Úvod do XSLT 2.0

Jirka Kosek
<jirka@kosek.cz>

Obsah

Úvod

Jazyk XSLT

- jedna z částí jazyka XSL (eXtensible Stylesheet Language)
  - jazyk XSL má dvě části – transformační (XSLT) a formátovací objekty (XSL-FO)
  - XSL realizuje jeden ze základních principů XML – oddělení obsahu od vzhledu
  - pro jedny data (XML) můžeme mít mnoho pohledů (stylů)
  - jeden styl můžeme použít pro jednotné zpracování mnoha dokumentů XML, pokud mají podobnou strukturu (vyhovují společnému XML schématu)
- XSLT umožňuje popsat transformaci z XML do XML, HTML, XHTML nebo čistého textu
- XSLT je vyvíjen a standardizován na půdě konsoricia W3C
- existují verze 1.0 (1999) a 2.0 (2007), připravuje se verze 3.0
- využití
  - prezentace informací v několika formátech (HTML, PDF, WML)
  - konverze XML zpráv používající odlišné schéma
  - obecná manipulace s XML

Specifikace XSLT 2.0

- doporučení W3C od ledna 2007
- mnoho společných částí s XQuery 1.0
- celé XSLT 2.0 je popsáno v několika dílčích specifikacích
  - XSLT 2.0[1]
  - XPath 2.0[2]
  - Datový model XQuery 1.0/XPath 2.0[3]
  - Funkce a operátory XQuery 1.0/XPath 2.0[4]
  - Serializace XQuery 1.0/XSLT 2.0[5]
- specifikace definuje dvě úrovni jazyka – Basic a Schema-Aware

[1] http://www.w3.org/TR/xslt20/
[2] http://www.w3.org/TR/xpath20/
[3] http://www.w3.org/TR/xpath-datamodel/
[4] http://www.w3.org/TR/xpath-functions/
[5] http://www.w3.org/TR/xslt-xquery-serialization/

Podpora XSLT

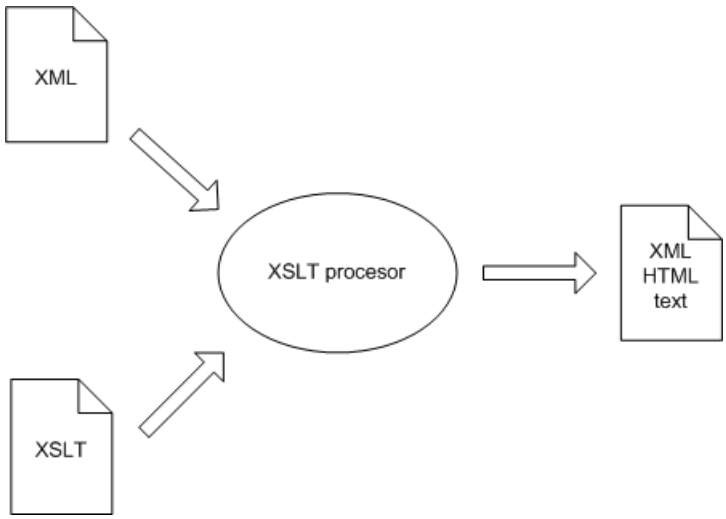
- verze 1.0
  - široce podporovaná
  - podporují všechny běžně používané prohlížeče
  - standardní součást Javy, .NET Frameworku
  - knihovny implementující XSLT 1.0 existují pro naprostou většinu běžně používaných programovacích jazyků
- verze 2.0
  - Saxon 9[6]
    - verze pro Java, .NET, Javascript a C/C++
    - verze Basic je open-source, Schema-Aware verze je komerční
  - AltovaXML[7]
    - Win32 aplikace
    - Schema-Aware, použití zdarma
  - Gestalt[8]
    - napsán v jazyce Eiffel, existují binární verze pro několik platforem
    - Basic, open-source
  - WebSphere[9]
    - Schema-Aware, komerční
    - součástí novějších verzí aplikačního serveru WebSphere
  - Intel® SOA Expressway XSLT 2.0 Processor[10]
    - Basic, komerční (pro osobní potřebu zdarma)
  - MarkLogic Server[11]
    - Schema-Aware, komerční (pro osobní potřebu zdarma)
    - součást XML databáze
  - Frameless[12]
    - Javascript

[6] http://saxonica.com
[7] http://www.altova.com/altovaxml.html
[8] http://gestalt.sourceforge.net/
[9] http://www-01.ibm.com/software/cz/websphere/
[10] http://software.intel.com/en-us/articles/intel-soa-expressway-xslt-20-processor/
[11] http://www.marklogic.com/product/marklogic-server.html
[12] http://frameless.io/

Podpora XSLT (Pokračování)

- vhodné zejména pro šablony (prohlížeč, node.js) nebo pro interaktivní aplikace v prohlížeči

Princip XSLT transformace



Ukázka transformace

Příklad 1. `clanek.xsl`

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

  <!-- Výstup bude do HTML v kódování utf-8 -->
  <xsl:output indent="yes" method="html" encoding="utf-8"/>

  <!-- Zpracování celého dokumentu, kořenový element vygeneruje
       kostru HTML dokumentu -->
  <xsl:template match="/">
    <html>
      <head>
        <title><xsl:value-of select="/clanek/zahlavi/autor"/>:
               <xsl:value-of select="/clanek/zahlavi/nazev"/></title>
        <style type="text/css">
          .zahlavi { text-align: center;
                     border: solid 2px red;
                     background-color: #F0E0E0;
                     padding: 1em; }
          .rubrika { font-size: xx-small;
                     float: right; }
        </style>
      </head>
      <body>
        <xsl:apply-templates/>
      </body>
    </html>
  </xsl:template>

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

  <xsl:template match="rubrika">
    <p class="rubrika">
      Rubrika: <b><xsl:apply-templates/></b>
    </p>
  </xsl:template>
```

Ukázka transformace (Pokračování)

```
  <xsl:template match="nazev">
    <h1 align="center"><xsl:apply-templates/></h1>
  </xsl:template>

  <xsl:template match="autor">
    <p align="center"><i><xsl:apply-templates/></i></p>
  </xsl:template>

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

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

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

</xsl:stylesheet>
```

Provedení transformace

- prohlížeče podporují instrukci `<?xml-stylesheet …?>`
  - zatím je podporováno jen XSLT 1.0
- ruční spuštění transformace
  - řádkové utility, XML editory, …

    např. Saxon:

```
$ java -jar saxon9he.jar -s:vstup.xml -xsl:styl.xsl -o:výstup

$ Transform -s:vstup.xml -xsl:styl.xsl -o:výstup
```

- automatická transformace XML na webovém serveru podle typu klienta/požadavku
- programové provedení transformace z aplikace

Základní principy

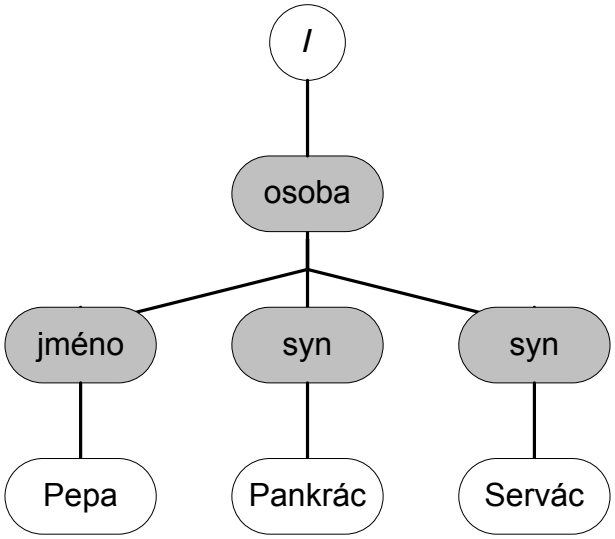
- styl/transformace obsahuje šablony, které určují, jak se budou jednotlivé části vstupního dokumentu převádět
- části dokumentu jsou v šablonách vybírány pomocí jazyka XPath
- kromě výkonného mechanismu šablon lze používat podmínky, cykly, proměnné, funkce, řazení části XML dokumentu, …
- pro zápis podmínek a všech dalších výrazů se používá jazyk XPath
- styl je sám o sobě XML dokumentem, který obsahuje dva druhy značek
  - instrukce pro XSLT procesor
  - značky výstupního formátu (HTML, XSL-FO, XML)
  - k odlišení se používají jmenné prostory

XPath 2.0

XPath

- jednoduchý dotazovací jazyk nad dokumenty XML
- využívá se v XSLT, XQuery, XML schématech a dalších jazycích, proto tvoří samostatný standard
- operuje nad stromovou strukturou dokumentu XML

