

Transformações em XML: XSLT

Vanessa Braganholo

Importância de XSLT

- ▶ XSLT é um padrão para transformação de documentos XML para qualquer representação textual
 - ▶ Templates de transformação são aplicados a objetos XML
 - ▶ Entrada: documento XML
 - ▶ Saída: qualquer documento em formato texto (HTML, XML, TXT, RTF, etc)



Princípio de funcionamento da transformação

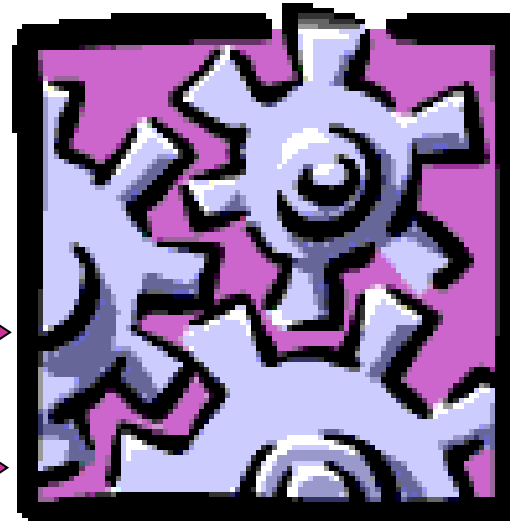
Documento de entrada

```
<carta>
  <cabecalho>
    . . .
  </cabecalho>

  <corpo>
    . . .
  </corpo>
</carta>
```

```
...
<template match="cabecalho">
  <apply-templates/>
</template>
...
```

Regras de transformação – stylesheets



Processador XSLT

Documento Transformado

WindStar 2000
Les rosières en buget
AB562 Saint Pétaouchnoqe



Tel: 012133564
Fax: 879765426

Saint Pétaouchnoqe,
Le 30 nivose 2004

Editions Duschmol,
12 rue Schmurz
YT123 Rapis

Objeto: Dívida

Prezado Senhot,

Bla bla bli, bli blo bla, kkkk vhlg
vckjdhklbg fidskjbvhw feje slc
ifehfe fhckh c jeflecj n khef
iheznf jùkvbc lkhdklvn v

sssinatura

Rodapé



XSLT

- ▶ Através do uso de XSLT, é possível:
 - ▶ adicionar prefixos e/ou sufixos a um conteúdo de texto;
 - ▶ eliminar, criar, reestruturar e ordenar elementos;
 - ▶ reusar elementos em qualquer parte do documento;
 - ▶ transformar dados de um formato XML para outro formato baseado em texto (XML, HTML, TeX, etc.);



XSLT

- ▶ Um processador XSLT
 - ▶ recebe como entrada um documento XML
 - ▶ gera na saída um outro documento em formato texto
- ▶ Se o documento de saída for um documento XML
 - ▶ ele pode estar estruturado de acordo com uma DTD diferente da DTD do documento de entrada
- ▶ A transformação é especificada em um *style sheet*
- ▶ Um *style sheet* segue a sintaxe do padrão XML



Style Sheets

- ▶ Um *style sheet* é formado por um conjunto de regras *template*
 - ▶ transformações são executadas de acordo com tais regras
- ▶ Cada regra "casa" com um tipo de elemento no documento de entrada utilizando expressões **XPath**
- ▶ As *tags* originais são substituídas por novas *tags* de saída



Estrutura Geral

- ▶ O elemento raiz é denominado stylesheet
- ▶ O namespace de XSLT deve ser declarado
xmlns:xsl=http://www.w3.org/1999/XSL/Transform

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

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

```
....
```

```
</xsl:stylesheet>
```



Estrutura Geral

- ▶ Templates (ou regras) são criadas através do elemento `<xsl:template>`

```
<xsl:template match="title">
```

```
...
```

```
</xsl:template>
```

- ▶ Recursividade de templates é criada através do elemento `<xsl:apply-templates>`

```
<xsl:template match="book">
```

```
...
```

```
  <xsl:apply-templates/>
```

```
</xsl:template>
```



Processamento

- ▶ A execução usa uma pilha auxiliar
 - ▶ Quem guia a execução é a pilha

- 1. Empilhar a raiz do documento XML
- 2. Desempilhar o topo da pilha **t**
- 3. Encontrar, no **style-sheet**, um template correspondente ao nodo **t**
- 4. Executar o template
- 5. Se template possui comando **apply-templates**, empilhar os filhos de **t**
- 6. Se pilha não está vazia, voltar ao passo 2



