Transformace XML dat

Obsah

Jazyk XSLT

Souvislosti, historie

- XSLT (eXtensible Stylesheet Language Transformation) [http://w3.org/style/XSL] je jazyk pro specifikaci *transformací XML dokumentů* na (obvykle) XML výstupy, případně *textové*, *HTML* či jiné výstupní formáty.
- Původní aplikační oblastí byla transformace XML dat na XSL:FO (formátovací objekty), tedy *vizualizace XML*.
- XSLT byl tedy součástí specifikací XSL [http://w3.org/style/XSL] (eXtensible Stylesheet Language).
- Později se z XSL vyčlenil a začal být chápán jako univerzální jazyk popisu obecných XML->XML(txt, HTML) transformací.
- Aktuální verze je dána specifikací XSLT 1.0 [http://www.w3.org/TR/xslt].

  Práce na verzi 1.1 byly zastaveny ve prospěch vývoje XSLT 2.0 [http://www.w3.org/TR/xslt20].

Hlavní principy

- XSLT je *funkcionálním jazykem*, kde *redukční pravidla* mají podobu *šablon*, které předepisují, jak se *uzly* zdrojového dokumentu přepisují do výstupního dokumentu.
- Specifikace XSLT transformace je obsažena v tzv. *stylu (stylesheet)*, což je XML dokument tvořený podle syntaxe XSLT. Kořenovým elementem je stylesheet nebo transformation (to jsou synonyma).
- XSLT styl obsahuje tzv. *šablony (template)*.

Hlavní principy (2)

- Šablony mají *výběrovou část* - která reprezentuje levou stranu funkcionálního redukčního pravidla a *konstrukční část* představující pravou stranu red. prav.
- Výběrovou část tvoří atribut match šablony.

  Konstrukční část představuje tělo elementu šablony.
- Vlastní transformace pak znamená, že interpreter XSLT stylů (*XSLT procesor*, *XSLT engine*) bere uzly vstupního dokumentu, vyhledá k nim vhodnou šablonu - podle její výběrové části - a vyprodukuje výsledek odpovídající konstrukční části pravidla daného touto šablonou.

Hlavní informační zdroje - specifikace, reference, tutoriály, FAQ

- XSLT 1.0 W3C Recommendation: http://www.w3.org/TR/xslt

- *What is XSLT?* na XML.COM: http://www.xml.com/pub/a/2000/08/holman/index.html
- Mulberrytech.com XSLT Quick Reference (2xA4, PDF): http://www.mulberrytech.com/quickref/XSLTquickref.pdf
- Dr. Pawson XSLT FAQ: http://www.dpawson.co.uk/xsl/xslfaq.html
- Zvon XSLT Tutorial: http://zvon.org/xxl/XSLTutorial/Books/Book1/index.html

Syntaxe XSLT

Struktura celého XSLT stylu

Kořenový element xsl:transform nebo xsl:stylesheet uzavírá celý XSLT styl a specifikuje NS prefix pro XSLT elementy.

V kořenovém elementu je:

- Deklarace *parametrů* (a jejich implic. hodnoty) - elt. xsl:param . Parametry lze nastavit při volání XSLT procesoru - např. **java net.sf.saxon.Transform -o outfile.xml infile.xml style.xsl -Dparam=paramvalue [http://www.google.com/search?q=java net.sf.saxon.Transform -o outfile.xml infile.xml style.xsl -Dparam=paramvalue]** [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=java net.sf.saxon.Transform -o outfile.xml infile.xml style.xsl -Dparam=paramvalue]
- Deklarace a inicializace *proměnných* - elt. xsl:variable - proměnné jsou de facto totéž, co parametry, ale nejsou nastavitelné zvenčí.
- Je třeba si uvědomit, že XSLT (bez procesorově-specifických rozšíření) je čistý funkcionální jazyk, tj. aplikace šablony nemá vedlejší efekt -> proměnné lze přiřadit jednou, pak už jen číst!