```
<osoba>
  <jméno>Pepa</jméno>
  <syn>Pankrác</syn>
  <syn>Servác</syn>
</osoba>
```



- výsledkem dotazu je posloupnost složená z uzlů stromu dokumentu nebo hodnot
- velmi úsporná syntaxe zaměřená především na navigaci a výběr hodnot ze stromu dokumentu

Datový model

- dotazy operují nad datovým modelem XML – XDM (XQuery and XPath Data Model)
- XPath je uzavřený nad XDM – výsledkem dotazu je opět instance XDM
- základním typem je posloupnost
- posloupnost se může skládat z atomických hodnot (číslo, řetězec, logická hodnota, …) nebo uzlů
- typy uzlů
  - dokument
  - element
  - atribut
  - textový uzel
  - komentář
  - instrukce pro zpracování
- každý uzel má své vlastnosti

Anatomie XPath výrazu

- nejtypičtější XPath výraz vybírá uzly ze vstupního dokumentu

  část cesty/*část cesty*/*část cesty*/…

- každá část cesty se skládá ze tří částí:
  - osa
  - test uzlu
  - predikát
- každá část cesty vrátí uzly ležící na dané ose, vyhovující testu uzlu a splňující predikát
- vybrané uzly se stávají aktuálním uzlem pro další část cesty
- výsledkem celého výrazu jsou uzly vrácené poslední částí cesty
- XPath výraz se začíná vyhodnocovat buď o kořenového uzlu (začíná-li na /), nebo od aktuálního uzlu (ten je definován v hostitelském jazyce jako je XSLT)

Osy

`ancestor::`
: Rodič aktuálního uzlu a všichni jeho další předci až ke kořenu stromu dokumentu. Uzly jsou uspořádány v opačném pořadí než v dokumentu.

`ancestor-or-self::`
: Aktuální uzel a všichni jeho další předci až ke kořenu stromu dokumentu. Uzly jsou uspořádány v opačném pořadí než v dokumentu.

`attribute::`
: Uzly odpovídající všem připojeným atributům, pořadí není nijak pevně definováno.

`child::`
: Všechny děti aktuálního uzlu ve stejném pořadí jako v dokumentu.

`descendant::`
: Všichni potomci aktuálního uzlu ve stejném pořadí jako v dokumentu.

`descendant-or-self::`
: Aktuální uzel a všichni jeho potomci ve stejném pořadí jako v dokumentu.

`following::`
: Všechny uzly následující za koncem aktuálního uzlu ve stejném pořadí jako v dokumentu.

`following-sibling::`
: Všechny následující uzly, které jsou sourozenci aktuálního uzlu, ve stejném pořadí jako v dokumentu.

`namespace::`
: Uzly odpovídající deklarovaným jmenným prostorům, pořadí není nijak pevně definováno.

`parent::`
: Rodič aktuálního uzlu.

`preceding::`
: Všechny uzly, jejichž všichni potomci jsou v dokumentu ještě před aktuálním uzlem, v opačném pořadí než v dokumentu.

`preceding-sibling::`
: Všechny předcházející uzly, které jsou sourozencem aktuálního uzlu. Uzly jsou uspořádány v opačném pořadí než v dokumentu.

`self::`
: Aktuální uzel.

Test uzlu

- `název` – element nebo atribut daného názvu
- `*` – název může být libovolný
- `prefix:*`, `prefix:název` – výběr elementů/atributů patřících do určitého jmenného prostoru (pozor, prefix musí být deklarován)
- `*:název` – výběr elementů/atributů daného jména bez ohledu na jmenný prostor
- `processing-instruction()` – vybere všechny uzly odpovídající instrukci pro zpracování
- `processing-instruction(`*`cíl`*`)` – vybere všechny instrukce pro zpracování s daným jménem
- `comment()` – vybere všechny komentáře
- `text()` – vybere všechny textové uzly
- `node()` – vybere všechny uzly bez ohledu na jejich typ
- `document-node()` – vybere uzel dokumentu

Zkrácený zápis

- samotný název uzlu nebo test uzlu je považován jako pohyb po ose dětí (`child::`)

```
para    child::para
text()  child::text()
*       child::*
```

- pokud se chceme pohybovat po atributech, můžeme místo `attribute::` použít zavináč (`` `@' ``)

```
@id attribute::id
```

- na aktuální uzel se můžeme odkazovat zkráceným zápisem `` `.' `` místo `self::node()`
- na rodičovský uzel se můžeme odkazovat zkráceným zápisem `` `..' `` místo `parent::node()`
- pokud potřebujeme prohledávat potomky do libovolné hloubky, můžeme místo poněkud dlouhého `/descendant-or-self::node()/` použít `` `//' ``

```
//nadpis /descendant-or-self::node()/child::nadpis
```

Predikáty

- predikát je nepovinnou částí cesty
- dále filtruje uzly vybrané na základě testu uzlu na zvolené ose
- do výsledku se dostanou jen uzly, pro které je predikát splněný
- *číselná hodnota* – pokud predikát obsahuje výraz, který vrací číslo, chápe se jako pozice v množině uzlů na ose, po které se v dané části cesty pohybujeme.

  Uzly jsou číslovány od jedničky. Pokud se pohybujeme po reverzních osách (ancestor, preceding), jsou uzly číslovány v opačném pořadí než v dokumentu. Číselná hodnota tak spíše odpovídá pojmu vzdálenost.

  Pravdivou hodnotu má predikát pouze pro uzel s daným pořadím.
- *množina uzlů* – jako výraz v predikátu můžeme použít XPath výraz, který vrací množinu uzlů. Pokud je množina uzlů neprázdná, má predikát hodnotu `true`, v opačném případě `false`.
- *ostatní výrazy* – ostatní výrazy jsou převedeny na logickou hodnotu a jejich výsledek je i hodnotou predikátu pro daný uzel.
- predikáty lze aplikovat na libovolnou posloupnost

Příklady XPath výrazů

`katalog`
Vybere všechny elementy `katalog`, které jsou dětmi aktuálního uzlu.

`./katalog`
Vybere všechny elementy `katalog`, které jsou dětmi aktuálního uzlu.

`/katalog`
Vybere všechny elementy `katalog`, které jsou dětmi kořenového uzlu. Takový element může být v dokumentu maximálně jeden a je to kořenový element.

`*`
Vybere všechny elementy, které jsou dětmi aktuálního uzlu.

`katalog/*`
Vybere všechny elementy, které jsou dětmi elementu `katalog`, který je dítětem aktuálního uzlu.

`text()`
Vybere všechny textové uzly, které jsou dětmi aktuálního uzlu.

`//text()`
Vybere všechny textové uzly v celém dokumentu.

`@href`
Vybere atribut `href` aktuálního uzlu.

`foto/@href`
Vybere atribut `href` elementu `foto`, který je dítětem aktuálního uzlu.

`@*`
Vybere všechny atributy aktuálního uzlu.

`//*[@*]`
Vybere všechny elementy dokumentu, které mají alespoň jeden atribut.

`polozka[1]`
Vybere první element `polozka`, který je dítětem aktuálního uzlu.

`/katalog/polozka[1]`
Vybere první element `polozka`, který je dítětem elementu `katalog`, který je kořenovým elementem dokumentu.

`polozka[last()]`
Vybere poslední element `polozka`, který je dítětem aktuálního uzlu.

`*/nazev`
Vybere všechny elementy `nazev`, které jsou vnoučaty (dětmi dětí) aktuálního uzlu.

Příklady XPath výrazů (Pokračování)

`katalog//cena`
: Vybere všechny elementy `cena`, které jsou potomky elementu `katalog`, který je dítětem aktuálního uzlu.

`//cena`
: Vybere všechny elementy `cena`, které jsou potomky kořenového uzlu. To v praxi znamená, že jsou vybrány všechny elementy `cena`, které se nacházejí ve stejném dokumentu jako aktuální uzel.

`.`
: Vybere aktuální uzel.

`.//b`
: Vybere všechny elementy `b`, které jsou potomky aktuálního uzlu.

`..`
: Vybere rodiče aktuálního uzlu.

`//polozka[cena > 10000]/popis`
: Vybere všechny elementy `popis`, které jsou dětmi elementu `polozka`, v případě, že jeho podelement `cena` je větší než 10000.

  Lidsky řečeno vybere popis všech položek faktury, které jsou dražší než 10000.

`/katalog/polozka[kategorie='MiniDisc']`
: Vybere element `polozka`, který je dítětem kořenového elementu `katalog`, pokud obsahuje jako dítě element `kategorie` s textem `MiniDisc`.

`self::polozka`
: Vybere aktuální uzel, pokud je to element se jménem `polozka`.

`preceding::*[1]`
: Vybere poslední element, který se nachází před aktuálním uzlem.

`preceding-sibling::*[1]`
: Vybere poslední element, který se nachází před aktuálním uzlem a je na stejné úrovni (je to sourozenec).

`following::*[1]`
: Vybere první element, který se nachází za aktuálním uzlem.

Kontext pro vyhodnocování

- vyhodnocování každého XPath výrazu probíhá v kontextu, který ovlivňuje výsledek dotazu
- kontext je tvořen následujícími údaji
  - kontextová položka (aktuální uzel);
  - pozice kontextové položky v posloupnosti právě zpracovávaných položek (vrací `position()`);
  - velikost kontextu (vrací funkce `last()`);
  - přiřazené proměnné;
  - dostupné funkce;
  - deklarace jmenných prostorů – použijí se pokud výraz používá prefixy jmenných prostorů;
  - další „detaily“ – aktuální datum a čas, časová zóna, dostupné dokumenty, …
- kontext se při vyhodnocování výrazu postupně mění

Posloupnosti

- základní datová struktura XPathu
- prvky mohou být uzly nebo atomické hodnoty (číslo, řetězec, …)
- všechny hodnoty v XPathu 2.0 jsou posloupnosti – atomická hodnota je posloupnost o délce jedna
- jednotlivé prvky posloupnosti mohou mít rozdílné typy
- prvky v posloupnosti jsou uspořádané
- prvky v posloupnosti mohou být duplicitní

Vytvoření posloupnosti

- XPath cesta vrací výsledek jako posloupnost
- mnoho funkcí vrací jako výsledek posloupnost
- posloupnost lze zkonstruovat ručně