Processamento recursivo

```
<?xml version="1.0"?>
<books>
  <book>
    <title>ABC</title>
    <author>John</author>
  </book>
  <book>
    <title>DEF</title>
    <author>Joseph</author>
  </book>
</books>
```

```
<xsl:stylesheet version="1.0"
xmlns:xsl:....>

  <xsl:template match="books">
    ...
    <xsl:apply-templates/>
  </xsl:template>

  <xsl:template match="book">
    ...
    <xsl:apply-templates/>
  </xsl:template>

</xsl:stylesheet>
```



Processamento recursivo

```
<?xml version="1.0"?>
<b>books</b>
  <book>
    <title>ABC</title>
    <author>John</author>
  </book>
  <book>
    <title>DEF</title>
    <author>Joseph</author>
  </book>
</b>
```

```
<xsl:stylesheet version="1.0"
xmlns:xsl:....>
  {
    <xsl:template match="books">
      ...
      <xsl:apply-templates/>
    </xsl:template>

    <xsl:template match="book">
      ...
      <xsl:apply-templates/>
    </xsl:template>
  }
</xsl:stylesheet>
```



Processamento recursivo

```
<?xml version="1.0"?>
<b>books</b>
  <b>book</b>
    <title>ABC</title>
    <author>John</author>
  </b>
  <b>book</b>
    <title>DEF</title>
    <author>Joseph</author>
  </b>
</b>
```

```
<xsl:stylesheet version="1.0"
xmlns:xsl:....>
  {
    <xsl:template match="books">
      ...
      <xsl:apply-templates/>
    </xsl:template>

    <xsl:template match="book">
      ...
      <xsl:apply-templates/>
    </xsl:template>
  }
</xsl:stylesheet>
```



Processamento recursivo

```
<?xml version="1.0"?>
<books>
  <book>
    <title>ABC</title>
    <author>John</author>
  </book>
  <book>
    <title>DEF</title>
    <author>Joseph</author>
  </book>
</books>
```

```
<xsl:stylesheet version="1.0"
xmlns:xsl:....>

  <xsl:template match="books">
    ...
    <xsl:apply-templates/>
  </xsl:template>

  <xsl:template match="book">
    ...
    <xsl:apply-templates/>
  </xsl:template>

</xsl:stylesheet>
```



Processamento recursivo

```
<?xml version="1.0"?>
<books>
  <book>
    <title>ABC</title>
    <author>John</author>
  </book>
  <book>
    <title>DEF</title>
    <author>Joseph</author>
  </book>
</books>
```

```
<xsl:stylesheet version="1.0"
xmlns:xsl:....>

  <xsl:template match="books">
    ...
    <xsl:apply-templates/>
  </xsl:template>

  <xsl:template match="book">
    ...
    <xsl:apply-templates/>
  </xsl:template>

</xsl:stylesheet>
```



Regras *default*

- ▶ XSLT possui algumas regras *default*
- ▶ Uma *style-sheet* vazia aplica as regras *default* ao documento XML que está sendo processado
 - ▶ Processa o documento todo
 - ▶ Coloca todo o conteúdo dos elementos texto na saída



Exemplo Documento XML

```
<booklist>
  <book id="BOX00">
    <author>Box, D. and Skonnard, A. and Lam, J.</author>
    <editor>Series</editor>
    <title>Essential XML - Beyond Markup</title>
    <publisher>Addison-Wesley</publisher>
    <year>2000</year>
    <key/>
    <volume/>
    <number/>
    <series/>
    <address/>
    <edition/>
    <month>July</month>
    <note/>
    <annotate/>
    <url>http://www.develop.com/books/essentialxml</url>
  </book>
```

.....



Exemplo Documento XML

.....

```
<book id="MAR99">
```

```
  <author>Maruyama, H. and Tamura, K. and Uramoto, N.</author>
```

```
  <title>XML and Java: Developing of Web Applications</title>
```

```
  <publisher>Addison-Wesley</publisher>
```

```
  <year>1999</year>
```

```
  <address>MA</address>
```

```
  <month>August</month>
```

```
</book>
```

```
<book id="BRA00">
```

```
  <author>Bradley, N.</author>
```

```
  <title>The XML Companion</title>
```

```
  <publisher>Addison-Wesley</publisher>
```

```
  <year>2000</year>
```

```
  <address>Great Britain</address>
```

```
  <edition>2</edition>
```

```
  <month>August</month>
```

```
</book>
```

```
</booklist>
```



Exemplo (01-sample.xsl)

