Y36XML – Technologie XML

Přednáší:

Irena Mlýnková (mlynkova@ksi.mff.cuni.cz)

Martin Nečaský (necasky@ksi.mff.cuni.cz)

ZS 2009

Stránka přednášky:
http://www.ksi.mff.cuni.cz/~mlynkova/Y36XML/

Osnova předmětu

- Úvod do principů formátu XML, přehled XML technologií, jazyk DTD
- Datové modely XML, rozhraní DOM a SAX
- Úvod do jazyka XPath
- Úvod do jazyka XSLT
- XPath 2.0, XSLT 2.0
- Úvod do jazyka XML Schema
- Pokročilé rysy jazyka XML Schema
- Přehled standardních XML formátů
- Úvod do jazyka XQuery
- Pokročilé rysy jazyka XQuery, XQuery Update
- Úvod do XML databází, nativní XML databáze, číslovací schémata, structural join
- Relační databáze s XML rozšířením, SQL/XML

Principy XSLT

- **X**ML **s**tylesheet **l**anguage for **t**ransformations
- Transformace XML dokumentů

Principy XSLT

- Vstup: 1 a více XML dokumentů
- Výstup: 1 a více dokumentů
  - Ne nutně XML
  - V základní verzi, později si ukážeme jak udělat více výstupních XML dokumentů

Principy XSLT

```
<?xml version="1.0"?>
<objednavka cislo="O322" datum="10/10/2008" stav="expedovana">
 <zakaznik cislo="C992">
  <jmeno>Martin Nečaský</jmeno>
  <email>martinnec@gmail.com</email>
 </zakaznik>
 <polozky>
  <polozka kod="48282811">
   <jmeno>CD</jmeno>
   <mnozstvi>5</mnozstvi><cena>22</cena>
  </polozka>
  <polozka kod="929118813">
   <jmeno>Dell Latitude D630</jmeno>
   <mnozstvi>1</mnozstvi><cena>30000</cena><barva>modra</barva>
  </polozka>
 </polozky>
</objednavka>
```

Principy XSLT

```
<?xml version="1.0"?>
<html>
 <head><title>Objednávka č. 322 – Martin Nečaský</title></head>
 <body>
  <table>
   <tr>
     <td>CD</td><td>22 Kč</td><td>5 ks</td>
   </tr>
   <tr>
    <td>Dell Latitude D630</td>
    <td>30000 Kč</td><td>1 ks</td>
   </tr>
  </table>
  <div>
   Cena celkem : 22x5 + 30000x1 = 30110 Kč
  </div>
 </body>
</html>
```

Principy XSLT

- Pomocí XSLT píšeme *transformační skripty*
  - XSLT je XML jazyk, tzn. skripty jsou opět XML dokumenty
- Skript se skládá z *šablon*
- Šablona se aplikuje na daný uzel vstupního XML dokumentu a produkuje nějaký výstup
  - Může iniciovat spouštění dalších šablon na stejném či jiných uzlech

Kostra XSLTskriptu

- XSLT je XML jazyk
  - Úvodní XML deklarace
  - Kořenový element **stylesheet**

```
<?xml version="1.0" encoding="windows-1250"?>
<stylesheet>

</stylesheet>
```

Kostra XSLTskriptu

- Kořenový element **xsl:stylesheet**
  - Zavedení jmenného prostoru jazyka XSLT
  - Příp. zavedení dalších jmenných prostorů
- Verze jazyka XSLT

```
<?xml version="1.0" encoding="windows-1250"?>
<xsl:stylesheet
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns="http://www.w3.org/1999/xhtml"
    version="1.0">

</xsl:stylesheet>
```

Kostra XSLT skriptu

- Element **xsl:output**
  - Pod kořenovým elementem **xsl:stylesheet**
  - Určuje typ výstupního dokumentu
    - xml, pdf, text
  - **indent = "yes"** určuje, že XSLT procesor může přidávat bílé znaky do výstupu (formátování)

```
<?xml version="1.0" encoding="windows-1250"?>
<xsl:stylesheet ...>
  <xsl:output type="xml" indent="yes" />
</xsl:stylesheet>
```

XSLT šablony

- Element **xsl:template**
  - Pod kořenovým elementem **xsl:stylesheet**
  - Popisuje jednu šablonu
  - Ve skriptu jich může být více
    - Všechny jsou pod **xsl:stylesheet**

```
<?xml version="1.0" encoding="windows-1250"?>
<xsl:stylesheet>
  <xsl:template> ... </xsl:template>
  <xsl:template> ... </xsl:template>
  ...
</xsl:stylesheet>
```

