9

XML
Fundamentos de XSLT

Helder da Rocha
(helder@argonavis.com.br)

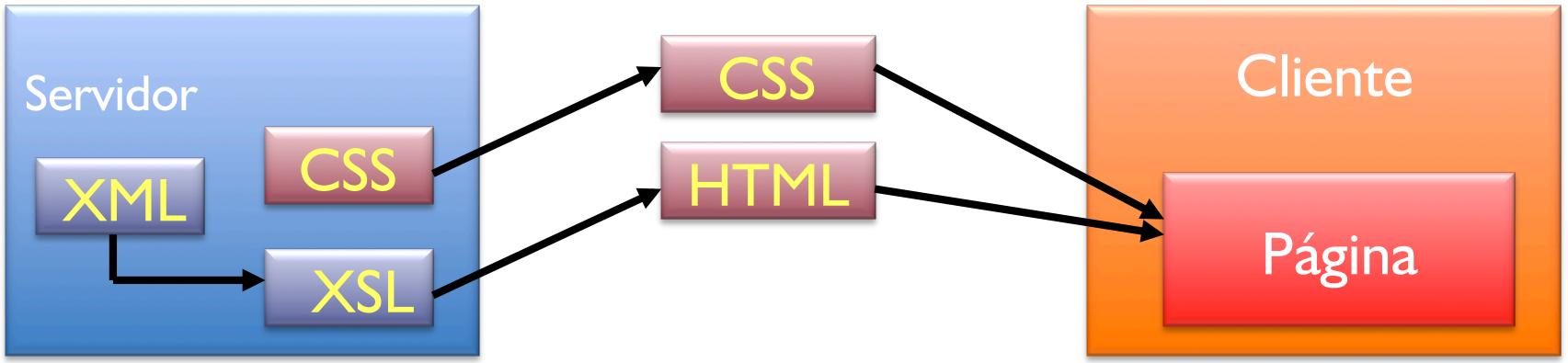
Fundamentos de transformação

- A transformação XSLT é realizada sobre a árvore de um documento-fonte
  - A localização dos nós é feita com XPath
- Todos os nós do documento são acessíveis
- Pode-se transformar qualquer documento-fonte XML em outro tipo de documento
  - HTML ou texto
  - XML (qualquer formato)
- XSLT é uma linguagem de programação completa
  - Ou seja, é complexa e não se aprende em um dia
  - Este curso apresentará fundamentos* do XSLT e exemplos
- A transformação XSLT pode acontecer dinamicamente ao carregar um documento XML com um XSL vinculado
  - Pode também ser usada para gerar um arquivo independente

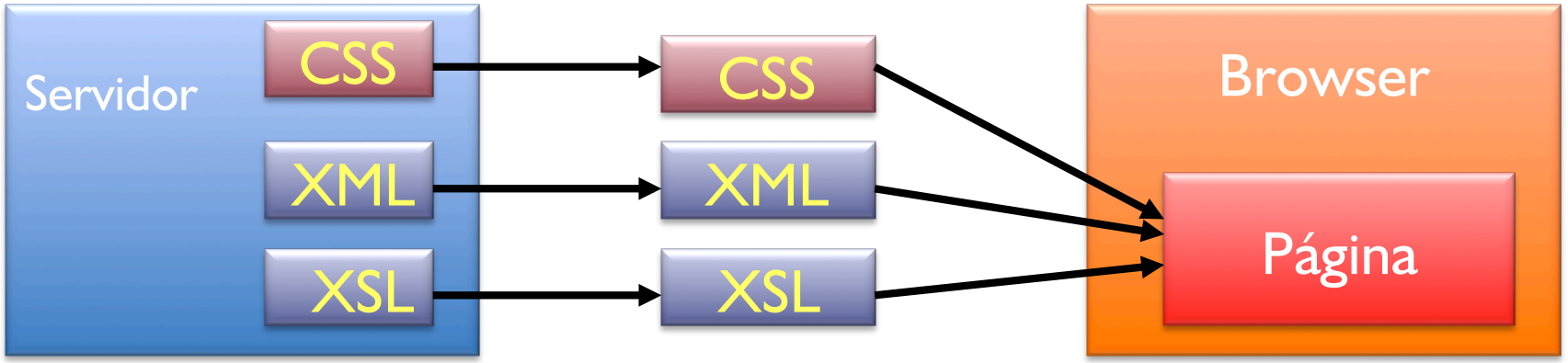
* Para uma abordagem mais profunda de XSLT, veja o curso e tutorial X300, que explora os temas abordados aqui na prática e em mais profundidade com carga-horária de 24 horas)

Alternativas para geração de HTML ou SVG

- Geração prévia no servidor



- Geração durante carga no browser (limitado)