```
1, 2, 3
"ahoj", 42, //cena
```

- spojitou posloupnost celých čísel lze generovat pomocí operátoru `to`

```
1 to 100
```

Zpracování posloupnosti

- můžeme využít prostředky XSLT (`xsl:for-each`, `xsl:apply-templates`)
- přímo XPath nabízí nový příkaz `for` pro pro průchod a zpracování posloupnosti

```
for $proměnná in posloupnost
return výraz používající proměnnou
```

- vypsání druhé mocniny čísel 1 až 10

```
for $i in 1 to 10 return $i * $i
```

- sečtení objednávky

```
sum(for $polozka in /objednavka/polozka
    return $polozka/cena * $polozka/pocet)
```

Základní operátory XPath

- matematické operátory
  - +, -, *, div, mod
- logické operátory
  - and, or, not()
- relační operátory
  - obecné porovnávání (`=`, `!=`, `<`, `<=`, `>`, `>=`)
    - operandy mohou být posloupnosti
    - stačí, aby podmínku splnil jeden z prvků posloupnosti
    - typ pro porovnávání se může určit automaticky
  - porovnání hodnot (`eq`, `ne`, `lt`, `le`, `gt`, `ge`)
    - porovnávají se vždy dvě hodnoty
    - typy operandů musí být stejné (s výjimkou číselných typů)

Operátory pro práci s uzly

- porovnání identity dvou uzlů

  `uzelA is uzelB`

- porovnávání vzájemné pozice uzlů
  - uzel A je před uzlem B – `uzelA << uzelB`
  - uzel A je za uzlem B – `uzelA >> uzelB`
- množinové operace s uzly
  - `union` – sjednocení (lze zapisovat i jako `|`)
  - `except` – rozdíl
  - `intersect` – průnik

Podmíněný výraz

- přímo na úrovni XPathu lze zapisovat podmínky

```
if (podmínka) then výraz else výraz
```

- číst `else` je povinná, ale může vracet prázdnou sekvenci `()`
- výrazy XPath lze libovolně kombinovat

```
for $i in 1 to 100 return
  if ($i mod 2 = 0) then $i else ()
```

Další možnosti

- část cesty může obsahovat sjednocení

  místo

  ```
  /kniha/priloha/nazev | /kniha/kapitola/nazev
  ```

  můžeme použít

  ```
  /kniha/(priloha|kapitola)/nazev
  ```

- funkce lze zapisovat jako část cesty

  ```
  /katalog/polozka/nazev/upper-case(.)
  ```

  vrátí posloupnost názvů položek katalogu převedených na velká písmena

  ```
  sum(//polozka/(cena * pocet))
  ```

- komentáře lze zapisovat pomocí `(: … :)`

  ```
  (: Výraz pro výběr všech elementů s atributy :)
  //*[@*]
  ```

- pořadí vracených uzlů
  - cesta XPath vrací vždy uzly v pořadí v jakém byly v dokumentu a eliminuje duplicity
  - posloupnost je vždy uspořádaná a může obsahovat duplicity
  - posloupnost lze použít jako část cesty
  - srovnání

    ```
    //polozka[1] | //polozka[2] | //polozka[1]
    //polozka[1], //polozka[2], //polozka[1]
    (//polozka[1], //polozka[2], //polozka[1])/.
    ```

Kvantifikátory

- existenční kvantifikátor

  some $*proměnná* in *posloupnost* satisfies *podmínka*

- všeobecný kvantifikátor

  every $*proměnná* in *posloupnost* satisfies *podmínka*

- pro podmínku se nemění kontextová položka, proto se zde obvykle odvoláváme na proměnnou
- příklad – výpis prvočísel

```
for $i in (2 to 100) return
  if (every $j in (2 to $i - 1) satisfies $i mod $j ne 0)
    then $i
    else ()
```

Funkce XPath

- funkce pro práci s uzly
  - last(), position(), count(), id(), local-name(), name(), namespace-uri()
- řetězcové funkce
  - string(), concat(), string-join(), starts-with(), ends-with(), contains(), substring-before(), substring-after(), substring(), string-length(), normalize-space(), translate(), codepoints-to-string(), string-to-codepoints(), compare(), normalize-unicode(), upper-case(), lower-case(), encode-for-uri(), iri-to-uri(), escape-html-uri()
- funkci pro práci s regulárními výrazy
  - matches(), tokenize(), replace()
- logické funkce
  - boolean(), not(), true(), false(), lang()
- matematické funkce
  - number(), sum(), avg(), min(), max(), floor(), ceiling(), round(), round-half-to-even(), abs()
- funkce pro práci s datem a časem
  - current-date(), current-time(), current-dateTime()
- funkce pro práci s posloupnostmi
  - index-of(), empty(), exists(), distinct-values(), insert-before(), remove(), reverse(), subsequence(), zero-or-one(), one-or-more(), exactly-one(), deep-equal()
- další funkce
  - doc(), collection(), id(), idref(), doc-available()
- rozšiřující funkce definované pro XSLT
  - document(), unparsed-text(), key(), format-number(), format-time(), format-date(), format-dateTime(), current(), unparsed-entity-uri(), generate-id(), system-property(), element-available(), function-available()

Základy XSLT

Nejdůležitější elementy

- <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  version="2.0">
  ... šablony ...
  </xsl:stylesheet>

- <xsl:template match="*XPath*">
  …
  </xsl:template>

- <xsl:apply-templates/>
  - hledá další šablony
  - šablony mohou generovat do výstupu další elementy
- <xsl:value-of select="*XPath*"/>
  - do výstupu se vloží pouze výsledek výrazu (řetězec)
  - v XSLT 1.0 se zpracovává pouze první vrácený uzel
  - v XSLT 2.0 celá posloupnost

Zpracování pomocí šablon

Příklad 2. **katalog-sablony.xsl**

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                version="2.0">

<!-- Nastavení výstupní metody a kódování -->
<xsl:output method="html" encoding="utf-8"/>

<!-- Šablona pro kořenový uzel generuje kostru HTML -->
<xsl:template match="/">
  <html>
    <head>
      <title><xsl:value-of select="/katalog/nazev"/></title>
      <style type="text/css">
        body   { color: black; background-color: white; }
        table  { border: solid 2px black; border-collapse: collapse; }
        th a   { text-decoration: none; color: black; }
        th     { background-color: #FF8000; }
        th, td { border: solid 1px black; }
      </style>
    </head>
    <body>
      <h1>
        <xsl:value-of select="/katalog/nazev"/>
      </h1>

      <table width="100%">
        <tr>
          <th>Název</th>
          <th>Kategorie</th>
          <th>Cena</th>
        </tr>
        <xsl:apply-templates/>
      </table>
    </body>
  </html>
</xsl:template>

<!-- Šablony pro jednotlivé elementy -->
<xsl:template match="polozka">
  <tr>
```