Struktura celého XSLT stylu (2)

V kořenovém elementu je dále:

- Deklarace (formátu) výstupu - elt. xsl:output
- ...kromě toho tam mohou být další, méně používané XSL elementy - viz např. dokumentace SAXONu [http://saxon.sf.net]
- pak následují vlastní šablony - elt. xsl:template

XSLT šablony

- Šablona (*template*) je specifikace *který uzel přepsat* a *na co (jak)*.
- Který uzel se přepisuje, je dáno *atributem* match .
- Na co se přepisuje, je uvedeno v *těle šablony*.
- Šablona může být explicitně *pojmenovaná* (*named template*), v tom případě ji lze volat přímo/explicitně pomocí xsl:call-template.

Sémantika XSLT

XSLT - postup zpracování vstupního dokumentu

- Nejdříve se za aktuální uzel zvolí kořen, tj. uzel odpovídající XPath výrazu /
- Najde se šablona (*explicitní* nebo *implicitní* - viz např. XSLT/XPath Quick Reference [http://www.mulberrytech.com/quickref/XSLTquickref.pdf]), jejíž match atribut chápaný jako XPath predikát vrátí v kontextu aktuálního uzlu true (tedy tzn. "matchuje" aktuální uzel).
- Pokud je jich více - nastává *nejednoznačnost* - pak je indikována chyba.
- Pokud je taková šablona právě jedna, aplikuje se, což znamenápřenesení jejího obsahu do výstupního *result tree fragmentu.*

XSLT - pořadí volání šablon

Je možné je specifikovat:

Přímo/explicitně — voláním (pojmenované) šablony - což ale odpovídá spíše *přístupu procedurálních jazyků*, takže se tomu spíše vyhýbáme.

Nepřímo/implicitně — tím, že se zavolá šablona, jejíž vzor (obsah atr. match) "pasuje" ("matchuje") na vybraný uzel - **funkcionální přístup**.

Výběr uzlu se přitom děje opět

- Explicitně ("řízeně") uvedením atributu select u apply-templates . Takto můžeme vybrat jak dceřinné elementy, tak dceřinné uzly, tak jakékoli jiné uzly odpovídající XPath výrazu uvedenému v select .
- Implicitně, necháme-li procesor sám "si uzel vybrat" (u apply-templates neuvádíme select). V tomto případě se ale vybírají pouze *dceřinné elementy* kontextového uzlu.

XSLT - specifikace výstupu/"výsledku" šablony

- Výstupem aplikace šablony je část tzv. *result tree fragmentu*.
- Výstupy jednotlivých šablon se "skládají" na to místo *result tree fragmentu*, který odpovídá pořadí volání šablon.
- Výstup celé transformace pak směřuje standardně do jednoho proudu, kde se z výstupního proudu událostí generuje výsledný (XML, text, HTML) dokument.
- Výstup bývá procesorem primárně generován jako sled událostí (např. SAX2), které jsou až druhotně převáděny na výsledný dokument - s uplatněním výstupního kódování, atd.

XSLT - výstup textových uzlů

Jak dostat text (textový uzel) na výstup?

1. Vepsat text přímo (jako literál) do výstupu (konstrukční části) šablony. Pozor na bílé znaky (mezery, CR/LF)!

2. vepsat text přímo (jako literál) do výstupu šablony. Pozor na bílé znaky (mezery, CR/LF)!

3. do speciálního elt. <xsl:text>textový uzel</xsl:text> . Bílé znaky jsou v něm vždy zachovány/ respektovány!

Implicitní šablony

Implicitní šablony jsou "vestavěné" v každém korektním procesoru XSLT:

- aby byly (alespoň jistým standardním "fallback" způsobem) zpracovány základní struktury (procházení stromu dokumentu)
- abychom "ušetřili psaní" často používaných šablon (ignorování komentářů a PI).
- Jsou překrytelné, abychom mohli chování změnit uvedením *vlastní šablony*, která bude mít stejnou (nebo překrývající se) klauzuli match= .

Přehled implicitních šablon

- "Default tree (do-nothing) traversal":

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

- "Default tree (do-nothing) traversal for specified mode":

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

Přehled implicitních šablon (2)

- "Copy text nodes and attributes" (do výsledku zkopíruje textové uzly a atributy):

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

- "Ignore PIs and comments" ignoruje (nezahrnuje do výsledku PI a komentáře):

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

Vybrané XSLT konstrukce podrobněji

Generování pevně daného elementu s atributy

Cíl: Vygenerovat na výstup předem daný element (s předem známým jménem), ale s atributy s hodnotami kalkulovanými při transformaci.