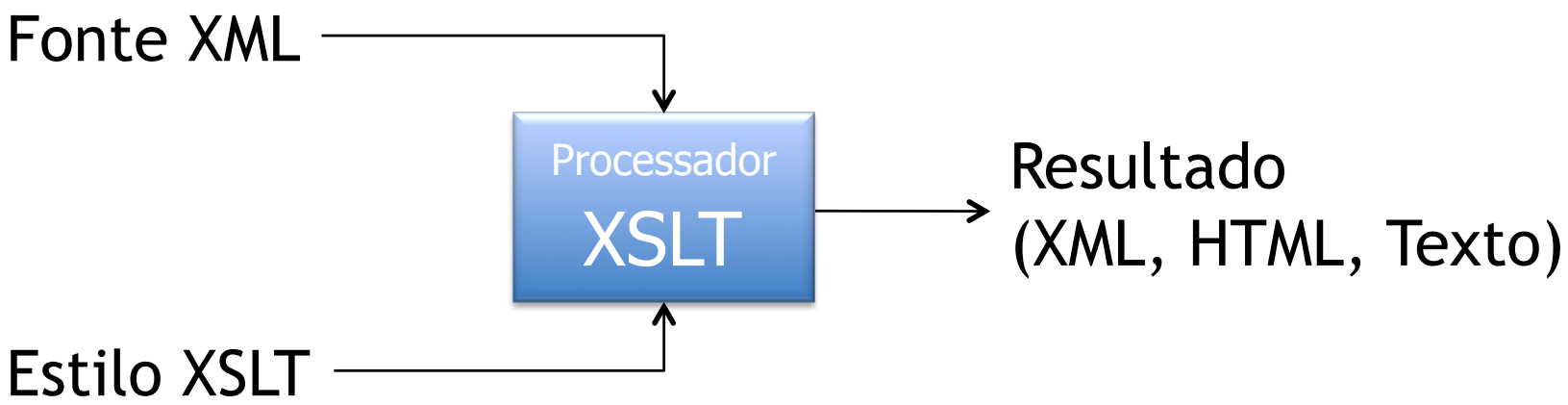
Uso de um XSLT no browser

- *Para que o documento seja transformado ao ser carregado, vincule-o a uma folha de estilos XSL com <?xml-stylesheet>*

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

<?xml-stylesheet version="1.0"
                 type="text/xsl" href="sol2HTML_2.xsl" ?>
<sistemaEstelar>
    <centro>
        <imagem href="sun.gif" />
        <estrela nome="Sol" diametrokm="1390000" />
    </centro>
    <orbita raioMedUA="0.387">
        <planeta id="p1" nome="Mercúrio" diametrokm="4879">
            <imagem href="mercury.jpg" />
        </planeta>
    </orbita>
...
```

Transformação standalone

- *Para fazer uma transformação **permanente**, gerando um arquivo de resultados, pode-se usar um processador XSLT como o **Saxon**, **Xalan**, **libxslt***
  - *xalan.apache.org*
  - *saxon.sourceforge.net*
  - *O Saxon ou Xalan pode ser embutido em projetos que usam linguagens de programação para processar XML*



Transformação XSLT em Java* (TrAX)

- *Inicialize o ambiente (DocumentBuilder, pacotes, etc.)*
- *Carregue o arquivo-fonte em uma árvore DOM*

```
Document document    = builder.parse("file:///fonte.xml");
```

- *Inicialize a árvore DOM do arquivo resultado*

```
Document resDocument = builder.newDocument();
```

- *Crie os objetos (e crie um InputStream com a folha XSLT)*

```
Source xmlSource = new DOMSource(document);
Result result    = new DOMResult(resDocument);
Source xslStyle  = new StreamSource(estilo);
```

- *Inicialize o transformador XSL*

```
TransformerFactory tf = TransformerFactory.newInstance();
Transformer t = tf.newTransformer(xslStyle);
```

- *Faça a transformação*

```
t.transform(xmlSource, result);
```

- *A árvore DOM resultante está em* ***resDocument***

* *Veja código completo em* exemplos/Java/TransformaXSLT.java

Transformação XSLT em C#

- *Importe System.Xml.XPath e System.Xml.Xsl*

```
using System.Xml;
using System.Xml.Xsl;
using System.Xml.XPath;
```

- *Inicialize o processador com a folha de estilos*

```
XslTransform transformer = new XslTransform();
transformer.Load("estilo.xsl");
```

- *Inicialize fluxo de saída*

```
XmlTextWriter writer =
          new XmlTextWriter("resultado.svg");
```

- *Obtenha fontes para transformar*

```
XPathDocument document = new XPathDocument("fonte.xml");
```

- *Realize a transformação*

```
transformer.Transform(document, null, writer);
```

Transformação XSLT em PHP5

- *Inicialize o processador*

```
$xp = new XsltProcessor();
```

- *Crie um objeto DOM para o documento da folha de estilos e importe para o processador*

```
$estilo= new DomDocument;
$estilo->load('estilo.xsl');
$xp->importStylesheet($estilo);
```

- *Crie um objeto DOM para o XML fonte*

```
$document = new DomDocument;
$document->load('arquivo.xml');
```

- *Passe parâmetros se precisar*
  - `$xp->setParameter('param1', 'valor');`
  - `$xp->setParameter('param2', 'valor');`
- *Faça a transformação*
  - `$resultado = $xp->transformToXML($document))`
- *O documento resultante está em* ***$resultado***

Transformação XSLT em Objective-C

- *Importe as bibliotecas libxml2.2.dylib e libxslt.dylib**
- *Inicialize o processador*

```
xmlDocPtr document, estilo, resDoc;
estilo   = xsltParseStylesheetFile((const xmlChar *)
  [@"estilo.xsl" cStringUsingEncoding: NSUTF8StringEncoding]);
document = xmlParseFile((const xmlChar *)
  [@"fonte.xml"  cStringUsingEncoding: NSUTF8StringEncoding]);
```

- *Realize a transformação*

```
resDoc = xsltApplyStylesheet(estilo, document, NULL);
```

- *Resultado em string*

```
char* buffer = nil; int length = 0;
xsltSaveResultToString(&buffer, &length, resDoc, estilo);
NSString* resultado = [NSString stringWithCString:buffer
                                         encoding:NSUTF8StringEncoding];
```

- *Libere os objetos*

```
free(buffer); xsltFreeStylesheet(estilo);
xmlFreeDoc(resDoc); xmlFreeDoc(document);
xsltCleanupGlobals();xmlCleanupParser();
```

** Apple Store rejeita dylibs importados: é preciso baixar os fontes e recompilar*

Como rodar o processador

- Para este curso podemos processar XSLT de duas formas
  - Através da ferramenta usada no curso (Eclipse, Oxygen, XML Spy, JEdit)
  - Através de linha de comando (usando Saxon ou outro processador)
- Saxon 9: baixe em saxon.sourceforge.net (use saxon9he.jar)
  - Voce deve ter um ambiente Java habilitado para execução de aplicações em linha de comando
- Exemplo de uso