XSLT šablony

- Vstup šablony: XML uzel, který lze vybrat pomocí XPath
  - Element, atribut, text, ...
- Výstup šablony: XML fragment (sekvence XML uzlů) nebo obecně nějaký text (HTML, pdf, txt, …)

XSLT šablony

- Dvě varianty šablon:
  - Nepojmenovaná šablona
    - element **xsl:template** s atributem **match**
    - Hodnotou atributu je sekvence XPath cest oddělených znakem '|'
    - Kroky XPath cest mohou využívat pouze osy child a attribute a navíc lze použít // (ale ne osu descendant-or-self)

```
<xsl:template match="[xpath cesta ['|' xpath cesta]*]">
  ...
</xsl:template>
```

XSLT šablony

- Dvě varianty šablon:
  - Pojmenovaná šablona
    - element **xsl:template** s atributem **name**
    - Hodnotou atributu je název šablony

```
<xsl:template name="[název šablony]">
  ...
</xsl:template>
```

XSLT šablony

```
<?xml version="1.0" encoding="windows-1250"?>
<xsl:stylesheet>

</xsl:stylesheet>
```

Jak to celé funguje?

- XSLT skript je vykonán programem, který se jmenuje XSTL procesor
  - saxon, xsltproc, ...
  - Také zabudovány v prohlížečích
- Spuštěn nad vstupním XML dokumentem
  - Může být i více vstupních dokumentů, ale je spuštěn jen na jednom, na ostatní se odkážeme přímo ve skriptu

Jak to celé funguje?

- XSLT procesor pracuje podle následujícího algoritmu
  - Vytvoř kontextovou množinu uzlů *C* a vlož do ní kořenový uzel vstupního XML dokumentu
  - Dokud je kontextová množina neprázdná:
    - Vyjmi první uzel *u* z *C* (pořadí je určeno pořadím uzlů v XML dokumentu)
    - Najdi nejvhodnější šablonu pro *u* a zpracuj pomocí ní *u*

Jak to celé funguje?

- Nalezení nevhodnější šablony pro zpracování uzlu *u*:
  - Hledá se pouze mezi nepojmenovanými šablonami
    - tj. těmi s atributem match
  - Berou se pouze ty, jejichž XPath výraz *P* v atributu match popisuje uzel *u*
    - tj. *u* je odněkud v XML dokumentu dosažitelný pomocí *P*

Jak to celé funguje?

- Co když se vhodných šablon najde více?:
  - Problém, protože lze aplikovat jen jednu
  - Bere se ta s nejvyšší prioritou
    - Může být nastavena explicitně pomocí atributu **priority**
    - Pokud není nastavena explicitně je vypočítána implicitní priorita
      - 0.5 : cesta s více než jedním krokem
      - 0: název elementu/atributu
      - -0.25: *
      - -0.5: node(), text(), …

Jak to celé funguje?

- Co když se nenajde žádná vhodná šablona?
  - Implicitní (předdefinované) šablony → vždy se nějaká vhodná najde
  - Mají nejnižší prioritu, tj. aplikují se pouze, když se jiná nenajde

Jak to celé funguje?

```
<xsl:template match="/">
 <!-- transformace kořenového uzlu dokumentu -->
</xsl:template>

<xsl:template match="polozka">
 <!-- transformace elementu polozka -->
</xsl:template>

<xsl:template match="jmeno">
 <!-- transformace elementu jmeno -->
</xsl:template>

<xsl:template match="zakaznik/jmeno">
 <!-- transformace elementu jmeno s rodicem zakaznik -->
</xsl:template>
```

Jak to celé funguje?

```
<xsl:template match="*|@*">
 <!-- transformace libovolného elementu nebo atributu -->
</xsl:template>

<xsl:template match="zakaznik/*">
 <!-- transformace libovolného elementu, který je dítětem
nějakého elementu zakaznik -->
</xsl:template>

<xsl:template match="text()">
 <!-- transformace libovolného textového uzlu -->
</xsl:template>

<xsl:template match="objednavka//node()">
 <!-- transformace libovolného potomka (kromě atributů) nějakého
elementu objednavka -->
</xsl:template>
```

Těla šablon – možnosti

1. Vytváření elementů a atributů
   - Přímo nebo nepřímo pomocí konstrukcí **xsl:element** a **xsl:attribute**
2. Vytváření textových uzlů
   - Přímo nebo nepřímo pomocí konstrukce **xsl:text**
3. Přístup k datům ze vstupu
   - Pomocí konstrukce **xsl:value-of**

