eXtensible Markup Language

Versão 1.1

Última Atualização – 09-FEV-2005

1

Agradecimentos

Agradeço a minha esposa e filha para mim dar força e tempo para estudar. Agradeço a Wanda Belo por ter ajudado na correção de erros gramáticos e outras inconsistências. Finalmente, agradeço todos os estudantes que tem lido o texto e dado sugestões de melhoria.

Feedback

Envie seus comentários para tewarid@msn.com.

Conteúdo

Tópico 1 – XML bem Formada – Sintaxe XML
Tópico 2 – XML Válida – DTD e Esquema
Tópico 3 – Acesso Programático – DOM e SAX
Tópico 4 – Transformando XML – CSS e XSL

2

Pré-requisitos

É desejável ter noção de programação em alguma linguagem como VB ou Java. Conhecimento de desenvolvimento Web utilizando ASP ou JSP é desejável.

Software Recomendado

- Visual Basic
- Microsoft .NET Framework
- Java 2 SDK
- Eclipse ou Netbeans
- Altova XMLSpy
- Microsoft IIS

Bibliografia

- **Livros**
  1. **Professional XML – Wrox e Ciência Moderna – Várias Autores.**
  2. **XML Conceitos e Aplicações – Benoît Marchal – QUE e Berkeley.**
  3. **Aprendendo XML – O'Reilly e Campus – Erik T. Ray.**
- **Sites**
  1. **Site do W3C – http://www.w3.org.**
  2. **Oasis – http://www.oasis-open.org.**

3

A XML e padrões aliados estão sendo desenvolvidos rapidamente. Por isso a maioria dos livros se encontram bastante desatualizados. Às vezes os livros discutem comandos e características que não existem mais dentro do padrão. Recomendamos usar o texto dos padrões localizado no site http://www.w3.org como uma referência autêntica.

Tópico 1

Sintaxe da XML – XML Bem Formada

4

Objetivo

•Sintaxe da XML

•Como construir XML bem formada

O que é XML?

- **XML - eXtensible Markup Language**
- **Baseada em marcadores (tags) como HTML**

Por Exemplo:

```
<livros>
   <livro nome="Aprenda Java"></livro>
   <livro nome="Aprenda XML"></livro>
</livros>
```

5

O que é XML?

eXtensible Markup Language (XML) é uma linguagem textual baseada em marcadores (tags) como HTML.

Marcador (Tag)

Um marcador é um delimitador de dados em XML como mostra o exemplo a seguir:

```
<livros>
    <livro nome="Aprenda XML em 21 Dias">Um bom
livro.</livro>
    <livro nome="Aprenda Java">Um livro sobre Java.</livro>
</livros>
```

Aqui <livros>, </livros>, <livro> e </livro> são marcadores. Dentro do marcador <livro> temos um atributo chamado “nome”.

Porque XML?

- **XML é uma HTML melhorada?**
- **Nenhum marcador predefinido**
  - **Extensível**
- **Sintaxe mas rígida**
  - **Fácil validar**

6

XML é uma HTML melhorada?

HTML é muito complexa porque contém dezenas de marcadores de **dados** (como P, SUP e SUB), **estrutura** (como DIV e TABLE) e **formatação** (como B e FONT) tornando difícil a tarefa de separar os dados (conteúdo / informação) da formatação visual. XML é uma linguagem também baseada em marcadores, mas sem nenhum marcador predefinido. Cada aplicação pode definir seu próprio esquema de marcadores XML. O esquema pode ser estendido adicionando novos marcadores sem causar impacto na funcionalidade atual. A funcionalidade atual apenas deixará de processar os novos marcadores.

A sintaxe XML é mais rígida do que HTML. XML pode ser validada usando dois critérios:

•XML bem formada.

•XML válida - Utilizando uma DTD ou um Esquema (Schema).

Quem controla XML?

- **É um padrão aberto**
- **Padronizada pelo World Wide Web Consortium (W3C)**
- **Evoluiu da SGML (padrão da ISO - ISO8879)**
- **Recomendação do W3C desde Fevereiro de 1998**

7

História

Em 1969 pesquisadores da IBM Research inventaram a primeira linguagem de marcação chamada General Markup Language (GML). Seu propósito era ser uma meta-linguagem para descrever outras linguagens. Isso se tornou SGML e em 1986 a ISO adotou a SGML como padrão para troca e armazenagem de dados (ISO8879). HTML é uma aplicação da SGML e teve um impacto profundo sobre a internet.

Em 1996 o W3C começou o projeto de uma abrangente linguagem de marcação para combinar a flexibilidade e capacidade da SGML com a ampla aceitação da HTML e assim surgiu a XML. A aplicação mais conhecida da XML é a XHTML cujo objetivo é padronizar a sintaxe da HTML.

Uso de XML

- **Armazenagem de Documentos**
- **Intercâmbio de Dados**
  - **Entre base de dados e aplicação**
  - **Entre aplicações**
- **Armazenagem de Dados**
  - **Base de dados**
  - **Arquivos de configuração**

8

Armazenagem de Documentos

XML está cada vez mais presente nas aplicações de produtividade. Aplicações como OpenOffice (http://www.openoffice.org) adotaram XML como a linguagem padrão para armazenagem de documentos.

Intercâmbio de dados

Padrões como Simple Object Access Protocol (SOAP) estão aplicando a XML para intercâmbio de dados entre aplicações. Também está cada vez mas comum a base de dados devolver dados de uma consulta no formato XML (como SQL Server e DB2).

Armazenagem de Dados

Estão aparecendo no mercado bancos de dados que armazenam dados em XML (Tamino da Software AG). Muitos softwares armazenam suas configurações em arquivos XML (como Servidores J2EE e .Net).

Outros Padrões Relacionados a XML

- **Espaços identificadores (namespaces)**
- **XPATH**
- **Folhas de Estilo (style-sheets)**
  - **CSS**
  - **XSL**
- **DOM e SAX**
- **XLink e XPointer**

9

Outros Padrões Relacionados a XML

Espaços Identificadores

•Usados para qualificar elementos e atributos XML.

XPATH

•Usado para referenciar e selecionar elementos e atributos no documento XML.

Folhas de Estilos

•CSS (Folha de estilo em cascata)

- •Usado para formatar elementos XML para visualização.

•XSL (Folha de estilo estendido)

- •Usado para transformar documento XML para HTML, texto, XSL-FO ou XML. É mais complexo do que CSS.

DOM e SAX

•Fornecem acesso programático aos documentos XML. Document Object Model (DOM) é mais complexo do que Simple API For XML (SAX). SAX é mais leve e baseada em eventos.

XLink e XPointer

•Usados para vincular diferentes documentos XML.

Trabalhando com XML

- **Browser XML**
- **Editor XML**
- **Parser XML**
- **Processador de XSL**

10

Browser XML

Usado para visualizar documentos XML. Geralmente mostra o documento XML como uma árvore de marcadores. Também permite utilizar um arquivo de folha de estilo em cascata (CSS / XSL) para formatar elementos. Exemplos são Internet Explorer e Mozilla.

Editor XML

Usado para editar e validar um documento XML. Exemplos são XMLSpy e Microsoft XML Notepad.

Parser XML

Usado para ler e validar documentos XML usando uma linguagem de programação. Exemplos são MSXML da Microsoft e XERCES da Apache.

Processador XSL

Usado para transformar um documento XML para um outro documento (HTML, XML, Texto, ...). Exemplos são MSXML da Microsoft e XALAN da Apache.

Estrutura de um Documento XML

- **A declaração “xml”**
- **Elementos**
- **Atributos**
- **Comentários**

11

A declaração “xml”

Opcionalmente colocado no início de um documento XML. Contém dois principais atributos: “version” informa a versão do padrão XML e “encoding” o padrão de codificação de caracteres (como UTF-8, UTF-16 e ISO-8859-1). Exemplo:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
```

Elementos

Definem a estrutura do documento XML. Cada elemento começa com um marcador de início e termina com um marcador de fim. Por exemplo, o marcador “<livro>” marca início do elemento “livro” e o marcador “</livro>” marca seu fim:

```
<livro nome="Programação SQL">Um livro bem didático</livro>
```

Atributos

Dentro do marcador de início de cada elemento podemos ter vários atributos. No exemplo acima “nome” é um atributo.

Comentários

Comentários podem ser colocados entre “<!--” e “-->” como em HTML.

Entidades XML

- **Usadas para criar dados predefinidos**
- **A ocorrência de uma entidade no documento XML é substituída pelo seu valor pelo parser**
- **A ocorrência começa com "&" e termina com ";"**
- **A XML utiliza entidades para representar alguns caracteres reservados pela linguagem, como ">" (>) e "<" (<)**
- **Novas Entidades podem ser criadas usando uma DTD**

12

Exemplo de uso de Entidades

Isto está errado:

```
<empresa>Marcos & Manoel</empresa>
```

Isto está certo:

```
<empresa>Marcos &amp; Manoel</empresa>
```

Algumas Entidades predefinidas

<	<
>	>
&	&
'	'
"	"

Observação

XML também suporta substituição de um caractere pelo seu valor numérico, por exemplo, a letra î pode ser substituída pelo "î". Para colocar o valor em hexadecimal basta usar um caractere "x" depois do caractere "#".

Atributos Especiais

- **xml:space**
  - **Preservar ou não espaços em branco duplicados.**
- **xml:lang**
  - **Linguagem natural do valor texto de um elemento.**

13

XML permite a utilização de dois atributos especiais para informar como o texto dos elementos deve ser tratado.

xml:space

Usado para avisar as aplicações se o espaço em branco duplicado deve ser preservado ou não. Neste exemplo:

```
<livro xml:space="preserve">Um   livro
muito bom</livro>
```

O parser vai preservar os espaços em branco a mais entre “Um” e “livro” e não vai descartar a quebra de linha depois da palavra “livro”.

xml:lang

Diz para a aplicação qual é a linguagem do texto dentro do elemento XML. Por, exemplo:

```
<livro xml:lang="pt-BR">Um livro excelente</livro>
<livro xml:lang="en-US">An excellent book</livro>
```

Instruções de Processamento

- **Podem ser usadas para associar arquivos CSS e XSL**
- **Iniciam com “<?” e terminam com “?>”**
- **Depois de “<?” aparece a aplicação destinatária**
- **Depois da aplicação aparece o texto da instrução**

Exemplo:

```
<?xml-stylesheet href="teste.xsl" type="text/xsl"?>
```

14

Instruções de processamento são usadas para estender a linguagem XML. No exemplo a seguir, demonstramos como especificar o documento XSL a ser utilizado para transformar um documento XML:

```
<?xml-stylesheet href="teste.xsl" type="text/xsl"?>
```

Aqui xml-stylesheet representa um gênero de aplicações que transformam XML usando um arquivo XSL. O local do arquivo XSL é especificado usando o atributo “href” e o tipo mime do arquivo é especificado usando o atributo “type”.

Instruções de processamento também podem ser utilizadas por editores XML especializados como XMetal para indicar texto substituível [Benoît Marchal]. Por exemplo:

```
<?xm-replace_text {texto qualquer sem marcação}?>
```

Seções CDATA

- **CDATA significa dados caractere**
- **É usada para colocar texto qualquer como scripts de código (como VBScript ou JScript) dentro da XML**
- **É delimitada pelo "<![CDATA[" e "]]>"**
- **Uma seção CDATA não pode ser aninhada dentro da outra**

15

Exemplo da Seção CDATA

Neste exemplo colocamos um documento XML embutido usando uma seção CDATA:

```
<catalogo>
  <livros>
<![CDATA[
Exemplo de um livro:
<livro>
  <nome>Java</nome>
  <autor>Gosling</autor>
</livro>
]]>
  </livros>
</catalogo>
```

O parser vai ignorar marcadores XML embutidos dentro da seção CDATA. Não é possível aninhar uma seção CDATA dentro da outra.

XML Bem Formada

- **Condições:**
  - **Apenas um elemento raiz**
  - **Elementos sem sobreposição**
  - **Início e fim de elementos corretamente demarcado**
  - **Nome de elementos e atributos corretamente formado**
  - **Valor de atributo demarcado com aspas duplas**
  - **Texto caractere sem caracteres reservados ou inválidos**

16

É fundamental que cada elemento XML tenha um marcador inicial e um marcador final. Um marcador deveria começar com “<” e terminar com “>”. O marcador final deveria ter um “/” antes do nome do elemento. Um elemento pode conter vários elementos filhos mas um documento XML pode conter apenas um elemento raiz. Os elementos não podem ser sobrepostos e devem ser completamente aninhados dentro de outros elementos.

Nome dos elementos e atributos são sensíveis ao caso. Eles não podem conter espaços em brancos, iniciar com números, ou conter caracteres reservados.

•Alguns nomes válidos:

Livro, LIVRO, moderna:catalogo, _livro, nome.completo, ...

•Alguns nomes inválidos:

-livro, 1Livro, livro$, livro&revista, ...

Um elemento não pode conter dois atributos com o mesmo nome. O valor do atributo deveria estar contido dentro de aspas duplas, por exemplo, nome="Java"

Um documento XML não será considerado bem formado se quebrar qualquer uma das regras especificadas acima. Editores como XMLSpy e Netbeans e muitos parsers podem verificar essas regras.

Projetando Documentos XML

- **Identificar nomes para coisas e conceitos**
- **Identificar hierarquia das coisas**
  - **Identificar os relacionamentos entre coisas**
- **Definir propriedades das coisas**
- **Isso é modelagem orientado objeto**
  - **Documentos XML podem ser gerados exportando uma hierarquia de objetos**

17

Projetando Documentos XML

Coisas e conceitos de um sistema de informação são representados pelo diferentes tipos de objetos. Cada tipo pode ser traduzido para um tipo de elemento em XML.

Os relacionamentos entre tipos de objetos podem ser representados em XML usando elementos aninhados, por exemplo, elementos do tipo “livro” podem ser colocados dentro de um elemento “catalogo”.

Para representar propriedades em XML podemos usar elementos filhos ou atributos.

Atributos ou Elementos Filhos

	Vantagens	Desvantagens
Atributos	•**Possível restringir valor usando DTD** •**Validação de ID e IDREF** •**Requer menos espaço** •**Fácil processar usando DOM e SAX**	•**Valores simples** •**Não suporta atributos sobre atributos**
Elementos filho	•**Suporta valores complexos** •**Suporta atributos sobre atributos** •**Extensível quando modelo de dados muda (um livro vários autores)**	•**Requer mais espaço** •**Mais difícil processar**

18

A escolha entre o uso de atributos ou elementos filhos para representar propriedades é uma escolha difícil. Historicamente, elementos têm contido texto e os atributos, meta-dados sobre o texto. É assim que HTML utiliza os dois.

Geralmente é mais fácil incorporar mudanças quando usamos elementos filhos. Por exemplo, uma propriedade “autor” pode ser repetida várias vezes se for representada usando um elemento. Por outro lado é mais fácil processar atributos quando usamos parsers DOM e SAX.

Exercício 1

1. **Como utilizar XMLSpy?**
2. **Crie um catálogo de livros usando XML**
    - **Sugira um modelo para o documento**
    - **Crie o documento usando o modelo**

19

XMLSpy

XMLSpy é um editor avançado de XML e pode ser utilizado para editar e testar documentos XML, DTD, XSD, XSL, XHTML, etc. Vamos usar o XMLSpy no modo de visualização de texto para editar nossos documentos. Existem outros modos para editar os documentos, mas esses fogem do escopo deste curso.

Tópico 2

XML Válida – DTD e Esquemas

20

Objetivo

•Porque DTD?

•Definindo a estrutura de documentos XML usando DTD.

•Validando documentos XML usando DTD.

•Entendendo espaços identificadores.

•Problema com DTD.

•Definindo estrutura de documentos XML usando Esquema.

•Validando documentos XML usando Esquema.

Porque Usar DTD?

- **DTD significa Definição do Tipo de Documento**
- **XML bem formada não basta**
  - **Precisa haver uma forma de definir o vocabulário do documento XML com precisão**
  - **Parsers XML podem usar a DTD para validação de uma instância do documento XML**
  - **Editores de XML (como XMLSpy) podem usar a DTD para fazer valer as regras contidas nele na hora de editar um documento XML**

21

A DTD foi a forma encontrada para garantir a validade de um documento XML. Um documento XML é válido se segue as regras especificadas na DTD. A DTD tem fortes vínculos históricos com a SGML.

Associando DTD com Documento XML

- **Usando a instrução de processamento DOCTYPE**
- **Duas formas**
  - **DTD Interna**
    - **DTD fica dentro do documento XML**
  - **DTD Externa**
    - **DTD fica num arquivo separado**
    - **Facilmente aplicada a vários documentos XML**

22

Exemplo – DOCTYPE

```
<!DOCTYPE catalogo SYSTEM "catalogo.dtd">
```

Aqui estamos estabelecendo “catalogo” como o elemento raiz. SYSTEM indica a forma padrão de acesso a uma DTD externa usando um protocolo como HTTP ou sistema de arquivo. SYSTEM pode ser substituído pelo PUBLIC para indicar uma forma de acesso próprio do aplicativo a uma DTD catalogada.

Exemplo de DTD Interna

```
<!DOCTYPE catalogo [
    <!ENTITY COPYRIGHT "2001 Empresa ABC">
    <!ELEMENT catalogo (propriedade)>
    <!ELEMENT propriedade (#PCDATA)>
]>
<catalogo>
    <propriedade>&COPYRIGHT;</propriedade>
</catalogo>
```

Construtos de DTD

- **Sintaxe da DTD não utiliza XML**
- **Uma DTD contém os construtos a seguir:**
  - **ELEMENT**
  - **ATTLIST**
  - **ENTITY**
  - **NOTATION**

23

Construtos da DTD

ELEMENT

Usado para declarar um novo tipo de elemento XML.

ATTLIST

Usado para declarar os atributos que podem ser designados a um tipo de elemento XML.

ENTITY

Usado para declarar informação (uma entidade) reutilizável.

NOTATION

Usado para associar um aplicativo externo ao conteúdo que não pode ser analisado (informação binária por exemplo).

Declaração ELEMENT

- **Declarar um novo tipo de elemento XML**
  - **"<!ELEMENT" seguido pelo nome do elemento**
  - **Nome tem que ser identificador XML válido**
  - **Nome seguido pela definição de conteúdo**

Exemplos:

1. **<!ELEMENT livro (#PCDATA)>**
2. **<!ELEMENT autor (titulo, nome)>**
3. **<!ELEMENT autor ANY>**

24

Declarar Tipos de Elementos

Um tipo de elemento é declarado dentro de "<!ELEMENT" e ">". O nome do elemento deveria ser um nome XML válido. O nome é seguido pela especificação de conteúdo. O conteúdo de um elemento pode ser:

Vazio

Especificado usando a palavra EMPTY. Indica que um elemento não tem conteúdo. Por exemplo, <!ELEMENT autor EMPTY>

Texto

Especificado usando a palavra PCDATA, por exemplo, <!ELEMENT autor (#PCDATA)>.

Elementos

Um elemento contém outros elementos filhos mais nenhum texto. Especificado usando modelo de conteúdo, por exemplo, <!ELEMENT livro (#PCDATA | autor)*>.

Misto – Texto e Elementos

Um elemento contém texto e elementos. Indicado usando um modelo de conteúdo, por exemplo, <!ELEMENT autor (#PCDATA | nome)>.

Qualquer

Caso não queira especificar conteúdo específico use a palavra chave ANY, por exemplo, <!ELEMENT autor ANY>. Qualquer elemento declarado na DTD poderá ser utilizado dentro do elemento "autor".

Declaração ELEMENT – Modelo de Conteúdo

- **Para especificar conteúdo dentro de um elemento**
- **Contém uma combinação de:**
  - **Texto – indicado usando #PCDATA**
  - **Elementos filhos**
  - **Operadores**

Exemplos:

1. `<!ELEMENT livro (#PCDATA | autor)*>`
2. `<!ELEMENT livro (autor+, custo?)>`
3. `<!ELEMENT livro (custo, (autor | editora)+)>`

25

Modelo de Conteúdo

Um modelo de conteúdo declara a estrutura dos elementos e consiste de uma combinação de elementos filhos, texto (indicado usando #PCDATA) e operadores. Os operadores são:

, (vírgula) uma seqüência estrita de elementos filhos.

| (barra) Um ou mais elementos filhos em qualquer ordem.

Parênteses podem ser utilizados para misturar os operadores como mostra o exemplo 3 acima.

Cardinalidade

É possível especificar a cardinalidade de elementos filhos no modelo de conteúdo de um elemento usando operadores de cardinalidade. Quando o conteúdo é misto a cardinalidade só pode ser especificada como mostra o exemplo 1 acima. Os operadores de cardinalidade são:

?	Opcional, pode ou não aparecer o elemento filho
*	Zero ou mais
+	Um ou mais

Quando não for especificada, a cardinalidade é Um.

Declaração ATTLIST

- **Atributos são propriedades de um elemento**
- **Declarados usando a sintaxe**

```
<!ATTLIST elemento atributo tipo uso>
```

- **Exemplo**

```
<!ATTLIST autor
   primeiro CDATA #REQUIRED
   ultimo CDATA #REQUIRED
  conveniado CDATA #FIXED "sim"
>
```

26

Atributos

Atributos complementam e modificam elementos, fornecendo um meio de associar propriedades aos elementos. Todos os atributos de um elemento são declarados dentro do mesmo ATTLIST como no exemplo acima.

Tipos de Atributos

O tipo geralmente utilizado é **CDATA** (dados caractere) mas pode ser **ID**, **IDREF**, **IDREFS**, **ENTITY**, **ENTITIES**, **NMTOKEN**, **NMTOKENS**, **NOTATION**, ou uma **opção de valores**.

Uso de Atributos

É possível especificar o uso de um atributo dentro de um elemento. O uso pode ser:

#REQUIRED	Deve aparecer.
#IMPLIED	Pode ou não aparecer dentro de um elemento.
#FIXED	Vai aparecer e tem um valor fixo.

Atributos tipo ID, IDREF e IDREFS

- **ID é usado para definir um nome único no documento**
  - **Uso de um ID é sempre #REQUIRED**
- **IDREF refere-se a um ID declarado no documento**
  - **Usado para criar relacionamentos um para um**
- **IDREFS refere-se a vários ID ao mesmo tempo**
  - **Usado para criar relacionamentos um para vários**

Exemplo

1. `<!ATTLIST livro nome ID #REQUIRED>`
2. `<!ATTLIST livro tipo IDREF #REQUIRED>`

27

Exemplo IDREF e IDREFS

```
<!ELEMENT livro EMPTY>
<!ELEMENT autor EMPTY>
<!ATTLIST livro
    nome ID #REQUIRED
    tipo IDREF #REQUIRED
    autores IDREFS #REQUIRED
>
<!ATTLIST autor
    codigo ID #REQUIRED
    nome CDATA #REQUIRED
>
<!ELEMENT tipo_livro EMPTY>
<!ATTLIST tipo_livro
    tipo ID #REQUIRED
>
```

Aqui um livro está definido com três atributos: nome, tipo e autores. O atributo autores é um IDREFS. Isso significa que no documento XML o valor do atributo vai conter códigos de vários autores separados com espaço em branco, por exemplo, autores="a1 a2".

Atributos tipo ENTITY e ENTITIES

- **Apontam para Entidades definidas no mesmo documento usando declaração ENTITY**
- **ENTITIES é uma lista de ENTITY separadas por espaços**
- **Exemplo**

```
<!ENTITY textoCopyright "(c) Livraria Cataloga">
<!ATTLIST livro
  copyright ENTITY #IMPLIED
>
```

O documento XML vai conter

```
<livro copyright="textoCopyright">
```

28

Declaração ENTITY

Atributos do tipo ENTITY apontam para texto especificado usando a declaração ENTITY.

A declaração ENTITY tem duas formas:

Entidades gerais

São usadas para criar texto substituível. No exemplo acima `textoCopyright` é uma entidade que pode ser usada dentro de qualquer texto (elemento com conteúdo #PCDATA ou atributo tipo CDATA) colocando **`&textoCopyright;`**

Entidades de parâmetros

São usadas para criar uma lista de construções reutilizáveis que são avaliadas sintaticamente.

Exemplo

```
<!ENTITY % parametrosNome "primeiro CDATA #REQUIRED ultimo
CDATA #REQUIRED">
<!ELEMENT autor EMPTY>
<!ATTLIST autor
    codigo ID #REQUIRED
    %parametrosNome;
>
```

Atributos tipo NMTOKEN e NMTOKENS

- **NMTOKEN = Fichas Nomeadas**
- **Os atributos desse tipo são atribuídos fichas válidas**
- **O processamento e checagem de validade das fichas fica por conta da aplicação**

Exemplo

```
<!ELEMENT livro EMPTY>
<!ATTLIST livro
   nome CDATA #REQUIRED
   tipo IDREF #REQUIRED
   autores IDREFS #REQUIRED
   copyright ENTITY #IMPLIED
   editora NMTOKEN #REQUIRED
>
```

29

No exemplo acima “editora” é um atributo do elemento “livro”. No documento XML o atributo deveria ser atribuído o nome de uma ficha válida. Uma ficha válida é um nome XML válido. Segue exemplo de um documento XML que utiliza a DTD acima:

```
<livro autores="a1 a2" tipo="historia" nome="Historia do
Mundo" copyright="copyright" editora="E0001"/>
```

Nesse exemplo “E0001” representa o código da editora. A informação sobre a editora pode estar gravada numa base de dados ou outro arquivo XML.

Isto seria inválido:

```
<livro autores="a1 a2" tipo="historia" nome="Historia do
Mundo" copyright="copyright" editora="0001"/>
```

Usar atributos do tipo NMTOKENS para associar várias NMTOKEN (ficha). As fichas devem estar separadas por espaços.

Atributo tipo NOTATION

- **NOTATION significa Notação**
- **O atributo desse tipo é usado em conjunto com a declaração NOTATION (notação)**
- **Uma notação declara um formato e associa um aplicativo externo para processar o formato**

Exemplo

```
<!NOTATION jpg SYSTEM "jpgviewer.exe">
<!NOTATION gif SYSTEM "gifviewer.exe">
<!ELEMENT livro (imagem)>
<!ELEMENT imagem (#PCDATA)>
<!ATTLIST imagem
    tipo NOTATION (gif | jpg) "gif"
>
```

30

No exemplo acima “tipo” é um atributo tipo NOTATION do elemento “imagem”. No documento XML vamos poder utilizar valores “gif” ou “jpg” para esse atributo. Se não especificarmos o atributo, o valor “gif” será assumido.

Os dados contidos dentro do elemento “imagem” podem ser dados binários (da imagem do livro) codificados como texto (usando Base64, por exemplo) e serão processados por aplicativos externos “jpgviewer.exe” ou “gifviewer.exe”.

Atributos tipo “opção de valores”

No exemplo acima especificamos a opção de valores possíveis para o atributo “tipo". Isso pode ser feito para qualquer atributo como mostra o exemplo a seguir:

```
<!ATTLIST livro
    grau (primeiro | segundo | terceiro | superior) #REQUIRED
>
```

DTD Interna

- **Elementos declarados na DTD interna sobrepõem elementos da DTD externa**
- **DTD interna é pouca utilizada porque:**
  - **Ocupa espaço em cada documento**
  - **Difícil mudar por estar em vários documentos XML**

31

Exemplo de DTD Interna

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE endereco [
<!ELEMENT endereco
(rua,complemento?,bairro,cidade,cep,uf,pais)>
<!ATTLIST endereco preferido (true | false) "false">
<!ELEMENT rua (#PCDATA)>
<!ELEMENT numero (#PCDATA)>
<!ELEMENT complemento (#PCDATA)>
<!ELEMENT bairro (#PCDATA)>
<!ELEMENT cidade (#PCDATA)>
<!ELEMENT cep (#PCDATA)>
<!ELEMENT uf (#PCDATA)>
<!ELEMENT pais (#PCDATA)> ]>
<endereco preferido="true">
    <rua>Av. Boa Viagem</rua>
    <numero>128</numero >
    <complemento>Apt° 101</complemento>
    <bairro>Boa Viagem</bairro>
    <cidade>Recife</cidade>
    <cep>50000-000</cep>
    <uf>PE</uf>
    <pais>Brasil</pais>
</endereco>
```

Exemplo Escola

- **Criar DTD para representar dados de uma escola**
- **Coisas**
  - **Escola**
  - **Estudante**
  - **Professor**
  - **Aula**

32

```
<!ELEMENT escola (professores, estudantes, aulas)>
<!ATTLIST escola
             nome CDATA #REQUIRED
>
<!ELEMENT professores (professor+)>
<!ELEMENT professor EMPTY>
<!ATTLIST professor
             codigo ID #REQUIRED
             nome CDATA #REQUIRED
>
<!ELEMENT estudantes (estudante+)>
<!ELEMENT estudante EMPTY>
<!ATTLIST estudante
             codigo ID #REQUIRED
             nome CDATA #REQUIRED
>
<!ELEMENT aulas (aula+)>
<!ELEMENT aula (estudante_aula+)>
<!ATTLIST aula
             professor IDREF #REQUIRED
             materia (Matemática | Física | Química) #REQUIRED
>
<!ELEMENT estudante_aula EMPTY>
<!ATTLIST estudante_aula
             codigo IDREF #REQUIRED
>
```

Limitação de DTD

- **Difícil de escrever e entender**
- **Difícil processar programaticamente**
- **Difícil de estender**
- **Sem suporte para espaços identificadores (namespace)**
- **Nenhum suporte para tipos de dados**
- **Nenhum suporte para herança**

33

Limitação de DTD

•Difícil de escrever e entender

Por seguir a Extended Backus Naur Form (EBNF).

•Difícil processar programaticamente

Isso torna difícil a tarefa de usar DTD para validar documentos XML. Se fosse em XML talvez fosse mais fácil criar e utilizar DTD.

•Difícil de estender

Por conter toda sintaxe dentro de um mesmo documento, é necessário copiar o documento inteiro para estender qualquer declaração.

•Sem suporte para espaços identificadores (namespace)

Espaços identificadores permitem qualificar tipos com mesmo nome.

•Nenhum suporte para tipos de dados

Não tem suporte para outros tipos de dados além de texto.

•Nenhum suporte para herança

Por isso é difícil derivar uma declaração de outra existente.

Exercício 2

1. **Crie uma DTD para o documento XML de catálogo de livros criado no Exercício 1**

34

Use XMLSpy para criar a DTD.

O que é Esquema XML?

- **Um esquema XML contém**
  - **Definição de tipos**
  - **Declaração de elementos**
- **Permite**
  - **Validar elementos e atributos XML e seus valores**
  - **Estender um esquema existente**
- **É uma recomendação do W3C**

35

Vantagens de usar Esquemas

•Arquivos esquemas são arquivos XML.

•Fácil de estender e reutilizar esquemas existentes.

•Suporte para espaços identificadores (namespaces).

•Suporte para diversos tipos de dados.

•Suporte para herança.

Componentes de Esquema – Componentes Primários

- **Definição de Tipos Simples**
- **Definição de Tipos Complexos**
- **Declaração de Atributos**
- **Declaração de Elementos**

36

Componentes Primários

Definição de Tipos Simples

Um tipo simples é definido usando o elemento “xs:simpleType” e pode ser associado a atributos e elementos com texto, mas sem elementos filhos.

Definição de Tipos Complexos

Um tipo complexo é definido usando o elemento “xs:complexType” e pode ser associado com elementos.

Declaração de Atributos

Um atributo é declarado usando o elemento “xs:attribute”. Podemos especificar o nome, uso e tipo do atributo no elemento.

Declaração de Elementos

O elemento “xs:element” pode ser usado para declarar elementos. Dentro do elemento podemos definir o tipo dele contendo declarações de outros elementos e atributos. Usando o atributo “type” podemos atribuir um tipo já definido.

Observação

Definição

“Definição” tem o mesmo sentido da palavra “criação”. Um tipo é criado dentro de um esquema, mas um elemento ou atributo não.

Declaração

Um elemento ou atributo é “declarado” dentro de um esquema, mas definido dentro do arquivo XML que utiliza o esquema.

Componentes de Esquema – Componentes Secundários

- **Declaração de Notações**
- **Definição de Grupos de Atributos**
- **Definição de Restrições de Identidade**
- **Definição de Grupos de Modelos**

37

Componentes Secundários

Declaração de Notações

São usados para especificar um processador externo para tipos de dados não XML. O elemento “xs:notation” é usado para criar uma notação.

Definição de Grupos de Atributos

Usados para definir um conjunto de atributos que podem ser reutilizados em vários declarações. O elemento “xs:attributeGroup” é usado para criar um grupo de atributo.

Definição de Restrições de Identidade

Usados para especificar chave primária, única ou estrangeira.

Definição de Grupos de Modelos

Podemos associar um grupo de modelo a um nome usando a definição de grupo de modelos. Assim o grupo de modelo pode ser reutilizado em vários lugares dentro do esquema. O elemento “xs:group” é usado para criar uma definição de grupos de modelos.

Componentes de Esquema – Componentes Auxiliares

- **Anotações**
- **Grupos de Modelos**
- **Partículas**
- **Curingas (wildcards)**
- **Uso de Atributos**

38

Componentes Auxiliares

Anotações

Podem ser usadas para associar texto de documentação ao modelo. Definidas usando o elemento “xs:annotations”.

Grupos de Modelos

Usados para definir uma seqüência de elementos dentro de uma definição de tipo complexo. Pode ser definido usando elementos “xs:all”, “xs:choice”, ou “xs:sequence”.

Partículas

Partícula é um termo semântico para conteúdo de elementos composta de declarações de elementos, curingas e grupos de modelos, juntamente com restrições de ocorrência.

Curingas (wildcards)

São partículas especiais que validam definição de elementos e atributos baseado no espaço identificador.

Uso de Atributos

É usado para definir o uso do atributo. Definido usando o atributo “use” dentro do elemento “xs:attribute”.

Estrutura do Arquivo de Esquema

- **O arquivo tem a extensão xsd**
- **Contém um elemento raiz chamado “schema”**
- **Elemento “schema” pode conter atributos:**
  - **attributeFormDefualt**
  - **elementFormDefault**
  - **id**
  - **targetNamespace**
  - **version**
  - **xmlns (mais de uma ocorrência)**

39

O que é esquema?

Um esquema é representado em XML por um, ou mais de um, documento de esquema com a extensão “xsd”. Um documento de esquema tem um elemento raiz chamado “schema” que contém vários componentes como definições de tipos e declarações de elementos todos pertencendo o mesmo espaço identificador especificado pelo atributo “targetNamespace”.

Exemplo do elemento xs:schema

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified" attributeFormDefault="unqualified"
version="1.0" id="catalogo" xmlns="http://www.exemplo.com/catalogo"
targetNamespace="http://www.exemplo.com/catalogo">
    ...
</xs:schema>
```

Importação

É possível importar outros esquemas dentro de um esquema usando o elemento xs:import, por exemplo, `<xs:import schemaLocation="exemplo.xsd"/>`. Isso permite a reutilização de tipos entre esquemas.

Associando Esquema a um Documento XML

- **Um documento XML pode ser considerado como uma instância de um esquema**
- **O elemento raiz do documento XML deve declarar o espaço identificador ao qual ele pertence e isso deve corresponder ao espaço identificador especificado pelo atributo “targetNamespace” no esquema**
- **O elemento raiz deve declarar o espaço identificador “http://www.w3.org/2000/10/XMLSchema-instance” e atribuir para o atributo “schemaLocation” deste espaço identificador o nome do arquivo contendo o esquema**

40

Exemplo

Se o documento esquema chamado “catologo.xsd” contiver:

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="unqualified"
attributeFormDefault="unqualified" version="1.0"
id="catalogo" xmlns="http://exemplo/abc"
targetNamespace="http://exemplo/abc">
  <xs:element name="e">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="e1"/>
        ...
```

Então o documento XML vai conter:

```
<abc:e xmlns:abc="http://exemplo/abc"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://exemplo/abc
abc.xsd">
  <e1/>
  ...
```

Se o valor do atributo “elementFormDefault” for “qualified” então o mesmo documento XML vai conter:

```
<abc:e xmlns:abc="http://exemplo/abc"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://exemplo/abc
abc.xsd">
  <abc:e1/>
  ...
```

Espaço Identificadores (namespaces)

- **Evitam ambigüidade e colisão de nomes quando um elemento XML utilizar dois esquemas diferentes**
- **O elemento declara a utilização de um esquema usando o atributo “xmlns” ou a forma “xmlns:prefixo”**
- **O prefixo é utilizado no nome de todos os elementos declarados no esquema**
- **O espaço identificador é identificado utilizando um URI (identificador universal de recursos)**
- **Geralmente o URI é um URL**

41

Exemplos

1. Declarar um espaço identificador.

```
xmlns="http://www.w3.org/2001/XMLSchema"
```

2. Fornecer um nome alternativo para o espaço identificador. No exemplo seguinte “xs” é um nome alternativo que será prefixado aos nomes de elementos e atributos declarados no esquema “`http://www.w3.org/2001/XMLSchema`”.

```
xmlns:xs="http://www.w3.org/2001/XMLSchema"
```

Observação

Namespace é uma recomendação do W3C desde Janeiro de 1999. Mais informações estão contidas na página http://www.w3.org/TR/1999/REC-xml-names-19990114/

Definição de Tipo Simples

- **Pode ser usada na declaração de:**
  - **Atributos**
  - **Elementos simples contendo apenas texto**

Por Exemplo

```
<xs:simpleType name="tipoTemperaturaAgua">
    <xs:restriction base="xs:number">
        <xs:minExclusive value="0.00"/>
        <xs:maxExclusive value="100.00"/>
    </xs:restriction>
</xs:simpleType>
```

42

A definição de tipo simples só pode ser usada com atributos e elementos simples contendo apenas texto. Cada definição de tipo simples é uma restrição aplicada sobre um outro tipo simples. A restrição pode especificar uma expressão regular usando o elemento “xs:pattern” para validar valor texto. Podemos definir tipos simples que representam uma lista de valores de um tipo simples (xs:list), ou cujos valores são uma união (xs:union) de tipos simples, por exemplo:

```
<xs:simpleType name="tipoTamanho">
    <xs:union>
        <xs:simpleType>
            <xs:restriction base="xs:positiveInteger">
                <xs:minInclusive value="8"/>
                <xs:maxInclusive value="72"/>
            </xs:restriction>
        </xs:simpleType>
        <xs:simpleType>
            <xs:restriction base="xs:NMTOKEN">
                <xs:enumeration value="pequeno"/>
                <xs:enumeration value="médio"/>
                <xs:enumeration value="grande"/>
            </xs:restriction>
        </xs:simpleType>
    </xs:union>
</xs:simpleType>
```

Restringindo Valores Usando Expressões Regular

- **Podemos restringir valores utilizando expressões regular (regular expressions)**
- **Uma expressão regular é especificada utilizando o elemento “xs:pattern” dentro do elemento “xs:restriction”**
- **Demonstrado embaixo utilizando o valor do CPF como exemplo**
  - **Observe o uso do atributo “memberTypes” dentro do elemento “xs:union”**

43

Esquema - cpf.xsd

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="cpf">
        <xs:simpleType>
            <xs:union memberTypes="tipoCPF tipoCPFFormatado"/>
        </xs:simpleType>
    </xs:element>
    <xs:simpleType name="tipoCPF">
        <xs:restriction base="xs:string">
            <xs:pattern value="[0-9]{11}"/>
        </xs:restriction>
    </xs:simpleType>
    <xs:simpleType name="tipoCPFFormatado">
        <xs:restriction base="xs:string">
            <xs:pattern value="[0-9]{3}\.[0-9]{3}\.[0-9]{3}-[0-9]{2}"/>
        </xs:restriction>
    </xs:simpleType>
</xs:schema>
```

Vamos ver um documento XML válido que utiliza o esquema.

Documento XML - cpf.xml

```
<cpf xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="cpf.xsd">011.077.604-61</cpf>
```

Definição de Tipo Complexo

- **Usada para declarar elementos contendo:**
  - **Atributos**
  - **Tipo do Conteúdo (elementos filhos)**
- **Tipos complexos podem estender outros tipos simples ou complexos**

44

Um tipo complexo é criado usando o elemento “xs:complexType” e pode ser usado na declaração de elementos. A definição de um tipo complexo contém uma série de atributos e tipo de conteúdos (elementos filhos). O tipo de conteúdo determina se o elemento associado:

-contém ou não elementos filhos

-tipo dos elementos filhos

-contém ou não texto caractere

Cada tipo complexo é uma restrição ou extensão de um outro tipo simples ou complexo especificado usando os elementos “xs:simpleContent”, “xs:complexContent”, “xs:restriction”, e “xs:extension”.

Exemplo de tipo complexo

```
<xs:complexType  name="tipoPedido">
    <xs:sequence>
        <xs:element name="produtoPara" type="tipoEnderecoBrasil"/>
        <xs:element name="notaPara" type="tipoEnderecoBrasil"/>
        <xs:element name="itens" type="tipoItens"/>
    </xs:sequence>
    <xs:attribute name="dataPedido" type="xs:date"/>
</xs:complexType>
```

Declaração de Atributos

- **Uma associação entre um nome e um tipo simples**
- **Pode conter um valor padrão**

45

Declaração de um atributo é uma associação entre um nome e uma definição de tipo simples feita usando o elemento “xs:attribute”. Adicionalmente, a declaração pode conter informação sobre uma ocorrência, usando o atributo “use”, e um valor padrão, usando o atributo “value”.

Exemplo 1