Řešení: Použít normální postup - literal result element - a hodnoty atributy specifikovat jako tzv. *attribute value templates (AVT)*:

Vstup:

```
<link ref="odkaz_dovnitr_dok">
   ...
</link>
```

Šablona:

```
<xsl:template match="link">
   <a href="#{@ref}"> ... </a>
</xsl:template>
```

Transformuje odkaz link na a , hodnotu atributu href spočte tak, že před hodnotu původního atributu ref přidá znak #

Generování elementu s kalkulovaným názvem i atributy

Cíl: Vygenerovat na výstup element, jehož název, atributy i obsah předem - při psaní stylu - neznáme.

Řešení: Použít do konstrukční části šablony `xsl:element`:

Vstup:

```
<generate element="elt_name"> ... </generate>
```

Šablona:

```
<xsl:template match="generate">
   <xsl:element name="{@element}">
      <xsl:attribute name="id">ID1</xsl:attribute>
   </xsl:element>
</xsl:template>
```

Vytvoří element s názvem `elt_name`, opatří jej atributem `id="ID1"` .

Řízení chodu transformace uvnitř šablony - větvení

Cíl: Větvit generování výstupu na základě podmínky.

Řešení: Použít do konstrukční části šablony větvení - jednoduché xsl:if nebo vícecestné xsl:choose / xsl:when / xsl:otherwise :

Vstup:

```
<rohlik cena="5"> ... </rohlik>
```

Šablona:

```
<xsl:template match="rohlik">
   <p>
      <xsl:if test="cena>2">
         <span class="expensive">Drahý</span>
      </xsl:if> rohlík - cena <xsl:value-of select="@cena"/> Kč </p>
</xsl:template>
```

Vytvoří element p , do něj vloží info o rohlíku - se zvýrazněním, je-li drahý.

Řízení chodu transformace uvnitř šablony - vícecestné větvení

Vstup:

```
<rohlik cena="5"> ... </rohlik>
<rohlik cena="2"> ... </rohlik>
<rohlik cena="0.9"> ... </rohlik>
```

Šablona:

```
<xsl:template match="rohlik">
   <p>
      <xsl:when test="cena>2">
         <span class="expensive">Drahý</span>
      </xsl:when>
      <xsl:when test="cena<1">
         <span class="strangely-cheap">Podezřele levný</span>
      </xsl:when>
      <xsl:otherwise>
         <span class="normal-price">Běžný</span>
      </xsl:otherwise> rohlík - cena <xsl:value-of select="@cena"/> Kč </p>
</xsl:template>
```

Odfiltruje dvě extrémní úrovně ceny - pro xsl:otherwise zůstane „normální“ cena.

Řízení chodu transformace uvnitř šablony - cykly

Cíl: Větvit generování výstupu na základě podmínky.

Řešení: Použít do konstrukční části šablony větvení - jednoduché xsl:if nebo vícecestné xsl:choose / xsl:when / xsl:otherwise :

Vstup:

```
<pecivo>
   <rohlik cena="5"> ... </rohlik>
   <rohlik cena="2"> ... </rohlik>
   <rohlik cena="0.9"> ... </rohlik>
</pecivo>
```

Šablona:

```
<xsl:template match="pecivo">
   <xsl:for-each select="rohlik">
      <p>Rohlík - cena <xsl:value-of select="@cena"/> Kč</p>
   </xsl:for-each>
</xsl:template>
```

Vytvoří element p , do něj vloží info o rohlíku - se zvýrazněním, je-li drahý.

Pozor: Konstrukce xsl:for-each má typicky procedurální charakter, je dobré s ní šetřit. Dává totiž minumum flexibility na obsah iterované množiny uzlů - tj. *předem musím vědět, co tam bude*.