```
java  -cp saxon9he.jar net.sf.saxon.Transform -t     \
      -s:fonte.xml -xsl:estilo.xsl -o:resultado.html
```

- Para passar parâmetros

```
java  -cp saxon9he.jar net.sf.saxon.Transform -t     \
      -s:fonte.xml -xsl:estilo.xsl -o:resultado.html   \
      param1='valor1'    \
      param2='valor2'
```

Hello World

- *Suponha que seu documento-fonte contenha o seguinte:*

```
<aeronave id="PTGWZ">
   <origem partida="08:15">Rio de Janeiro</origem>
   <destino>Sao Paulo</destino>
</aeronave>
```

- *Nós para seleção em XPath*
  - `/aeronave/@id` `PTGWZ`
  - `/aeronave/origem` `Rio de Janeiro`
  - `/aeronave/origem/@partida` `08:15`
  - `/aeronave/destino` `Sao Paulo`

Uma regra de template

- *O seguinte template poderia extrair seus dados*

```
<xsl:template match="aeronave">
    <p>A aeronave de prefixo
    <xsl:value-of select="@id" /> decolou
    de <xsl:value-of select="origem" /> às
    <xsl:value-of select="origem/@partida" />
    tendo como destino o aeroporto de
    <xsl:value-of select="destino" />.</p>
</xsl:template>
```

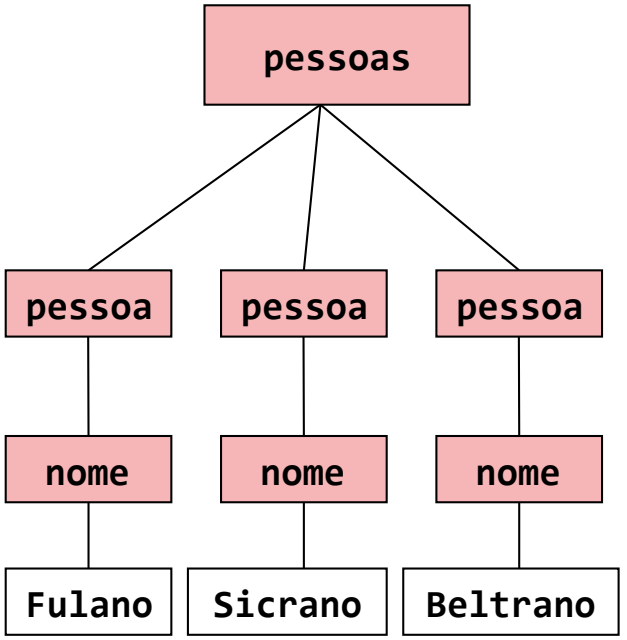
O resultado

- *O resultado será*

```
<p>A aeronave de prefixo
    PTGWZ decolou
    de Rio de Janeiro às
    8:15
    tendo como destino o aeroporto de
    Sao Paulo.</p>
```

Caminhos vs. Padrões

- *Templates usam **padrões** XPath (**match**) no lugar de **caminhos** (**select**)*
  - *Caminho:*
    `[contexto atual]/nome`
  - *Padrão:*
    `[quaisquer ancestrais]/nome`



- *Na árvore ao lado, se o nó de contexto for* `<pessoas>`
  - *o **caminho** devolverá um **conjunto vazio***
  - *o **padrão** devolverá um **conjunto de três nós***

Exemplos de padrões/caminhos

- *Atributos XSLT* ***match****, recebem* ***padrões*** *para instanciar templates:*
  - `<xsl:template match="autor"> ...`
  - `<xsl:template match="livro//pagina[25]"> ...`
- *Atributos XSLT* ***select*** *e* ***test****, usados dentro dos templates, recebem* ***caminhos****:*
  - `<xsl:value-of select="../livro/titulo" />`
  - `<xsl:for-each select="@numero" > ...`
  - `<xsl:if test="not(titulo)"> ...`

Cabeçalho e <xsl:stylesheet>

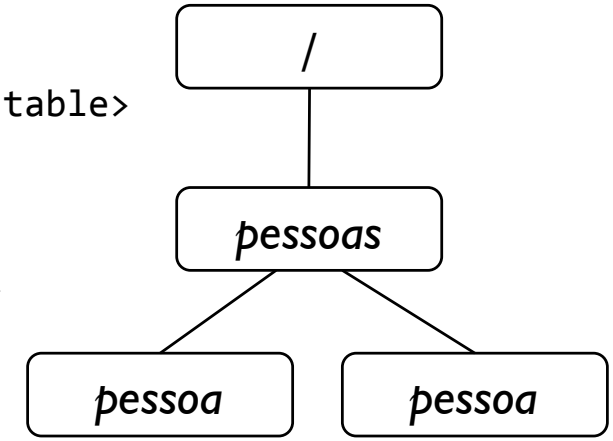
- *Todos os elementos de uma folha de estilos são definidos dentro de* ***<xsl:stylesheet>***
- *O esqueleto básico para uma folha de estilos é*

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet
     xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
     version="1.0">



</xsl:stylesheet>
```

<xsl:template> e uma folha de estilos básica

- *Uma folha de estilos básica possui um ou mais templates*
  - *Boa prática é ter um template para cada nó que envolver transformação: divide a complexidade*

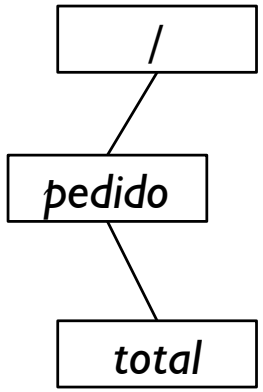
```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
version="1.0">

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

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

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

</xsl:stylesheet>
```



