02

Personalizando o serviço

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-cap2.zip). **Você só precisar baixar o arquivo se você não fez os exercícios do capítulo anterior.**

Revisão

No capítulo anterior, escrevemos e publicamos o primeiro serviço SOAP. Apesar de ser a forma mais simples possível, já foi o bastante para conhecermos alguns artefatos importantes. Qualquer serviço define um contrato, o **WSDL**. Os dados que trafegam entre cliente e servidor são apresentados através das mensagens **SOAP**. Vimos que no nosso serviço temos uma mensagem de **ida** e uma de **volta**. Publicamos o nosso serviço usando **JAX-WS**. Especificação Java EE que é especialista neste assunto. Por fim criamos um cliente do nosso primeiro serviço. Nesse capítulo, vamos ajustar a mensagem e o WSDL.

Entendendo as operations

Vamos subir o serviço pelo Eclipse e chamar o WSDL pelo navegador.

No SoapUI temos a mensagem SOAP gerada. Repare que no `Body` temos um elemento com o nome `getItens` :

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:ws="http://ws.estc
   <soapenv:Header/>
   <soapenv:Body>
      <ws:getItens/>
   </soapenv:Body>
</soapenv:Envelope>
```

O elemento possui o mesmo nome que o método na classe Java. Como criamos a implementação do serviço primeiro, foi utilizado a nomenclatura dela para criar o contrato WSDL que é a base para o SOAP.

Podemos ver esse elemento `operation` que faz parte do `portType` definido no WSDL:

```
<definitions ...>
    <types>
        <!-- definições dos tipos xsd-->
    </types>
    <message name="getItens">
        <part name="parameters" element="tns:getItens" />
    </message>
    <message name="getItensResponse">
        <part name="parameters" element="tns:getItensResponse" />
    </message>
    <portType name="EstoqueWS">
        <operation name="getItens">
            <input  message="tns:getItens" />
            <output  message="tns:getItensResponse" />
        </operation>
    </portType>
```

```
    <!-- bindings e endereços omitidos-->
</definitions>
```

Repare que o elemento `portType` está parecido com uma interface Java: declara um nome (`EstoqueWS`) e define as operações com cada entrada e saída.

A mensagem SOAP se baseia no `input` ou `output` das operações do `portType` .

Melhorando o serviço

Nosso serviço é simples, mas podemos (e iremos) melhorar muito. Aquele elemento `getItens` faz sentido para quem é desenvolvedor Java. Mas será que é um bom nome para desenvolvedores de outras plataformas como Ruby e Python? Precisamos lembrar que nosso serviço é independente de plataforma.

Não há uma nomenclatura padrão de serviços SOAP, por isso devemos deixar nosso contrato o mais expressivo possível. Já que ferramentas o usarão para gerar clientes do nosso serviço.

Observe, por exemplo, a resposta SOAP. Lá temos um elemento `return` , que novamente aparenta ser um nome não muito expressivo, não acha?

Precisamos alterar a classe `EstoqueWS` para definir nomes melhores, portanto nosso primeiro passo será escolher um outro nome para o método `getItens` . Poderiamos então dar um `rename` e fugir da nomenclatura padrão de Java. Porém dessa forma estariamos prejudicando nosso código Java, abrindo mão das convenções de nomenclatura, por conta do WSDL.

@WebMethod e @WebResult

Para evitar esse problema, o JAX-WS nos oferece uma alternativa: a anotação `@WebMethod` . Com ela, podemos redefinir o nome da *operation* no WSDL e assim também manipular a mensagem SOAP.

```
@WebMethod(operationName="todosOsItens")
public List<Item> getItens() {
    System.out.println("Chamando getItens()");
    return dao.todosItens();
}
```

Essa mudança simples faz com que a requisição SOAP tenha agora um elemento com o nome da operação:

```
<soapenv:Envelope ...>
   <soapenv:Header/>
   <soapenv:Body>
      <ws:todosOsItens/>
   </soapenv:Body>
</soapenv:Envelope>
```

Segundo passo é melhorar a resposta SOAP, já que aquele `return` não deve existir. Isso acontece porque o JAX-B (que veremos mais a frente) que é o responsável por gerar o XML, não conhece a interface List. Portanto, ele substitui por um nome genérico (`return`).

Repare que na resposta SOAP aparece para cada item um elemento `return`. Uma forma fácil de resolver isso é usar a anotação `@WebResult`:

