

Expandindo nossas validações

Transcrição

No vídeo anterior aprendemos a validar um pedaço do nosso XML, e agora repetiremos o processo para toda a parte de produtos. Começaremos descomentando o código e alocando-o dentro da tag `<venda>`.

```
<?xml version="1.0" encoding="UTF-8"?>
<venda moeda="real" xsi:noNamespaceSchemaLocation="formatoVenda.xsd" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <formaDePagamento>Cartão</formaDePagamento>
  <produtos>
    <produto>
      <nome>Livro de xml</nome>
      <preco>29.90</preco>
    </produto>
    <produto>
      <nome>Livro de 0.0. java</nome>
      <preco>29.90</preco>
    </produto>
  </produtos>
</venda>
```

Se rodarmos nosso sistema dessa forma, receberemos um erro indicando que `produtos` não é válido, pois nenhum elemento filho é esperado nesse ponto. Então, em `formatoVenda.xsd`, indicaremos a existência desse elemento com a tag `<element>`. Além disso, informaremos que `produtos` também é um tipo complexo.

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema">
  <element name="venda">
    <complexType>
      <sequence>
        <element name="formaDePagamento" type="string" />
        <element name="produtos">
          <complexType>
            <sequence>
              <element name="produto" type="complexType" />
            </sequence>
          </complexType>
        </element>
      </sequence>
      <attribute name="moeda" type="string"/>
    </complexType>
  </element>
</schema>
```

Dentro da tag `<complexType>`, adicionaremos mais um elemento `produto`, que por sua vez também é um elemento complexo. Prosseguindo, adicionaremos os elementos `nome`, que é uma `string`, e `preco`, que é um `double`. Também devemos nos lembrar que, para termos um elemento dentro de um tipo complexo, precisamos adicionar uma tag `<sequence>`.

```

<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema">
  <element name="venda">
    <complexType>
      <sequence>
        <element name="formaDePagamento" type="string" />
        <element name="produtos">
          <complexType>
            <sequence>
              <element name="produto">
                <complexType>
                  <sequence>
                    <element name="nome" type="string"/>
                    <element name="preco" type="double"/>
                  </sequence>
                </complexType>
              </element>
            </sequence>
          </complexType>
        </element>
      </sequence>
      <attribute name="moeda" type="string"/>
    </complexType>
  </element>
</schema>

```

Feitas essas definições, se tivermos uma tag `<nomes>` no nosso XML, ao invés de `<nome>`, teremos como retorno um erro. Ou seja, a validação que gostaríamos de implementar já foi realizada. Porém, mesmo corrigindo esse erro, se rodarmos novamente o sistema teremos outro aviso de que um `produto` foi encontrado em um ponto onde nenhum elemento filho era esperado - nesse caso, se referindo ao segundo produto que cadastramos.

Nosso `formatoVenda.xsd` informa que temos um elemento `produto` dentro de `produtos`. Porém, por padrão, o XSD considera que um elemento existe uma única vez. Se várias tags `<produto>` devem ser permitidas, teremos que adicionar um atributo `maxOccurs` cujo valor definiremos como `unbounded` (não determinado).

```

<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema">
  <element name="venda">
    <complexType>
      <sequence>
        <element name="formaDePagamento" type="string" />
        <element name="produtos">
          <complexType>
            <sequence>
              <element name="produto" maxOccurs="unbounded">
                <complexType>
                  <sequence>
                    <element name="nome" type="string"/>
                    <element name="preco" type="double"/>
                  </sequence>
                </complexType>
              </element>
            </sequence>
          </complexType>
        </element>
      </sequence>
    </complexType>
  </element>
</schema>

```

```

        </element>
    </sequence>
    <attribute name="moeda" type="string"/>
</complexType>
</element>
</schema>

```

Com isso, não teremos mais nenhuma mensagem de erro. Perceba que nosso arquivo XSD ficou um pouco grande e confuso, com diversos elementos encadeados. Será que podemos melhorá-lo? Para isso, pegaremos a definição do elemento `produto` e a exportaremos para fora do código. Em seguida, removeremos o atributo `maxOccurs` e, dentro da tag `<sequence>` que ficou vazia, faremos uma conexão por meio de uma tag `element ref="produto"/>`, na qual definiremos o máximo de ocorrências com `maxOccurs="unbounded"`.

Para que o Java consiga encontrar essa conexão, precisaremos definir um apelido `xsd` para o nosso namespace. Em seguida, precisaremos adicionar o prefixo `xsd:` a todas as tags que vêm do módulo `xmlns:xsd`.

```

<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:element name="produto">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="nome" type="xsd:string"/>
                <xsd:element name="preco" type="xsd:double"/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>

    <xsd:element name="venda">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="formaDePagamento" type="xsd:string" />
                <xsd:element name="produtos">
                    <xsd:complexType>
                        <xsd:sequence>
                            <xsd:element ref="produto" maxOccurs="unbounded"/>
                        </xsd:sequence>
                    </xsd:complexType>
                </xsd:element>
            </xsd:sequence>
            <xsd:attribute name="moeda" type="xsd:string"/>
        </xsd:complexType>
    </xsd:element>
</xsd:schema>

```

Rodando novamente o sistema, tudo funcionará corretamente, e o código do nosso XSD está um pouco mais legível. Para não termos conflito, é uma boa prática darmos um apelido para todo namespace externo que é importado, e assim evitaremos problemas futuros.