

# eXtensible Markup Language

Devendra Tewari

February 09, 2005

# eXtensible Markup Language

# Conteúdo

- ▶ XML bem Formada – Sintaxe XML
- ▶ XML Válida – DTD e Esquema
- ▶ Acesso Programático – DOM e SAX
- ▶ Transformando XML – CSS e XSL

# Bibliografia

## Livros

- ▶ Professional XML – Wrox e Ciência Moderna – Vários Autores
- ▶ XML Conceitos e Aplicações – Benoît Marchal – QUE e Berkeley
- ▶ Aprendendo XML – O'Reilly e Campus – Erik T. Ray

## Sites

- ▶ Site do W3C
- ▶ Site da OASIS

## Estado de padrões do W3C

---

	1996	1997	1998	1999	2000	2001	2002
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							

## Sintaxe da XML – 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>
```

# Porque XML?

- ▶ XML é uma HTML melhorada?
- ▶ Nenhum marcador predefinido
  - ▶ Extensível
- ▶ Sintaxe mais rígida
  - ▶ Fácil validar



# 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

# 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

# Outros Padrões Relacionados a XML

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

# Trabalhando com XML

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

# Estrutura de um Documento XML

- ▶ A declaração `xml`
- ▶ Elementos
- ▶ Atributos
- ▶ Comentários

# 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 > (&gt;) e < (&lt;)
- ▶ Novas entidades podem ser criadas usando uma DTD

# Atributos Especiais

`xml:space`

- ▶ Preservar ou não espaços em branco duplicados

`xml:lang`

- ▶ Linguagem natural do valor texto de um elemento

# 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"?>
```



## Seção 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

# 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 do atributo demarcado com aspas duplas
- ▶ Texto caractere sem caracteres reservados ou inválidos

# 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

# Quando usar Atributos para Propriedades

- ▶ Vantagens

- ▶ Possível restringir valor usando DTD
- ▶ Validação de ID e IDREF
- ▶ Requer menos espaço
- ▶ Fácil processar usando DOM e SAX

- ▶ Desvantagens

- ▶ Valores simples
- ▶ Não suporta atributos sobre atributos

# Quando usar Elementos Filhos para Propriedades

- ▶ Vantagens

- ▶ Suporta valores complexos
- ▶ Suporta atributos sobre atributos
- ▶ Extensível quando modelo de dados muda (um livro vários autores)

- ▶ Desvantagens

- ▶ Requer mais espaço
- ▶ Mais difícil processar

# Exercício 1

- ▶ Como utilizar XMLSpy
- ▶ Crie um catálogo de livros usando XML
  - ▶ Sugira um modelo para o documento
  - ▶ Crie o documento usando o modelo

XML Válida – DTD

# Porque Utilizar 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



# 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

# Componentes de DTD

- ▶ Sintaxe da DTD não utiliza XML
- ▶ Uma DTD contém os componentes a seguir
  - ▶ ELEMENT
  - ▶ ATTLIST
  - ▶ ENTITY
  - ▶ NOTATION

# 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

```
<!ELEMENT livro (#PCDATA)>
```

```
<!ELEMENT autor (titulo, nome)>
```

```
<!ELEMENT autor ANY>
```

# Declaração ELEMENT – Modelo de Conteúdo

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

```
<!ELEMENT livro (#PCDATA | autor)*>
```

```
<!ELEMENT livro (autor+, custo?)>
```

```
<!ELEMENT livro (custo, (autor | editora)+)>
```

# 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"  
>
```

# Atributos do 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
- ▶ Exemplos

```
<!ATTLIST livro nome ID #REQUIRED>
```

```
<!ATTLIST livro tipo IDREF #REQUIRED>
```

# Atributos do 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">
```