```
<xs:attribute name="idade" type="xs:positiveInteger" use="required"
value="1"/>
```

Exemplo 2

```
<xs:attribute name="tipo" use="required">
    <xs:simpleType>
        <xs:restriction base="xs:NMTOKEN">
            <xs:enumeration value="a"/>
            <xs:enumeration value="b"/>
        </xs:restriction>
    </xs:simpleType>
</xs:attribute>
```

Declaração de Elementos

- **É uma associação entre um nome e tipos simples ou complexos**
- **Pode está contida dentro de uma definição de tipo complexo**
- **Equivalente as declarações ELEMENT e ATTLIST do DTD**

46

A declaração de um elemento é uma associação de um nome com uma definição de tipo (simples ou complexo) feita usando o elemento “xs:element”. A definição do tipo pode ser feita dentro da declaração do elemento. A declaração de um elemento também pode aparecer dentro da definição de um tipo complexo.

Exemplo 1 – Um elemento simples

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

Exemplo 2 – Um elemento complexo

```
<xs:element name="Endereco">
    <xs:complexType name ="tipoEnderecoBrasil">
        <xs:attribute name="logradouro" type="xs:string"/>
        <xs:attribute name="numero" type="xs:string"/>
        <xs:attribute name="complemento" type="xs:string"/>
        <xs:attribute name="bairro" type="xs:string"/>
        <xs:attribute name="cidade" type="xs:string"/>
        <xs:attribute name="uf" type="xs:string"/>
        <xs:attribute name="cep" type="xs:positiveInteger"/>
    </xs:complexType>
</xs:element>
```

Declaração de Notações

- **É uma associação entre um nome e um identificador de notação**
- **Uma notação serve para identificar conteúdo não XML e o programa que processa o conteúdo**

47

Uma notação é uma associação entre um nome e um identificador de notação e serve para identificar conteúdo não XML e o programa que processa o conteúdo. O nome de uma notação válida deve ser atribuído a um atributo que tiver o tipo simples “xs:NOTATION”. Neste exemplo declaramos um atributo do tipo “xs:NOTATION”:

```
<xs:element name="imagem">
  <xs:complexType mixed="true"> <!-- mixed – conteúdo misto -->
    <xs:attribute name="imagemTipo" type="xs:NOTATION" use="required"/>
  </xs:complexType>
</xs:element>
```

Exemplo - Declaração da Notação

```
<xs:notation name="jpeg" public="image/jpeg" system="viewer.exe"/>
```

Exemplo - Documento XML para os trechos de esquema acima

```
<imagem imagemTipo="jpeg">dados da imagem</imagem>
```

Definição de Grupos de Atributo

- **É usada para reutilizar grupos de atributos em várias definições de tipos complexos**

48

Uma definição de grupo de atributo é uma associação entre um nome e uma série de declarações de atributos facilitando a reutilização das declarações em várias definições de tipos complexos.

Exemplo

```
<xs:attributeGroup name="grupoNome">
    <xs:attribute name="primeiroNome"/>
    <xs:attribute name="sobreNome"/>
</xs:attributeGroup>

<xs:complexType name="tipoAutor">
    <xs:attributeGroup ref="grupoNome"/>
</xs:complexType>
```

Definição de Restrições de Identidade

- **É uma associação entre um nome e qualquer tipo de restrição como chave primária, chave única ou chave estrangeira**
- **Utiliza a especificação XPATH para indicar os registros aos quais se aplica a restrição**

49

Uma restrição de identidade é uma associação entre um nome e qualquer um dos tipos de restrições como chave primária, chave única e chave estrangeira. Todos os tipos de restrição utilizam XPATH para indicar os registros aos quais a restrição aplicará.

Exemplo 1 – Restrição Chave Única

```
<xs:unique name="nomeLivroUnico">
    <xs:selector xpath="livros/livro"/>
    <xs:field xpath="@nome"/>
</xs:unique>
```

Exemplo 2 – Restrições Chave Primária e Estrangeira

```
<xs:key name="keyAutor">
    <xs:selector xpath="autores/autor"/>
    <xs:field xpath="primeiroNome"/>
    <xs:field xpath="sobreNome"/>
</xs:key>
<xs:keyref name="keyrefAutorLivro" refer="keyAutor">
    <xs:selector xpath="livros/livro/autor"/>
    <xs:field xpath="primeiroNome"/>
    <xs:field xpath="sobreNome"/>
</xs:keyref>
```

Definição de Grupos de Modelos

- **É uma associação entre um nome e um grupo de modelo**
- **Facilita a reutilização do mesmo grupo de modelo em vários tipos complexos**

50

Uma definição de grupo de modelo é uma associação entre um nome e um grupo de modelo que pode ser reutilizado em vários tipos complexos.

Exemplo

```
<xs:group name="grupoNome">
    <xs:sequence>
        <xs:element name="primeiroNome" type="xs:string"/>
        <xs:element name="sobreNome" type="xs:string"/>
    </xs:sequence>
</xs:group>
<xs:complexType name="tipoNome">
    <xs:group ref="grupoNome"/>
</xs:complexType>
```

Anotações

- **Contém informação sobre o modelo**
- **Usadas por leitores humanos ou máquinas**

51

Uma anotação é alguma informação sobre o modelo usada por leitores humanos ou máquinas.

Exemplo – Documentação

```
<xs:simpleType name="tipoTemperaturaAgua">
    <xs:annotation>
        <xs:documentation source="Temperatura em farenheits"/>
    </xs:annotation>
    <xs:restriction base="xs:number">
        <xs:minExclusive value="0.00"/>
        <xs:maxExclusive value="100.00"/>
    </xs:restriction>
</xs:simpleType>
```

Grupos de Modelos

- **Compostos por uma lista de**
  - **Elementos**
  - **Curingas e**
  - **Grupos de Modelos**
- **Temos três tipos de grupos de modelos**
  - **Seqüência**
  - **Conjunção**
  - **Disjunção**

52

Um grupo de modelo é um fragmento semântico que pode ser aplicado a uma lista de elementos de informação e é composto de declaração de elementos, curingas e grupos de modelos. Tem três tipos de grupos de modelos:

Seqüência

Todos os elementos deveriam ser especificados e seguem uma seqüência rígida. Uma seqüência é criada usando elemento “sequence” (xs:sequence).

Conjunção

Todos os elementos deveriam ser especificados mas não seguem uma seqüência rígida. Uma conjunção é criada usando elemento “all” (xs:all).

Disjunção

Qualquer um dos elementos precisa ser especificado. Uma disjunção é criada usando o elemento “choice” (xs:choice).

Exemplo – sequence e choice

```
<xs:sequence>
    <xs:choice>
        <xs:element name="esquerda"/>
        <xs:element name="direita"/>
    </xs:choice>
    <xs:element name="pontoReferencia"/>
</xs:sequence>
```

Curingas

- **O elemento “xs:any” fornece um mecanismo para introduzir um curinga para o conteúdo dos elementos**
- **O elemento “xs:anyAttribute” é um curinga para qualquer atributo do espaço identificador especificado no atributo “namespace” do elemento**

53

Formas de utilização do “xs:any”

1. A forma para representar qualquer construção de elementos XML de qualquer espaço identificador:

```
<xs:any namespace="##any"/>
```

2. A forma para representar qualquer construção de elementos XML de qualquer espaço identificador menos o atual:

```
<xs:any namespace="##other"/>
```

3. A forma para representar qualquer construção de elementos XML do espaço identificador especificado:

```
<xs:any namespace="http://www.w3.org/2001/XMLSchema"/>
```

4. A forma para representar qualquer construção de elementos XML do espaço identificador atual:

```
<xs:any/>
```

Exemplo – any e anyAttribute

```
<xs:element name="contem_curingas">
  <xs:complexType>
    <xs:sequence>
      <xs:any/>
    </xs:sequence>
    <xs:anyAttribute/>
  </xs:complexType>
</xs:element>
```

Uso de Atributos

- **Usado para especificar se a utilização de qualquer atributo é:**
  - **Requerido**
  - **Opcional**
  - **Proibido**
- **O atributo pode ter um valor fixo (constante)**

54

Usado para especificar a obrigatoriedade de uso de um atributo. O uso é especificado usando o atributo “use” do elemento “xs:attribute” que pode conter o valor:

required	Requerido
optional	Opcional
prohibited	Proibido

O atributo pode conter um valor fixo e isso pode ser especificado usando o atributo “fixed” do elemento “xs:attribute”.

Exemplos

```
<xs:attribute name="codigo" use="required" type="xs:positiveInteger"/>
<xs:attribute name="segundoNome" use="optional"/>
<xs:attribute name="copyright" type="xs:string" fixed="© Livraria ABC"/>
```

Tipos de Dados Internos

string	**boolean**	**decimal**
float	**double**	**duration**
dateTime	**time**	**date**
gYearMonth	**gYear**	**gMonthDay**
gDay	**gMonth**	**hexBinary**
base64Binary	**anyURI**	**QName**
NOTATION		

55

Tipos primitivos internos fazem parte da especificação de Esquemas XML.

Tipos Derivados Internos

normalizedString	**ID**	**IDREF**	**IDREFS**
nonPositiveInteger	**int**	**unsignedInt**	**positiveInteger**
nonNegativeInteger	**long**	**short**	**negativeInteger**
unsignedLong	**byte**	**integer**	**ENTITY**
unsignedShort	**Name**	**NCName**	**ENTITIES**
unsignedByte	**token**	**NMTOKEN**	**NMTOKENS**
language			

56

Tipos derivados de outros tipos de dados primitivos internos.

Exemplo Escola

- **Mesmo exemplo como no caso da DTD**
- **Esquema permite maior flexibilidade:**
  - **Definição das chaves:**
    - **Única**
    - **Primária**
    - **Estrangeira**
  - **Definição e extensão de tipos simples e complexos**

57

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="escola">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="professores"/>
                <xs:element ref="estudantes"/>
                <xs:element ref="aulas"/>
            </xs:sequence>
            <xs:attribute name="nome" type="xs:string" use="required"/>
        </xs:complexType>
        <xs:key name="keyProfessor">
            <xs:selector xpath="professores/professor"/>
            <xs:field xpath="@codigo"/>
        </xs:key>
        <xs:key name="keyEstudante">
            <xs:selector xpath="estudantes/estudante"/>
            <xs:field xpath="@codigo"/>
        </xs:key>
        <xs:unique name="uniqueEstudanteAula">
            <xs:selector xpath="aulas/aula/estudante_aula"/>
            <xs:field xpath="@codigo"/>
        </xs:unique>
        <xs:keyref name="fkeyProfessorAula" refer="keyProfessor">
            <xs:selector xpath="aulas/aula"/>
            <xs:field xpath="@professor"/>
        </xs:keyref>
        <xs:keyref name="fkeyEstudanteAula" refer="keyEstudante">
            <xs:selector xpath="aulas/aula/estudante_aula"/>
            <xs:field xpath="@codigo"/>
        </xs:keyref>
    </xs:element> <!--escola-->
```

```
    <xs:element name="professores">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="professor" maxOccurs="unbounded"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:element name="professor">
        <xs:complexType>
            <xs:attribute name="codigo" type="xs:positiveInteger" use="required"/>
            <xs:attribute name="nome" type="xs:string" use="required"/>
        </xs:complexType>
    </xs:element>
    <xs:element name="estudantes">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="estudante" maxOccurs="unbounded"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:element name="estudante">
        <xs:complexType>
            <xs:attribute name="codigo" type="xs:positiveInteger" use="required"/>
            <xs:attribute name="nome" type="xs:string" use="required"/>
        </xs:complexType>
    </xs:element>
    <xs:element name="aulas">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="aula" maxOccurs="unbounded"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:element name="aula">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="estudante" type="estudante_aula"
maxOccurs="unbounded"/>
            </xs:sequence>
            <xs:attribute name="professor" type="xs:positiveInteger" use="required"/>
            <xs:attribute name="materia" type="nome_materia" use="required"/>
        </xs:complexType>
    </xs:element>
    <xs:simpleType name="nome_materia">
        <xs:restriction base="xs:string">
            <xs:enumeration value="Matematica"/>
            <xs:enumeration value="Quimica"/>
            <xs:enumeration value="Fisica"/>
        </xs:restriction>
    </xs:simpleType>
    <xs:complexType name="estudante_aula">
        <xs:attribute name="codigo" type="xs:positiveInteger" use="required"/>
    </xs:complexType>
</xs:schema>
```

Exercício 3

1. **Crie um esquema XML para a DTD de catálogo de livros criada no Exercício 2**
2. **Estude tópicos avançados:**
   - **Expressões Regular**
   - **Tipos abstratos**
   - **Redefinir esquema - elemento xs:redefine**
   - **Importando esquemas - elemento xs:include**
   - **Derivando tipos - elementos xs:extension e xs:restriction**
   - **Restringindo derivação de tipos - atributos final e block do elemento xs:complexType**
   - **Formas de restringir valores de tipos simples (facets)**

59

Use XMLSpy para criar o Esquema. Para estudar tópicos avançados use o documento “XML Schema Part 0: Primer” disponível no endereço http://www.w3.org/TR/xmlschema-0/.