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

```
<xsl:stylesheet version="1.0"  
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">  
  <xsl:output method="text" indent="yes"/>
```

```
</xsl:stylesheet>
```



Resultado

Box, D. and Skonnard, A. and Lam, J. Series Essential XML - Beyond Markup Addison-Wesley 2000 July <http://www.develop.com/books/essentialxml>

Maruyama, H. and Tamura, K. and Uramoto, N. XML and Java: Developing of Web Applications Addison-Wesley 1999 MA August

Bradley, N. The XML Companion Addison-Wesley 2000 Great Britain 2 August



Regras *default*

- ▶ Regra 1:
 - ▶ Processar todo o documento XML

```
<xsl:template match="/|*">  
  <xsl:apply-templates/>  
</xsl:template>
```

- ▶ `xsl:apply-templates` faz com que os filhos do nodo atual sejam processados recursivamente



Regras *default*

- ▶ Regra 2:

- ▶ Copiar o conteúdo texto dos elementos para a saída

```
<xsl:template match="text()">  
  <xsl:value-of select="."/>  
</xsl:template>
```

- ▶ `xsl:value-of select="."` faz com que o conteúdo do nodo atual seja copiado para a saída



Exemplo (02-sample.xsl)

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  version="1.0">
  <xsl:output method="text" indent="yes"/>

  <xsl:template match="booklist">
    =====Lista de Livros=====
    <xsl:apply-templates/>
    =====Fim da lista=====
  </xsl:template>

</xsl:stylesheet>
```



Resultado

=====Lista de livros=====

Box, D. and Skonnard, A. and Lam, J. Series Essential XML - Beyond Markup Addison-Wesley 2000 July <http://www.develop.com/books/essentialxml>
Maruyama, H. and Tamura, K. and Uramoto, N. XML and Java: Developing of Web Applications Addison-Wesley 1999 MA August
Bradley, N. The XML Companion Addison-Wesley 2000 Great Britain 2 August

=====Fim da lista=====



Função name()

- ▶ XPath possui uma função name() que pode ser usada para imprimir o nome do elemento que casou com uma determinada regra

```
<xsl:template match="book">  
  <xsl:value-of select="name()" />  
</xsl:template>
```



Exemplo (02-1-sample.xsl)

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  version="1.0">
  <xsl:output method="text" indent="yes"/>

  <xsl:template match="*">
    <xsl:value-of select="name()"/>:
    <xsl:apply-templates/>
  </xsl:template>

</xsl:stylesheet>
```



Resultado

booklist: book: author: Box, D. and Skonnard, A. and Lam, J.editor: Seri
title: Essential XML - Beyond Markuppublisher: Addison-Wesleyyear: 2000key:
volume: number: series: address: edition: month: Julynote: annote: url:
<http://www.develop.com/books/essentialxml>book: author: Maruyama, H. and
Tamura, K. and Uramoto, N.title: XML and Java: Developing of Web
Applicationspublisher: Addison-Wesleyyear: 1999address: MAmonth:
Augustbook: author: Bradley, N.title: The XML Companionpublisher: Addison-
Wesleyyear: 2000address: Great Britainedition: 2month: August



Exemplo (03-sample.xsl)

```
<?xml version="1.0"?>  
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"  
  version="1.0">  
  <xsl:output method="text" indent="yes"/>
```

```
  <xsl:template match="*">  
  </xsl:template>
```

```
  <xsl:template match="booklist">  
    =====Lista de livros=====  
    <xsl:apply-templates/>  
    =====Fim da lista=====  
  </xsl:template>
```

```
</xsl:stylesheet>
```



Resultado

=====Lista de livros=====

=====Fim da lista=====



Exemplo (04-sample.xsl)

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  version="1.0">
  <xsl:output method="text" indent="yes"/>

  <xsl:template match="author">
    =====Livro do autor=====
    <xsl:apply-templates/>
    =====Detalhes do livro=====
  </xsl:template>

</xsl:stylesheet>
```



Resultado

====Livro do autor=====

Box, D. and Skonnard, A. and Lam, J.

====Detalhes do livro=====

SeriesEssential XML - Beyond MarkupAddison-
Wesley2000July<http://www.develop.com/books/essentialxml>

====Livro do autor=====

Maruyama, H. and Tamura, K. and Uramoto, N.

====Detalhes do livro=====

XML and Java: Developing of Web ApplicationsAddison-Wesley1999MAAugust

====Livro do autor=====

Bradley, N.

====Detalhes do livro=====

The XML CompanionAddison-Wesley2000Great Britain2August



Exemplo (05-sample.xsl)

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  version="1.0">
  <xsl:output method="text" indent="yes"/>

  <xsl:template match="booklist">
    =====Lista de livros=====
    <xsl:apply-templates/>
    =====Fim da lista=====
  </xsl:template>

  <xsl:template match="author">
    =====Livro do autor=====
    <xsl:apply-templates/>
    =====Detalhes do livro=====
  </xsl:template>
</xsl:stylesheet>
```



Resultado

=====Lista de livros=====

=====Livro do autor=====

Box, D. and Skonnard, A. and Lam, J.