# Atributos tipo NMTOKEN e NMTOKENS

- ▶ NMTOKEN são fichas nomeadas
- ▶ 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  
>
```



## Atributo tipo NOTATION

- ▶ O atributo desse tipo é usado em conjunto com a declaração NOTATION (notação)
- ▶ Uma notação declara um formato e o aplicativo externo que deve processá-lo
- ▶ Exemplo

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

# DTD Interna

- ▶ Elementos declarados na DTD interna sobrepõem elementos da DTD externa
- ▶ Uso menos frequente porque
  - ▶ Ocupa espaço em cada documento
  - ▶ Difícil alterar por estar em vários documentos XML

# Exemplo Escola

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

# 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

## Exercício 2

- ▶ Crie uma DTD para o documento XML de catálogo de livros criado no Exercício 1

## XML Válida – Esquemas

# 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

# Componentes Primários

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



# Componentes Secundários

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

# Componentes Auxiliares

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

# Estrutura do Arquivo de Esquema

- ▶ O arquivo tem a extensão `xsd`
- ▶ Contém um elemento raiz chamado `schema`
- ▶ Elemento `schema` pode conter atributos
  - ▶ `attributeFormDefault`
  - ▶ `elementFormDefault`
  - ▶ `id`
  - ▶ `targetNamespace`
  - ▶ `version`
  - ▶ `xmlns` (mais de uma ocorrência)

# Associando Esquema ao 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

## Espaço Identificadores (namespaces)

- ▶ Evitam ambiguidade e colisão de nomes quando um elemento XML utilizar 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

# Definição de Tipos 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>
```

# Restringindo Valores Usando Expressões Regular

- ▶ Podemos restringir valores utilizando expressões regulares
- ▶ Uma expressão regular é especificada utilizando o elemento `xs:pattern` dentro do elemento `xs:restriction`
- ▶ Exemplo - `cpf.xml`

```
<cpf xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:noNamespaceSchemaLocation="cpf.xsd">011.077.604-6
```

## Esquema para validar CPF - 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}" />
    </xs:restriction>
  </xs:simpleType>
</xs:schema>
```



# Definição de Tipos Complexos

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

```
<xs:complexType name="tipoPedido">  
  <xs:sequence>  
    <xs:element name="produtoPara" type="tipoEnderecoBr">  
    <xs:element name="notaPara" type="tipoEnderecoBrasi">  
    <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
- ▶ Exemplo

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

# Declaração de Elementos

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

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

# 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
- ▶ Exemplo

```
<xs:notation name="jpeg" public="image/jpeg" system="vi  
<xs:element name="imagem">  
  <xs:complexType mixed="true">  
    <xs:attribute name="tipo" type="xs:NOTATION" use="r  
  </xs:complexType>  
</xs:element>
```

## Definição de Grupos de Atributo

- ▶ É usada para reutilizar grupos de atributos 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
- ▶ Exemplo

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

# Anotações

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

```
<xs:annotation>  
  <xs:documentation  
    source="Temperatura em farenheits"/>  
</xs:annotation>
```

# Grupos de Modelos

- ▶ Compostos por uma lista de
  - ▶ Elementos
  - ▶ Curingas
  - ▶ Grupos de Modelos
- ▶ Temos três tipos de grupos de modelos
  - ▶ Sequência
  - ▶ Conjunção
  - ▶ Disjunção



# Sequência

- ▶ Todos os elementos deveriam ser especificados e seguem uma sequência rígida
- ▶ Exemplo

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

# Conjunção e Disjunção

- ▶ Conjunção

- ▶ Todos os elementos deveriam ser especificados mas não seguem uma sequência rígida
- ▶ Uma conjunção é criada usando elemento `xs:all`

- ▶ Disjunção

- ▶ Qualquer um dos elementos precisa ser especificado
- ▶ Uma disjunção é criada usando o elemento `xs:choice`

# 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
- ▶ 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>
```

# 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
- ▶ Exemplo