Těla šablon – možnosti

4. Vyvolávání aplikace šablon
   - **xsl:apply-templates** a **xsl:call-template**
5. Proměnné a parametry
   - **xsl:variable** a **xsl:param**
6. Opakování
   - **xsl:for-each**
7. Větvení
   - **xsl:if** a **xsl:choose**

Vytváření elementů a atributů

```
<xsl:template match="/">
 <html>
  <head>
   <title>
    <!-- vygenerování nadpisu objednávky -->
   </title>
  </head>
  <table border="1">
   <!-- vygenerování řádků pro položky objednávky -->
  </table>
  <!-- vygenerování celkové ceny -->
 </html>
</xsl:template>
```

Vytváření elementů a atributů

- Nebo použijeme konstrukci **xsl:element**
  - Vytvoří element s daným názvem
  - Název je určen příznakem **name**
- a konstrukci **xsl:attribute**
  - Vytvoří atribut s daným názvem a hodnotou
  - Název je určen příznakem **name**
  - Hodnota je určena obsahem
- Umožňují "počítat" název elementu/atributu
  - Např. z hodnot na vstupu

Vytváření elementů a atributů

```
<xsl:template match="/">
 <html>
  <head>
   <xsl:element name="title">
    <!-- vygenerování nadpisu objednávky -->
   </xsl:element>
  </head>
  <table>
   <xsl:attribute name="border">1</xsl:attribute>
   <!-- vygenerování řádků pro položky objednávky -->
  </table>
  <!-- vygenerování celkové ceny -->
 </html>
</xsl:template>
```

Vytváření elementů a atributů

```
<xsl:template match="/">
 <objednavky>
  <xsl:for-each select="//objednavka">
   <objednavka>
    <xsl:if test="./@stav">
     <xsl:element name="{./@stav}">
      ANO
     </xsl:element>
    </xsl:if>
   </objednavka>
  </xsl:for-each>
 </objednavky>
</xsl:template>
```

Vytváření elementů a atributů

- example1.xslt
- example2.xslt

Vytváření textů

- V těle šablony píšeme rovnou výstupní text

```
<xsl:template match="/">
 <html>
  <head>
   <title>
    Objednávka č. <!-- číslo objednávky --> - <!-- jméno
    zákazníka -->
   </title>
  </head>
  ...
 </html>
</xsl:template>
```

Vytváření textů

- Pomocí konstrukce xsl:text

```
<xsl:template match="/">
 <html>
  <head>
   <title>
    <xsl:text>Objednávka č.</xsl:text>
    <!-- číslo objednávky -->
    <xsl:text>-</xsl:text>
    <!-- jméno zákazníka -->
   </title>
  </head>
  ...
 </html>
</xsl:template>
```

Vytváření textů

- example3.xslt

Data ze vstupu

- Pro přístup k datům ze vstupu použijeme konstrukci **xsl:value-of**
  - Příznakem **select** vybíráme hodnotu ze vstupu
    - Zadáme pomocí XPath výrazu
    - Výraz se vyhodnocuje vzhledem k aktuálnímu uzlu, který je šablonou zpracováván
  - Vybraná hodnota jde na výstup

Data ze vstupu

```
<xsl:template match="/">
 <html>
  <head>
   <title>
    <xsl:text>Objednávka č.</xsl:text>
    <xsl:value-of select="./objednavka/@cislo" />
    <xsl:text>-</xsl:text>
    <xsl:value-of select=".//zakaznik/jmeno" />
   </title>
  </head>
  ...
 </html>
</xsl:template>
```

Data ze vstupu

- example4.xslt

Vyvolání aplikace šablon

- XSLT procesor nalezne vhodnou šablonu pro transformaci kořenového uzlu vstupního XML dokumentu
  - Ale co dál? Chceme transformovat i uzly v podstromu.
  - example5.xslt

Vyvolání aplikace šablon

- Konstrukce **xsl:apply-templates**
  - V místě zavolání iniciuje transformaci dětí aktuálně zpracovávaného uzlu
  - Pomocí příznaku **select** lze explicitně určit uzly, které se mají transformovat
    - Tj. ne děti, ale něco jiného
    - Pomocí XPath výrazu
  - Vybrané uzly jsou transformovány stejně jako aktuální uzel
    - Tj. je nalezena nejvhodnější šablona pro jejich transformaci a ta je aplikována

Vyvolání aplikace šablon

```
<xsl:template match="/">
 <html>
  <head>
   ...
  </head>
  <table>
   <xsl:apply-templates />
  </table>
  ...
 </html>
</xsl:template>
```