=====Detalhes do livro=====

SeriesEssential XML - Beyond MarkupAddison-
Wesley2000July<http://www.develop.com/books/essentialxml>

=====Livro do autor=====

Maruyama, H. and Tamura, K. and Uramoto, N.

=====Detalhes do livro=====

XML and Java: Developing of Web ApplicationsAddison-Wesley1999MAAugust

=====Livro do autor=====

Bradley, N.

=====Detalhes do livro=====

The XML CompanionAddison-Wesley2000Great Britain2August

=====Fim da lista=====



Exemplo (06-sample.xsl)

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xsl:output method="text" indent="yes"/>

  <xsl:template match="text()"/>

  <xsl:template match="booklist">
    <xsl:apply-templates/>
  </xsl:template>

  <xsl:template match="book">
    _____ Livro _____
    <xsl:apply-templates/>
  </xsl:template>

  <xsl:template match="author">
    AUTOR: <xsl:value-of select="."/>
  </xsl:template>
</xsl:stylesheet>
```



Resultado

_____ Livro _____

AUTOR: Box, D. and Skonnard, A. and Lam, J.

_____ Livro _____

AUTOR: Maruyama, H. and Tamura, K. and Uramoto, N.

_____ Livro _____

AUTOR: Bradley, N.



Exercício 1

- ▶ Utilizando o arquivo pedido4.xml, obtenha a seguinte saída:

Pedido

Cliente: ABC

CGC: 00.000.000/0001-00

Local:

Rua das Flores, 75 - Porto Alegre – RS

Dica: para colocar espaços na saída:

<xsl:text> </xsl:text>



Processamento seletivo

- ▶ O atributo **select** do elemento **apply-templates** é utilizado para selecionar determinados filhos para serem processados e ignorar o restante

```
<xsl:template match="booklist">  
  <xsl:apply-templates  
    select="book[@id='MAR99']"/>  
</xsl:template>
```



Exemplo (07-sample.xsl)

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xsl:output method="text" indent="yes"/>

  <xsl:template match="text()"/>

  <xsl:template match="booklist">
    <xsl:apply-templates select="book[@id='MAR99']"/>
  </xsl:template>

  <xsl:template match="book">
    _____ Livro _____
    <xsl:apply-templates/>
  </xsl:template>

  <xsl:template match="author">
    AUTOR: <xsl:value-of select="."/>
  </xsl:template>
</xsl:stylesheet>
```



Resultado

_____Livro_____

AUTOR: Maruyama, H. and Tamura, K. and Uramoto, N.



Exercício 2

- ▶ Utilizando o arquivo pedido4.xml, obtenha a seguinte saída:
 - ▶ Observe que apenas o item “caneta azul” está na saída

Pedido no. 1000

Cliente: ABC

CGC: 00.000.000/0001-00

Itens

* caneta azul

100

2

- ▶ Por enquanto, não se preocupem com alinhamento
-



Gerando um novo documento XML

- ▶ É possível gerar novas *tags* XML na saída, produzindo um novo documento XML de saída a partir da entrada
- ▶ Tudo o que não possui o *namespace* de XSLT é copiado para a saída

```
<xsl:template match="book">  
  <livro>  
    <xsl:apply-templates/>  
  </livro>  
</xsl:template>
```



Exemplo (09-sample.xsl)

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
<xsl:output indent="yes"/>


  <xsl:template match="text()"/>

  <xsl:template match="booklist">
    <ListaLivros><xsl:apply-templates/></ListaLivros>
  </xsl:template>

  <xsl:template match="book">
    <Livro><xsl:apply-templates/></Livro>
  </xsl:template>

  <xsl:template match="author">
    <Autor><xsl:value-of select="."/></Autor>
  </xsl:template>

  <xsl:template match="title">
    <Titulo><xsl:value-of select="."/></Titulo>
  </xsl:template>
</xsl:stylesheet>
```



Resultado