```
<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)
- ▶ Exemplo

```
<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

- ▶ Tipos primitivos internos fazem parte da especificação de Esquemas XML

---

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

---

# Tipos Derivados Internos

- Tipos derivados a partir de tipos de dados primitivos 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			

---

## Exemplo Escola - Esquema

- ▶ Mesmos dados como no caso da DTD
- ▶ Esquema permite maior flexibilidade
  - ▶ Definição das chaves
    - ▶ Única
    - ▶ Primária
    - ▶ Estrangeira
  - ▶ Definição e extensão de tipos simples e complexos
  - ▶ Link



## Exercício 3

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

## Acesso Programático - DOM

# DOM – Document Object Model

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

# Objetos do DOM

- ▶ O Node representa um nó na árvore do DOM
- ▶ NodeList e NamedNodeMap são utilizados para agregar nós
- ▶ O DOM é composto de vários tipos de objetos que herdam de Node

---

Element	Attr	Text	CDATAsection
EntityReference	Entity	ProcessingInstruction	Comment
Document	DocumentType	DocumentFragment	Notation

---

## Objeto Node

- ▶ O objeto Node disponibiliza várias propriedades para navegar a árvore do DOM
  - ▶ `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 caso o nó atual tiver atributos
  - ▶ `nodeName`: Nome do nó
  - ▶ `nodeValue`: Valor texto do nó

# Métodos do Objeto Node

- ▶ Disponibilizados pelo objeto Node para inserir, duplicar, remover e substituir outros 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 recurso do DOM
  - ▶ `normalize`: Normaliza a estrutura do nó
  - ▶ `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

## 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



# 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

# 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

# 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

# Usando DOM para criar Documentos XML

- ▶ Usaremos MSXML nos exemplos
- ▶ O exemplo a seguir
  - ▶ 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
  - ▶ Gera a saída

```
<?xml version="1.0"?>
```

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

## 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"
```

# 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

## 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
```

## 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>
```

- ▶ Resultado

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



## Exercício 4

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

## Acesso Programático - SAX

# Simple API for XML

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

## Lendo um Documento XML Usando SAX

- ▶ Utilizamos parser SAX do pacote MSXML 3.0
- ▶ 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` passando como parâmetro o nome do arquivo XML

## Lendo um Documento XML Usando SAX em VB

Implements IVBSAXContentHandler

Const MAX\_DEPTH = 100

Dim nDepth As Long, sArray(MAX\_DEPTH) As String

```
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_startElement( _  
    strNamespaceURI As String, strLocalName As String, _  
    strQName As String, ByVal oAttributes As MSXML2.IVBSAXAttributeCollection)  
    sArray(nDepth) = strQName  
    Debug.Print String(nDepth, vbTab) & strQName  
    nDepth = nDepth + 1
```

End Sub

Public Function saxParse(ByVal fileName As String) As Boolean

## Lendo um Documento XML Usando SAX - Resultado

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

# 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

# Usando JAXP - DOM

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



## Usando JAXP DOM - Exemplo Java

```
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 {
        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());
            }
        }
    }
}
```

# 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`
    - ▶ Classes `SAXParser`, `SAXParserFactory` e `ParserConfigurationException`
  - ▶ `org.xml.sax`
    - ▶ Classe `SAXException`
  - ▶ `org.xml.sax.helpers`
    - ▶ Classe `DefaultHandler`

## Usando JAXP SAX - Exemplo Java

```
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 {
        SAXParser leitor = SAXParserFactory.newInstance().newSAXParser();
        leitor.parse("catalogo.xml", new LeitorSAX());
    }
    public void startElement(String namespaceURI, String lName,
        System.out.println(qName);
    }
}
```

- Empty methods have been pruned

## Exercício 5

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

# JDOM

- ▶ Limitações do DOM
  - ▶ Independente da Linguagem
    - ▶ Não foi desenhado com a linguagem Java em mente
    - ▶ Não faz uso das classes de coleção existentes
  - ▶ Herança de classes estrita
    - ▶ Demais objetos são derivados do objeto Node
  - ▶ Baseado em Interface
- ▶ Vantagens da JDOM
  - ▶ Para Java
  - ▶ Hierarquias de classes inexistentes

# 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

## Método Pull para ler documento XML em C#

```
using System;
using System.Xml;

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

## Transformando XML – CSS



# 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 superior
  - ▶ Netscape 6.x ou superior
- ▶ Padrão atualmente no nível 2 (CSS2)

## Exemplo de uso 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 es
    reunidos, um oficial trouxe a lista de convidados.
  </paragrafo>
</artigo>
```

## Exemplo de uso de CSS - artigo.css

```
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 }
```



### 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 ate ele 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 }

