Programação Orientada a Serviços

Prof. Diego Cirilo

Aula 08: XML Schema

XML Schema

- O DTD apresenta algumas limitações:
  - Não é escrito em XML, criando confusão com duas linguagens diferentes
  - Não suporta namespaces
  - Não suporta tipos de dados (string, integer, float, etc)
- O XML Schema Definition (XSD) é uma recomendação do W3C que soluciona essas limitações.
- De forma geral é um XML que descreve os componentes de outro XML (ou uma classe deles).

XML Schema

- Tipos de dados são importantes para garantir integridade dos dados. Ex.
  - Como garantir que `<data></data>` tenha uma data com DTD? E como garantir um padrão na escrita dessa data?
- Também há problemas com o XSD:
  - A ordem continua importando (nem sempre é necessário)
  - Muito complexa (a primeira parte da especificação tem 150 páginas)
  - ...

Exemplo

- XML

```
<cartao xmlns="http://cartaodevisitas.org"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://cartaodevisitas.org
                            cartao.xsd">
  <nome>Luiz Silva</nome>
  <titulo>Gerente, Lojas Pernambucanas S.A.</titulo>
  <email>luiz.silva@pernambucanas.com.br</email>
  <telefone>(88)99456-1414</telefone>
  <logo url="logo.gif"/>
</cartao>
```

Exemplo

- XSD (cartao.xsd)

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:b="http://cartaodevisitas.org"
        targetNamespace="http://cartaodevisitas.org"
        elementFormDefault="qualified">

  <element name="cartao" type="b:cartao_type"/>
  <element name="nome" type="string"/>
  <element name="titulo" type="string"/>
  <element name="email" type="string"/>
  <element name="telefone" type="string"/>
  <element name="logo" type="b:logo_type"/>

  <complexType name="cartao_type">
    <sequence>
      <element ref="b:nome"/>
      <element ref="b:titulo"/>
      <element ref="b:email"/>
      <element ref="b:telefone" minOccurs="0"/>
      <element ref="b:logo" minOccurs="0"/>
    </sequence>
  </complexType>

  <complexType name="logo_type">
    <attribute name="url" type="anyURI"/>
  </complexType>
```

Sintaxe

- O elemento raiz do XSD é o `<schema>`
- Alguns possíveis atributos do `<schema>`:
  - `xmlns="http://www.w3.org/2001/XMLSchema"` namespace do XML Schema padrão.
  - `xmlns:b="http://cartaodevisitas.org"` define o prefixo `b` para o namespace do cartão.
  - `targetNamespace="http://cartaodevisitas.org"` indica qual namespace que esse XSD descreve.
  - `elementFormDefault="qualified"` indica se os elementos devem ser qualificados por um namespace.
- Detalhes

Sintaxe

- Para incluir o XSD no XML:
  - `xmlns="http://cartaodevisitas.org"` namespace padrão.
  - `xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"` namespace da instância do XSD.
  - `xsi:schemaLocation="http://cartaodevisitas.org cartao.xsd"` chamada do arquivo XSD.
    - O primeiro argumento é o namespace, e o segundo o arquivo
    - Perceba o uso do prefixo `xsi`
  - `elementFormDefault="qualified"`

Sintaxe

- Elementos *simples*
- Armazena apenas um dado (não tem sub-elementos)
  - `<element name="nome" type="tipodedado"/>`
- Exemplos de tipos de dados padrão:
  - `string`, `decimal`, `integer`, `boolean`, `date`, `time`, ...
- É possível definir tipos customizados.

Restrições

- É possível limitar valores usando restrições ou *facets*.
- Ex.

```
<element name="idade">
  <simpleType>
    <restriction base="integer">
      <minInclusive value="0"/>
      <maxInclusive value="120"/>
    </restriction>
  </simpleType>
</element>
```

Restrições

- Algumas possibilidades:
  - `length`, `minLength`, `maxLength`
  - `maxInclusive`, `maxExclusive`, `minInclusive`, `minExclusive`
  - `totalDigits`, `fractionDigits`
  - `pattern`
  - `enumeration`
  - `whiteSpace`

Restrições

```
<element name="percentual">
    <simpleType>
        <restriction base="string">
            <pattern value="([0-9]|[1-9][0-9]|100)%"/>
        </restriction>
    </simpleType>
</element>
```

```
<element name="marca">
  <simpleType>
    <restriction base="string">
      <enumeration value="Audi"/>
      <enumeration value="Fiat"/>
      <enumeration value="BMW"/>
    </restriction>
  </simpleType>
</element>
```

Tipos simples customizados

- Para possibilitar o reuso das restrições, é possível criar tipos customizados.