Tópico 3

Acesso Programático - DOM e SAX

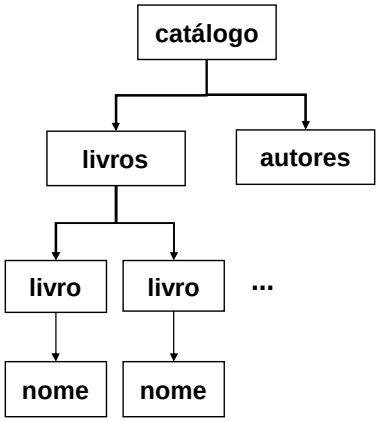
60

Objetivo

- O que é DOM?
  - Usando DOM para criar e validar documentos XML
- O que é SAX?
  - Usando SAX para ler documentos XML
- O que é JAXP?
  - Como usar DOM de JAXP
  - Como usar SAX de JAXP
- O que é JDOM?
- Leitura de XML usando um método PULL em Microsoft .NET

DOM – Document Object Model

- **Representa todo conteúdo do documento XML como uma árvore**
- **Cada elemento, texto do elemento, atributo, comentário, instrução de processamento etc. é representado como um nó nesta árvore**
- **É uma especificação recomendada pelo W3C**



61

Vantagens do DOM

•Todo o documento é representado como uma árvore, o que facilita a recuperação e a alteração aleatória de dados.

•Facilita criação de documentos XML bem formados porque o texto do documento é gerado automaticamente. O DOM gera erro quando você for fazer operações ilegais como tentar colocar dois elementos raiz.

Desvantagens do DOM

•Lento para processar documentos imensos repetidas vezes.

•A árvore de objetos ocupa muita memória.

Objetos do DOM

- **DOM composto de vários tipos de objetos para representar documento XML, elemento, atributo etc**
- **Existem objetos para representar e agregar nó:**
  - **Node: Um único nó do documento**
  - **NodeList: Apresenta uma lista de nó**
  - **NamedNodeMap: Fornece acesso aos nós de atributos pelo nome**

62

Objetos do DOM

Element: Um elemento.

Attr: Um atributo de um elemento.

Text: Conteúdo texto de um elemento ou atributo.

CDATAsection: Uma seção CDATA.

EntityReference: Referência a uma entidade.

Entity: Uma entidade (parsed ou unparsed).

ProcessingInstruction: Uma instrução de processamento.

Comment: Um comentário.

Document: Documento XML.

DocumentType: Referência para o elemento <!DOCTYPE>.

DocumentFragment: Referência para um fragmento de um documento.

Notation: Uma notação XML.

Observação: A especificação DOM do W3C defini os objetos em cima como interfaces Java contendo declaração de métodos e atributos. A implementação das interfaces é feita pelos parsers, por exemplo, o objeto DOMDocument é uma implementação da interface Document feito pelo parser MSXML 3.0.

Objeto Node

- **Objeto Node representa um nó**
- **Cada objeto (Element, Attribute, etc.) herda características básicas do objeto Node**
- **O objeto Node disponibiliza várias propriedades para navegar a árvore de elementos**

63

Propriedades expostas pelo objeto Node

nodeType: Tipo do objeto representado pelo nó

parentNode: Objeto pai do nó atual

childNodes: Lista de nós filhos do nó atual

firstChild: Primeiro nó filho

lastChild: Último nó filho

previousSibling: Nó anterior no mesmo nível que o atual (último irmão)

nextSibling: Nó posterior no mesmo nível que o atual (próximo irmão)

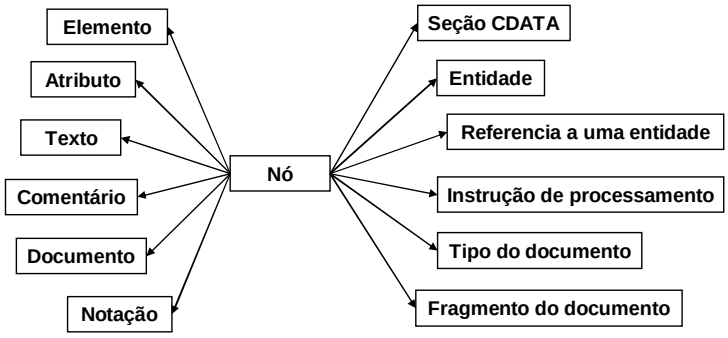
attributes: Lista de atributos se o nó atual tiver atributos

nodeName: Nome do nó

nodeValue: Valor texto do nó

Propriedade nodeType

- **Contém valor numérico identificando o tipo do nó**



64

Valores da propriedade nodeType

1	Elemento
2	Atributo
3	Texto
4	Seção CDATA
5	Referencia a uma entidade
6	Entidade
7	Instrução de processamento
8	Comentário
9	Documento
10	Tipo do documento
11	Fragmento do documento
12	Notação

Métodos do Objeto Node

- **Disponibilizados pelo objeto Node para inserir, duplicar, remover e substituir outros objeto Node**

65

Métodos do objeto Node

appendChild: Adiciona um novo nó filho.

cloneNode: Cria cópia de um nó.

hasAttributes: Retorna verdadeiro se o nó tiver atributos.

hasChildNodes: Retorna verdadeiro se o nó tiver filhos

insertBefore: Insere um nó filho antes do nó filho especificado

isSupported: Retorna verdadeiro se tiver suporte para um determinado recurso do DOM.

normalize: Normaliza a estrutura do nó. Junta nó de texto adjacentes dentro de elementos em um único nó texto.

removeChild: Remove um nó filho.

replaceChild: Substitui um nó filho.

Objetos NodeList e NamedNodeMap

- **Objeto NodeList**
  - **Fornece uma lista de nós**
  - **É uma coleção indexada de nós começando com o índice 0**
- **Objeto NamedNodeMap**
  - **Fornece uma coleção de nós**
  - **Os nós podem ser acessados usando nomes**
  - **Os nós também podem ser acessados usando um índice numérico**

66

Objeto NodeList

Propriedades

length: Quantidade de nós na lista.

Métodos

item: Retorna um item indexado.

Objeto NamedNodeMap

Propriedades

length: Quantidade de nós na coleção.

Métodos

getNamedItem: Retorna um nó com o nome especificado.

getNamedItemNS: Retorna um nó com o nome e espaço identificador especificado.

item: Retorna um nó contido na posição especificada.

removeNamedItem: Remove o nó com o nome especificado.

removeNamedItemNS: Remove o nó com o nome e espaço identificador especificado.

setNamedItem: Adiciona ou substitui um nó com o nome especificado.

setNamedItemNS: Adiciona ou substitui um nó com o nome e espaço identificador especificado.

Objeto Attr

- **Representa atributos**
- **Objeto Attr herda do objeto Node mas não faz parte da árvore de documento**
    - **Por isso atributos parentNode, previousSibling e nextSibling do objeto Node tem valor nulo**
- **Em MSXML 3.0 a interface é chamada de IXMLDOMAttribute**

67

Propriedades do objeto Attr

name: Nome do atributo.

ownerElement: Referencia o objeto Element que contém o atributo.

specified: Contém verdadeiro se foi atribuído algum valor ao atributo.

value: Valor do atributo.

Objeto Element

- **Representa um elemento XML**
- **Herda do objeto Node**
  - **Tem todas as características e operações do objeto Node**
- **Contém métodos para recuperar objetos do tipo Attr pelo nome ou pelo índice**
- **Propriedades**
  - **tagName: Nome do elemento**

68

Métodos

getAttribute e getAttributeNS: Recupera o valor de um atributo pelo nome.

getAttributeNode e getAttributeNodeNS: Recupera um objeto Attr pelo nome.

getElementsByTagName e getElementsByTagNameNS: Retorna um objeto NodeList contendo os elementos filhos que têm o nome especificado.

hasAttribute e hasAttributeNS: Retorna verdadeiro se o elemento tiver atributos.

removeAttribute e removeAttributeNS: Remove o atributo especificado.

removeAttributeNode: Remove o objeto Attr especificado.

setAttribute e setAttributeNS : Adiciona ou substitui um atributo, dado seu nome e valor.

setAttributeNode e setAttributeNodeNS: Adiciona ou substitui o objeto Attr especificado.

Objetos CharacterData e Text

- **Objeto CharacterData**
  - **Representa dados caractere usando Unicode (UTF-16)**
  - **Estende o objeto Node**
- **Objeto Text**
  - **Representa dados texto dentro de um elemento.**
  - **Herda do Objeto CharacterData**

69

Objeto CharacterData

Propriedades

data: Dados caractere disponível.

length: Número de caracteres disponível.

Métodos

appendData: Anexar dados string ao final dos dados caractere.

deleteData: Excluir caracteres na faixa especificada.

insertData: Inserir um string na posição especificada.

replaceData: Substituir caracteres começando na posição especificada pelo string especificado.

substringData: Extrair caracteres da faixa especificada.

Objeto Text

Método

splitText: Quebrar o objeto Texto em dois, começando da posição especificada.

Objeto Document

- **Representa um documento XML**
- **Herda do objeto Node**
- **Acrescenta algumas propriedades a mais**
  - **documentElement: Elemento raiz**
  - **doctype: Tipo do documento**
- **Acrescenta alguns métodos**
- **Métodos específicos do parser MSXML 3.0:**
  - **loadXML: Carregar texto XML**
  - **load: Carregar arquivo XML**

70

Métodos do Objeto Document

createAttribute e createAttributeNS: Criar um novo atributo.

createCDATASection: Criar uma nova seção CDATA.

createComment: Criar um novo comentário.

createDocumentFragment: Criar um novo fragmento de documento vazio.

createElement e createElementNS: Criar um novo elemento.

createEntityReference: Criar uma nova referência a entidade.

createProcessingInstruction: Criar uma nova instrução de processamento.

createTextNode: Criar um novo nó texto.

getElementById: Retornar um elemento com o identificador especificado.

getElementsByTagName e getElementsByTagNameNS: Retornar um elemento com o marcador especificado.

importNode: Importar um nó de um outro documento.

Usando DOM para criar Documentos XML

- **Usaremos MSXML nos exemplos**
- **Nosso exemplo:**
  - **Utiliza o objeto DOMDocument para criar um novo documento XML**
    - **Insere uma nova instrução de processamento**
    - **Cria nó do tipo “NODE_ELEMENT” e do tipo “NODE_ATTRIBUTE”**

71

Exemplo usando MSXML Parser versão 3.0

```
Dim oDom As New MSXML2.DOMDocument
Dim oElemento As MSXML2.IXMLDOMNode
Dim oAtributo As MSXML2.IXMLDOMNode
oDom.appendChild oDom.createProcessingInstruction( _
    "xml", "version=""1.0""")
Set oElemento = oDom.appendChild( _
    oDom.createNode(NODE_ELEMENT, "catalogo", ""))
Set oAtributo = oElemento.Attributes.setNamedItem( _
    oDom.createNode(NODE_ATTRIBUTE, "data", ""))
oAtributo.Text = "12/01/2001"
Set oElemento = oElemento.appendChild( _
    oDom.createElement("livro"))
Set oAtributo = oElemento.Attributes.setNamedItem( _
    oDom.createAttribute("nome"))
oAtributo.Text = "Programando em Java"
```

Saída - Documento XML Gerado

<?xml version="1.0"?>

<catalogo data="12/01/2001"><livro nome="Programando em Java"/></catalogo>

Usando DOM para ler Documentos XML

- **Usaremos MSXML neste exemplo**
- **O objeto DOMDocument é usado para ler XML**
  - **O método load carrega um arquivo XML**
- **No exemplo ilustramos**
  - **Como tratar erros usando atributo parseError**
  - **Como ler todos os nós de um documento XML usando um método recursivo**

72

Exemplo – Ler documento DOM usando método recursivo

```
oDom.Load "nome do arquivo XML"
If oDom.parseError.errorCode = 0 Then
    Debug.Print parse(oDom.documentElement, 0)
End If

Function parse(oNode As IXMLDOMNode, Nivel As Long) As String
    Dim i As Long
    Dim s As String
    s = s & String(Nivel, vbTab) & oNode.nodeName & vbCrLf
    For i = 0 To oNode.Attributes.length - 1
        s = s & String(Nivel + 1, vbTab) _
              & oNode.Attributes(i).nodeName _
              & "=" & oNode.Attributes(i).nodeValue & vbCrLf
    Next
    For i = 0 To oNode.childNodes.length - 1
        s = s & parse(oNode.childNodes.Item(i), Nivel + 1)
    Next
    parse = s
End Function
```

Resultado da Leitura

- **Aqui mostramos a saída gerada pela leitura do documento XML**
- **O documento XML utilizado foi:**

```
<catalogo>
    <livro nome="Programação Java">
        <autor nome="Paulo Junior"/>
    </livro>
</catalogo>
```

73

Saída – Resultado da execução

```
catalogo
    livro
        nome=Programação Java
        autor
            nome=Paulo Junior
```

Exercício 4

1. **Use DOM para ler o catálogo de livros criado anteriormente e mostrar nome de todos os livros dentro de um controle como ListBox**

74

Use MSXML 3.0 e VB 6.0 para fazer o exercício.

Simple API for XML

- **Está baseada em eventos**
- **Não é um padrão do W3C**
- **É um padrão "de fato"**
- **A versão preliminar chamado de SAX**
- **A versão atual é chamado SAX2**
  - **Oficialmente disponível para Java**
  - **Tem suporte para espaços identificadores**

75

SAX é uma API baseada em eventos. As APIs baseadas em eventos geram eventos dentro do aplicativo usando o mecanismo de callback. No caso de SAX os eventos podem ser:

•Start Document

•Start Element

•End Element

•End Document

Aplicativos implementam métodos para tratar esses eventos. SAX fornece acesso aos documentos XML utilizando um mecanismo simples e de baixo nível. Isso possibilita analisar documentos muito maiores que a memória do sistema. Isso é uma limitação do DOM.

Aplicativos tratam os eventos e constroem suas próprias estruturas de dados. DOM gera uma árvore completa, o que pode não ser a estrutura de dados requerida.