# Tipos de Seletores

- ▶ Seletor Universal

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

- ▶ Seletor de tipo

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

- ▶ Seletor de classe (válido somente para HTML)

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

- ▶ Seletor de ID

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

# Mais Tipos de Seletores

- ▶ 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 }
```

# Pseudo-elementos e Pseudo-Classes

## ► Pseudo-classes

- :first-child, :link, :visited, :hover, :active, :focus e :lang

```
DIV > P:first-child { text-indent: 0 }  
A:link { color: red }  
A:focus { background: yellow }  
HTML:lang(fr) { quotes: '« ' ' »' }
```

## ► Pseudo-elementos

- :first-line, :first-letter, :before e :after

```
P:first-line { text-transform: uppercase }  
P:first-letter { color: green; font-size: 200% }  
H1:before {content: counter(chapno, upper-roman) ". "}
```

# Valor das Propriedades

- ▶ Propriedades estão contidas dentro de um bloco de declaração
- ▶ Valores de alguns tipos básicos
- ▶ Comprimento
  - ▶ Um número seguido de uma unidade de medida como em, px, cm e mm
- ▶ Porcentagem
- ▶ Cor
  - ▶ Um valor hexadecimal, por exemplo, #FFFFFF é a cor branca
  - ▶ Uma especificação RGB explícita, por exemplo, rgb(0, 0, 255) ou rgb(0%, 0%, 100%)
  - ▶ Palavras chaves
- ▶ URL (para imagens)



# Propriedades da Caixa

- ▶ Os elementos são pintados dentro de uma caixa, retângulo por padrão
- ▶ Display
  - ▶ `display: block` cria um novo bloco contendo o texto do elemento
  - ▶ `display: inline` evita criar um novo bloco e mostra o texto dentro do bloco do elemento pai
  - ▶ `display: none` suprime o elemento
- ▶ Margem (externo à borda)
  - ▶ `margin-top`, `margin-right`, `margin-bottom` e `margin-left`, ou apenas `margin`

# Mais Propriedades da Caixa

- ▶ Borda
  - ▶ Tipo none, dotted, dashed, solid, double, groove, ridge, inset, ou outset, por exemplo, `border: solid, dotted, double, inset;`
  - ▶ Largura `border-top-width`, `border-right-width`, `border-bottom-width` e `border-left-width`, ou apenas `border-width`
- ▶ Padding (enchimento interno à borda)
  - ▶ `padding-top`, `padding-right`, `padding-bottom` e `padding-left`, ou apenas `padding`

# Propriedades de Texto e Fonte

- ▶ Nome da fonte - `font-family`
- ▶ Comprimento da fonte - `font-size`
- ▶ Estilo e peso - `font-style` e `font-weight`
- ▶ Alinhamento do Texto - `text-align` e `vertical-align`
- ▶ Endentação e Altura de Linha - `text-indent` e `line-height`

# Propriedades da Cor e Fundo

- ▶ Cor do primeiro plano - `color`
- ▶ Cor de fundo - `background-color`
- ▶ Cor da borda - `border-color`
- ▶ Imagem de fundo - `background-image`

# 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
- ▶ Exemplo

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

## Exercício 6

- ▶ Crie um documento CSS para mostrar o catálogo de livros dentro do browser

## Transformando XML – XSL

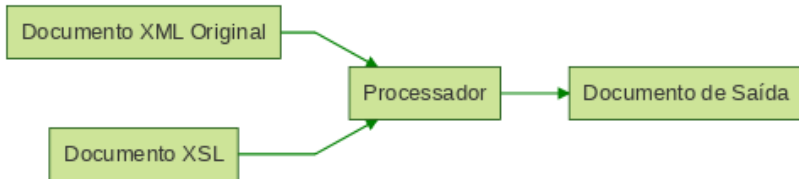
# eXtended Stylesheet Language

- ▶ CSS faz transformações básicas
- ▶ Queremos transformar
  - ▶ XML para XML
  - ▶ XML para HTML
  - ▶ Agrupar, filtrar, calcular e classificar
- ▶ Não estamos falando em deixar de usar a CSS



# Conceito de XSL

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



## Exemplo de XSLT - Documento Original

- ▶ 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>
```