Zpracování pomocí šablon (Pokračování)

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

<xsl:template match="nazev">
  <td align="center"><xsl:apply-templates/></td>
</xsl:template>

<xsl:template match="kategorie">
  <td align="center"><xsl:apply-templates/></td>
</xsl:template>

<xsl:template match="cena">
  <td align="right"><xsl:apply-templates/> Kč</td>
</xsl:template>

<!-- Prázdné šablony pro elementy, které se mají ignorovat -->
<xsl:template match="vyrobce"/>

<xsl:template match="katalog/nazev"/>

</xsl:stylesheet>
```

Cykly

- iterace přes množinu uzlů/prvky posloupnosti

```
<xsl:for-each select="XPath výraz">
  …
</xsl:for-each>
```

- každý zpracovávaný uzel se stává při průchodu tělem cyklu aktuálním uzlem
- funkce `last()` vrací celkový počet iterací
- funkce `position()` vrací kolikátou iteraci provádíme

Zpracování pomocí iterace

Příklad 3. **katalog-jinak.xsl**

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                version="2.0">

<xsl:output method="html" encoding="utf-8"/>

<xsl:template match="/">
  <html>
    <head>
      <title><xsl:value-of select="/katalog/nazev"/></title>
      <style type="text/css">
        body   { color: black; background-color: white; }
        table  { border: solid 2px black; border-collapse: collapse; }
        th a   { text-decoration: none; color: black; }
        th     { background-color: #FF8000; }
        th, td { border: solid 1px black; }
      </style>
    </head>
    <body>
      <h1>
        <xsl:value-of select="/katalog/nazev"/>
      </h1>

      <table width="100%">
        <tr>
          <th>#</th>
          <th>Název</th>
          <th>Kategorie</th>
          <th>Cena</th>
        </tr>
        <!-- Postupně zpracujeme všechny položky -->
        <xsl:for-each select="/katalog/polozka">
          <tr>
            <td align="center"><xsl:value-of select="position()"/>.</td>
            <td align="center"><xsl:value-of select="nazev"/></td>
            <td align="center"><xsl:value-of select="kategorie"/></td>
            <td align="right"><xsl:value-of select="cena"/> Kč</td>
          </tr>
        </xsl:for-each>
      </table>
```

Zpracování pomocí iterace (Pokračování)

```
    </body>
  </html>
</xsl:template>

</xsl:stylesheet>
```

Zpracování pomocí iterace

Příklad 4. **katalog-otoceny.xsl**

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                version="2.0">

<xsl:output method="html" encoding="utf-8"/>

<xsl:template match="/">
  <html>
    <head>
      <title><xsl:value-of select="/katalog/nazev"/></title>
      <style type="text/css">
        body   { color: black; background-color: white; }
        table  { border: solid 2px black; border-collapse: collapse; }
        th a   { text-decoration: none; color: black; }
        th     { background-color: #FF8000; }
        th, td { border: solid 1px black; }
      </style>
    </head>
    <body>
      <h1>
        <xsl:value-of select="/katalog/nazev"/>
      </h1>

      <table width="100%">
        <tr>
          <th>Název</th>
          <!-- Vybereme všechny položky a vypíšeme jejich názvy -->
          <xsl:for-each select="/katalog/polozka">
            <td align="center"><xsl:value-of select="nazev"/></td>
          </xsl:for-each>
        </tr>
        <tr>
          <th>Kategorie</th>
          <!-- Vybereme všechny položky a vypíšeme jejich kategorie -->
          <xsl:for-each select="/katalog/polozka">
            <td align="center"><xsl:value-of select="kategorie"/></td>
          </xsl:for-each>
        </tr>
        <tr>
          <th>Cena</th>
```

Zpracování pomocí iterace (Pokračování)

```
          <!-- Vybereme všechny položky a vypíšeme jejich ceny -->
          <xsl:for-each select="/katalog/polozka">
            <td align="center"><xsl:value-of select="cena"/></td>
          </xsl:for-each>
        </tr>
      </table>

    </body>
  </html>
</xsl:template>

</xsl:stylesheet>
```

Přístupy ke psaní stylu

- `xsl:template` a `xsl:apply-templates`
  - procesor XSLT automaticky prochází strom vstupního dokumentu a volá odpovídající šablony
  - struktura vstupního i výstupního dokumentu je podobná
- `xsl:for-each`
  - ručně si vybíráme data ke zpracování v pořadí a struktuře jakou potřebujeme
  - struktura výstupního a vstupního dokumentu může být zcela odlišná
- oba přístupy lze libovolně kombinovat

Podmínky

- <xsl:if test="*podmínka*">
  ...
  </xsl:if>

- náhrada if-then-else

```
<xsl:choose>
  <xsl:when test="podmínka">
    …
  </xsl:when>
  <xsl:otherwise>
    …
  </xsl:otherwise>
</xsl:choose>
```

- výběr z více variant

```
<xsl:choose>
  <xsl:when test="podmínka">
    …
  </xsl:when>
  <xsl:when test="podmínka">
    …
  </xsl:when>
  <xsl:otherwise>
    …
  </xsl:otherwise>
</xsl:choose>
```

Použití podmínek

Příklad 5. **katalog-podminky.xsl**

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                version="2.0">

<xsl:output method="html" encoding="utf-8"/>

<xsl:template match="/">
  <html>
    <head>
      <title><xsl:value-of select="/katalog/nazev"/></title>
      <style type="text/css">
        body   { color: black; background-color: white; }
        table  { border: solid 2px black; border-collapse: collapse; }
        th a   { text-decoration: none; color: black; }
        th     { background-color: #FF8000; }
        th, td { border: solid 1px black; }
        .even  { background-color: #FFFF80; }
      </style>
    </head>
    <body>
      <h1>
        <xsl:value-of select="/katalog/nazev"/>
      </h1>

      <table width="100%">
        <tr>
          <th>#</th>
          <th>Název</th>
          <th>Kategorie</th>
          <th>Cena</th>
        </tr>
        <xsl:for-each select="/katalog/polozka">
          <tr>
            <!-- Podmíněné obarvení sudých/lichých řádek -->
            <xsl:if test="position() mod 2 = 0">
              <xsl:attribute name="class">even</xsl:attribute>
            </xsl:if>
            <td align="center"><xsl:value-of select="position()"/>.</td>
            <td align="center">
              <xsl:value-of select="nazev"/><br/>
```