```
<?xml version="1.0" encoding="UTF-16"?>
```

```
<ListaLivros>
```

```
  <Livro>
```

```
    <Autor>Box, D. and Skonnard, A. and Lam, J.</Autor>
```

```
    <Titulo>Essential XML - Beyond Markup</Titulo>
```

```
  </Livro>
```

```
  <Livro>
```

```
    <Autor>Maruyama, H. and Tamura, K. and Uramoto, N.</Autor>
```

```
    <Titulo>XML and Java: Developing of Web Applications</Titulo>
```

```
  </Livro>
```

```
  <Livro>
```

```
    <Autor>Bradley, N.</Autor>
```

```
    <Titulo>The XML Companion</Titulo>
```

```
  </Livro>
```

```
</ListaLivros>
```



Exercício 3

- ▶ Utilizando o arquivo pedido4.xml, obtenha a seguinte saída:

```
<pedido>
  <numero>1000</numero>
  <cliente>
    <nome>ABC</nome>
    <cgc>00.000.000/0001-00</cgc>
    <rua>Rua das Flores,75 </rua>
    <cidade>Porto Alegre</cidade>
    <estado>RS</estado>
  </cliente>
  <itens_pedido>
    <item>
      <produto>caneta azul</produto>
      <quant>100</quant>
      <preco>2</preco>
    </item>
    <item>
      <produto>papel</produto>
      <quant>100</quant>
      <preco>8</preco>
    </item>
  </itens_pedido>
</pedido>
```



Gerando HTML

- ▶ Da mesma forma que podemos gerar novos elementos XML na saída, também podemos gerar elementos HTML
- ▶ Basta fazer com que o documento gerado possua a estrutura básica de HTML

```
<HTML>  
  <HEAD>  
    <TITLE>título</TITLE>  
  </HEAD>  
  <BODY>  
    conteúdo da página HTML  
  </BODY>  
</HTML>
```



Exercício 4

- ▶ Exiba o pedido4.xml no formato HTML
- ▶ Utilize uma tabela para exibir os itens do pedido



Ordenando elementos

- ▶ O elemento **sort** é utilizado para ordenar uma lista de elementos
 - ▶ Atua sobre os elementos **selecionados pelo apply-templates**
 - ▶ Os elementos são ordenados no documento de saída
- ▶ Exemplo: coloca em ordem alfabética a lista de itens

```
<list>
```

```
  <item>ZZZ</item>
```

```
  <item>AAA</item>
```

```
  <item>MMM</item>
```

```
</list>
```

```
<xsl:template match="list">
```

```
  <xsl:apply-templates><xsl:sort/></xsl:apply-templates>
```

```
</xsl:template>
```



Ordenando elementos

- ▶ O atributo `select` do elemento **sort** é utilizado para especificar o critério pelo qual ordenar

```
<xsl:template match="booklist">  
  <xsl:apply-templates>  
    <xsl:sort select="year" />  
  </xsl:apply-templates>  
</xsl:template>
```



Atributos de sort

- ▶ O elemento **sort** possui atributos opcionais que podem ser utilizados
 - ▶ o atributo **order** especifica a ordem:
 - ▶ ascending (o *default*)
 - ▶ descending
 - ▶ o atributo **data-type** especifica o tipo:
 - ▶ text (o *default*)
 - ▶ number para ordenar valores numéricos
 - ▶ o atributo **case-order** indica se
 - ▶ “a” aparece antes de “A” (lower-first)
 - ▶ ou o contrário (upper-first)



Exemplo (1 1-sample.xsl)

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xsl:output method="text" indent="yes"/>

  <xsl:template match="text()"/>

  <xsl:template match="booklist">
    <xsl:apply-templates><xsl:sort select="year" /></xsl:apply-templates>
  </xsl:template>

  <xsl:template match="book">
    _____ Livro _____
    <xsl:apply-templates/>
  </xsl:template>

  <xsl:template match="author">
    AUTOR: <xsl:value-of select="."/>
  </xsl:template>

</xsl:stylesheet>
```



Resultado

_____ Livro _____

AUTOR: Maruyama, H. and Tamura, K. and Uramoto, N.

_____ Livro _____

AUTOR: Box, D. and Skonnard, A. and Lam, J.

_____ Livro _____

AUTOR: Bradley, N.