## Exemplo de XSLT

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

## Exemplo de XSLT - Resultado

---

Livros

---

Professional XML  
XML By Example

---

# 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
- ▶ Exemplo

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

# Caminhos

Caminho	Descrição
/	Elemento raiz
//	Seleciona todos os nós descendentes
*	Qualquer elemento
.	Nó atual
	Seleciona entre várias caminhos (OU)

## Controlando a Saída

- ▶ Podemos controlar a saída do processador XSLT usando `xsl:output`
- ▶ Podemos escolher a saída atribuindo os seguinte valores para o atributo `method`
  - ▶ `xml`
  - ▶ `text`
  - ▶ `html`
- ▶ Deve aparecer logo após `xsl:stylesheet`
- ▶ Exemplo

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.  
  <xsl:output method="xml"/>  
  <!-- ... -->  
</xsl:stylesheet>
```

## Definindo Templates

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



# Variáveis e Parâmetros

- ▶ Elemento `xsl:variable`
  - ▶ O tipo do valor pode ser qualquer texto ou nó selecionado usando expressões XPATH ou constante
- ▶ Elemento `xsl:param`
  - ▶ Parâmetros podem ser passados usando o elemento `xsl:with-param`
- ▶ 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

# Processamento Condicional

- ▶ Elemento `xsl:if`
  - ▶ Expressão lógica atribuída ao atributo `test`
  - ▶ Não tem “Se Não”

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

- ▶ 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

# Repetição

- ▶ Usando elemento `xsl:for-each`

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

- ▶ Atributo `select` é atribuído uma expressão XPATH que deverá retornar uma lista de nós
- ▶ Dentro do elemento deverá ser definido o template a ser aplicado nos nós selecionados

## Classificação

- ▶ Usando elemento `xsl:sort`

```
<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>
```

- ▶ Pode aparecer dentro dos elementos `xsl:apply-templates` ou `xsl:for-each`
- ▶ Pode aparecer uma sequencia 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

# Formatação de Números

- ▶ O elemento `xsl:number` pode ser usado para inserir um número formatado na saída

```
<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>
```

- ▶ O atributo `value` deve conter uma expressão
- ▶ O atributo `format` especifica o formato de conversão do número para texto

# Importando Folhas de Estilo XSL

- ▶ Elemento `xsl:import`
  - ▶ Importa o arquivo especificado pelo 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

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

# Chamando Templates Nomeados

- ▶ Usando elemento `xsl:call-template`
- ▶ O atributo `name` do elemento deve especificar o nome do template à chamar

```
<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 `name`
- ▶ O espaço identificador pode ser especificado usando atributo `namespace`
- ▶ Um conjunto de atributos pode ser especificado usando o atributo `use-attribute-sets`

```
<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 à atributo `use-attribute-sets` dos elementos `xsl:element`, `xsl:copy` ou `xsl:attribute-set`

## Criando Atributos - Exemplo

### ► 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>
```

### ► Saída

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

# 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

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

Saida - A & B

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

Saida - 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

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

## Exemplo de Criação de Menu - XML

- ▶ 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>
```

## Exemplo de Criação de Menu - XSL

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/
  <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>
  <xsl:template name="link">
    <xsl:element name="a">
      <xsl:attribute name="href">
```

# Exemplo de 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



# 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

## Transformação usando JAXP - Código

```
import java.io.*;
import org.xml.sax.*;           // SAXException, SAXParser
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
    public static void main (String argv []) {
        DocumentBuilderFactory factory = DocumentBuilderFactory
        try {
            DocumentBuilder builder = factory.newDocumentBuilder();
            document = builder.parse("catalogo.xml");
            TransformerFactory tFactory = TransformerFactory
            StreamSource stylesheet = new StreamSource("cat
            Transformer transformer = tFactory.newTransform
            DOMSource source = new DOMSource(document);
```

# 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

## Exercício 7

- ▶ Crie um documento XSL para gerar uma página HTML contendo uma tabela listando informações sobre os livros dentro do catálogo