Použití podmínek (Pokračování)

```
            <small>
              <b>Doporučené příslušenství: </b>
              <!-- Doplnění popisu na základě kategorie -->
              <xsl:choose>
                <xsl:when test="kategorie = 'MiniDisc'">
                  prázdná MD média
                </xsl:when>
                <xsl:when test="kategorie = 'Receiver'">
                  anténa a nezapomeňte
                 na <a href="http://www.rozhlas.cz/poplatek">rozhlasový
                  poplatek</a>
                </xsl:when>
                <xsl:otherwise>-</xsl:otherwise>
              </xsl:choose>
            </small>
          </td>
          <td align="center"><xsl:value-of select="kategorie"/></td>
          <td align="right"><xsl:value-of select="cena"/> Kč</td>
        </tr>
      </xsl:for-each>
    </table>
  </body>
 </html>
</xsl:template>

</xsl:stylesheet>
```

Generování výstupu

- elementy nepatřící do jmenného prostoru XSLT se beze změn kopírují do výstupu
- stejně tak textové uzly
- textové uzly obsahující pouze bílé znaky se uvnitř šablon ignorují
- lepší kontrolu nad bílými znaky lze získat pomocí `xsl:text`
- spočítanou hodnotu lze vypsat pomocí `xsl:value-of`
- hodnotu generovaného atributu lze vygenerovat dynamicky pomocí šablon pro hodnotu atributu (AVT = Attribute Value Template)

```
<a href="{@filename}.html">…</a>
```

- elementy a atributy s předem neznámým jménem lze generovat pomocí instrukcí `xsl:element` a `xsl:attribute`

Ukázka generování výstupu

Příklad 6. **katalog-generovani-vystupu.xsl**

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                version="2.0">

<xsl:output method="html" encoding="utf-8"/>

<xsl:template match="/">
  <html>
    <head>
      <title><xsl:value-of select="/katalog/nazev"/></title>
      <style type="text/css">
        body   { color: black; background-color: white; }
        table  { border: solid 2px black; border-collapse: collapse; }
        th a   { text-decoration: none; color: black; }
        th     { background-color: #FF8000; }
        th, td { border: solid 1px black; }
      </style>
    </head>
    <body>
      <h1>
        <xsl:value-of select="/katalog/nazev"/>
        <!-- Vypsání mezery do výstupu, bez xsl:text by byla zrušena -->
        <xsl:text> </xsl:text>
        <xsl:value-of select="current-date()"/>
      </h1>

      <table width="100%">
        <tr>
          <th>#</th>
          <th>Název</th>
          <th>Kategorie</th>
          <th>Cena</th>
        </tr>
        <xsl:for-each select="/katalog/polozka">
          <tr>
            <td align="center"><xsl:value-of select="position()"/>.</td>
            <td align="center">
              <span title="Cena: {cena}">
                <xsl:value-of select="nazev"/>
              </span>
```

Ukázka generování výstupu (Pokračování)

```
            </td>
            <td align="center"><xsl:value-of select="kategorie"/></td>
            <td align="right"><xsl:value-of select="cena"/> Kč</td>
          </tr>
        </xsl:for-each>
      </table>
    </body>
  </html>
</xsl:template>

</xsl:stylesheet>
```

Šablony

Volání šablon jen pro vybrané uzly

- `<xsl:apply-templates select="`*XPath*`"/>`

- standardně `xsl:apply-templates` zpracovává všechny dětské uzly (`select="child::node()"`)
- v atributu `match` lze používat pouze vzory (patterns) ne úplné výrazy (expressions)
  - lze používat pouze osy child, descendant, attribute
  - ostatní osy lze použít až v případném predikátu

Příklad 7. `katalog-sablony-vyber-uzlu.xsl`

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                version="2.0">

<!-- Nastavení výstupní metody a kódování -->
<xsl:output method="html" encoding="utf-8"/>

<!-- Šablona pro kořenový uzel generuje kostru HTML -->
<xsl:template match="/">
  <html>
    <head>
      <title><xsl:value-of select="/katalog/nazev"/></title>
      <style type="text/css">
        body   { color: black; background-color: white; }
        table  { border: solid 2px black; border-collapse: collapse; }
        th a   { text-decoration: none; color: black; }
        th     { background-color: #FF8000; }
        th, td { border: solid 1px black; }
      </style>
    </head>
    <body>
      <h1>
        <xsl:value-of select="/katalog/nazev"/>
      </h1>

      <table width="100%">
        <tr>
          <th>Název</th>
          <th>Kategorie</th>
          <th>Cena</th>
```

Volání šablon jen pro vybrané uzly (Pokračování)

```
        </tr>
        <!-- K dalšímu zpracování vybereme pouze položky -->
        <xsl:apply-templates select="/katalog/polozka"/>
      </table>
    </body>
  </html>
</xsl:template>

<!-- Šablony pro jednotlivé elementy -->
<xsl:template match="polozka">
  <tr>
    <!-- Zpracováváme jen vybrané podlementy položky -->
    <xsl:apply-templates select="nazev | kategorie | cena"/>
  </tr>
</xsl:template>

<xsl:template match="nazev">
  <td align="center"><xsl:apply-templates/></td>
</xsl:template>

<xsl:template match="kategorie">
  <td align="center"><xsl:apply-templates/></td>
</xsl:template>

<xsl:template match="cena">
  <td align="right"><xsl:apply-templates/> Kč</td>
</xsl:template>

</xsl:stylesheet>
```

Režimy zpracování

- v praxi často potřebujeme jednotlivé části dokumentu XML zpracovat několikrát různými způsoby
- pro jeden uzel můžeme mít více šablon odlišených pomocí režimu

```
<xsl:template match="…" mode="režim">
```

- při volání šablon můžeme určit režim, v jakém se mají šablony hledat

```
<xsl:apply-templates mode="režim"/>
```

- speciální názvy režimů – `#all` (vyhoví všem režimům), `#default` (výchozí režim, tj. žádný), `#current` (režim právě prováděné šablony)

Příklad 8. **`katalog-rezimy.xsl`**

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                version="2.0">

  <xsl:output method="html" encoding="utf-8"/>

  <xsl:template match="/">
    <html>
      <head>
       <title>Katalog <xsl:value-of select="/katalog/info/firma"/></title>
      </head>
      <body>
        <h1>Katalog <xsl:value-of select="/katalog/info/firma"/></h1>

        <table WIDTH="100%" BORDER="1">
          <xsl:apply-templates select="/katalog/polozka"/>
        </table>

       <xsl:apply-templates select="/katalog/polozka" mode="detailni"/> ▶

      </body>
    </html>
  </xsl:template>

  <xsl:template match="polozka">
    <tr>
      <xsl:apply-templates select="nazev|kategorie|cena"/>
```

Režimy zpracování (Pokračování)

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

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

  <xsl:template match="kategorie">
    <td align="right"><xsl:apply-templates/></td>
  </xsl:template>

  <xsl:template match="cena">
    <td align="right"><xsl:apply-templates/> Kč</td>
  </xsl:template>

  <xsl:template match="polozka" mode="detailni">
    <xsl:apply-templates mode="detailni"/>
    <hr/>
  </xsl:template>

  <xsl:template match="nazev" mode="detailni">
    <h2><xsl:apply-templates mode="detailni"/></h2>
  </xsl:template>

  <xsl:template match="kategorie" mode="detailni">
    <em><xsl:apply-templates mode="detailni"/></em>
  </xsl:template>

  <xsl:template match="cena" mode="detailni">
    <strong> - <xsl:apply-templates mode="detailni"/> Kč</strong>
  </xsl:template>

  <xsl:template match="popis" mode="detailni">
    <p><xsl:apply-templates mode="detailni"/></p>
  </xsl:template>

  <xsl:template match="br" mode="detailni">
    <br/>
  </xsl:template>

</xsl:stylesheet>
```

Zabudované šablony

- aplikují se, pokud daný uzel není obsloužen šablonou ve stylu
- ```
  <!-- Šablona pro postupné zpracování celého dokumentu
       ve všech režimech -->
  <xsl:template match="*|/" mode="#all">
    <xsl:apply-templates mode="#current"/>
  </xsl:template>

  <!-- Kopírování textových uzlů -->
  <xsl:template match="text()|@*" mode="#all">
    <xsl:value-of select="string(.)"/>
  </xsl:template>

  <!-- Ignorování komentářů a instrukcí pro zpracování -->
  <xsl:template match="comment()|processing-instruction()" mode="#all"/>
  ```
  ```

Konflikty mezi šablonami

- pokud uzlu vyhovuje více šablon, vybere se šablona s vyšší prioritou
- prioritu lze určit ručně pomocí atributu `priority`