```
<element name="aproveitamento" type="percentual"/>

<simpleType name="percentual">
    <restriction base="string">
        <pattern value="([0-9]|[1-9][0-9]|100)%"/>
    </restriction>
</simpleType>
```

Exemplos

```
<element name="idade" type="limite_idade">
<simpleType name="limite_idade">
    <restriction base="integer">
        <minInclusive value="0"/>
        <maxInclusive value="120"/>
    </restriction>
</simpleType>
```

```
<simpleType name="tamanho">
    <restriction base="string">
        <pattern value="PP|P|M|G|GG"/>
    </restriction>
</simpleType>
```

Elementos *complexos*

- Permitem a definição de elementos com sub-elementos e atributos
- Podem ser vazios ou conter elementos, texto ou elementos e texto

```
<produto id="p123"/>

<funcionário>
    <nome>James Bond</nome>
    <matrícula>007</matrícula>
</funcionário>

<comida tipo="sobremesa">Sorvete</comida>
```

Elementos *complexos*

```
<element name="funcionário">
  <complexType>
    <sequence>
      <element name="matrícula" type="integer"/>
      <element name="nome" type="string"/>
    </sequence>
  </complexType>
</element>
```

Elementos *complexos* - reuso

```
<element name="funcionário" type="cadPessoa"/>
<element name="aluno" type="cadPessoa"/>

<complexType name="cadPessoa">
    <sequence>
        <element name="matrícula" type="integer"/>
        <element name="nome" type="string"/>
    </sequence>
</complexType>
```

Elementos *complexos* - extensão

```
<complexType name="cadPessoa">
    <sequence>
        <element name="matrícula" type="integer"/>
        <element name="nome" type="string"/>
    </sequence>
</complexType>

<complexType name="cadPessoaCompleto">
  <complexContent>
    <extension base="cadPessoa">
      <sequence>
        <element name="endereço" type="string"/>
        <element name="cidade" type="string"/>
        <element name="país" type="string"/>
      </sequence>
    </extension>
  </complexContent>
</complexType>
```

Cardinalidade

- `maxOccurs` e `minOccurs`
- `maxOccurs="unbounded"` para infinitas ocorrências

```
<complexType name="cadPessoa">
    <sequence>
        <element name="matrícula" type="integer"/>
        <element name="nome" type="string"/>
        <element name="telefone" type="string" minOccurs="0" maxOccurs="unbounded"/>
    </sequence>
</complexType>
```

Exemplo

- Crie um XML Schema básico para um prato (elemento raiz) que tenha os elementos:
  - `nome` (string com 100 caracteres no máximo)
  - `refeição` (`café`, `almoço` ou `janta`)
  - `ingredientes` com os sub-elementos `ingrediente` (pelo menos um)
  - `preço` (decimal com duas casas)
  - `dataCadastro` (data)
- Crie um XML que use o schema acima e valide no VS Code.

Atributos

- Elementos *simples* não tem atributos.
- Atributos são opcionais por padrão, é preciso utilizar `use="required"` para ser obrigatório.
- `<attribute name="nome" type="tipodedado" use="required"/>`

Elementos Vazios

- Descrevo um tipo complexo, mas não adiciono elementos

```
<element name="produto" type="tipoProduto"/>

<complexType name="tipoProduto">
  <attribute name="idProd" type="positiveInteger"/>
</complexType>
```

Elementos apenas texto (com atributos)

- Apenas elementos complexos podem ter atributos
- Declaro um elemento complexo, mas com conteúdo simples, e os atributos

```
<element name="tamanhoSapato" type="tipoSapato"/>

<complexType name="tipoSapato">
  <simpleContent>
    <extension base="integer">
      <attribute name="país" type="string" />
    </extension>
  </simpleContent>
</complexType>
```

```
<tamanhoSapato país="Brasil">42</tamanhoSapato>
```

Elementos mistos (texto, sub-elementos e atributos)

```
<element name="carta" type="tipoCarta"/>

<complexType name="tipoCarta" mixed="true">
  <sequence>
    <element name="nome" type="string"/>
    <element name="idPedido" type="positiveInteger"/>
    <element name="dataEnvio" type="date"/>
  </sequence>
</complexType>
```

```
<carta>
    Prezado <nome>Mr. Burns</nome>.
    Seu pedido <idPedido>23</idPedido> será enviado
    em <dataEnvio>2023-06-23</dataEnvio>.
</carta>
```

Indicadores

- *Indicadores* são utilizados para controlar como os elementos serão utilizados no documento.
- São sete:
  - Ordem: `all`, `choice`, `sequence`
  - Cardinalidade: `maxOccurs`, `minOccurs`
  - Agrupamento: `group name` e `attributeGroup name`

Indicadores de ordem

- `all` - os elementos podem aparecer em qualquer ordem, mas no máximo uma vez. `minOccurs` pode ser 0 ou 1.
- `choice` - apenas um dos elementos pode ocorrer.
- `sequence` - os elementos devem aparecer na sequência.

Exemplos

