INVESTICE DO ROZVOJE VZDĚLÁVÁNÍ

Realizováno za finanční podpory ESF a státního rozpočtu ČR v rámci v projektu *Zkvalitnění a rozšíření možností studia na TUL pro studenty se SVP* reg. č. CZ.1.07/2.2.00/29.0011

Cykly a koncepce transformace

Zpracování všech

- <xsl:for-each select="*XPath výraz*">
  šablona
  </xsl:for-each>
- pro každý uzel vyhovující *XPath výrazu* bude do výstupního stromu vložena *šablona* (**vyhodnocuje se v kontextu uzlu vyhovujícího atributu select**)
- vhodné zejména pro sumarizace informací
- není skutečný cyklus, může provádět paralelně

Příklad: Seznam obrázků

- na začátku stránky chceme mít seznam obrázků

```
<xsl:template match="/">
...
<h2>Seznam obrázků:</h2>
<ul>
<xsl:for-each select="//figure">
    <li><xsl:value-of select="@title"/></li>
</xsl:for-each>
</ul>
...
</xsl:template>
```

Základní kostra transformace (1)

- **varianta 1: Přirozená rekurze**
  - šablony odpovídající zhruba 1:1 prvkům původního dokumentu
  - mnoho šablon, časté uplatnění xsl:apply-templates
  - výstupní strom je sestavován na základě rekurzivního procházení stromu vstupního
  - trochu připomíná podprogramy – neděláme složité kroky, řešení rozložíme do série kroků jednoduchých
  - nejčastější a typický přístup

Příklad vstupních dat

```
<?xml version="1.0"?>
<cenik>
    <zbozi id="zb001" druh=“potravina“>
        <nazev>Houska</nazev>
        <cena>1.70</cena>
    </zbozi>
    <zbozi id=“zb004“ druh=“potravina“>
        <nazev>Voda</nazev>
        <cena>7.50</cena>
    </zbozi>
</cenik>
```

Rekurzivní styl (1)

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    version="1.0">
<xsl:template match="/">
    <html>
    <head><title>Ceník</title></head>
    <body>
    <h1>Ceník</h1>
    <xsl:apply-templates/>
    </body>
    </html>
</xsl:template>
```

Rekurzivní styl (2)

```
<xsl:template match="cenik">
    <table><xsl:apply-templates/></table>
</xsl:template>
<xsl:template match="zbozi">
    <tr><xsl:apply-templates/></tr>
</xsl:template>
<xsl:template match="nazev">
    <td><xsl:apply-templates/></td>
</xsl:template>
<xsl:template match="cena">
    <td align="right"><xsl:apply-templates/></td>
</xsl:template>
</xsl:stylesheet>
```

Základní kostra transformace (2)

- **varianta 2: Řízení shora**
  - složité šablony využívající cykly k procházení obsahu vstupních prvků
  - málo šablon, xsl:apply-templates se používá zřídka či vůbec ne
  - sestavení výstupního stromu je explicitně řízeno stylovým předpisem
  - kompaktnější, ale obtížněji modifikovatelné
  - nepoužitelné pro rekurzivní data (různá hloubka vnoření)
  - příliš se nepoužívá

Řízený styl (1)

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    version="1.0">
<xsl:template match="/">
    <html>
    <head><title>Ceník</title></head>
    <body>
    <h1>Ceník</h1>
    <table>
```

Řízený styl (2)

```
<xsl:for-each select="cenik/zbozi">
    <tr>
    <td><xsl:value-of select="nazev"/></td>
    <td align="right"><xsl:value-of select="cena"/></td>
    </tr>
</xsl:for-each>
</table>
</body>
</html>
</xsl:template>
</xsl:stylesheet>
```

Řazení

Řazení

- pořadí uplatnění šablon standardně odpovídá pořadí prvků ve vstupním stromě, lze změnit
- <xsl:sort select="*hodnota pro uspořádání*" />
- lze jen uvnitř xsl:for-each nebo xsl:apply-templates
- příklad:

```
<xsl:for-each select="cenik/zbozi">
    <xsl:sort select="nazev" />
    <tr>...</tr>
</xsl:for-each>
```

Atributy xsl:sort

- ovlivňují způsob řazení
  - select="*výraz*" určuje hodnoty, podle nichž se třídí (implicitně uzel převedený na řetězec)
  - data-type="text" | "number" abecední/číselné
  - order="ascending" | "descending" vzestupné/sestupné
  - lang="*kód jazyka*" jazyková pravidla třídění (naše „ch")
  - case-order="upper-first" | "lower-first" pořadí malých/velkých písmen

Řazení podle více kritérií

- několik xsl:sort za sebou
- priorita klesá – rozhodne první, kde se uzly liší

```
<xsl:template match="cenik">
    <table>
    <xsl:apply-templates>
        <xsl:sort select="nazev"/>
        <xsl:sort select="cena" data-type="number"/>
    </xsl:apply-templates>
    </table>
</xsl:template>
```

Větvení

Podmíněné zpracování

- <xsl:if test="*XPath výraz*">
  šablona
  </xsl:if>
- šablona bude použita, pokud boolovský XPath výraz vydá hodnotu true
- neexistuje else

Příklad

```
<xsl:template match="zbozi">
    <xsl:element name="h2">
        <xsl:if test="@kod">
            <xsl:attribute name="id"/>
                <xsl:value-of select="@kod"/>
            </xsl:attribute>
        </xsl:if>
        <xsl:value-of select="nazev"/>
    </xsl:element>
    ...
</xsl:template>
```

```
<zbozi kod="zb001" druh="potravina">
    <nazev>Houska</nazev>
    <cena>1.70</cena>
</zbozi>
```

```
<h2 id="zb001">Houska</h2>
...
```