```
<xsl:template match="…" priority="100">
```

- není-li priorita explicitně určena, spočítá se automaticky
  - šablony testující jméno elementu mají prioritu 0
  - šablony používající hodně obecné testy (např. `match="*"`) mají prioritu -0.5
  - ostatní šablony mají prioritu 0.5
  - výše uvedená pravidla jsou ve skutečnosti složitější

Další možnosti jazyka

Řazení dat

- data před zpracováním pomocí instrukcí `xsl:apply-templates` a `xsl:for-each` můžeme seřadit
- `<xsl:sort select="`*výraz*`" data-type="`*typ*`"/>`

- data v dokumentu XML jsou neotypovaná a chápou se jako text (pokud nepoužíváme XML schéma a Schema-Aware verzi XSLT), takže musíme určovat datový typ nebo provést ruční přetypování
- lze určit mnoho dalších parametrů, např. jazyk pro správné řazení podle národních specifik

Příklad 9. `katalog-razeni.xsl`

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                version="2.0">

<xsl:output method="html" encoding="utf-8"/>

<xsl:template match="/">
  <html>
    <head>
      <title><xsl:value-of select="/katalog/nazev"/></title>
      <style type="text/css">
        body   { color: black; background-color: white; }
        table  { border: solid 2px black; border-collapse: collapse; }
        th a   { text-decoration: none; color: black; }
        th     { background-color: #FF8000; }
        th, td { border: solid 1px black; }
      </style>
    </head>
    <body>
      <h1>
        <xsl:value-of select="/katalog/nazev"/>
      </h1>

      <p>Položky jsou seřazeny podle ceny.</p>

      <table width="100%">
        <tr>
          <th>Název</th>
          <th>Kategorie</th>
          <th>Cena</th>
```

Řazení dat (Pokračování)

```
        </tr>
        <!-- Pomocí šablon zpracujeme všechny položky -->
        <!-- Před voláním šablon jsou položky seřazeny -->
        <xsl:apply-templates select="/katalog/polozka">
          <xsl:sort select="cena" data-type="number"/>
        </xsl:apply-templates>
      </table>
    </body>
  </html>
</xsl:template>

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

<xsl:template match="nazev">
  <td align="center"><xsl:apply-templates/></td>
</xsl:template>

<xsl:template match="kategorie">
  <td align="center"><xsl:apply-templates/></td>
</xsl:template>

<xsl:template match="cena">
  <td align="right"><xsl:apply-templates/> Kč</td>
</xsl:template>

<xsl:template match="vyrobce"/>

<xsl:template match="katalog/nazev"/>

</xsl:stylesheet>
```

Seskupování

- instrukce `xsl:for-each-group` pro seskupení uzlů a jejich zpracování po skupinách
- uzly se vybírají pomocí výrazu v atributu `select`
- několik způsobů vytváření skupin
  - `group-by` – seskupení podle společné hodnoty
  - `group-adjacent` – seskupení po sobě jdoucích uzlů se společnou hodnotou
  - `group-starting-with` – každá skupina je určena svým počátečním uzlem
  - `group-ending-with` – každá skupina je určena svým koncovým uzlem
- uvnitř `xsl:for-each-group` jsou všechny prvky skupiny dostupné pomocí funkce `current-group()`
- `current-grouping-key()` vrací společnou hodnotu, jíž odpovídající skupina se právě zpracovává (pro `group-by` a `group-adjacent`)

Ukázka seskupování

Příklad 10. `citaty-seskupeni.xsl`

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="2.0"
                xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="html" encoding="utf-8"/>

  <xsl:template match="citaty">
    <html>
      <head>
        <title>Citáty podle autorů</title>
      </head>
      <body>
        <xsl:for-each-group select="citat" group-by="autor">
          <xsl:sort select="current-grouping-key()" lang="cs"/>
          <h1><xsl:value-of select="current-grouping-key()"/></h1>
          <xsl:for-each select="current-group()">
            <xsl:sort select="text" lang="cs"/>
            <p>
              <xsl:value-of select="text"/>
            </p>
          </xsl:for-each>
        </xsl:for-each-group>
      </body>
    </html>
  </xsl:template>

</xsl:stylesheet>
```

Číslování

- ruční číslování

```
<xsl:number value="…" format="…"/>
```

- automatické číslování

```
<xsl:number count="…" from="…" level="any|single|multiple"/>
```

Příklad 11. **cislovani.xsl**

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                version="2.0">

<xsl:template match="*">
  <xsl:number count="*" level="multiple" format="1. "/>
  <xsl:value-of select="name()"/>
  <br/>
  <xsl:apply-templates/>
</xsl:template>

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

</xsl:stylesheet>
```

Ovládání formátu výstupu

- transformace může produkovat různé výstupy – HTML, XHTML, XML, prostý text
- transformace probíhá z XDM do XDM
- formát výstupu se bere v potaz až při serializaci
- formát výstupu se nastavuje pomocí instrukce `xsl:output`
- nejdůležitější je parametr `method`, určující výstupní metodu
  - `method="xml"` – při serializaci se používají běžné konvence XML
  - `method="html"` – respektují se specifika syntaxe HTML – např. místo `<br/>` se vygeneruje `<br>`
  - `method="xhtml"` – používají se pravidla pro XML a navíc je výstup uzpůsoben čtení pomocí parseru HTML; generované elementy musí být ve jmenném prostoru XHTML
  - `method="text"` – do výstupu se dostanou jen textové uzly
  - standardně se používá metoda XML, pokud je kořenový element `html` automaticky se přepne na metodu HTML
- `<xsl:output method="`*{html|xhtml|xml|text}*`"`
  `encoding="`*kódování*`"`
  `indent="`*{yes|no}*`"`
  `doctype-system="`*URI*`"`
  `doctype-public="`*FPI*`"/>`

Příklady použití xsl:output

- generování stránky v HTML 4.01 Strict:

```
<xsl:output method="html" encoding="utf-8"
            doctype-public="-//W3C//DTD HTML 4.01//EN"
            doctype-system="http://www.w3.org/TR/html4/strict.dtd"/>
```

- generování stránky v XHTML 1.0 Transitional:

```
<xsl:output method="xhtml" encoding="utf-8"
            doctype-public="-//W3C//DTD XHTML 1.0 Transitional//EN"
            ▶
doctype-system="http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd"/>
```

- generování HTML5:

```
<xsl:output method="html" encoding="utf-8"
            html-version="5.0"/>

<xsl:output method="html" encoding="utf-8"
            doctype-system="about:legacy-compat"/>
```

  případně je možné !DOCTYPE generovat bez xsl:output nebo použít !DOCTYPE pro XHTML

- generování stránky ve WML:

```
<xsl:output method="xml" encoding="us-ascii" indent="yes"
            doctype-public="-//WAPFORUM//DTD WML 1.1//EN"
            doctype-system="http://www.wapforum.org/DTD/wml_1.1.xml"/>
```

Výstup do více souborů

- výstup generovaný uvnitř instrukce `xsl:result-document` se zapisuje do samostatného souboru
- jméno souboru se určuje v atributu `href`
- v atributu `href` je možné používat AVT

Příklad 12. `katalog-soubory.xsl`

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                version="2.0">

  <xsl:output method="html" encoding="utf-8"/>

  <!-- Do hlavního souboru se uloží úvodní stránka s obsahem -->
  <xsl:template match="/">
    <xsl:result-document href="katalog.html">
      <html>
        <head>
          <title>Katalog <xsl:value-of ▶
select="katalog/info/firma"/></title>
        </head>
        <body>
          <h1>Katalog
              <xsl:value-of select="katalog/info/firma"/></h1>