```
<element name="pessoa">
    <complexType>
        <all>
            <element name="nome" type="string"/>
            <element name="sobrenome" type="string"/>
        </all>
    </complexType>
</element>
```

```
<element name="pessoa">
    <complexType>
        <choice>
            <element name="funcionario" type="funcionario"/>
            <element name="membro" type="membro"/>
        </choice>
    </complexType>
</element>
```

Agrupamento

- `group name="nomedogrupo"` - definição de grupo. Deve conter um indicador de ordem.
- `attributeGroup name="nomedogrupo"` - grupo de atributos.

Exemplos

```
<group name="grupoPessoa">
    <sequence>
        <element name="nome" type="string"/>
        <element name="sobrenome" type="string"/>
        <element name="dataNascimento" type="date"/>
    </sequence>
</group>
```

```
<attributeGroup name="grupoAttrPessoa">
    <attribute name="nome" type="string"/>
    <attribute name="sobrenome" type="string"/>
    <attribute name="dataNascimento" type="date"/>
</attributeGroup>
```

`any` e `anyAttribute`

- Os elementos `any` e `anyAttribute` permitem que o usuário adicione qualquer elemento ou atributo no XML.
- Permite a extensão dos documentos, sem violar o XSD.

```
<element name="pessoa">
    <complexType>
        <sequence>
            <element name="nome" type="string"/>
            <element name="sobrenome" type="string"/>
            <any minOccurs="0"/>
        </sequence>
    </complexType>
</element>
```

Substituição de elementos

- Permite fazer com que dois elementos sejam equivalentes
- Útil para dar suporte a múltiplas línguas

```
<xs:element name="name" type="xs:string"/>
<xs:element name="nome" substitutionGroup="name"/>

<xs:complexType name="infoCliente">
  <xs:sequence>
    <xs:element ref="nome"/>
  </xs:sequence>
</xs:complexType>

<xs:element name="cliente" type="infoCliente"/>
<xs:element name="customer" substitutionGroup="cliente"/>
```

Tipos de implementação

- Aninhada
  - Estrutura lógica, porém dificulta o reuso e é mais complexa para manter
- Uso do atributo `ref`
  - Todos os elementos simples são definidos no início do código e os complexos depois, usando o atributo `ref` para referenciar os elementos simples.
- Definição de tipos customizados
  - Tipos customizados são definidos de maneira análoga a classes (POO) e depois os elementos/atributos são criados usando os tipos customizados.

Exemplos

- Nota Fiscal Eletrônica

Tarefa

- Crie um XSD `cardapio.xsd` para o XML da aula 05 (`cardapio.xml`), os requisitos são os seguintes:
  - Elemento raiz é `cardapio` e deve conter um ou mais elementos `prato`.
  - Elemento `prato` deve conter obrigatoriamente um atributo `id` do tipo `integer`;
  - Elemento `prato` deve conter os sub-elementos `nome`, `descricao`, `ingredientes`, `preco`, `calorias` e `tempoPreparo`;
  - Elemento `ingredientes` deve conter 1 ou mais sub-elementos `ingrediente`;
  - `calorias` deve ser `integer` e `preco` `decimal` com duas casas.
  - Os demais elementos são do tipo `string`;
- Valide seu XML com esse XSD

Tarefa

- Crie um XSD `imobiliaria.xsd` para os dados de uma imobiliária que cumpra os seguintes requisitos:
  - O elemento raiz é `imobiliária` e deve conter um ou mais elementos `imovel`.
  - O elemento `imovel` deve conter os sub-elementos `descricao`, `proprietario`, `endereco`, `caracteristicas` e `valor`.
  - O elemento `proprietario` deve conter os sub-elementos `nome` e os elementos `email` ou `telefone` são opcionais (pode ter mais de um).
  - O elemento `endereco` deve conter os sub-elementos `rua`, `bairro`, `cidade` e `número`, que deve ser opcional.
  - O elemento `caracteristicas` deve conter os sub-elementos `tamanho`, `numQuartos` e `numBanheiros`.
  - `número`, `numQuartos` e `numBanheiros` são `integer` e `tamanho` é `decimal`
  - O tipo de dado padrão para os demais elementos é `string`.
- Crie pelo menos 5 imóveis válidos.
  - Inclua pelo menos um proprietário com 2 telefones e um email e outro apenas com um telefone.
  - Teste também um imóvel sem número.

Tarefa

- Crie um XSD equivalente ao DTD quiz.dtd
- Use o VSCode para validar o XML criado para essa questão na aula 06.

Dúvidas?