02

WSDL abstrato e concreto

Aluno, você está começando nesse capitulo? Não tem problema, você pode baixar o projeto a importar no Eclipse aqui (https://s3.amazonaws.com/caelum-online-public/soap/stage/estoquews-cap4.zip). Você só precisar baixar o arquivo se você não fez os exercícios do capítulo anterior.

Revisão

No último capítulo vimos como trabalhar com cabeçalhos e exceções na mensagem SOAP. Personalizamos ambos usando as anotações disponíveis no JAX-WS. Para criar um cabeçalho da mensagem SOAP basta adicionar um parâmetro no método e declará-lo com `@WebParam(header=true)` . Vimos que os cabeçalhos são úteis para declarar meta informações como auditoria, dados da transação e autenticação e autorização. Muitas vezes os dados nos cabeçalhos são processados pelos intermediários entre cliente e servidor. Pode ter vários desses intermediários ou *SOAP Nodes* que realmente usam esses cabeçalhos.

Também vimos como o SOAP trabalha com exceções. Para ser correto os SOAP chama as exceções do mundo Java de **Fault**. Então é preciso traduzir as exceções para os `Fault` s do mundo SOAP. As exceções *checked* automaticamente fazem parte do contrato, ou seja, fazem parte do WSDL. As exceções *unchecked* serão traduzidos para um `Fault` padrão.

Neste capítulo veremos mais detalhes sobre o contrato e vamos falar sobre as seções `types` , `message` e `portType` do WSDL.

Definição dos tipos

Já falamos bastante e sobre o WSDL, vamos dar uma olhada com mais detalhes nas seções do WSDL. A primeira seção são os tipos usados no contrato (`<types>`). São aqueles definições do XSD. Este define como, por exemplo, se compor um item ou como se define aquele *token* do usuário. O XSD define também as regras de validação e é bastante rico nesse sentido. Esse arquivo até pode ser utilizado separadamente para validar um item ou o payload (os dados principais) da mensagem SOAP!Exemplo do item no XSD:

```
<xs:simpleType name="tipoItem">
    <xs:restriction base="xs:string">
        <xs:enumeration value="Livro"/>
        <xs:enumeration value="Celular"/>
        <xs:enumeration value="Tablet"/>
    </xs:restriction>
</xs:simpleType>
```

Podemos dizer, que tudo que está trafegando dentro de uma mensagem SOAP deve estar declarado de alguma forma no XSD. Em geral, qualquer serviço expõe um modelo, e no mundo SOAP este modelo está definido o XSD!

Mensagens no WSDL

A segunda parte do WSDL são as mensagens. As mensagens se baseiam no XSD e cada uma representa uma entrada ou saída do serviço. Repare o elemento `CadastrarItem` :

```
<message name="CadastrarItem">
    <part name="parameters" element="tns:CadastrarItem"/>
```

```
    <part name="tokenUsuario" element="tns:tokenUsuario"/>
</message>
```

Ele possui duas partes, a primeira é o que vem no corpo da mensagem (no `Body`), a segunda parte é do cabeçalho (`Header`). Repare também que nosso `Fault` faz parte de uma mensagem:

```
<message name="AutorizacaoFault">
    <part name="fault" element="tns:AutorizacaoFault"/>
</message>
```

Isso faz sentido já que o `Fault` também é uma saída.

A interface: o elemento PortType

Logo após as mensagem, encontramos a seção `portType` que associa as mensagem a uma operação. Repare a operação `TodosOsItens`, ela possui uma entrada e saída, cada uma representada por uma mensagem:

```
<operation name="TodosOsItens">
   <input wsam:Action="http://ws.estoque.caelum.com.br/EstoqueWS/TodosOsItensRequest" message="tns:T
  <output wsam:Action="http://ws.estoque.caelum.com.br/EstoqueWS/TodosOsItensResponse" message="tns:
</operation>
```

O que importa aqui é o atributo `message`. Nele temos uma referência a mensagem, por exemplo `tns:TodosOsItens`.

O atributo `wsam:Action` é relacionado com o WS-Addressing que pode ser útil para chamadas assíncronas, quando queremos devolver a mensagem de resposta para algum outro endereço. O JAX-WS define uma anotação `@Action` para manipular estes valores.

WSDL abstrato e concreto

Até agora, nesses elementos do WSDL não definimos o protocolo concreto a ser utilizado. Em nenhum momento está escrito que realmente queremos usar o SOAP! Também não tem nenhuma informação sobro o endereço de serviço. Essas definições mais concretas vem na segunda parte do WSDL.

De certa forma o contrato é dividido em duas partes: A primeira parte que já vimos, com as operações, mensagens e tipos que chamamos de **WSDL abstrato**. A segunda parte que terá definições sobre o protocolo, endereço e codificação das mensagens chamamos de **WSDL concreto**.

Visualizando o WSDL

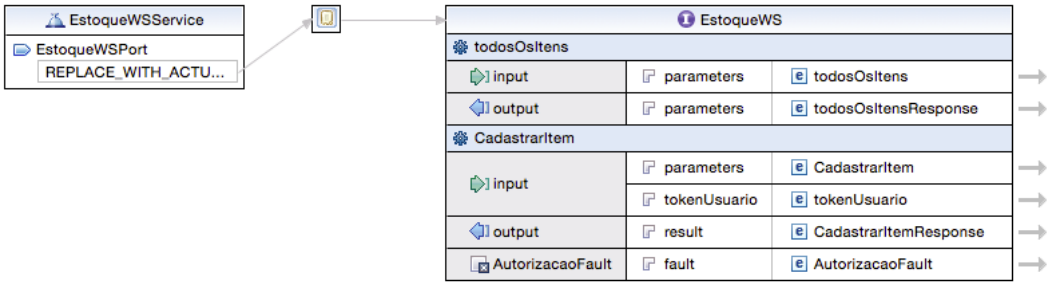
Para simplificar vamos visualizar o WSDL no Eclipse, mas antes iremos gerar o WSDL a partir da classe.

Entre na pasta do projeto e execute:

```
wsgen -wsdl -inlineSchemas -cp bin br.com.caelum.estoque.ws.EstoqueWS
```

OBS: O JDK deverá estar instalado, caso não tenha, faça o download do WSDL aqui (https://s3.amazonaws.com/caelum-online-public/soap/EstoqueWSServiceCap5.wsdl) ou copie do navegador mesmo.

Repare que no comando usamos a configuração `inlineSchemas` para criar um arquivo apenas com XSD E WSDL. Agora abra o arquivo gerado no Eclipse que possui um editor dedicado ao WSDL:



Podemos ver no lado esquerdo a parte concreta do WSDL e no lado direito a parte abstrata. Entre as duas parte tem uma ligação (*binding*). Repare que a parte abstrata é apresentado como se fosse um interface Java (usa-se o mesmo símbolo). Claro que não é Java e sim XML, mas trata-se do contrato do serviço.

O editor possui algumas funções para editar e refatorar os dados do serviço, no entanto devemos ter em mente que o WSDL publicado pelo serviço não é criado ao vivo, na hora de rodar o serviço.

No próximo capítulo vamos focar no WSDL concreto mas agora é a hora dos exercícios.

O que você aprendeu nesse capítulo?

- existe uma parte abstrata e concreto no WSDL
- a parte abstrata é parecido com uma interface
- a parte concreta é para definir o protocolo e endereço
- a parte abstrata define os tipos, mensagens e operações
- a interface no WSDL se chama portType
- podemos usar wsgen para gerar o arquivo WSDL