Vyvolání aplikace šablon

```
<xsl:template match="/">
 <html>
  <head>
   ...
  </head>
  <table>
   <xsl:apply-templates select=".//polozka"/>
  </table>
  ...
 </html>
</xsl:template>
```

Vyvolání aplikace šablon

```
<xsl:template match="polozka">
 <tr>
  <td><xsl:value-of select="jmeno" /></td>
  <td>
   <xsl:value-of select="cena" />
   <xsl:text> Kč</xsl:text>
  </td>
  <td>
   <xsl:value-of select="mnozstvi" />
   <xsl:text> ks</xsl:text>
  </td>
 </tr>
</xsl:template>
```

Vyvolání aplikace šablon

- example5.xslt

Vyvolání aplikace šablon

- Konstrukce **xsl:call-template**
  - Aplikace konkrétní šablony na aktuální uzel
    - Na šablonu se odkážeme pomocí jejího jména (příznak **name**)
  - Čili XSLT procesor nehledá pro uzel nejvhodnější šablonu, ale rovnou aplikuje tu s daným jménem

Vyvolání aplikace šablon

```
<xsl:template match="polozka">
 <tr>
  ...
  <td>
   <xsl:call-template name="cena-s-dph" />
   <xsl:text> Kč</xsl:text>
  </td>
  ...
 </tr>
</xsl:template>

<xsl:template name="cena-s-dph">
 <xsl:value-of select = "./cena * 1.19" />
</xsl:template>
```

Vyvolání aplikace šablon

- example6.xslt

Proměnné a parametry

- *Proměnná* umožňuje uložit nějakou hodnotu a odkazovat se na ní
  - Konstrukce **xsl:variable** s příznakem **name** a nepovinně **select**
- *Parametr* je proměnná, která je "vidět" vně šablony
  - Při volání šabolny musejí být specifikovány také hodnoty jejích parametrů
  - Konstrukce **xsl:param** s příznakem **name** a nepovinně **select**
- Lokální (uvnitř šablon) i globální (pod kořenovým elementem **xsl:stylesheet**)

Proměnné a parametry

```
<xsl:variable name="pocet-polozek">
 <xsl:value-of select="count(//polozka)" />
</xsl:variable>

<xsl:template match="/">
 <tr>
  ...
  <xsl:text>Celkem položek: </xsl:text>
  <xsl:value-of select="$pocet-polozek" />
 </tr>
</xsl:template>
```

Proměnné a parametry

```
<xsl:template match="polozka">
 <tr>
  ...
  <td>
   <xsl:call-template name="cena-s-dph">
    <xsl:with-param name="cena" select="./cena" />
   </xsl:call-template>
   <xsl:text> Kč</xsl:text>
  </td>
  ...
 </tr>
</xsl:template>

<xsl:template name="cena-s-dph">
 <xsl:param name="cena" select="0" />
 <xsl:value-of select = "$cena * 1.19" />
</xsl:template>
```

Proměnné a parametry

- example7.xslt

Proměnné a parametry

- Pozor: proměnným a parametrům nelze měnit hodnoty
  - Jakmile je hodnota nastavena, už ji nelze změnit

Špatné použití proměnných

```
<xsl:variable name="cena-celkem">
 <xsl:value-of select="0" />
</xsl:variable>

<xsl:template match="/">
 ...<xsl:apply-tempates select=".//polozka" />...
 <xsl:text>Cena celkem: </xsl:text>
 <xsl:value-of select="$cena-celkem" />
</xsl:template>

<xsl:template match="polozka">
 <tr>
  ...
 </tr>
 <xsl:variable name="cena-celkem"
               select="$cena-celkem + (./cena * ./mnozstvi)"/>
</xsl:template>
```

Nefunguje

Example8.xslt

Opakování transformace

- pomocí **xsl:for-each**
  - Obdoba for cyklů
  - Příznak **select** vybere množinu uzlů, na kterou se aplikuje tělo for cyklu

```
<xsl:for-each select=".//polozka">
 <xsl:call-template name="zpracuj-polozku" />
</xsl:for-each>
```

Podmíněná transformace

- Pomocí konstrukce **xsl:if** můžeme část šablony vykonat pouze, pokud je splněna daná podmínka
  - Příznak **test** obsahující logickou XPath podmínku
  - Nemá **else** větev

```
<xsl:if test="./@expedovana">
 <xsl:text>Objednávka byla expedována dne </xsl:text>
 <xsl:value-of select="./@expedovana" />
</xsl:if>
```

Větvení transformace

- Zobecnění **xsl:if** je **xsl:choose**
  - Jedna a více větví **xsl:when**
    - S příznakem **test** obsahující podmínku
    - Vykoná se, pokud je podmínka splněna, ostatní se přeskočí
  - Jedna větev **xsl:otherwise**
    - Vykoná se, pokud nebyla vykonána ani jedna větev **xsl:when**