Procesory XSLT (XSLT Transformation Engines)

Populárními volně dostupnými procesory XSLT v javovém prostředí jsou:

- SAXON (autor M.H.Kay) [http://saxon.sf.net]
- XALAN (autor Apache Software Foundation) [http://xml.apache.org/xalan-j/index.html]
- ... další free i komerční procesory k nalezení na XML Software [http://www.xmlsoftware.com/xslt.html]

Pokročilá témata

Režimy (módy) zpracování

Motivace: Módy umožňují mít paralelně sadu šablon se stejnými vzory match , používaných ale pro různé účely, např.:

- jedna sada pro generování obsahu (*index*) dokumentu
- druhá pro formátování plného textu dokumentu

Při explicitním vyvolání aplikace šablon (`apply-templates`) lze uvést mód (atributem `mode=`):

- uvede-li se, aplikují se pouze šablony se stejným módem, jaký byl uveden v `xsl:apply-templates mode="mód"` .
- neuvede-li se, aplikují se pouze šablony *bez* specifikace módu (bez atributu `mode=`).

Deklarace a volání pojmenovaných šablon

Deklarace - `xsl:template name="jmeno_sablony"`

Šablona smí obsahovat deklarace parametrů:

- `<xsl:param name="jmenoParametru"/>`

Volání - `<xsl:call-template name="jmenoSablony">`

volání smí specifikovat parametry:

- `<xsl:with-param name="jmenoParametru" select="hodnotaParametru"/>` nebo
- `<xsl:with-param name="jmenoParametru">hodnota parametru</xsl:with-param>`

Automatické (generované) číslování

Vložíme-li do konstrukční části šablony (do těla šablony) element xsl:number , zajistí nám vygenerování čísla daného čítačem.

Je možné uvést, podle čeho se má číslovat, např.:

- pořadového čísla zdrojového elementu v rámci jeho rodičovského elementu

  - a to i víceúrovňově, např. číslo kapitoly 1.1. apod.

Příklad 1. Automatické číslování podle pozice elementu

Aplikujeme-li tento styl

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                version="1.0">
   <xsl:template match="/">
      <html>
         <body>
            <xsl:for-each select="devguru_staff/programmer">
               <xsl:number value="position()" format="1. " />
               <xsl:value-of select="name" />
               <br/>
            </xsl:for-each>
         </body>
      </html>
   </xsl:template>
</xsl:stylesheet>
```

na následující zdrojový soubor

```
<devguru_staff>
    <programmer>
        <name>Bugs Bunny</name>
        <dob>03/21/1970</dob>
        <age>31</age>
        <address>4895 Wabbit Hole Road</address>
        <phone>865-111-1111</phone>
    </programmer>
    <programmer>
        <name>Daisy Duck</name>
        <dob>08/09/1949</dob>
        <age>51</age>
        <address>748 Golden Pond</address>
        <phone>865-222-2222</phone>
    </programmer>
    <programmer>
        <name>Minnie Mouse</name>
        <dob>04/13/1977</dob>
        <age>24</age>
        <address>4064 Cheese Factory Blvd</address>
        <phone>865-333-3333</phone>
    </programmer>
</devguru_staff>
```

dostaneme výslednou HTML stránku (nebrat v úvahu odsazení - to bude jiné...)

```
<html>
    <body>1. Bugs Bunny<br>
          2. Daisy Duck<br>
          3. Minnie Mouse<br>
    </body>
</html>
```

Automatické číslování (2)

Příklad 2. Automatické víceúrovňové číslování

Aplikujeme-li tento styl

```
<xsl:stylesheet
        xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
        version="1.0">
    <xsl:template match="/book">
        <html>
            <body>
                <xsl:for-each select="chapter">
                    <h2>
                        <xsl:number count="chapter" format="1. "/>
                        <xsl:value-of select="title" />
                    </h2>
                    <xsl:for-each select="sect1">
                        <h3>
                            <xsl:number count="chapter" format="1. "/>
                            <xsl:number count="sect1" format="a. "/>
                            <xsl:value-of select="title" />
                        </h3>
                        <xsl:apply-templates select="para"/>
                    </xsl:for-each>
                </xsl:for-each>
            </body>
        </html>
    </xsl:template>
</xsl:stylesheet>
```

na následující zdrojový soubor