```
@WebMethod(operationName="todosOsItens")
@WebResult(name="item")
public List<Item> getItens() {
    System.out.println("Chamando getItens()");
    return dao.todosItens();
}
```

A resposta SOAP já melhorou bastante, veja o XML:

```
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
   <S:Body>
      <ns2:todosOsItensResponse xmlns:ns2="http://ws.estoque.caelum.com.br/">
         <item>
            <codigo>GAL</codigo>
            <nome>Galaxy Tab</nome>
            <quantidade>3</quantidade>
            <tipo>Tablet</tipo>
         </item>
         <item>
            <codigo>MOX</codigo>
            <nome>Moto X</nome>
            <quantidade>6</quantidade>
            <tipo>Celular</tipo>
         </item>
         <!-- outros itens -->
      </ns2:todosOsItensResponse>
   </S:Body>
</S:Envelope>
```

Mapeamento com JAX-B

Houve uma mudança significativa do XML SOAP. No entanto há mais para melhorar: é um tanto estranho chamar o `@WebResult` de `item` já que estamos trabalhando com uma lista de itens, não acha?. Faz mais sentido usar um elemento `itens` que possui vários filhos `item`, algo assim:

```
<itens>
         <item>
                  ....
         </item>
         <item>
                  ....
         </item>
</itens>
```

Precisamos resolver isso, mas nosso método `getItens` possui um pequeno problema: ele retorna um `List` e já vimos que esta interface não é conhecida do JAX-B. Por isso foi preciso usar a anotação `@WebResult` para mapear cada um dos elementos da lista como `item` no XML. Porém dessa forma, eles ficam órfãos (sem uma tag pai).

Então, para resolvermos esse problema, precisaremos criar uma classe separada para esse propósito. Vamos chamar a classe de `ListaItens` que vai existir para embrulhar a lista original:

```
public class ListaItens {

    private List<Item> itens;

    public ListaItens(List<Item> itens) {
        this.itens = itens;
    }

    //esse construtor também é necessário
    ListaItens() {
    }
}
```

Para o JAX-B funcionar corretamente devemos colocar a anotação `@XmlRootElement` e, para não criar um getter e setter, vamos definir o acesso pelo atributo pela anotação `@XmlAccessorType(XmlAccessType.FIELD)` . Repare também que definimos que cada elemento na lista é um `item` :

```
@XmlRootElement
@XmlAccessorType(XmlAccessType.FIELD)
public class ListaItens {

    @XmlElement(name="item")
    private List<Item> itens;

    public ListaItens(List<Item> itens) {
        this.itens = itens;
    }

    //esse construtor também é necessário
    ListaItens() {
    }
}
```

Pronto, no método do serviço colocaremos como retorno `ListaItens` :

```
@WebService()
public class EstoqueWS {

    private ItemDao dao = new ItemDao();

    @WebMethod(operationName="todosOsItens")
    @WebResult(name="itens")
    public ListaItens getItens() {
        System.out.println("Chamando getItens()");
        return new ListaItens(dao.todosItens()); //criando uma ListaItens
    }

}
```

Tudo pronto para testar de novo! Vamos republicar o serviço e verificar o WSDL. Acesse novamente a URL:

http://localhost:8080/estoquews?wsdl (http://localhost:8080/estoquews?wsdl)

Vá ao SoapUI e atualize o cliente (aperte F5 no `EstoqueWSPortBinding`). Abra o *request* e execute a requisição SOAP. Na resposta deve haver um elemento `itens` seguido pelos elementos `item`:

```
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
   <S:Body>
      <ns2:todosOsItensResponse xmlns:ns2="http://ws.estoque.caelum.com.br/">
         <itens>
            <item>
               <codigo>SEO</codigo>
               <nome>SEO na Prática</nome>
               <quantidade>4</quantidade>
               <tipo>Livro</tipo>
            </item>

            <!-- outros itens omitidos -->
         </itens>
      </ns2:todosOsItensResponse>
   </S:Body>
</S:Envelope>
```

O que você aprendeu nesse capítulo?

- os métodos Java se tornam `operation`s no WSDL
- as `operation`s fazem parte do portType
- as anotações do JAX-WS servem para personalizar o WSDL
- a especificação JAX-B gera o XML por baixo dos panos