Outro exemplo

- O processador irá navegar pela árvore
  - Deve haver um template para cada nó a ser transformado
  - Não é necessário que haja um template para cada nó (se não houver, ele é executado por default sem transformação)

```
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns="http://www.w3.org/TR/xhtml1/strict">
    <xsl:template match="/">
        <html>
            <body>
                <xsl:apply-templates />
            </body>
        </html>
    </xsl:template>

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

    <xsl:template match="total">
        <p>Total do Pedido: <xsl:value-of select="."/></p>
    </xsl:template>
</xsl:stylesheet>
```



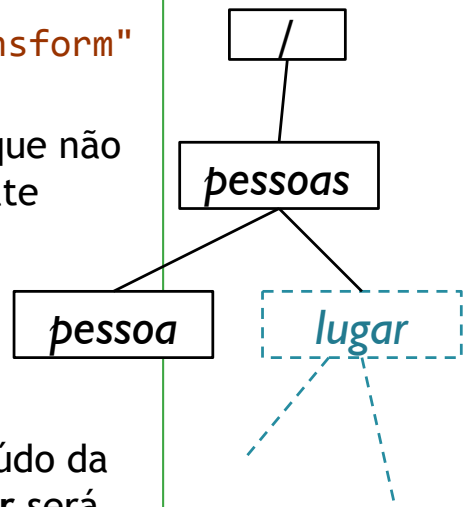
Uso do template com <xsl:apply-templates>

- *<xsl:apply-templates /> processa todos os filhos (inclusive nós de texto)*
  - *Se algum elemento filho combinar com o **match** de um template existente, esse template será processado*
  - *Se algum template não tiver um **<xsl:apply-templates>** o processamento da árvore irá terminar*

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                version="1.0">

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

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

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

O nó pessoas, que não tem um template também será processado!

Todo conteúdo da árvore **lugar** será omitido!



<xsl:apply-templates> com select

- *O atributo **select** pode ser usado para o **<xsl:apply-templates>** pular a outro nó da árvore fonte*
  - *As expressões XPath dentro dos elementos **select** são consideradas **no contexto** dos nós selecionados pelo **match** do template onde ocorrem*

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
version="1.0">

    <xsl:template match="/">
        <body><xsl:apply-templates select="pessoas/pessoa" /></body>
    </xsl:template>

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

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

</xsl:stylesheet>
```

Contexto é "/"

Geração de texto com <xsl:value-of>

- ***<xsl:value-of>*** *pode ser usado para gerar texto a partir de dados do documento-fonte*
  - *Converte todo o conteúdo para texto*
  - *Expressão XPath é relativa ao nó corrente.*

```
<xsl:template match="contxt">
  <xsl:value-of select="elemento" />
</xsl:template>
```

  - *No exemplo acima, select seleciona qualquer nó que contenha o string* ***contxt/elemento***
  - *Use "." para valor de elemento corrente*
  - *Use "/" para usar caminhos absolutos*

Criação de texto com <xsl:text>

- ***<xsl:text>** pode ser usado para gerar texto estático, formatar a saída, etc.*
  - *Preserva espaços, novas-linhas e tabuações*
  - *Útil para controlar forma de impressão do texto*

```
        <xsl:template match="x">
                <xsl:text>Quebra linha depois
</xsl:text>
        </xsl:template>

        <xsl:template match="y">
                <xsl:text>Não quebra linha</xsl:text>
        </xsl:template>
```

Regras de template nativas

- *Várias regras de template estão embutidas*
  - *Pode-se re-declará-las localmente para mudar o comportamento default*

1. Processamento da raiz e todos os elementos

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

2. Processamento de atributos e nós de texto

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

3. Processamento de texto

```
<xsl:template
      match="processing-instruction()|comment()"/>
```

<xsl:attribute>

- *O elemento <xsl:attribute> permite criar atributos na árvore-resultado*

```
<xsl:template match="link">
    <a>
    <xsl:attribute name="href">
        <xsl:value-of select="@ref" />
    </xsl:attribute>
    <xsl:value-of select="." />
    </a>
</xsl:template>
```

- *Resultado*

```
<a href="... conteúdo de @ref ...">
          ... conteúdo de link ... </a>
```

Attribute value templates

- *Em vez de usar* ***<xsl:attribute>*** *para criar os atributos dinâmicamente, é possível usar uma sintaxe especial*
  - `{expressão}`
  - *Sintaxe chamada de "atrribute value templates"*
- *As chaves consideram o valor da expressão resolvida*
  - *É usada dentro de atributos*

```
<xsl:template match="link">
    <a href="{@ref}">
    <xsl:value-of select="." /></a>
</xsl:template>
```

<xsl:element>

- *Assim como é possível criar atributos, é possível criar elementos, usando* ***<xsl:element>***:

```
<xsl:template match="coisa">
    <xsl:element name="elemento">
        <xsl:value-of select="." />
    </xsl:element>
</xsl:template>
```

- *Resultado:*

```
<elemento> ... </elemento>
```

<xsl:sort>

- *Ordenação*
  - *Aplica-se a cada nó do conjunto de nós do contexto. Use dentro de* ***<xsl:apply-templates>***