Lendo um Documento XML Usando SAX

- **Utilizamos parser SAX do pacote MSXML**
- **Implementamos a interface “ContentHandler” para receber e tratar callbacks (eventos) SAX**
- **Usamos os eventos:**
  - **startElement**
  - **endElement**
- **Para executar o exemplo podemos chamar o método**
  - **saxParse(nome do arquivo XML)**

76

Exemplo – Código fonte contida numa Classe VB

```
Implements IVBSAXContentHandler
Const MAX_DEPTH = 100
Dim nDepth As Long, sArray(MAX_DEPTH) As String
Private Sub IVBSAXContentHandler_characters(strChars As String)
'
End Sub
Private Property Set IVBSAXContentHandler_documentLocator( _
  ByVal RHS As MSXML2.IVBSAXLocator)
'
End Property
Private Sub IVBSAXContentHandler_endDocument()
'
End Sub
Private Sub IVBSAXContentHandler_endElement( _
  strNamespaceURI As String, strLocalName As String, _
  strQName As String)
    If strQName = sArray(nDepth - 1) Then 'Found closing tag
        nDepth = nDepth - 1
    End If
End Sub
Private Sub IVBSAXContentHandler_endPrefixMapping(strPrefix As String)
'
End Sub
Private Sub IVBSAXContentHandler_ignorableWhitespace(strChars As String)
'
End Sub
Private Sub IVBSAXContentHandler_processingInstruction( _
  strTarget As String, strData As String)
'
End Sub
```

Lendo um Documento XML Usando SAX

- **Resultado da execução**

```
catalogo
    autores
        autor
            primeiroNome
            sobreNome
    ...
    livros
        livro
            autor
                primeiroNome
                sobreNome
```

77

```
Private Sub IVBSAXContentHandler_skippedEntity(strName As String)
'
End Sub
Private Sub IVBSAXContentHandler_startDocument()
'
End Sub
Private Sub IVBSAXContentHandler_startElement( _
  strNamespaceURI As String, strLocalName As String, _
  strQName As String, ByVal oAttributes As MSXML2.IVBSAXAttributes)
    sArray(nDepth) = strQName
    Debug.Print String(nDepth, vbTab) & strQName
    nDepth = nDepth + 1
End Sub
Private Sub IVBSAXContentHandler_startPrefixMapping( _
  strPrefix As String, strURI As String)
'
End Sub
Public Function saxParse(ByVal fileName As String) As Boolean
    Dim oReader As New SAXXMLReader
    Set oReader.contentHandler = Me
    oReader.parseURL fileName
    If nDepth = 0 Then
        saxParse = True
    Else
        saxParse = False
    End If
End Function
```

XML e Java

- **O mundo Java tem diversos APIs e parsers XML**
- **APIs**
  - **JAXP da SUN**
  - **JDOM**
- **Parsers DOM**
  - **XML4J da IBM**
  - **Xerces do Apache**
- **Parsers SAX**
  - **Xerces do Apache**

78

Todos os parsers implementam as interfaces padrão especificadas pelo W3C. As APIs XML de Java (JDOM e JAXP) permitem utilizar diversos parsers DOM ou SAX ao mesmo tempo.

JDOM é uma API simples feita especificamente para Java e no momento está na versão beta. A JDOM foi submetida ao Java Community Process (JCP) para ser incorporada na plataforma Java no futuro.

A JAXP é uma API da SUN para Java e foi incorporada na versão 1.4 do SDK Java.

Usando JAXP - DOM

- **JAXP gera uma árvore DOM usando um parser SAX**
- **Pacotes JAXP utilizados no exemplo:**
  - **javax.xml.parsers**
    - **DocumentBuilder, DocumentBuilderFactory, ParserConfigurationException**
  - **org.w3c.dom**
    - **Document, Node, NamedNodeMap, NodeList**
  - **org.xml.sax**
    - **SAXException**

79

Exemplo JAXP-DOM – lista todos os elementos e atributos no documento

```
import java.io.*;
import javax.xml.parsers.*;
import org.w3c.dom.*;
import org.xml.sax.*;
class LeitorDOM {
    public static void main(String args[])
    throws ParserConfigurationException, IOException, SAXException {
        DocumentBuilder leitor =
            DocumentBuilderFactory.newInstance().newDocumentBuilder();
        Document documento = leitor.parse("catalogo.xml");
        processNode(documento);
    }
    private static void processNode(Node no) {
        System.out.println(no.getNodeName());
        if (no.hasAttributes()) {
            NamedNodeMap attr = no.getAttributes();
            for (int i=0; i < attr.getLength(); i++) {
                System.out.println(attr.item(i).getNodeName());
            }
        }
        if (no.hasChildNodes()) {
            NodeList nos = no.getChildNodes();
            for (int i=0; i < nos.getLength(); i++) {
                processNode(nos.item(i));
            }
        }
    }
}
```

Usando JAXP – SAX

- **JAXP vem com um parser SAX**
  - **Isso não impede a utilização de outro parser SAX qualquer**
- **Pacotes JAXP utilizados no exemplo:**
  - **javax.xml.parsers**
    - **SAXParser, SAXParserFactory, ParserConfigurationException**
  - **org.xml.sax**
    - **SAXException**
  - **org.xml.sax.helpers**
    - **DefaultHandler**

80

Exemplo JAXP-SAX – lista todos os elementos encontrados

```
import java.io.*;
import javax.xml.parsers.*;
import org.xml.sax.*;
import org.xml.sax.helpers.*;
class LeitorSAX extends DefaultHandler {
    public static void main(String args[])
    throws SAXException, ParserConfigurationException, IOException {
        SAXParser leitor = SAXParserFactory.newInstance().newSAXParser();
        leitor.parse("catalogo.xml", new LeitorSAX());
    }
    public void startDocument() throws SAXException { }
    public void endDocument() throws SAXException { }
    public void startElement(String namespaceURI,
                             String lName, // local name
                             String qName, // qualified name
                             Attributes attrs) throws SAXException {
        System.out.println(qName);
    }
    public void endElement(String namespaceURI,
                           String sName, // simple name
                           String qName  // qualified name
                          ) throws SAXException { }
}
```

Exercício 5

1. **Utilize SAX para ler o catálogo de livros criado anteriormente e mostrar o nome de todos os livros dentro de um controle como ListBox**

81

Use MSXML 3.0 e VB 6.0 para fazer o exercício.

JDOM

- **Limitações do DOM:**
  - **Independente da Linguagem**
  - **Hierarquias estritas de classes**
  - **Baseado em Interface**
- **Vantagens da JDOM:**
  - **Para Java**
  - **Hierarquias de classes inexistentes**
- **Ainda na versão beta (consulte www.jdom.org)**

82

A API JDOM pode ser utilizada como uma alternativa do pacote org.w3c.dom, porém JDOM e DOM podem coexistir sem problemas. JDOM utiliza outros parsers para ler texto XML. Para facilitar a leitura de texto XML o JDOM disponibiliza as classes wrapper para alguns parsers.

Limitações do DOM

- Independente da Linguagem: O DOM não foi desenhado com a linguagem Java em mente. DOM não faz uso das classes de coleção existentes na linguagem Java.
- Hierarquias Estritas: O DOM segue a especificação XML a risca. Em XML, tudo é um nó. Todos os outros objetos são derivados do objeto Node.
- Baseado em Interface: Isto torna a tarefa de construir objetos XML um pouco complexa.

Vantagens do JDOM

- Feito para Java: Usa objetos Java.
- Nenhuma hierarquia: Por exemplo, um elemento XML é uma instância da classe Element, e um atributo XML é uma instância da classe Attribute. Isso facilita programação.

Método Pull em Microsoft .NET

- **A API SAX utiliza callbacks para avisar quando elementos são encontrados**
- **O método Pull utiliza chamadas explícitas a um método que recupera um elemento após outro**
- **Implementada pela classe XmlTextReader do espaço identificador System.Xml**
- **O exemplo utiliza a linguagem C# da plataforma .NET**

83

Exemplo – C# usando método Pull para ler documento XML

```
using System;
using System.Xml;

public class TestXMLReader : object {
  public static void Main() {
    XmlTextReader reader = new XmlTextReader("teste.xml");
    while (reader.Read()) {
      if (reader.NodeType == XmlNodeType.Element) {
        System.Console.WriteLine(
                "Found Element: " + reader.Name);
        if (reader.HasAttributes) {
          while (reader.MoveToNextAttribute())  {
            System.Console.WriteLine(
                  "Found Attribute: " + reader.Name);
          } //while
        } //if
      } //if
    } //while
    reader.Close();
  } //Main
}
```

Tópico 4

Transformando XML – CSS e XSL

84

Objetivo

•O que quer dizer transformar?

•Usando CSS para formatar documentos XML.

•Usando XSLT para transformar documentos XML.

Cascading Style Sheets (CSS)

- **CSS significa Folhas de Estilo Em Cascata**
- **É utilizada com HTML**
- **Pode ser utilizada com XML**
- **Determina a formatação visual dos elementos**
- **Qualquer navegador com suporte a CSS pode formatar os elementos:**
  - **IE 5.x ou melhor**
  - **Netscape 6.x ou melhor**
- **Padrão atualmente no nível 2 (CSS2)**

85

A CSS surgiu para facilitar a separação entre dados e formatação visual. HTML tem o marcador FONT que pode ser utilizado para especificar o estilo da fonte do texto, mas o W3C recomenda a utilização da CSS para formatar documentos HTML, porque isso separa a especificação da formatação do conteúdo. Por exemplo, um texto como:

```
<P><FONT face="Arial" size="1">Isto é um parágrafo</FONT></P>
<P><FONT face="Arial" size="4">Isto é outro
parágrafo</FONT></P>
```

contém conteúdo dos parágrafos misturado com informação de formatação. Isto tem duas desvantagens:

-É mais difícil separar o conteúdo das páginas da formatação.

-É difícil fazer alteração de formatação em todas as páginas mais tarde.

A CSS foi desenvolvida como uma forma mais poderosa de fazer a formatação de estilo. Um arquivo de folha de estilo contém especificação de formatação de diversos elementos HTML. Assim, alterando o arquivo de folha de estilo, é possível alterar a formatação de várias páginas ao mesmo tempo.

Exemplo da CSS com XML

- **Vamos abrir este documento XML em IE 5.0**

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<?XML:stylesheet type="text/css" href="artigo.css"?>
<artigo>
   <titulo>Frederico o Grande encontra Bach</titulo>
   <autor>Johann Nikolaus Forkel</autor>
   <paragrafo>
     Uma noite, quando ele estava acabando de preparar a
     <instrumento>flauta</instrumento> e seus músicos estavam
     reunidos, um oficial trouxe a lista de convidados.
   </paragrafo>
</artigo>
```

86

Arquivo de folha de estilo artigo.css

```
/* Folha de estilo */
instrumento { display: inline }
artigo, titulo, autor, paragrafo { display: block }
titulo { font-size: 1.3em }
autor { font-style: italic }
artigo, titulo, autor, paragrafo { margin: 0.5em }
instrumento { font-style: italic }
```

O resultado mostrado pelo IE 5.0

Frederico o Grande encontra Bach

Johann Nikolaus Forkel

Uma noite, quando ele estava acabando de preparar a *flauta* e seus músicos estavam reunidos, um oficial trouxe a lista de convidados.

Seletores

- **Uma folha de estilo contém um conjunto de regras compostas de:**
  - **Seletores**
  - **Blocos de declarações**
- **Um seletor especifica um ou mais elemento XML**
- **Exemplo:**

```
artigo, titulo, autor, paragrafo { display: block }
```

  - **Artigo, titulo, autor e paragrafo são seletores**
  - **A declaração CSS começa depois de { e termina com }**

87

O seletor consiste de tudo até a primeira chave '{'. Um seletor é sempre seguido pelo bloco {}. Quando um aplicativo não pode entender o seletor o bloco {} é ignorado.

Exemplos de Seletores Ilegais

Por exemplo, como "&" não é um símbolo válido em CSS2, um navegador CSS2 vai ignorar a linha dois inteiramente e não vai atribuir a cor vermelha ao elemento H3:

```
H1, H2 {color: green }
H3, H4 & H5 {color: red }
H6 {color: black }
```

Uma declaração contém propriedades ou pode está vazia. Uma propriedade é seguida por dois pontos ':' e um valor. Várias propriedades podem ser separadas dentro da mesma declaração pelo ponto e virgula ';'. Por exemplo,

```
instrumento { color: blue; font-style: italic }
```

Tipos de Seletores

- **Seletor universal**
- **Seletor de tipo**
- **Seletor de descendente**
- **Seletor de filho (elemento filho)**
- **Seletor de irmão (elementos adjacentes)**
- **Seletor de atributo**
- **Seletor de classe**
- **Seletor de ID**

88

Exemplos de Tipos de Seletores

Seletor Universal

```
* { color: blue }
```

Seletor de tipo

```
H1 { color: blue }
```

Seletor de descendente

```
paragrafo { color: blue }
instrumento { color: blue; font-style: italic }
paragrafo instrumento { color: red }
```

Seletor de filho

```
BODY > P { line-height: 1.3 }
```

Seletor de irmão adjacente

```
H1 + H2 { margin-top: -5mm }
```

Seletor de atributos

```
*[LANG=fr] { display: none }
```

Seletor de classe (válido somente para HTML)

```
p.titulo { color: red }
```

Seletor de ID

```
H1#chapter1 { text-align: center }
```

Pseudo-elementos e Pseudo-Classes

- **Pseudo-classes**
  - **:first-child**
  - **:link e :visited**
  - **:hover, :active, e :focus**
  - **:lang**
- **Pseudo-elementos**
  - **:first-line**
  - **:first-letter**
  - **:before e :after**

89

Pseudo-classes

:first-child