Větvení (1)

- podmíněný příkaz s více větvemi:

```
<xsl:choose>
    <xsl:when test="XPath výraz">
        šablona
    </xsl:when>
    <xsl:when test="XPath výraz">
        šablona
    </xsl:when>
    ...
    <xsl:otherwise>
        šablona
    </xsl:otherwise>
</xsl:choose>
```

Větvení (2)

- analogie
  if
  elsif
  ...
  else
- uvnitř xsl:choose mohou být jen xsl:when a xsl:otherwise
- použije první *šablonu*, jejíž test vydal hodnotu true

Příklad větvení

- řekněme, že <zbozi> obsahuje prvek <skladem> obsahující počet kusů na skladě

```
<xsl:template match="zbozi">
    <tr>...
    <xsl:choose>
        <xsl:when test="skladem &lt; 10">
            <td>nakupujte rychle</td>
        </xsl:when>
        <xsl:when test="skladem &lt; 100">
            <td>mame toho dost</td>
        </xsl:when>
        <xsl:otherwise>
            <td>akce!</td>
        </xsl:otherwise>
    </xsl:choose>
    </tr>
</xsl:template>
```

znak „<“ entitou

Proměnné

Proměnné

- spíše konstanty – hodnotu nelze změnit
- definice – dvě možnosti:
  <xsl:variable name="*jméno*">hodnota</xsl:variable>
  <xsl:variable name="*jméno*" select="*XPath výraz*" />
- použití: $*jméno* (ve výrazech: {...} a xsl:value-of vyhodnotí; xsl:copy-of zkopíruje)
- mohou být globální i lokální v šabloně (má přednost)
- vhodné pro opakující se konstrukce (snadná změna)

Příklad – textová proměnná

```
<xsl:variable name="tuladr">
Technická univerzita v Liberci,<br />
Studentská 2,<br />
461 17 Liberec 1
</xsl:variable>

...
<xsl:template match="zahlavi">
    <div class="zahlavi">
    <xsl:apply-templates/>
    <xsl:copy-of select="$tuladr" />
    </div>
</xsl:template>
```

Příklad – mezery zleva

```
<xsl:variable name="sirkaPole">30</xsl:variable>

<xsl:template match="nazev">
    <xsl:variable name="delka" select="string-length(.)"/>
    <xsl:variable name="vypln"
                  select="$sirkaPole – $delka"/>

        <!-- vloží mezery -->
    <xsl:value-of select="substring('                ',1,$vypln)"/>
        <!-- za ně vlastní název -->
    <xsl:value-of select="."/>
</xsl:template>
```

alespoň 30 mezer

Výstupy

XSLT a mezery (1)

- prázdné znaky: mezera, tabelátor, CR, LF
- XSLT zachází s volným místem celkem liberálně
- **<xsl:strip-space elements="cenik zbozi"/>**
  vypustí z prvků cenik a zbozi prázdné textové uzly (volné místo mezi prvky); jen přímé potomky
- existuje i xsl:preserve-space, implicitní chování
- Pozor: <nazev> Cukr </nazev>
  neobsahuje prázdné textové uzly, ale textový uzel
  „ Cukr “

XSLT a mezery (2)

- často přecházejí do výstupu uplatněním implicitní šablony (opis textu) na prázdné uzly
- explicitní vkládání do výstupu zajistí xsl:text

```
<xsl:text>   </xsl:text><!-- mezery -->
<xsl:text>
</xsl:text>                    <!-- konec řádku -->
```

- normalizace mezer pomocí xsl:value-of a XPath funkce normalize-space()

```
<xsl:value-of select="normalize-space(nazev)"/>
```

Odřádkování za výstupem (1)

- ze <zbozi> odstraněny prázdné textové uzly pomocí xsl:strip-space, chceme vypsat název a odřádkovat
- **varianta 1: doslovná**

```
<xsl:template match="zbozi">
  <em><xsl:value-of select="nazev" /></em>
  <xsl:text>
</xsl:text>
</xsl:template>
```

nevýhoda: </xsl:text> musí být na začátku řádku – narušuje odsazení

Odřádkování za výstupem (2)

- **varianta 2: použití proměnné**

```
<xsl:variable name="NL">
  <xsl:text>
</xsl:text>
</xsl:variable>
<xsl:template match="zbozi">
  <em><xsl:value-of select="nazev" /></em>
  <xsl:value-of select="$NL">
</xsl:template>
```

vyžaduje přípravu, použití je elegantní

Odřádkování za výstupem (3)

- **varianta 3: proměnná + concat**

```
<xsl:variable name="NL">
  <xsl:text>
</xsl:text>
</xsl:variable>
<xsl:template match="zbozi">
  <xsl:value-of select="concat('<em>', nazev,
                               '</em>', $NL)">
</xsl:template>
```

velmi kompaktní, ale méně přehledné

Odřádkování za výstupem (4)

- **varianta 4: kódy znaků**

```
<xsl:template match="zbozi">
  <em><xsl:value-of select="nazev" /></em>
  <xsl:text>&#10;</xsl:text>
</xsl:template>
```

lze zapisovat přímo nebo uložit do proměnné

kódy závisí na OS, pro MS Windows

```
  <xsl:text>&#13;&#10;</xsl:text>
```

Formát výstupu

- **<xsl:output method="*metoda*"/>**
  dostupné metody: xml (implicitní), html, text
- další atributy xsl:output:
  - doctype-system="*URI*", doctype-public="*URI*" – určují obsah <!DOCTYPE...> výstupního dokumentu
  - encoding="*kódování*" – kódování výstupu
  - omit-xml-declaration="yes" | "no" – vypustit úvodní XML deklaraci
  - indent="yes" | "no" – odsazovat prvky