```
<xsl:apply-templates>
   <xsl:sort select="@codigo" />
   <xsl:value-of select="." />
</xsl:apply-templates>
```

  - *Para ordenar números use o atributo* `data-type="number"` *(default é ordem alfabetica)*

Criação de comentários

- `<xsl:comment> texto </xsl:comment>`
  - *Insere um comentário*
- `<xsl:processing-instruction name="alvo">`
  `conteúdo </xsl:processing-instruction>`
  - *Insere uma instrução de processamento*

Cópia rasa <xsl:copy>

- **<xsl:copy>**
  - *Usado para copiar o nó corrente (no contexto do template) para o documento-resultado*
  - *Não copia atributos, filhos*
  - *Copia apenas elemento e namespace*
- *Para copiar árvore inteira, precisa ser chamado recursivamente via <xsl:apply-templates>*

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

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

</xsl:stylesheet>
```

Oferece controle sobre o que deve ser incluído na cópia (no match do template)

Cópia completa <xsl:copy-of>

- **<xsl:copy-of select="expressão">**
  - *Copia coisas (árvores, texto) para a árvore resultado*
  - *Se o select identifica um nodeset, todos os nós do nodeset são copiados para a árvore resultado*
  - *O nó é copiado por completo (inclusive com atributos, instruções de processamento, etc.)*
- *Exemplo: XSLT que copia a entrada na saída:*

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

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

</xsl:stylesheet>
```

Não controla o que deve ser incluído na cópia (copia o nó inteiro)

Blocos condicionais: <xsl:if> e <xsl:choose>

- *Há dois tipos de blocos condicionais em XSLT*

```
<xsl:if test="expressao">…</xsl:if>

<xsl:choose>
   <xsl:when test="expressao 1"> …</xsl:when>
   <xsl:when test="expressao 2"> …</xsl:when>
      …
   <xsl:when test="expressao n"> …</xsl:when>
   <xsl:otherwise> … </xsl:otherwise>
</xsl:choose>
```

  - ***<xsl:if>*** *processa o conteúdo se a expressão resultar* ***true()***
  - ***<xsl:choose>*** *é um bloco do tipo "case" ou "if-elseif-else"*

Exemplos de <xsl:if>

```
<xsl:template match="elemento">
   <xsl:if test="/raiz/elemento/filho[position()
                          = last() or @attr[.='alta']]">
      <xsl:value-of select "@atrib_do_elemento" />
   </xsl:if>
</xsl:template>

<xsl:if test="(5 &gt; 3) and not(9 &lt; 4))">
    ....
</xsl:if>

<xsl:if test="starts-with(@nome, 'Sra. ')">
   ...
</xsl:if>

<xsl:if test="not(//elemento)">
   ...
</xsl:if>
```

Exemplo com <xsl:choose>

```
<xsl:choose>
     <xsl:when test="../carro[@novo='true']">
       ...
     </xsl:when>
     <xsl:when test="../casa">
       ...
     </xsl:when>
     <xsl:when test="../salario/text() &gt; 10000">
       ...
     </xsl:when>
     <xsl:otherwise>
       ...
     </xsl:otherwise>
</xsl:choose>
```

Valores booleanos em XPath

- *Em XPath:*
  - **true** — **elemento** *chamado* **true** *(ou* **child::true***)*
  - **'false'** — *string contendo o* **texto** *'false'*
  - **true()** — *valor* **booleano** *true*
- *É fácil cometer erros primários*

```
<xsl:if test="true">…</xsl:if>
```

  - *O bloco somente é processado* **se existir um elemento <true>** *no contexto do teste (nodeset vazio = false())*

```
<xsl:if test="'false'">…</xsl:if>
```

  - *O bloco sempre é processado porque o* **string 'false' tem mais de zero caracteres** *(string vazio = false())*

```
<xsl:if test="true() and 'false' or true">…</xsl:if>
```

  - *O bloco sempre é processado porque o* **valor booleano true()** *é verdadeiro e o* **string 'false'** *não é vazio*

Looping com <xsl:for-each>

- *Permite processar um conjunto de nós dentro da mesma regra de template (sem recursão)*
  - `<xsl:for-each select="expressão">`
    `... </for-each>`
- *O atributo select recebe uma expressão XPath que retorna um node-set.*
  - *O node-set é a lista de nós correntes*
  - *O nó sendo processado a cada repetição é o nó corrente*
  - *O conteúdo de <xsl:for-each> está no contexto do nó corrente*
- *Exemplo*