Numeração automática

- ▶ O elemento **number** é utilizado para numerar os elementos em uma lista de elementos
 - ▶ Facilita a manutenção de uma lista de itens
- ▶ Exemplo: insere o número de sequência antes do conteúdo do elemento

```
<xsl:template match="item">  
  <xsl:number /><xsl:apply-templates />  
</xsl:template>
```



Exemplo (12-sample.xsl)

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xsl:output method="text" indent="yes"/>

  <xsl:template match="text()"/>

  <xsl:template match="booklist">
    <xsl:apply-templates/>
  </xsl:template>

  <xsl:template match="book">
    _____ Livro _____
    Numero: <xsl:number /> <xsl:apply-templates />
  </xsl:template>

  <xsl:template match="author">
    AUTOR: <xsl:value-of select="."/>
  </xsl:template>

</xsl:stylesheet>
```



Resultado

_____ Livro _____

Numero:1

AUTOR: Box, D. and Skonnard, A. and Lam, J.

_____ Livro _____

Numero:2

AUTOR: Maruyama, H. and Tamura, K. and Uramoto, N.

_____ Livro _____

Numero:3

AUTOR: Bradley, N.



Modos

- ▶ O atributo `mode` do elemento `template` é utilizado para definir vários templates associados a um elemento
 - ▶ útil quando um elemento é formatado de diferentes formas dependendo do caso



Exemplo (13-sample.xsl)

```
<xsl:template match="text()" />
```

```
<xsl:template match="booklist">
```

```
Lista de livros dos autores:=====
```

```
    <xsl:apply-templates select="//booklist/book/author" mode="list" />
```

```
=====
```

```
    <xsl:apply-templates />
```

```
</xsl:template>
```

```
<xsl:template match="book">
```

```
    _____ Livro _____
```

```
    <xsl:apply-templates />
```

```
</xsl:template>
```

```
<xsl:template match="author">
```

```
AUTOR: <xsl:value-of select="." />
```

```
</xsl:template>
```

```
<xsl:template match="author" mode="list">
```

```
    <xsl:value-of select="." /> ,
```

```
</xsl:template>
```

```
<xsl:template match="title">
```

```
TITULO: <xsl:value-of select="." />
```

```
</xsl:template>
```



Resultado

Lista de livros dos autores:===== Box, D.
and Skonnard, A. and Lam, J., Maruyama, H. and Tamura, K. and
Uramoto, N., Bradley, N.,

=====

_____ Livro _____

AUTOR: Box, D. and Skonnard, A. and Lam, J. TITULO:
Essential XML - Beyond Markup

_____ Livro _____

AUTOR: Maruyama, H. and Tamura, K. and Uramoto, N.
TITULO: XML and Java: Developing of Web Applications

_____ Livro _____

AUTOR: Bradley, N. TITULO: The XML Companion



Reusando templates

- ▶ O elemento **call-template** é utilizado para invocar um **template**
 - ▶ útil para reusar a mesma formatação em vários lugares
- ▶ **Funcionamento:**
 - ▶ o template com a formatação comum é nomeado
 - ▶ o template é invocado através do elemento **call-template**



Reusando templates

- ▶ O atributo name é utilizado para associar um nome único ao template

```
<xsl:template name="CreateHeader">  
  <hr/>  
  <h2>**** <xsl:apply-templates/> ****</h2>  
  <hr/>  
</xsl:template>
```

- ▶ O atributo name do elemento call-template é utilizado para invocar o template

```
<xsl:template match="title">  
  <xsl:call-template name="CreateHeader"/>  
</xsl:template>  
<xsl:template match="head">  
  <xsl:call-template name="CreateHeader"/>  
</xsl:template>
```



Criando elementos com `element`

- ▶ Existem dois modos de criar um elemento na saída
 - ▶ escrever o elemento, por exemplo `<pedido>`
 - ▶ utilizar o elemento `<xsl:element>`
- ▶ Ao utilizar o elemento `<xsl:element>`
 - ▶ o atributo `name` indica o nome do elemento a ser criado
 - ▶ o atributo opcional `namespace` indica o *namespace* do elemento a ser criado

```
<xsl:template match="cliente">  
  <xsl:element namespace="html" name="h3">  
    <xsl:apply=templates/>  
  </xsl:element>  
</xsl:template>
```



Copiando elementos

- ▶ O elemento **copy** é utilizado para copiar elementos do documento de entrada para o documento de saída
- ▶ Exemplo: agrega um prefixo ao conteúdo dos elementos cliente e item