```
<book>
    <title>Moje nová kniha</title>
    <chapter>
        <title>První kapitola</title>
        <sect1>
            <title>První sekce první kapitoly</title>
            <para>Text</para>
        </sect1>
        <sect1>
            <title>Druhá sekce první kapitoly</title>
            <para>Text druhé sekce</para>
        </sect1>
    </chapter>
    <chapter>
        <title>Druhá kapitola</title>
        <sect1>
            <title>První sekce druhé kapitoly</title>
            <para>Text</para>
        </sect1>
        <sect1>
            <title>Druhá sekce druhé kapitoly</title>
            <para>Text druhé sekce</para>
        </sect1>
    </chapter>
</book>
```

dostaneme výslednou HTML stránku

Co používat raději?

- Preferovat funkcionální přístup - např. `xsl:template match=` a `xsl:apply-templates select=`
- před procedurálním přístupem - `xsl:template name=` a `xsl:call-template name=`

Používat módy zpracování (xsl:template ... mode= a xsl:apply-templates ... mode=)

módy lze dobře kombinovat s funkcionálním přístupem:

- xsl:apply-templates select=... mode=...
- xsl:template match=... mode=...

Znovupoužitelnost stylů

Co pro ni můžeme udělat?

- Členit styly do menších znovupoužitelných celků (souborů) a podle potřeby je vřazovat pomocí xsl:include a nebo, ještě lépe, xsl:import - protože import upřednostňuje šablony uvedené přímo v základním stylu nad šablonami importovanými.

Podrobněji viz příspěvek TP pro DATAKON 2001 - fulltext příspěvku [http://www.fi.muni.cz/~tomp/xml03/pitner.doc] a slidy [http://www.fi.muni.cz/~tomp/xml03/prezentace.ppt].

Návrhové vzory

Identická transformace 1 (nepřevede do výsledku atributy kořenového elementu!) http://wwbota.free.fr/XSLT_models/identquery.xslt

Identická transformace 2 http://wwbota.free.fr/XSLT_models/identquery2.xslt

Identická transformace s potlačením elementů, které nemají na ose // (v dceřinných uzlech ani jejich potomcích) žádné textové uzly http://wwbota.free.fr/XSLT_models/suppressEmptyElements.xslt

Nahradí atributy pomocí elementů http://wwbota.free.fr/XSLT_models/attributes2elements.xslt

Dtto, ale elementy vzniklé z atributů jsou ve zvláštním jmenném prostoru xslt/attributes2elements.xslt [http://www.fi.muni.cz/~tomp/xml03/xslt/attributes2elements.xslt]

Reverzní transformace xslt/elements2attributes.xslt [http://www.fi.muni.cz/~tomp/xml03/xslt/elements2attributes.xslt]

Odkazy na pokročilá témata

XSLT Design Patterns - výběr [http://www.dpawson.co.uk/xsl/sect1/N169.html]

The Functional Programming Language XSLT [http://www.topxml.com/xsl/articles/fp/1.asp]

Motivace a hlavní principy

Co jsou STX?

Proudové transformace pro XML (Streaming Transformations for XML [http://stx.sf.net], STX) jsou alternativním mechanizmem, jak specifikovat i prakticky vykonávat netriviální transformace XML zdrojů na XML a dalších formátů.

STX nabízejí transformační postup založený na jednoprůchodovém, tedy proudovém (=streaming), zpracování XML vstupu bez nutnosti budovat stromovou podobu zdrojového dokumentu, což:

- snižuje paměťové nároky a umožňuje zpracovat širší škálu (velkých) zdrojových dokumentů/dat;
- je potenciálně rychlejší (ve stávajících implementacích zatím bohužel ne - chybí optimalizace)

Proudové zpracování na bázi SAX

- SAX (Simple API for XML Processing [http://saxproject.org]) určuje proudové rozhraní pro zpracování XML;
- proudové transformace XML bychom tedy mohli psát jako tzv. SAX filtry -
- tento postup je ale programátorsky náročný a nepohodlný:
  - je to hodně nízkoúrovňová záležitost,
  - ve filtru je třeba explicitně udržovat stav zpracování,
  - vytvářet pomocné datové struktury... atd.

Proudové zpracování na bázi SAX - příklad řetězce filtrů

Proudové zpracování na bázi SAX (2)

- udržovatelnost a rozšiřitelnost netriviálních filtrů je obtížná, proto
- vznikaly různé "berličky" jako Programmable SAX Filter [http://raritantechnologies.com/sax.shtml] (open-source, Raritan Technologies);
- všechny měly společný imperativní přístup ke specifikaci transformace.
- STX nabízí možnost popsat transformace deklarativně, podobně jako v jazyce XSLT, na nějž jsou programátoři již zvyklí,
- STX se také nazývá "SAX se syntaxí XSLT".

Vztah STX a SAX

- STX popisují proudové transformace,
- SAX (Simple API for XML Processing [http://saxproject.org]) určuje proudové rozhraní.

- STX jsou budovány jako nadstavba SAX a mohou být se SAX kombinovány,
- programovat (psát transformační styly) pro STX je méně náročné než programovat SAX (transformační) filtry.

Hlavní charakteristiky proudových transformací

Dobrým podrobnějším zdrojem info je http://www.informatik.hu-berlin.de/~obecker/Docs/EML2003/. [http://www.informatik.hu-berlin.de/~obecker/Docs/EML2003/]

Zpracování STX

Model zpracování

Charakteristika STX transformačního stylu a zpracování je podobná XSLT:

- styl je XML dokument obsahující šablony podobné jako XSLT
- šablony jsou voleny a aplikovány podle vzorů v klauzulích match
- v šablonách mohou být buďto:
  - "literal result elements", čili přímo napsaný text, elementy mimo jmenný prostor XSLT... nebo
  - instrukce XSLT
- namísto XPath jako pro XSLT je u STX používán jazyk *STXPath*.

Co je při zpracování dokumentu k dispozici

Při zpracování vstupního dokumentu šablonami máme k dispozici

u XSLT strom celého dokumentu a

u STX jen zásobník předchůdců.

Lze tedy přistupovat

u XSLT k libovolnému uzlu vstupního dokumentu a

u STX jen k aktuálnímu uzlu a jeho předchůdcům.

Není to omezující?

STX styl a zpracování - příklad

Příklad převzat z Extended SAX Filter Processing with STX by Oliver Becker [http://www.informatik.hu-berlin.de/~obecker/Docs/EML2003/script.html]

„Modify the input by adding a consecutive ID attribute to certain elements (footnotes that appear within chapters):“