```
<xsl:template match="livro">
   <xsl:for-each select="capitulo">
      <xsl:value-of select="position()" />
      <xsl:text>. </xsl:text>
      <xsl:value-of select="titulo" />
   </xsl:for-each>
</xsl:template>
```

Nó de contexto

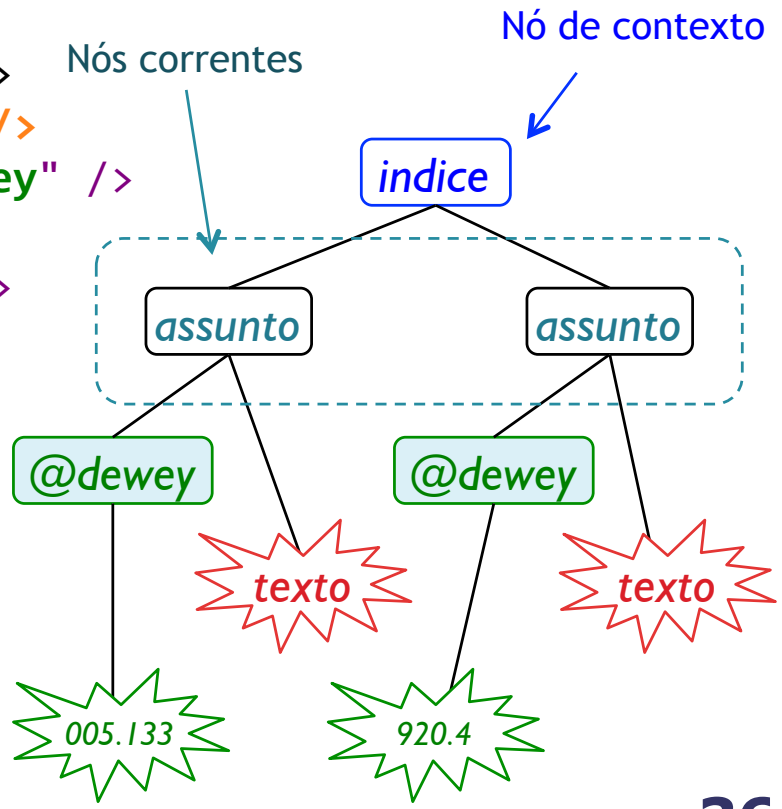
Node-set com nós correntes

Posição do nó corrente dentro da lista de nós correntes

Nó dentro do contexto do nó corrente

<xsl:for-each> e <xsl:sort>

- ***<xsl:sort>*** *pode ser usado dentro de* ***<xsl:for-each>*** *para ordenar os elementos de acordo com um campo*
  - *O atributo* ***select*** *é uma expressão que retorna o valor associado ao nó corrente que será usado como critério de ordenação*
  -
```
<xsl:template match="indice">
    <xsl:for-each select="assunto">
        <xsl:sort select="@dewey" />
        <xsl:value-of select="@dewey" />
        <xsl:text>: </xsl:text>
        <xsl:value-of select="." />
    </xsl:for-each>
</xsl:template>
```
  - *Pode haver mais de um* ***<xsl:sort>****, com outro* ***select****, para ordenar por outros campos associados ao nó corrente*



Variáveis <xsl:variable>

- Podem ser
  - **Locais**: definidas dentro do escopo de uma uma regra de template, bloco <xsl:for-each>, etc.
  - **Globais**: definidas no primeiro nível, como filhas de <xsl:stylesheet>
  - Uma vez que tenham valor, tornam-se constantes
  - Podem conter qualquer tipo (node-set, string, boolean ou number)
- Para definir

```
<xsl:variable name="usr">Morpheus</xsl:variable>

<xsl:variable name="nome" select="nome[4]" />
```

- Para usar

```
<xsl:value-of select="$nome" />

<a href="{$usr}" ></a>
```

Exemplos com <xsl:variable>

```
<xsl:stylesheet ...>
    <xsl:variable name="titulo">Relatório de serviços prestados</xsl:variable>
    <xsl:variable name="logotipo">http://www.a.com/imagem/logo.png</xsl:variable>
    <xsl:variable name="css">relatorio2_css.xml</xsl:variable>

    <xsl:template match="/">
        <html>
            <head><title>Argo Navis: <xsl:value-of select="$titulo"/></title>
                <xsl:copy-of select="document($css)"/>
            </head>
            <body>
                <img src="{$logotipo}" />
                <h2><xsl:value-of select="$titulo"/></h2>
          ...
```

```
 <xsl:template match="intervalo">
        <xsl:variable name="total" select="$fim - $inicio" />
        <xsl:variable name="horas" select="floor($total div 60)" />
        <xsl:variable name="minutos" select="$total - ($horas * 60)" />
        <p>Intervalo: <xsl:value-of
                            select="concat(format-number($horas, '00'),
                                           ':',
                                           format-number($minutos, '00'))" /></p>
    </xsl:template>
```

<xsl:variable> é uma constante

- *Apesar do nome do elemento sugerir o contrário, não é possível mudar o valor de uma variável. Isto não funciona:*

```
<xsl:variable name="mensagem" />
<xsl:if test="@xml:lang['pt']">
    <xsl:variable name="mensagem" select="/mensagens/portugues"/>
</xsl:if>
<xsl:if test="@xml:lang['en']">
    <xsl:variable name="mensagem" select="/mensagens/ingles"/>
</xsl:if>
```

- *A solução é realizar operações dentro de* ***<xsl:variable>*** *que produzam o valor final em um* ***<xsl:value-of>****. Por exemplo:*

```
<xsl:variable name="mensagem">
    <xsl:if test="@xml:lang['pt']">
        <xsl:value-of select="/mensagens/portugues"/>
    </xsl:if>
    <xsl:if test="@xml:lang['en']">
        <xsl:value-of select="/mensagens/ingles"/>
    </xsl:if>
</xsl:variable>
```

Exemplos com <xsl:variable>

```
<intervalo millis="129646644" />
```

```
<xsl:template match="intervalo">
        <xsl:variable name="segundos" select="@millis div 1000"  />

        <!- formata hh:mm:ss a partir de tempo em segundos -->
        <xsl:variable name="formatado">
            <xsl:variable name="tmpHora" select="$segundos div 3600" />
            <xsl:variable name="hora" select="floor($tmpHora)" />
            <xsl:variable name="tmpMin" select="($tmpHora - $hora) * 60" />
            <xsl:variable name="minuto" select="floor($tmpMin)" />
            <xsl:variable name="segundo"
                          select="floor( ($tmpMin - $minuto) * 60)" />
            <xsl:value-of select="concat(format-number($hora, '00'), ':',
                                         format-number($minuto, '00'), ':',
                                         format-number($segundo, '00'))" />
        </xsl:variable>

        <p>Duração: <xsl:value-of select="$formatado" /></p>
</xsl:template>
```

Subrotinas com <xsl:call-template>

- *Templates podem ser usados como subrotinas!*
  - *Ao definir um* ***<xsl:template>****, em vez do atributo* ***match****, use o atributo name:*

```
<xsl:template name="formatar">
... regras para formatar
</xsl:template>
```

  - *Templates sem match não são chamados automaticamente durante o processamento de nós*
  - *Templates com name podem ser chamados de dentro de outros templates, de definições de variáveis, etc.*
  - *O valor retornado pelo template substitui a chamada*
- *Use* ***<xsl:call-template>*** *para realizar a chamada de templates pelo nome*
  - `<xsl:call-template name="formatar" />`

Parâmetros <xsl:param>

- *Parâmetros são quase o mesmo que variáveis*