```
DIV > P:first-child { text-indent: 0 }
```

:link e :visited

```
A:link { color: red }
```

:hover, :active, e :focus

```
A:focus { background: yellow }
```

:lang

```
HTML:lang(fr) { quotes: '« ' ' »' }
```

Pseudo-elementos

:first-line

```
P:first-line { text-transform: uppercase }
```

:first-letter

```
P:first-letter { color: green; font-size: 200% }
```

:before e :after

```
H1:before {content: counter(chapno, upper-roman) ". "}
```

Valor das Propriedades

- **Propriedades estão contidas dentro de um bloco de declaração**
- **Basicamente têm os seguinte tipos de valores:**
  - **Comprimento**
  - **Porcentagem**
  - **Cor**
  - **URL**

90

Comprimento

Comprimento é usado para definir altura, largura e tamanho em geral. Um valor comprimento é um número seguido por uma unidade de medida. As unidades de medida são abreviadas usando dois caracteres: em – altura da fonte de um elemento; px – pixeis; in – polegadas; cm – centímetros; mm – milímetros; pt – pontos, aonde cada ponto é 1/72 polegadas; pc – picas, aonde cada pica é 12 pontos.

Porcentagem

Porcentagens são usadas para largura, altura e posição. Uma porcentagem é um número seguido por sinal de porcentagem “%”.

Cor

Existem várias formas de especificar cor. A primeira forma usa um valor hexadecimal para especificar a cor (RGB), por exemplo, #FFFFFF é a cor branca, #FF0000 é a cor vermelha saturada, e #00FF00 é a cor verde saturada. A segunda forma usa uma especificação RGB explicita, por exemplo, `color: rgb(0, 0, 255)` ou `color: rgb(0%, 0%, 100%)`. A terceira forma utiliza as palavras chave: black, maroon, green, navy, silver, red, lime, blue, gray, purple, olive, teal, white, fuchsia, yellow, e aqua.

URL

URL é usado para imagens. Por exemplo:

```
background: url(http://www.w3.org/images/logo.gif)
```

Propriedades da Caixa

- **Os elementos são pintados dentro de uma caixa**
- **Uma caixa padrão é um retângulo**
- **As propriedades da caixa são divididas nas seguinte categorias:**
  - **Display**
  - **Margem**
  - **Padding**
  - **Borda**

91

Display

Determina como um elemento será mostrado. Essa propriedade tem os seguintes valores: `block`, `inline`, `list-item`, `none`. A declaração `"display: block"` cria um novo bloco contendo o texto do elemento. A declaração `"display: inline"` evita criar um novo bloco e mostra o texto dentro do bloco do elemento pai. A declaração `"display: none"` suprime o elemento.

Margem

Tem quatro propriedades de margem para uma caixa: `margin-top`, `margin-right`, `margin-bottom`, e `margin-left`, por exemplo, `margin-top: 10px;`. A outra forma de definir a margem é utilizando a propriedade "margin" para especificar as quatro margens de uma só vez.

Padding

É especificado exatamente como margem usando as propriedades `padding-top`, `padding-right`, `padding-bottom` e `padding-left`, ou apenas `padding`.

Borda

Usando a propriedade "border" podemos definir o estilo da borda. Os valores válidos são: `none`, `dotted`, `dashed`, `solid`, `double`, `groove`, `ridge`, `inset`, ou `outset`. Podemos especificar um valor diferente para bordas de topo, direita, inferior, e esquerda, por exemplo, `border: solid, dotted, double, inset;`.

Podemos especificar a largura da borda usando propriedades `border-top-width`, `border-right-width`, `border-bottom-width` e `border-left-width`, ou apenas `border-width`.

Propriedades de Texto e Fonte

- **Propriedades de fonte atribuam:**
  - **Nome da fonte**
  - **Comprimento da fonte**
  - **Estilo e peso**
  - **Alinhamento do Texto**
  - **Endentação e Altura de Linha**

92

Nome da fonte

Pode ser especificada usando a propriedade `font-family`. Podemos especificar a fonte desejada e fontes alternativas caso a fonte desejada não exista no computador, por exemplo, `font-family: Palatino, Garamond, "Times New Roman", Serif;`.

Comprimento da Fonte

Podemos especificar o comprimento da fonte usando a propriedade `font-size`. Este valor pode ser especificado usando um valor ou as palavras chaves `xx-small`, `x-small`, `small`, `medium`, `large`, `x-large`, e `xx-large`.

Estilo e Peso da Fonte

O estilo pode ser especificado usando a propriedade `font-style` e o peso usando a propriedade `font-weight`. Podemos especificar os valores `normal`, `italic` e `oblique` para `font-style`. Para `font-weight` podemos especificar `normal`, `bold`, `bolder`, `lighter`, 100, 200, 300, 400, 500, 600, 700, 800, 900.

Alinhamento do Texto

Pode ser especificado usando as propriedades `text-align` e `vertical-align`. `text-align` aceita valores `left`, `right`, `center`, e `justify`. `vertical-align` aceita valores `baseline`, `sub`, `super`, `top`, `text-top`, `middle`, `bottom`, `text-bottom`, ou uma porcentagem.

Endentação e Altura de Linha

A propriedade `text-indent` especifica endentação da primeira linha do texto e a propriedade `line-height` especifica espaçamento entre linhas adjacentes.

Propriedades da Cor e Fundo

- **Podemos especificar propriedades para:**
  - **Cor do primeiro plano**
  - **Cor de fundo**
  - **Cor da borda**
  - **Imagem de fundo**

93

Cor do Primeiro Plano

Podemos especificar a cor do primeiro plano usando a propriedade color. Por exemplo:

```
color: blue;
color: rgb(0,0,100%);
color: #0000FF;
```

Cor de Fundo

Podemos especificar a cor de fundo usando a propriedade background-color. Por exemplo:

```
background-color: white;
background-color: rgb(0,0,0);
background-color: #000000;
```

Cor da Borda

Podemos especificar a cor da borda usando a propriedade border-color. Por exemplo:

```
border-color: olive;
```

Imagem de Fundo

Podemos especificar a imagem de fundo usando a propriedade background-image. Por exemplo:

```
background-image: url(logo.gif);
```

Importando uma Folha de Estilo

- **Uma folha de estilo pode ser importada dentro da outra**
- **Regras especificadas no arquivo atual sobrepõem as regras do arquivo importado**
- **Utilizamos @import para fazer a importação**

94

Exemplo da Importação

@import url(http://www.w3.org/css/default.css);

Aqui o arquivo "default.css" no caminho (URL) especificado vai ser importado dentro do arquivo atual.

Exercício 6

1. **Crie um documento CSS para mostrar o catálogo de livros dentro do browser**

95

Use XMLSpy para concluir o exercício acima.

eXtended Stylesheet Language

- **CSS é muito básico**
- **Precisamos transformar**
  - **XML para XML**
  - **XML para HTML**
  - **Agrupar, filtrar, calcular e classificar**
- **Não estamos falando em abolir CSS**

96

CSS é um mecanismo de estilo simples e eficiente. Porém a CSS é limitada a definir o estilo do documento e não pode reorganizar ou processar o documento. CSS não pode:

•Converter um documento XML para outro menor ou mais completo.

•Converter XML para HTML, por exemplo, uma lista de elementos XML para tabela HTML.

•Filtrar informação, por exemplo, listar livros de um determinado autor.

Podemos continuar usando CSS e XSL juntas. XSL para processar os documentos e CSS para formatar os documentos que serão visualizados.

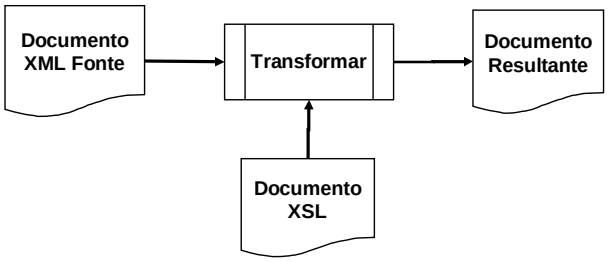
O padrão XSL é dividido em duas partes:

•XSLT – Transformações XSL.

•XSLFO – Objetos de Formatação XSL.

Conceito de XSL

- **O processo de usar XSLT:**
  - **Ler documento XML**
  - **Aplicar Transformações**
  - **Gravar documento resultante**



97

XSLT transforma um documento XML num outro documento baseado nas transformações especificadas no documento XSL.

Um documento XSL tem vários "templates" para processar cada elemento XML desejado. A seleção dos elementos é feita usando a linguagem XPATH que é um padrão que serve como base para XSLT. XSLT processa o documento XML recursivamente nó a nó até chegar ao final do documento XML.

Vamos ver um exemplo simples em seguida.

Exemplo de XSLT

- **Vamos aplicar transformações neste documento:**

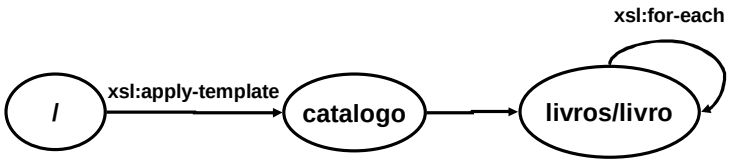
```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="catalogo.xsl"?>
<catalogo>
  <livros>
    <livro nome="Professional XML"/>
    <livro nome="XML By Example"/>
  </livros>
</catalogo>
```

98

XSLT usado para aplicar Transformações

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform" xmlns:msxsl="urn:schemas-
microsoft-com:xslt">
    <xsl:template match="/">
        <html>
            <body>
                <xsl:apply-templates/>
            </body>
        </html>
    </xsl:template>
    <xsl:template match="catalogo">
        <table width="100%" border="1">
            <tr>
                <td><p>Livros</p></td>
            </tr>
            <xsl:for-each select = "livros/livro">
            <tr>
                <td>
                    <p><xsl:value-of select="@nome"/></p>
                 </td>
            </tr>
            </xsl:for-each>
        </table>
    </xsl:template>
</xsl:stylesheet>
```

Execução do Exemplo



Resultado:

Livros
Professional XML
XML By Example

99

A transformação do exemplo anterior gera o resultado mostrado no slide acima.

Formato do Documento XSL

Um documento XSL é um documento XML válido. Todos os comandos XSLT estão contidos dentro do elemento raiz `xsl:stylesheet`. Podemos misturar elementos HTML válidos com elementos XSL. Esses elementos não são processados pelo transformador e aparecerão sem alteração no documento XML gerado pelo processo de transformação.

Dentro do elemento raiz precisamos ter pelo menos um elemento `xsl:template`. O atributo `match` desse elemento contém uma expressão em XPATH apontando para o elemento que será processado pelo template. A expressão XPATH "/" indica o elemento raiz.

XPATH

- **Padrão base para XSLT**
- **Usado para selecionar elementos e atributos**
- **Trata documento como sistema de arquivos (elementos = pastas)**
- **Um caminho pode ser relativo ou absoluto**
- **Permite usar expressões lógicas e funções**

/	**Elemento raiz**
//	**Seleciona todos os nós descendentes**
*	**Qualquer elemento**
.	**Nó atual**
|	**Seleciona entre várias caminhos (Ou)**

100

Selecionando Atributos

Usamos "@" antes do nome para especificar um atributo. "@*" seleciona qualquer atributo. Por exemplo `@nome` é o atributo chamado nome do elemento atual. XPATH tem o conceito de nó atual como temos o conceito de diretório atual no sistema de arquivos.

Algumas Funções utilizadas na expressão XPATH

`position()` – Retorna posição do nó atual dentro de uma lista de nós.

`text()` – Retorna texto de um elemento.

`last()` – Retorna posição do último nó na lista atual de nós.

`count()` – Retorna o número de nós na lista atual de nós.

`not()` – Nega o argumento.

`contains()` – Retorna verdadeiro se o primeiro argumento contém o segundo.

`starts-with()` – Retorna verdadeiro se o primeiro argumento começa com o segundo.

Exemplo de Seleção – Lendo Texto do Elemento

```
/artigo/secao[position()=2]/titulo/text()
```

Observação

Para mais detalhes veja http://www.w3.org/TR/1999/REC-xpath-19991116

Controlando a Saída

- **Podemos controlar a saída do processador XSLT usando o elemento `xsl:output`**
- **Podemos escolher a saída atribuindo os seguinte valores para o atributo `method`:**
  - **xml**
  - **text**
  - **html**
- **O elemento `xsl:output` deve aparecer logo após do marcador inicial do elemento `xsl:stylesheet`**

101

Exemplo

```
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:fo="http://www.w3.org/1999/XSL/Format">
  <xsl:output method="xml"/>
  <xsl:template match="/">
    <xsl:apply-templates/>
  </xsl:template>
  ...
</xsl:stylesheet>
```

Definindo Templates

- **Documento XSL contém pelo menos um elemento `xsl:template`**
- **O atributo `match` desse elemento seleciona o nó que será processado por esse elemento**
- **Ao atributo `match` deve ser atribuído uma expressão XPATH válida**
- **Dentro do elemento `xsl:template` podemos ter um elemento `xsl:apply-templates` para pedir o processador XSLT para aplicar os demais templates do documento**

102

Atributos do elemento xsl:template

`match` – Expressão XPATH.

`name` – Nome do template.

`priority` – Prioridade da execução (resolução de conflito).

`mode` – Modo (permite processamento múltiplo de elementos).

O atributo `match` é uma expressão XPATH que identifica os nós aos quais o template pode ser aplicado. Se o atributo `name` não for especificado o atributo é obrigatório.

Aplicando Templates – xsl:apply-templates

Usado para aplicar templates para nós filhos do nó atual. O elemento pode conter os atributos:

`select` – Template a ser aplicado.

`mode` – Modo (corresponde a um template com o mesmo modo).