          <!-- Vygenerování tabulky s obsahem -->
          <table border="1">
            <tr>
              <th>Název</th>
              <th>Cena</th>
            </tr>
            <xsl:for-each select="katalog/polozka">
              <tr>
                <td>
                  <a href="{generate-id()}.html">
                    <xsl:value-of select="nazev"/>
                  </a>
                </td>
                <td>
                  <xsl:value-of select="cena"/>
                </td>
              </tr>
```

Výstup do více souborů (Pokračování)

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

        <!-- Zpracování všech položek pomocí dalších šablon -->
        <xsl:apply-templates select="katalog/polozka"/>
      </body>
    </html>
  </xsl:result-document>
</xsl:template>

<!-- Každá položka se uloží do samostatného souboru -->
<xsl:template match="polozka">
  <xsl:result-document href="{generate-id()}.html">
    <html>
      <head>
        <title><xsl:value-of select="nazev"/></title>
      </head>
      <body>
        <xsl:apply-templates/>
        <p align="center"><a href="katalog.html">Zpět</a></p>
      </body>
    </html>
  </xsl:result-document>
</xsl:template>

<xsl:template match="nazev">
  <h2><xsl:apply-templates/></h2>
</xsl:template>

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

<xsl:template match="cena">
  <strong> - <xsl:apply-templates/> Kč</strong>
</xsl:template>

<xsl:template match="popis">
  <ul>
    <xsl:for-each-group select="node()" group-starting-with="br">
      <li>
        <xsl:apply-templates select="current-group()"/>
```

Výstup do více souborů (Pokračování)

```
        </li>
      </xsl:for-each-group>
    </ul>
  </xsl:template>

  <xsl:template match="b">
    <font color="red"><xsl:apply-templates/></font>
  </xsl:template>

</xsl:stylesheet>
```

Práce se jmennými prostory

- používají-li vstupní dokumenty jmenné prostory, nesmíme zapomenout na prefixy v XPath výrazech
- pomocí atributu `xpath-default-namespace` můžeme určit jmenný prostor pro elementy ve výrazech XPath, které nemají prefix
  - lze tak podstatně zkrátit výrazy, zvláště pokud zpracovávaný dokument má elementy pouze v jednom jmenném prostoru
  - platnost atributu je na elementu, kde je uveden, a všech jeho potomcích

Příklad 13. **`atom2html.xsl`**

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                xmlns:atom="http://www.w3.org/2005/Atom"
                exclude-result-prefixes="atom"
                version="2.0">

<xsl:output method="html" encoding="utf-8"/>

<xsl:template match="/">
  <html>
    <head>
      <title><xsl:value-of select="/atom:feed/atom:title"/></title>
    </head>
    <body>
      <h1><xsl:value-of select="/atom:feed/atom:title"/></h1>

      <xsl:for-each select="/atom:feed/atom:entry">
        <a href="{atom:link[not(@rel) or @rel='alternate'][1]/@href}">
          <xsl:value-of select="atom:title"/>
        </a>
        <br/>
      </xsl:for-each>
    </body>
  </html>
</xsl:template>

</xsl:stylesheet>
```

Ukázka výchozího jmenného prostoru pro XPath

Příklad 14. **atom2html-vychozi-ns.xsl**

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                xpath-default-namespace="http://www.w3.org/2005/Atom"
                version="2.0">

<xsl:output method="html" encoding="utf-8"/>

<xsl:template match="/">
  <html>
    <head>
      <title><xsl:value-of select="/feed/title"/></title>
    </head>
    <body>
      <h1><xsl:value-of select="/feed/title"/></h1>

      <xsl:for-each select="/feed/entry">
        <a href="{link[not(@rel) or @rel='alternate'][1]/@href}">
          <xsl:value-of select="title"/>
        </a>
        <br/>
      </xsl:for-each>
    </body>
  </html>
</xsl:template>

</xsl:stylesheet>
```

Formátování čísel

- čísla lze na výstupu formátovat pomocí funkce `format-number(`*`číslo`*`,` *`formátovací řetězec`*`)`
- struktura formátovacího řetězce

  [*prefix*]*celá část*[*desetinná část*][*suffix*]
  [;[*prefix*]*celá část*[*desetinná část*][*suffix*]]

- pokud chceme používat např. desetinou čárku místo tečky musíme si vytvořit vlastní desetinný formát pomocí instrukce `xsl:decimal-format`
- význam znaků ve formátovacím řetězci

Znak	**Význam**
0	číslice
#	číslice – pokud je 0, nezobrazuje se
.	znak oddělující specifikaci pro celou a desetinnou část čísla
,	znak vyznačující pozici oddělovače řádu (nejčastěji tisíců)
;	oddělovač specifikace pro kladná a záporná čísla
-	znaménko
%	číslo se vynásobí 100 a zobrazí jako procenta
‰	číslo se vynásobí 1000 a zobrazí jako promile (pro zápis můžeme použít znakovou entitu `&#x2030;`)
'*speciální znak*'	uzavření znaku mezi apostrofy vypne jeho speciální význam, pokud je použit v prefixu nebo suffixu

Příklad 15. Ukázka formátování čísla 123456789.87654

```
Formátovací řetězec          Výstup
#.#                          123456789.9
#.###                        123456789.877
#                            123456790
#,###.###                    123,456,789.877
$#,###.###                   $123,456,789.877
#,###.## USD                 123,456,789.88 USD
'#'#,###.##                  #123,456,789.88
```

Formátování čísel (Pokračování)

Příklad 16. Ukázka formátovaní čísla -357.4

Formátovací řetězec	Výstup
#.#	-357.4
#.00	-357.40
#.#;(#.#)	(357.4)

Formátování data a času

- k dispozici jsou tři nové funkce – `format-date()`, `format-dateTime()` a `format-time()`
- funkcím můžeme předat například aktuální datum/čas – `current-date()`, `current-dateTime()` a `current-time()`
- nebo můžeme předat jakékoliv jiné datum, ale musí mít správný typ – např. `xs:date('2008-12-04')`
- popis formátovacího řetězce[13]

Příklad 17. **formatovani-data-a-casu.xsl**

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                version="2.0"
                xmlns:xs="http://www.w3.org/2001/XMLSchema">

<xsl:output method="text"/>

<xsl:template match="/">
  <xsl:value-of select="format-date(current-date(), '[D]. [M]. [Y]')"/>
  <xsl:text>&#10;</xsl:text>
  <xsl:value-of select="format-date(current-date(), '[D,2]/[M,2]/[Y,4]')"/>
  <xsl:text>&#10;</xsl:text>
  <xsl:value-of select="format-date(current-date(), '[F], [MNn] [D1o], ▶
[Y]')"/>
  <xsl:text>&#10;</xsl:text>
  <xsl:value-of select="format-date(xs:date('2008-12-04'), '[DWwo] [MNn] ▶
[Y]', 'de', 'AD', 'GE')"/>
  <xsl:text>&#10;</xsl:text>
  <xsl:value-of select="format-date(current-date(), '[D]. [M]. [Y] [E]')"/>
  <xsl:text>&#10;</xsl:text>
  <xsl:value-of select="current-time()"/>
  <xsl:text>&#10;</xsl:text>
  <xsl:value-of select="format-time(current-time(), '[H]:[m]:[s]')"/>
  <xsl:text>&#10;</xsl:text>
  <xsl:value-of select="format-time(current-time(), '[h]:[m]:[s] [P]')"/>
</xsl:template>