```
<xsl:param name="nome1" select="expr">
<xsl:param name="nome2">...</xsl:param>
...
<xsl:value-of select="$nome1" />
<a name="#{$nome2}"></a>
```

- *São semelhantes a variáveis, mas diferem um pouco:*
  - ***<xsl:variable>** recebe o valor (resultante da expressão XPath) como constante (não aceita outro valor); o **select** ausente (e elemento vazio) equivale a um string vazio.*
  - *O select de **<xsl:param>** é opcional pois pode ser atribuído através de chamadas **<xsl:call-template>** e **<xsl:with-param>** ou, se for global, através de parâmetros passados externamente.*

<xsl:param> e <xsl:with-param>

- ***<xsl:with-param>*** *permite definir parâmetros em um template que são repassados ao template que chamá-lo*
- *Chamada de um template:*

```
<xsl:call-template name="formatar">
    <xsl:with-param name="abc" select="@ident" />
</xsl:call-template>
```

- *Valores default*
  - *Um* ***<xsl:param>*** *pode definir um valor default que será usado caso a chamada não envie um valor novo*
- *Template destino:*

```
<xsl:template name="formatar">
   <xsl:param name="abc" select="/servicos/defaults/abc"/>
   <xsl:value-of select="$abc" />
</xsl:template>
```

Exemplos com <xsl:call-template>

```
...
<xsl:template name="normalizeTime">
        <xsl:param name="timeString" /> <!-- hh:mm:ss -->

        <xsl:variable name="hora"
                      select="substring-before($timeString, ':')" />
        <xsl:variable name="tmpMinSec"
                      select="substring-after($timeString, ':')" />
        <xsl:variable name="minuto"
                      select="substring-before($tmpMinSec, ':')" />
        <xsl:variable name="segundo"
                      select="substring-after($tmpMinSec, ':')" />

        <xsl:value-of select="$hora * 3600 + $minuto * 60 + $segundo" />
</xsl:template>

<xsl:variable name="segundos">
       <xsl:call-template name="normalizeTime">
            <xsl:with-param name="timeString" select="campos" />
       </xsl:call-template>
</xsl:variable>
...
```

<xsl:call-template> recursivo

```
<xsl:template name="calcularTotal">
   <xsl:param name="camposDuracao" />
   <xsl:param name="segundos" />
   <xsl:param name="valorIni" select="0" />
   <xsl:param name="contador" select="1" />

   <xsl:variable name="totalSegundos">
      <xsl:choose>
         <!-- Se este for o último, devolva a soma do valor inicial + segundos -->
         <xsl:when test="$camposDuracao[$contador = last()]">
            <xsl:value-of select="$valorIni + $segundos" />
         </xsl:when>
         <!-- Caso contrário, passe o total como valor inicial, incremente o
              contador e chame este template novamente com novos parametros -->
          <xsl:otherwise>
              <xsl:call-template name="calcularTotal">
                  <xsl:with-param name="camposDuracao" select="$camposDuracao" />
                  <xsl:with-param name="contador" select="$contador + 1" />
                  <xsl:with-param name="valorIni" select="$valorIni + $segundos" />
              </xsl:call-template>
          </xsl:otherwise>
      </xsl:choose>
   </xsl:variable>
   <!-- A duração total em segundos é retornada aqui -->
   <xsl:value-of select="$totalSegundos"/>
</xsl:template>
```

Debugging e <xsl:message>

- *O elemento* ***<xsl:message>*** *pode ser usado para imprimir na tela durante a execução do processador*
- *É uma boa ferramenta para debugging*
  - *Pode-se usá-la para imprimir valores esperados enquanto se testa a aplicação*
- *Exemplo*

```
<xsl:template name="calcularTotal">
    <xsl:param name="campos" />
    <xsl:param name="contador" select="1" />

    <xsl:message><xsl:value-of select="$campos [$contador]"/></xsl:message>

    <xsl:variable name="totalSegundos">
        <xsl:choose><xsl:when test="...">...</xsl:when>
            <xsl:otherwise>
                <xsl:call-template name="calcularTotal">...</xsl:call-template>
            </xsl:otherwise>
        </xsl:choose>
     </xsl:variable>
</xsl:template>
```

Carregar arquivos externos com document()

- *Função XPath (extensão XSLT):* ***document(uri)***
  - *carrega um documento XML externo e retorna um* ***node-set*** *contendo a raiz do documento*
- *Seleção de um documento externo*

```
<xsl:apply-templates select="document
  ('header.xml')" />
```

- *Constante para representar documento*

nó inicial

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

<xsl:apply-templates select="$tot/menu/item"/>
```

- *Seleção de nó em documento encontrado em nó de arquivo-fonte*

```
<xsl:apply-templates
                  select="document(/a/b)/x[@a='2']" />
```

- *XSLT 1.1 possui* ***<xsl:document>*** *com recursos adicionais*

Fusão de documentos-fonte

- *É possível, com **document()**, gerar um resultado a partir de múltiplos documentos de entrada*

```
<news show="false">
  Noticia 1
</news>
```
1.xml

```
<news show="true">
  Noticia 2
</news>
```
2.xml

```
<news show="false">
  Noticia 3
</news>
```
3.xml

```
<news show="true">
  Noticia 4
</news>
```
4.xml

resultado.xml

```
<arquivos>
  <news show="false">
    Noticia 1
  </news>
...
  <news show="true">
    Noticia 4
  </news>
</arquivos>
```

```
<page>
  <folder>1.xml</folder>
  <folder>2.xml</folder>
  <folder>3.xml</folder>
  <folder>4.xml</folder>
</page>
```
docs.xml (arvore-fonte)

```
<xsl:stylesheet ...>
<xsl:variable name="docs"
              select="document(/page/folder)" />
  <xsl:template match="/">
    <arquivos><xsl:for-each select="$docs">
      <xsl:copy-of select="." />
    </xsl:for-each></arquivos>
  </xsl:template>
</xsl:stylesheet>
```
merge.xsl

generate-id(expressão)