```
<stx:transform xmlns:stx="http://stx.sourceforge.net/2002/ns"
               xmlns:ex="http://www.example.com/"
               version="1.0" pass-through="all">
  <stx:variable name="count" select="1" />
  <stx:template match="chapter//footnote">
    <stx:copy attributes="@*">
      <stx:attribute name="ex:id" select="$count" />
      <stx:assign name="count" select="$count + 1" />
      <stx:process-children />
    </stx:copy>
  </stx:template>
</stx:transform>
```

Jak transformaci spouštět z Javy

STX podporuje rozhraní JAXP, čili STX procesor lze nastavit jako transformační nástroj v JAXP a následně místo XSLT stylů používat STX styly:

```
// use Joost as transformation engine
System.setProperty("javax.xml.transform.TransformerFactory",
                   "net.sf.joost.trax.TransformerFactoryImpl");

// the remaining code is known area
TransformerFactory tFactory = TransformerFactory.newInstance();
SAXTransformerFactory saxTFactory = (SAXTransformerFactory) tFactory;

// of course the transformation source must be different
TransformerHandler tHandler1 =
    saxTFactory.newTransformerHandler(new StreamSource("trans.stx"));
...
myReader.setContentHandler(tHandler1);
myReader.setProperty("http://xml.org/sax/properties/lexical-handler",
                     tHandler1);
tHandler1.setResult(new SAXResult(tHandler2));
...
```

Jazyk stylů STX

Zdroje

STX procesor(y)

Vzhledem k novosti standardu je počet implementujících procesorů dosud malý.

K dispozici jsou:

Joost	open-source javová implementace dostupná na http://stx.sourceforge.net/
XML::STX	open-source implementace pro čistý Perl dostupná na http://www.gingerall.org/charlie/ga/xml/p_stx.xml

Informace, dokumentace

Odkaz na většinu publikovaných článků i disertaci O. Beckera je na http://stx.sourceforge.net/.