Větvení transformace

```
<xsl:choose>
 <xsl:when test="./@expedovana">
  <xsl:text>Objednávka byla expedována.</xsl:text>
 </xsl:when>
 <xsl:when test="./@dorucena">
  <xsl:text>Objednávka byla doručena.</xsl:text>
 </xsl:when>
 <xsl:otherwise>
  <xsl:text>Objednávka se připravuje.</xsl:text>
 </xsl:otherwise>
</xsl:choose>
```

Počítání pomocí mezivýsledku

```
<xsl:template match="/">
 ...
 <xsl:text>Cena celkem: </xsl:text>
 <xsl:variable name="cena-mezisoucty">
  <mz:mezisoucty>
   <xsl:for-each select="//polozka">
    <mz:mezisoucet>
     <xsl:value-of select="./cena * ./mnozstvi" />
    </mz:mezisoucet>
   </xsl:for-each>
  </mz:mezisoucty>
 </xsl:variable>
 <xsl:call-template name="cena-s-dph">
  <xsl:with-param name="cena"
       select="sum($cena-mezisoucty//mz:mezisoucet)" />
 </xsl:call-template>
</xsl:template>
```

example9.xslt

Počítání pomocí rekurze (1)

```
<xsl:template name="cena-celkem">
 <xsl:param name="mezivysledek" /><xsl:param name="polozka" />
 <xsl:variable name="novymezivysledek"
       select="$mezivysledek + ($polozka/cena * $polozka/mnozstvi)" />
 <xsl:choose>
  <xsl:when test="count($polozka/following-sibling::polozka)>0">
   <xsl:call-template name="cena-celkem">
    <xsl:with-param name="mezivysledek" select="$novymezivysledek" />
    <xsl:with-param name="polozka"
                    select="$polozka/following-sibling::polozka[1]" />
   </xsl:call-template>
  </xsl:when>
  <xsl:otherwise>
   <xsl:call-template name="cena-s-dph">
    <xsl:with-param name="cena" select="$novymezivysledek" />
   </xsl:call-template>
  </xsl:otherwise>
 </xsl:choose>
</xsl:template>
```

Počítání pomocí rekurze (1)

```
<xsl:template match="/">
 ...
 <xsl:call-template name="cena-celkem">
  <xsl:with-param name="mezivysledek" select="0" />
  <xsl:with-param name="polozka"
                  select="./objednavka/polozky/polozka[1]"
/>
 </xsl:call-template>
</xsl:template>
```

- example10.xslt

Počítání pomocí rekurze (2)

```
<xsl:template name="cena-celkem">
 <xsl:param name="mezivysledek" /><xsl:param name="polozka-poz" />
 <xsl:variable name="polozka"
               select="/descendant::polozka[$polozka-poz]" />
 <xsl:variable name="novymezivysledek"
       select="$mezivysledek + ($polozka/cena * $polozka/mnozstvi)" />
 <xsl:choose>
  <xsl:when test="count($polozka/following-sibling::polozka)>0">
   <xsl:call-template name="cena-celkem">
    <xsl:with-param name="mezivysledek" select="$novymezivysledek" />
    <xsl:with-param name="polozka-poz" select="$polozka-poz + 1" />
   </xsl:call-template>
  </xsl:when>
  <xsl:otherwise>
   <xsl:call-template name="cena-s-dph">
    <xsl:with-param name="cena" select="$novymezivysledek" />
   </xsl:call-template>
  </xsl:otherwise>
 </xsl:choose>
</xsl:template>
```

Počítání pomocí rekurze (2)

```
<xsl:template match="/">
 ...
 <xsl:call-template name="cena-celkem">
  <xsl:with-param name="mezivysledek" select="0" />
  <xsl:with-param name="polozka-poz"
                  select="1" />
 </xsl:call-template>
</xsl:template>
```

Implicitní šablony

- Prázdný XSLT skript dává na neprázdném vstupu výsledek
  - example11.xslt
  - Proč? Díky implicitním šablonám.
  - Má-li se transformovat uzel, ale nelze nalézt vhodnou šablonu, aplikuje se implicitní šablona.

Implicitní šablony

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

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

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

Implicitní šablony

- Jak zakázat implicitní šablony?
  - Předefinovat je

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

- Říká, že pro žádný uzel se nemá nic dělat
- Všechny naše další šablony s příznakem **`match`** jiným než **`node()`** mají větší prioritu

Konec