```
<xsl:template match="cliente|item">  
  <xsl:copy>  
    ****: <xsl:apply-templates/>  
  </xsl:copy>  
</xsl:template>
```

- ▶ Os **atributos** do elemento de entrada NÃO são copiados!
-



Criando atributos

- ▶ O elemento **attribute** é utilizado para criar atributos em um elemento de saída
 - ▶ o atributo name indica o nome do atributo criado
 - ▶ o atributo namespace indica o *namespace* do atributo criado
 - ▶ o conteúdo do elemento representa o valor do atributo

```
<xsl:template match="cliente">  
  <xsl:copy>  
    <xsl:attribute name="nome">João</xsl:attribute>  
    <xsl:apply-templates/>  
  </xsl:copy>  
</xsl:template>
```



Obtendo maior controle sob a execução

- ▶ Além do elemento `apply-templates`, é possível utilizar o elemento `for-each`
- ▶ O elemento `for-each` faz com que o processamento se repita para cada elemento selecionado pelo atributo `select`

```
<xsl:template match="text()"/>
```

```
<xsl:template match="booklist">  
  <xsl:for-each select="book">  
    LIVRO-----  
    AUTOR: <xsl:value-of select="author"/>  
  </xsl:for-each>  
</xsl:template>
```



Comandos condicionais

- ▶ choose
- ▶ if



Comandos condicionais: choose

```
<xsl:template match="book">  
  LIVRO -----  
  <xsl:choose>  
    <xsl:when test="author='Bradley, N.'">  
      AUTOR 1  
    </xsl:when>  
    <xsl:otherwise>  
      OUTRO AUTOR....  
    </xsl:otherwise>  
  </xsl:choose>  
</xsl:template>
```



Comandos condicionais: if

```
<xsl:template match="book">  
  LIVRO -----  
  <xsl:if test="author='Bradley, N.'">AUTOR 1</xsl:if>  
</xsl:template>
```



Fazendo cálculos

```
<xsl:template match="numbers">
```

A. $4 + 3.2$ = `<xsl:value-of select="x + y"/>`

B. $3.2 - 4$ = `<xsl:value-of select="y - x"/>`

C. $4 * 3.2$ = `<xsl:value-of select="x * y"/>`

D. $11/3.2$ = `<xsl:value-of select="z div y"/>`

E. $4 + 3.2 * 11$ = `<xsl:value-of select="x+y*z"/>`

F. $(4 + 3.2) * 11$ = `<xsl:value-of select="(x+y)*z"/>`

G. $11 \bmod 4$ = `<xsl:value-of select="z mod x"/>`

H. $4 + 3.2 + 11$ = `<xsl:value-of select="sum(*)"/>`

I. $\text{floor}(3.2)$ = `<xsl:value-of select="floor(y)"/>`

J. $\text{ceiling}(3.2)$ = `<xsl:value-of select="ceiling(y)"/>`

K. $\text{round}(3.2)$ = `<xsl:value-of select="round(y)"/>`

L. $11 + \text{count}(*)$ = `<xsl:value-of select="11+count(*)"/>`

M. $3.2 + \text{string-length}("3.2")$ = `<xsl:value-of select="y + string-length(y)"/>`

N. $11 + \text{"hello"}$ = `<xsl:value-of select="z + 'hello'"/>`

```
</xsl:template>
```

```
<numbers>  
  <x>4</x>  
  <y>3.2</y>  
  <z>11</z>  
</numbers>
```

Exercício 5

- ▶ Exiba o pedido4.xml no formato HTML
- ▶ Utilize uma tabela para exibir os itens do pedido
- ▶ Adicione uma coluna para o subtotal de cada item



Exercício 6 – Desafio

- ▶ Desafio: calcule o total do pedido (para isso, é necessário pesquisar na especificação de XSLT, pois será necessário usar comandos que não aprendemos em aula)
- ▶ Para quem tiver curiosidade, eu disponibilizo a resposta na próxima aula



Mais de um arquivo fonte

- ▶ Ver dicas em <http://www.stylusstudio.com/xsllist/200110/post40030.htm>
|



Mais de um arquivo fonte...

- ▶ Para usar mais de um arquivo fonte para a transformação, deve-se usar a função `document()`

```
<xsl:variable name="doc1" select="document('doc1.xml')"/>
```

```
<xsl:variable name="doc2" select="document('doc2.xml')"/>
```



Exemplo

- ▶ Processar dois arquivos de livro
- ▶ Assuma que o arquivo que temos usado até agora é o books1.xml
- ▶ Queremos incluir no resultado também os livros do arquivo books2.xml (exibido na próxima transparência)



books2.xml

