho o JAX-WS sabe resolver o método correto e podemos executar a requisição sem problemas.

Literal e encoded

Continuando na nossa viagem pelo WSDL temos em cada `operation` um `input` , `output` e um possível `fault` . Aqui podemos ver qual mensagem aparece onde. Ela pode ser um `input` ou `output` , e fazer parte do `body` , `header` ou `fault` :

```
<operation name="CadastrarItem">
   <soap:operation soapAction="CadastrarItem"/>
   <input>
      <soap:body use="literal" parts="item"/>
      <soap:header message="tns:CadastrarItem" part="tokenUsuario" use="literal"/>
   </input>
   <output>
      <soap:body use="literal"/>
   </output>
   <fault name="AutorizacaoException">
      <soap:fault name="AutorizacaoException" use="literal"/>
   </fault>
</operation>
```

Além disso, tem uma configuração importante, `use="literal"` ! Ela faz parte da **codificação da mensagem** e significa que na mensagem SOAP apenas dados (literais trafegam, sem nenhuma informação tipos ou regras de validação. Isso faz sentido pois o lugar correto para os tipos e as regras de validação é o XSD.

Infelizmente, essa boa separação dos dados e tipos nem sempre foi assim, pois existe uma outra forma de codificação: o `encoded` que significa que na mensagem SOAP os tipos são enviados junto aos dados literais. Por isso, não há XSD! Lembra do estilo RPC e os problemas de compatibilidade? Pois é, era muito comum usar `RPC/encoded` . A mensagem fica parecida com a mensagem a seguir onde também estamos usando `RPC/encoded` :

```
<soapenv:Envelope ...>
   <soapenv:Body>
    <ws:CadastrarItem>
      <ws:item>
         <codigo type="xsd:string">MEA</codigo>
         <nome type="xsd:string">MEAN</nome>
         <tipo type="xsd:string">Livro</tipo>
         <quantidade type="xs:int">5</quantidade>
      </ws:item>
    </ws:CadastrarItem>
   </soapenv:Body>
</soapenv:Envelope>
```

Os dados na mensagem já vem com os tipos. Torna-se mais fácil para nós humanos entendermos, mas gera uma dor de cabeça terrível para validadores de XML. Enfim, não adianta nem discutir muito pois é algo legado, não é aderente a especificação de compatibilidade de serviços SOAP do W3C e deve ser evitado, ok?

Essa forma de codificação fez bastante sucesso (*Encoded*) e ainda existem serviços web legados que usam isso, mas saiba que o JAX-WS nem dá mais suporte a isso.

O elemento Service - o endereço

Por fim temos o elemento `service` que é muito mais simples que define duas coisas. O *binding* utilizado e o endereço concreto para chamar serviço SOAP. Nesse caso é

```
<service name="EstoqueWSService">
   <port name="EstoqueWSPort" binding="tns:EstoqueWSPortBinding">
      <soap:address location="http://localhost:8080/estoquews"/>
   </port>
</service>
```

Todo esse tópico merecem uma boa revisão, não? Bora fazer os exercícios :)

O que você aprendeu nesse capítulo?

- Os estilos Document e RPC
- A codificação da mensagem SOAP: literal e encoded
- Usamos Document/literal/Wrapped para fazer RPC
- Encoded é algo legado e JAX-WS não suporta mais
- O elemento service define o endereço