- *É uma função XPath do XSLT (extensão)*
- *Gera um id unívoco para um conjunto de nós.*
  - *Cada vez que **generate-id()** for chamado em um nó, gera sempre **o mesmo id**.*
  - *Pode receber como parâmetro a expressão XPath ou aplicar-se ao nó do contexto se estiver vazio*
- *Exemplos:*

```
<xsl:value-of select="generate-id(/nome)" />
<xsl:template match="nome">
  <xsl:value-of select="generate-id()" />
</xsl:template>
<a name="{generate-id(.)}"></a>
```

Indexação por chave unívoca <xsl:key>

- Define um índice com três atributos
- Sintaxe

```
<xsl:key name="nome" match="padrão" use="expr">
```

  - **name**: nome do índice
  - **match** – expressão XPath que descreve os nós a serem indexados
  - **use** – expressão XPath que define a propriedade usada para criar o índice.
- Exemplos
  -
```
<xsl:key name="planetas" match="planeta"
         use="@nome" />
```
  -
```
<xsl:key name="astros"
         match="planeta|asteroide|satelite"
         use="@diametrokm" />
```

Acesso a chave unívoca: key()

- *Função XSLT que referencia uma relação definida com um elemento* ***<xsl:key>***
- *Sintaxe:*

```
key(nome_da_chave, indice)
```

- *Exemplo. Se houver um*

```
<xsl:key name="planetas" match="planeta"
         use="@nome" />
```

a chamada

```
key('planetas', 'Jupiter')
```

retorna um node-set que contém todos os elementos ***<planeta>*** *do documento que tenha um atributo* ***nome*** *igual a* ***Jupiter***

Desafio: agrupamento

- *Como ordenar o documento-fonte abaixo **por grupos**?*
  - *Agrupar por área, ordenar áreas, ordenar cursos (códigos) por área*

```
<catalogo>
    <curso codigo="J530" area="Java">Enterprise Java Beans</item>
    <curso codigo="X500" area="XML">Scalable Vector Graphics</item>
    <curso codigo="X170" area="XML">XML Schema</item>
    <curso codigo="W600" area="Web">JavaScript</curso>
</catalogo>
```

- *Este é resultado esperado. Como fazer?*

```
<?xml version="1.0" encoding="UTF-8"?>
<dl>
    <dt>Java</dt>
        <dd>J530: Enterprise Java Beans</dd>
    <dt>Web</dt>
        <dd>W600: JavaScript</dd>
    <dt>XML</dt>
        <dd>X170: XML Schema</dd>
        <dd>X500: Scalable Vector Graphics</dd>
</dl>
```

Como agrupar um item por grupo

- *XSLT 1.0 não oferece uma solução simples (existe em XSLT 2.0)*
- *Esta é melhor técnica em XSLT 1.0 ("método Münch"):*
  1. *Crie uma **chave** **<xsl:key>** para indexar o **item** que será agrupado, com base no **grupo** (elemento ou atributo usado para agrupar o **item**)*

```
<xsl:key name="chave" match="item" use="grupo" />
```

  2. *Crie um **<xsl:for-each>** com a seguinte expressão **select**:*

```
item[generate-id(.) = generate-id( key('chave', grupo)[1] )]
```

  3. *IDs gerados do mesmo nó são sempre idênticos. A expressão compara o **nó corrente** **(.)** com o primeiro **item** do **grupo**, logo a lista de nós correntes do **<xsl:for-each>** inclui **apenas um item de cada grupo***
  4. *Em um segundo **<xsl:for-each>** aninhado selecione cada **item** da **chave** (é um node-set) que pertença ao **grupo**:*

```
key('chave', grupo)
```

Solução: ordenação com agrupamento

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                version="1.0">
  <xsl:key name="cat_key" match="curso" use="@area" />

  <xsl:template match="catalogo">
     <dl><xsl:for-each
          select="curso[generate-id(.) =
                        generate-id(key('cat_key', @area)[1])]">
        <xsl:sort select="@area" />
        <xsl:for-each select="key('cat_key', @area)">
            <xsl:sort select="@codigo" />
            <xsl:if test="position() = 1">
                <dt><xsl:value-of select="@area" /></dt>
            </xsl:if>
            <dd><xsl:value-of select="@codigo" />:
                <xsl:value-of select="." /></dd>
        </xsl:for-each>
     </xsl:for-each></dl>
  </xsl:template>
</xsl:stylesheet>
```

Mais XSLT

- *Foge do escopo deste curso introdutório uma abordagem mais profunda do XSLT*
  - *Mesmo assim, alguns elementos abaixo são comuns e devem ser investigados: explore-os e analise os exemplos fornecidos*
- *Método de geraçao de saída (muito usado)*
  - `<xsl:output method="html|xml|text" indent="true" />`
- *Política de espaços e geração de texto (organiza a saída)*
  - `<xsl:preserve-space elements="pre, code">`
  - `<xsl:strip-space elements="*">`
  - *atributo* `disable-output escaping` *de* `<xsl:text>`
- *Recursos de extensão e compatibilidade (essencial se você usa extensões, XSLT 2.0 e ainda quer funcionar em browsers)*
  - `<xsl:fallback>`
  - `<xsl:namespace-alias>`
- *Formatação default para números (essencial para trocar ponto decimal por vírgula)*
  - `<xsl:decimal-format>` e função `format-number()`