```
<booklist>  
  <book id="HEU00">  
    <author>Heuser, Carlos</author>  
    <title>Projeto de Banco de Dados</title>  
    <publisher>Sagra Luzzato</publisher>  
    <year>2000</year>  
  </book>  
</booklist>
```



XSLT a ser processado a partir de books1.xml

```
<!-- definição dos arquivos adicionais a serem processados -->
```

```
<!-- eles podem ser adicionados ao arquivo via programação, para não amarrar muito o XSLT -->
```

```
<xsl:variable name="doc2" select="document('books2.xml')"/>
```

```
<xsl:template match="text()"/>
```

```
<xsl:template match="booklist">
```

```
  <xsl:for-each select="book">
```

```
    TITULO: <xsl:value-of select="title"/>
```

```
    AUTOR: <xsl:value-of select="author"/>
```

```
  </xsl:for-each>
```

```
<xsl:for-each select="$doc2/booklist/book">
```

```
  TITULO: <xsl:value-of select="title"/>
```

```
  AUTOR: <xsl:value-of select="author"/>
```

```
</xsl:for-each>
```

```
</xsl:template>
```



Resultado

TITULO: Essential XML - Beyond Markup

AUTOR: Box, D. and Skonnard, A. and Lam, J.

TITULO: XML and Java: Developing of Web Applications

AUTOR: Maruyama, H. and Tamura, K. and Uramoto, N.

TITULO: The XML Companion

AUTOR: Bradley, N.

TITULO: Projeto de Banco de Dados

AUTOR: Heuser, Carlos

Notem que os dados do último livro foram retirados do books2.xml



RESUMO

Element	Description	IE	NN
xsl:apply-imports	Applies a template rule from an imported style sheet	6.0	
xsl:apply-templates	Applies a template rule to the current element or to the current element's child nodes	5.0	6.0
xsl:attribute	Adds an attribute	5.0	6.0
xsl:attribute-set	Defines a named set of attributes	6.0	6.0
xsl:call-template	Calls a named template	6.0	6.0
xsl:choose	Used in conjunction with <xsl:when> and <xsl:otherwise> to express multiple conditional tests	5.0	6.0
xsl:comment	Creates a comment node in the result tree	5.0	6.0
xsl:copy	Creates a copy of the current node (without child nodes and attributes)	5.0	6.0
xsl:copy-of	Creates a copy of the current node (with child nodes and attributes)	6.0	6.0
xsl:decimal-format	Defines the characters and symbols to be used when converting numbers into strings, with the format-number() function	6.0	
xsl:element	Creates an element node in the output document	5.0	6.0
xsl:fallback	Specifies an alternate code to run if the XSL processor does not support an XSL element	6.0	



RESUMO

Element	Description	IE	NN
xsl:for-each	Loops through each node in a specified node set	5.0	6.0
xsl:if	Contains a template that will be applied only if a specified condition is true	5.0	6.0
xsl:import	Imports the contents of one style sheet into another. Note: An imported style sheet has lower precedence than the importing style sheet	6.0	6.0
xsl:include	Includes the contents of one style sheet into another. Note: An included style sheet has the same precedence as the including style sheet	6.0	6.0
xsl:key	Declares a named key that can be used in the style sheet with the key() function	6.0	6.0
xsl:message	Writes a message to the output (used to report errors)	6.0	6.0
xsl:namespace-alias	Replaces a namespace in the style sheet to a different namespace in the output	6.0	
xsl:number	Determines the integer position of the current node and formats a number	6.0	6.0
xsl:otherwise	Specifies a default action for the <xsl:choose> element	5.0	6.0
xsl:output	Defines the format of the output document	6.0	6.0
xsl:param	Declares a local or global parameter	6.0	6.0



RESUMO

Element	Description	IE	NN
xsl:processing-instruction	Writes a processing instruction to the output	5.0	6.0
xsl:sort	Sorts the output	6.0	6.0
xsl:strip-space	Defines the elements for which white space should be removed	6.0	6.0
xsl:stylesheet	Defines the root element of a style sheet	5.0	6.0
xsl:template	Rules to apply when a specified node is matched	5.0	6.0
xsl:text	Writes literal text to the output	5.0	6.0
xsl:transform	Defines the root element of a style sheet	6.0	6.0
xsl:value-of	Extracts the value of a selected node	5.0	6.0
xsl:variable	Declares a local or global variable	6.0	6.0
xsl:when	Specifies an action for the <xsl:choose> element	5.0	6.0
xsl:with-param	Defines the value of a parameter to be passed into a template	6.0	6.0