</xsl:stylesheet>
```

[13] http://www.w3.org/TR/xslt20/#date-picture-string

Regulární výrazy

- funkce XPathu
  - `matches(`*`řetězec`*`, `*`vzor`*`)` – testování shody řetězce se vzorem
  - `replace(`*`řetězec`*`, `*`vzor`*`, `*`náhrada`*`)` – nahrazení vzoru v řetězci
  - `tokenize(`*`řetězec`*`, `*`vzor`*`)` – rozdělení řetězce podle vzoru
- instrukce XSLT
  - `xsl:analyze-string`
- syntaxe regulárních výrazů je velice podobná např. regulárním výrazům ve W3C XML Schema nebo Perlu

Parametry a funkce

Předávání parametrů

- jednotlivé šablony i celý styl mohou mít deklarovány parametry
- při volání šablony nebo stylu lze parametrům předávat hodnoty
- parametry lze předávat přes příkazovou řádku, API, funkci XSLT IDE, …

Příklad 18. **katalog-parametry.xsl**

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                version="2.0">

<!-- Podle jakého elementu se má výsledek seřadit -->
<xsl:param name="raditPodle">cena</xsl:param>

<!-- Datový typ položky, podle které se řadí -->
<xsl:param name="typ">number</xsl:param>

<!-- Směr řazení dat (vzestupně/sestupně) -->
<xsl:param name="smer">ascending</xsl:param>

<xsl:template match="/">
  <html>
    <head>
      <title><xsl:value-of select="/katalog/nazev"/></title>
      <style type="text/css">
        body { color: black; background-color: white; }
        table { border: solid 2px black; border-collapse: collapse; }
        th a { text-decoration: none; color: black; }
        th   { background-color: #FF8000; }
        th, td { border: solid 1px black; }
      </style>
    </head>
    <body>
      <h1><xsl:value-of select="/katalog/nazev"/></h1>

      <table width="100%">
        <tr>
          <th>Název</th>
          <th>Kategorie</th>
          <th>Cena</th>
        </tr>
        <!-- Před zpracováním jsou položky seřazeny
```

Předávání parametrů (Pokračování)

```
            pole vybraných kritétií -->
        <xsl:apply-templates select="//polozka">
          <xsl:sort select="*[local-name() = $raditPodle]"
                    data-type="{$typ}" order="{$smer}"/>
        </xsl:apply-templates>
      </table>
    </body>
  </html>
</xsl:template>

<xsl:template match="polozka">
  <tr>
    <xsl:apply-templates select="nazev"/>
    <xsl:apply-templates select="kategorie"/>
    <xsl:apply-templates select="cena"/>
  </tr>
</xsl:template>

<xsl:template match="nazev">
  <td align="center"><xsl:apply-templates/></td>
</xsl:template>

<xsl:template match="kategorie">
  <td align="center"><xsl:apply-templates/></td>
</xsl:template>

<xsl:template match="cena">
  <td align="right"><xsl:apply-templates/> Kč</td>
</xsl:template>

</xsl:stylesheet>
```

Pojmenované šablony

- šablonu lze pojmenovat
- lze ji pak volat pomocí instrukce `xsl:call-template`
- umožňuje lepší modularizaci kódu

Příklad 19. `pojmenovana-sablona.xsl`

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                version="2.0">

<!-- Pojmenovaná šablona se dvěma parametry -->
<xsl:template name="kontakt">
  <xsl:param name="jmeno"/>
  <xsl:param name="email"/>

  <xsl:value-of select="$jmeno"/>
  <xsl:if test="$email != ''">
    <xsl:text> (e-mail: </xsl:text>
    <a href="mailto:{$email}"><xsl:value-of select="$email"/></a>
    <xsl:text>)</xsl:text>
  </xsl:if>
</xsl:template>

<xsl:template match="/">
  <html>
    <head>
      <title>Kontakty</title>
    </head>
    <body>
      <h1>Kontakty na vedení firmy</h1>

      <p>Ředitel:
        <xsl:call-template name="kontakt">
          <xsl:with-param name="jmeno">Jan Novák</xsl:with-param>
         <xsl:with-param name="email">novak@example.org</xsl:with-param>
        </xsl:call-template>
      </p>

      <p>Skladník:
        <xsl:call-template name="kontakt">
          <xsl:with-param name="jmeno">Josef Procházka</xsl:with-param>
        </xsl:call-template>
```

Pojmenované šablony (Pokračování)

```
      </p>
    </body>
  </html>
</xsl:template>

</xsl:stylesheet>
```

Proměnné

- proměnné fungují podobně jako parametry, jen nelze měnit jejich hodnotu při volání šablony/stylu
- mohou být definovány globálně pro celý styl nebo lokálně kdekoliv uvnitř šablony
- `<xsl:variable name="`*jméno*`" select="`*hodnota*`"/>`

```
<xsl:variable name="jméno">
  … nastavení obsahu proměnné …
</xsl:variable>
```

- hodnotu proměnné nelze po přiřazení měnit
- používá se pro uložení často používaných hodnot a pro zpřehlednění kódu

Uživatelsky definované funkce

- pohodlnější náhrada pojmenovaných šablon
- nově definovaná funkce je dostupná přímo v XPathu, nemusí se volat pomocí `xsl:call-template`
- na rozdíl od `xsl:call-template` nedostává funkce kontext, v případě potřeby je nutné jej předat jako explicitní parametr
- pomocí `as` lze určit typy parametrů a výsledku
- uvnitř těla funkce lze používat `xsl:sequence` pro generování výsledku
- jméno funkce musí být v našem vlastním jmenném prostoru, aby nekolidovalo se zabudovanými funkcemi

Příklad 20. **`bmi.xsl`**

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                version="2.0"
                xmlns:f="http://example.com/functions"
                xmlns:xs="http://www.w3.org/2001/XMLSchema">

<!-- Funce pro výpočet BMI -->
<xsl:function name="f:bmi" as="xs:double">
  <xsl:param name="hmotnost" as="xs:double"/>
  <xsl:param name="vyska" as="xs:double"/>

  <xsl:sequence select="$hmotnost div ($vyska * $vyska)"/>
</xsl:function>

<!-- Vlastní formát čísel podle českých zvyklostí -->
<xsl:decimal-format decimal-separator="," grouping-separator="."/>

<xsl:template match="/">
  <html>
    <head>
      <title>Tabulka BMI</title>
    </head>
    <body>
      <table border="1">
        <tr>
          <th>Jméno</th>
          <th>BMI</th>
        </tr>
        <xsl:for-each select="/adresář/osoba">
```

Uživatelsky definované funkce (Pokračování)

```
          <tr>
            <td><xsl:value-of select="jméno"/></td>

            <!-- Volání funkce v XPathu -->
            <td><xsl:value-of select="format-number(
                                      f:bmi(váha, výška div 100),
                                      '#,##')"/></td>
          </tr>
        </xsl:for-each>
      </table>
    </body>
  </html>
</xsl:template>

</xsl:stylesheet>
```

Určování typů parametrů

- u proměnných, parametrů a funkcí lze určit jejich typ pomocí atributu `as`
- určení typu vede k robustnějším transformacím, některé chyby mohou být odhaleny již při statické analýze kódu před spuštěním transformace
- deskriptory typu posloupnosti:
  - typ XML schémat, např. `xs:date`, `xs:integer`, …
  - obdoba testu uzlu – `node()`, `element()`, `attribute()`, `text()`
  - jakýkoliv prvek posloupnosti – `item()`
  - předchozí deskriptory lze kombinovat s příznaky opakování – `?`, `*` a `+`
  - prázdná posloupnost – `empty-sequence()`

Závěr

Na co nezbyl čas

- klíče a `xsl:key` – důležité při zpracování velkých dokumentů, kde se provádí mnoho dotazů
- kopírování uzlů pomocí `xsl:copy` a `xsl:copy-of`
- vytvoření seřazené posloupnosti pomocí `xsl:perform-sort`
- výpis hlášení pomocí `xsl:message`
- modularizace schémat pomocí `xsl:include`, `xsl:import`, `xsl:apply-imports` a `xsl:next-match`
- práce s typy a Schema-Aware transformace

Další zdroje informací

- Michael Kay: XSLT 2.0 and XPath 2.0 Programmer's Reference, 4th Edition – kniha od autora Saxonu a editora specifikace XSLT 2.0; výborná reference
- Jeni Tennison: Beginning XSLT 2.0: From Novice to Professional – dobrá hlavně pro začátečníky
- diskusní skupina xsl-list[14]
- dokumentace k implementacím XSLT 2.0
- články na serverech xml.com a IBM developerWorks
- specifikace W3C
- Kernow[15] – šikovná utilitka pro testování XSLT a pohodlné spouštění Saxonu

[14] http://www.mulberrytech.com/xsl/xsl-list/
[15] http://kernowforsaxon.sourceforge.net/