Variáveis e Parâmetros

- **Usando elementos `xsl:variable` e `xsl:param`**
- **Elemento `xsl:variable`**
  - **Associa um valor a um nome**
  - **O tipo do valor pode ser qualquer texto ou nó selecionado usando expressões XPATH ou constante**
- **Elemento `xsl:param`**
  - **Usado para passar parâmetros para uma folha de estilo ou template**
  - **Parâmetros podem ser passados usando o elemento `xsl:with-param`**

103

Atributos do xsl:variable

name – Nome da variável

select – Expressão XPATH ou valor constante

Atributos do xsl:param

name – Nome do parâmetro

select – Valor padrão

Exemplo

```
<xsl:template name="qualquer">
  <xsl:param name="param-1" select="1"/>
  <xsl:variable name="var-1" select="2"/>
  <xsl:variable name="var-2">valor texto</xsl:variable >
</xsl:template>
...
<xsl:call-template name="qualquer">
  <xsl:with-param name="param-1">3</xsl:with-param>
</xsl:call-template>
```

Lendo valores e Copiando Fragmentos do Documento

- **Elemento `xsl:value-of`**
  - **Recupera valor de um nó ou variável**
  - **Atributo `select` usado para selecionar o nó**
  - **O valor recuperado é escrito para a saída**
- **Elemento `xsl:copy`**
  - **Copia nó atual para a saída**
  - **Não copia atributos e sub nós**
- **Elemento `xsl:copy-of`**
  - **Copia um nó, seus atributos e sub nós para a saída**

104

Atributos do xsl:value-of

select – Expressão XPATH ou nome de um variável

disable-output-escaping – Desabilitar tratamento de saída “yes” ou “no”

Exemplos

1. xsl:value-of

```
<xsl:variable name="var-1">qualquer
texto<xsl:variable>
<xsl:value-of select="$var-1"/>
```

2. xsl:copy – criar cópia idêntica do documento de entrada

```
<xsl:template match="@*|node()">
  <xsl:copy>
    <xsl:apply-templates select="@*|node()"/>
  </xsl:copy>
</xsl:template>
```

3. xsl:copy-of – criar cópia idêntica

```
<xsl:template match="/">
  <xsl:copy-of select="."/>
</xsl:template>
```

Processamento Condicional

- **Elemento `xsl:if`**
  - **Expressão lógica atribuída ao atributo `test`**
  - **Não tem "Se Não"**
- **Elemento `xsl:choose`**
  - **Seleciona uma entre várias possibilidades**
  - **Possibilidades especificadas usando atributo `test` do elemento `xsl:when`**
  - **Elemento `xsl:otherwise` especifica o processamento caso nenhuma das possibilidades for válida**

105

Exemplos

1. xsl:if

```
<xsl:template match="livros/livro">
  <xsl:value-of select="@nome"/>
  <xsl:if test="not(position()=last())">, </xsl:if>
</xsl:template>
```

2. xsl:choose

```
<xsl:template match="livros/livro">
  <xsl:choose>
  <xsl:when test="@tipo = 1">tipo 1</xsl:when>
  <xsl:when test="@tipo = 2">tipo 2</xsl:when>
  <xsl:otherwise>Desconhecido</xsl:otherwise>
  </xsl:choose>
</xsl:template>
```

Repetição

- **Usando elemento `xsl:for-each`**
- **Atributo `select` é atribuído uma expressão XPATH que deverá retornar uma lista de nós**
- **Dentro do elemento `xsl:for-each` deverá ser definido o template a ser aplicado nos nós selecionados**

106

Exemplo

```
<xsl:template match="catalogo">
  <xsl:for-each select="livros/livro">
    <xsl:value-of select="@nome"/>
  </xsl:for-each>
</xsl:template>
```

Classificação

- **Usando elemento `xsl:sort`**
- **Pode aparecer dentro dos elementos `xsl:apply-templates` ou `xsl:for-each`**
- **Pode aparecer uma série de elementos `xsl:sort`**
    - **Isso permite classificar por várias colunas**
- **O valor do atributo `select` deve ser uma expressão XPATH apontando para um objeto válido**
    - **O valor padrão do atributo `select` é “.”**
- **O valor do atributo `data-type` pode ser “text” ou “number” indicando se o valor usado para classificação será string ou numérico**

107

Exemplo

Ordenando os livros dentro de um catalogo pelo nome.

```
<xsl:template match="catalogo">
    <xsl:apply-templates select="livros/livro">
        <xsl:sort select="@nome"/>
    </xsl:apply-templates>
</xsl:template>
<xsl:template match="livros/livro">
    <p><xsl:value-of select="@nome"/></p>
</xsl:template>
```

Formatação de Números

- **O elemento `xsl:number` pode ser usado para inserir um número formatado na saída**
- **O atributo `value` deve conter uma expressão**
- **O atributo `format` especifica o formato do número convertido para string**

108

Exemplo

Este exemplo vai imprimir uma lista de livros classificada pelo nome do livro e prefixar um número seqüencial ao nome.

```
<xsl:template match="livros">
  <xsl:for-each select="livro">
    <xsl:sort select="@nome"/>
    <p>
      <xsl:number value="position()" format="1. "/>
      <xsl:value-of select="@nome"/>
    </p>
  </xsl:for-each>
</xsl:template>
```

Resultado

1. Java
2. XML
3. XSL

...

Importando Folhas de Estilo XSL

- **Elemento `xsl:import`**
  - **Importa o arquivo especificado no atributo `href` dentro do arquivo XSL atual**
  - **Os templates do arquivo importado têm uma precedência menor**
  - **Só pode aparecer dentro do elemento `xsl:stylesheet` e antes de qualquer outro elemento**
- **Elemento `xsl:include`**
  - **Igual ao elemento `xsl:import`**
  - **Templates do arquivo XSL importado têm a mesma precedência dos templates locais**

109

Exemplo

xsl:import

```
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:import href="marca_corporativa.xsl"/>
  .
  .
  .
</xsl:stylesheet>
```

Chamando Templates Nomeados

- **Usando elemento `xsl:call-template`**
- **O template chamado tem que ter o mesmo nome especificado no atributo nome do elemento**

110

Exemplo

```
<xsl:template match="/">
    <xsl:call-template name="imprimir-titulo"/>
</xsl:template>
<xsl:template name="imprimir-titulo">
    <h1>Título</h1>
</xsl:template>
```

Criando Elementos

- **Usando elemento `xsl:element`**
- **O nome do elemento criado é especificado usando atributo nome**
- **O espaço identificador pode ser especificado usando atributo `namespace`**
- **Um conjunto de atributos pode ser especificado usando o atributo `use-attribute-sets`**

111

Exemplo

```
<xsl:template match="/">
    <xsl:call-template name="imprimir-titulo"/>
</xsl:template>
<xsl:template name="imprimir-titulo">
    <xsl:element name="h1" namespace="">
        Título
    </xsl:element>
</xsl:template>
```

Criando Atributos

- **Elemento `xsl:attribute`**
  - **Atributo `name` especifica o nome**
  - **Atributo `namespace` especifica o espaço identificador**
- **Elemento `xsl:attribute-set` para criar conjuntos**
  - **Contém uma série de elementos `xsl:attribute`**
  - **Atributo `name` especifica o nome do conjunto**
  - **Nome de um conjunto pode ser atribuído a atributo `use-attribute-sets` dos elementos `xsl:element`, `xsl:copy` ou outro `xsl:attribute-set`**

112

Exemplo

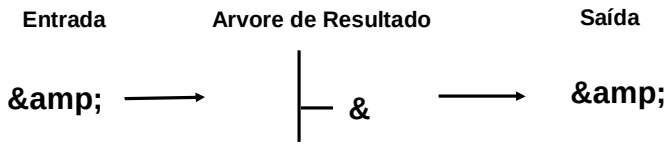
```
<xsl:template match="/">
    <xsl:call-template name="imprimir-titulo"/>
</xsl:template>
<xsl:attribute-set name="atrib-titulo">
    <xsl:attribute name="class">classe-h1</xsl:attribute>
    <xsl:attribute name="title">Título</xsl:attribute>
</xsl:attribute-set>
<xsl:template name="imprimir-titulo">
    <xsl:element name="h1" use-attribute-sets="atrib-titulo">
        Título
    </xsl:element>
</xsl:template>
```

Resultado

```
<?xml version="1.0" encoding="UTF-16"?>
<h1 class="classe-h1" title="Título">
        Título
</h1>
```

Criando Texto

- **Usando elemento `xsl:text`**
- **Usado para colocar texto na saída**
- **Atributo `disable-output-escaping` especifica se o tratamento de saída deve ser desabilitado**



disable-output-escaping = "no"

113

Exemplo 1

```
<xsl:text disable-output-escaping="no">A &amp; B</xsl:text>
```

Resultado

A & B

Exemplo 2

```
<xsl:text disable-output-escaping="yes">A &amp; B</xsl:text>
```

Resultado

A & B

Criando Comentários e Instruções de Processamento

- **Elemento `xsl:comment`**
  - **Para criar comentário**
- **Elemento `xsl:processing-instruction`**
  - **Para criar instrução de processamento**
  - **Pode ser usado para atribuir uma CSS ao documento de saída**

114

Exemplo xsl:comment

```
<xsl:comment>Isto é um comentário</xsl:comment>
```

Exemplo xsl:porcessing-instruction

xsl:porcessing-instruction pode ser usado para atribuir um documento CSS ao resultado XML.

```
<xsl:template match="/">
  <xsl:processing-instruction name="xml-stylesheet">
    href="article.css" type="text/css"
  </xsl:processing-instruction>
  <xsl:apply-templates/>
</xsl:template>
```

Exemplo – Criação de Menu

- **Vamos colocar os dados representando o menu num arquivo XML como mostrado a seguir**

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet href="menu.xsl" type="text/xsl"?>
<menu>
   <link url="1.asp" nome="1"/>
   <link url="2.asp" nome="2"/>
   <titulo nome="t1">
       <link url="3.asp" nome="3"/>
   </titulo>
   <titulo nome="t2">
       <link url="4.asp" nome="4"/>
   </titulo>
</menu>
```

115

Segue a XSL que gera a HTML do menu.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output encoding="UTF-8" method="html"/>
  <xsl:template match="menu">
    <xsl:for-each select="link | titulo">
      <xsl:choose>
        <xsl:when test="name()='link'">
          <xsl:call-template name="link"/>
        </xsl:when>
        <xsl:when test="name()='titulo'">
          <xsl:call-template name="titulo"/>
        </xsl:when>
      </xsl:choose>
    </xsl:for-each>
  </xsl:template>
```

Exemplo – Criação de Menu

- **Resultado**

1
2
t1
3
t2
4

- **Melhorias**
  - **Aninhar os itens do menu com títulos**
  - **Permitir criar títulos dentro de títulos**
  - **Gerar DHTML para fechar e abrir títulos**

116

... Continued

```
  <xsl:template name="link">
    <xsl:element name="a">
      <xsl:attribute name="href">
        <xsl:value-of select="@url"/>
      </xsl:attribute><xsl:value-of select="@nome"/>
    </xsl:element><br/>
  </xsl:template>
  <xsl:template name="titulo">
    <xsl:value-of select="@nome"/><br/>
    <xsl:for-each select="link">
      <xsl:call-template name="link"/>
    </xsl:for-each>
  </xsl:template>
</xsl:stylesheet>
```

Transformação usando JAXP

- **Utilizando JAXP para transformar XML via programa**
- **Pacotes utilizados**
  - **javax.xml.parsers**
  - **org.xml.sax**
  - **org.w3c.dom**
  - **javax.xml.transform**
  - **javax.xml.transform.dom**
  - **javax.xml.transform.stream**
  - **java.io**

117

```
import java.io.*;
import org.xml.sax.*;             // SAXException, SAXParseException
import org.w3c.dom.*;             // Document, DOMException
import javax.xml.parsers.*;       // DocumentBuilder, DocumentBuilderFactory
import javax.xml.transform.*;     // Transformer, TransformerFactory
import javax.xml.transform.dom.*; // DOMSource;
import javax.xml.transform.stream.*; // StreamResult, StreamSource
public class XSLTransform {
    static Document document; // Valor global referenciado pelo tree-adapter
    public static void main (String argv []) {
        DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
        try {
            DocumentBuilder builder = factory.newDocumentBuilder();
            document = builder.parse("catalogo.xml");
            TransformerFactory tFactory = TransformerFactory.newInstance();
            StreamSource stylesheet = new StreamSource("catalogo.xsl");
            Transformer transformer = tFactory.newTransformer(stylesheet);
            DOMSource source = new DOMSource(document);
            StreamResult result = new StreamResult(System.out);
            transformer.transform(source, result);
        } catch (TransformerConfigurationException tce) {
            // Erro gerado pelo parser
            System.out.println ("\n** Erro de Transformer Factory");
```

Transformação usando JAXP

- **Classes utilizadas**
  - **DocumentBuilderFactory e DocumentBuilder**
    - **Para ler o documento XML**
  - **TransformerFactory, Transformer**
    - **Para transformar documento XML**
  - **DOMSource, StreamSource, StreamResult**
    - **Entrada e saída do transformador**

118

... Continued

```
            System.out.println(" " + tce.getMessage() );
        } catch (TransformerException te) {
            // Erro gerado pelo transformador
            System.out.println ("\n** Erro de Transformação");
            System.out.println(" " + te.getMessage() );
        } catch (SAXParseException spe) { // Erro gerado pelo parser
            System.out.println ("\n** Erro de Parser");
            System.out.println(" " + spe.getMessage() );
        } catch (SAXException sxe) {
            // Erro gerado pelo aplicativo
            // (ou erro de inicialização do parser)
            Exception x = sxe;
            if (sxe.getException() != null) x = sxe.getException();
            x.printStackTrace();
        } catch (ParserConfigurationException pce) {
            // Opções especificadas não podem criar o Parser
            pce.printStackTrace();
        } catch (IOException ioe) { // Erro de E/S
            ioe.printStackTrace();
        }
    } // main
}
```

Exercício 7

1. **Crie um documento XSL para gerar uma página HTML contendo uma tabela listando informações sobre os livros dentro do catálogo**

119

Utilize XMLSpy para concluir o exercício acima.

Estado de padrões do W3C

	1996	1997	1998	1999	2000	2001	2002	2003	2004
CSS	1.0		2.0						
DOM			Level-1		Level-2				
HTML		4.0							
MathML				1.01		2.0			
NS				1.0					
RDF				Parcial					
SMIL			1.0			2.0			
SOAP									
SVG						1.0			
XHTML					1.0				
XLink						1.0			
XML/DTD			1.0						
XMLBase						1.0			
XMLProtocol									
XPATH				1.0					
XPointer							1.0		
XQuery									
XS						1.0			
XSL:FO						1.0			
XSLT				1.0					

120