

XSLT

Patryk Czarnik

Instytut Informatyki UW

XML i nowoczesne technologie zarządzania treścią – 2007/08

1 XSLT

- Budowa arkusza
- Wywoływanie szablonów
- Instrukcje sterujące
- Tworzenie wyniku
- Zmienne i parametry
- Szablony nazwane i funkcje
- Sortowanie i grupowanie

2 Tekst i serializacja

- Analiza tekstu
- Metody serializacji
- XSLT 1.0 a XSLT 2.0
- Sposoby przetwarzania

Plan

- 1 XSLT
 - Budowa arkusza
 - Wywoływanie szablonów
 - Instrukcje sterujące
 - Tworzenie wyniku
 - Zmienne i parametry
 - Szablony nazwane i funkcje
 - Sortowanie i grupowanie

- 2 Tekst i serializacja
 - Analiza tekstu
 - Metody serializacji
 - XSLT 1.0 a XSLT 2.0
 - Sposoby przetwarzania

XSLT – status

- Wersja 1.0
 - listopad 1999
 - powiązane z XPath 1.0
- Wersja 2.0
 - styczeń 2007
 - powiązane z XPath 2.0, XPath DM, XPath FO
- Powstał w ramach *Extensible Stylesheet Language*.
- Zastosowania wykraczają poza wizualizację XML.

Budowa arkusza

- **Arkusz** zbudowany z **szablonów**.
- Elementy (z prz. nazw) XSLT – deklaracje i **instrukcje**.
- **Konstruktory sekwencji** – fragmenty traktowane jako wyrażenia
 - ciała szablonów, funkcji, wartościowania zmiennych i parametrów,
 - tworzą drzewo wynikowe,
 - mogą zawierać instrukcje XSLT.

Struktura arkusza

- element główny

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

  <xsl:output method="html" encoding="iso-8859-2" />
  <xsl:import href="inny_arkusz.xml"/>
  <xsl:param name="css"/>

  <xsl:template match="/">
    <html>
      <head><link rel="stylesheet" type="text/css" href="{ $css }"/></head>
      <body><xsl:apply-templates/></body>
    </html>
  </xsl:template>

  <xsl:template match="akapit">
    <p><xsl:value-of select="."/></p>
  </xsl:template>
</xsl:stylesheet>
```

Struktura arkusza

- deklaracje, "konfiguracja"

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

  <xsl:output method="html" encoding="iso-8859-2" />
  <xsl:import href="inny_arkusz.xsl"/>
  <xsl:param name="css"/>

  <xsl:template match="/">
    <html>
      <head><link rel="stylesheet" type="text/css" href="{ $css }"/></head>
      <body><xsl:apply-templates/></body>
    </html>
  </xsl:template>

  <xsl:template match="akapit">
    <p><xsl:value-of select="."/></p>
  </xsl:template>
</xsl:stylesheet>
```

Struktura arkusza

- szablony

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

  <xsl:output method="html" encoding="iso-8859-2" />
  <xsl:import href="inny_arkusz.xsl"/>
  <xsl:param name="css"/>

  <xsl:template match="/">
    <html>
      <head><link rel="styeshheet" type="text/css" href="{ $css }"/></head>
      <body><xsl:apply-templates/></body>
    </html>
  </xsl:template>

  <xsl:template match="akapit">
    <p><xsl:value-of select="."/></p>
  </xsl:template>
</xsl:stylesheet>
```

Struktura arkusza

- konstruktory sekwencji

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

  <xsl:output method="html" encoding="iso-8859-2" />
  <xsl:import href="inny_arkusz.xsl"/>
  <xsl:param name="css"/>

  <xsl:template match="/">
    <html>
      <head><link rel="styeshheet" type="text/css" href="{ $css }"/></head>
      <body><xsl:apply-templates/></body>
    </html>
  </xsl:template>

  <xsl:template match="akapit">
    <p><xsl:value-of select="."/></p>
  </xsl:template>
</xsl:stylesheet>
```

Model przetwarzania

- Szablon – przekształcenie węzła we fragment drzewa.
- Start – szablon pasujący do korzenia.
- `apply-templates` – rekurencyjne wywołania szablonów.
- „Przechodzenie” drzewa także za pomocą `for-each`.

Dopasowywanie wzorców

```
<xsl:template match="osoby">
  <ul>
    <xsl:apply-templates select="osoba"/>
  </ul>
</xsl:template>
```

```
<xsl:template match="osoba">
  <li><xsl:apply-templates /></li>
</xsl:template>
```

- **Zawartość** `match` – **wzorzec**:
 - ograniczona postać ścieżek XPath,
 - osie tylko włąb (`child` i `attribute`).
- Dla każdego węzła z wyniku `select`:
 - dobierany szablon o najściślej podanym `match`,
 - chyba że ręcznie podano `priority`,
 - konflikt – błąd lub wybierany późniejszy szablon (zależne od implementacji).
- Brak `select` – `child::node()`.

Tryby przetwarzania (*modes*)

```
<xsl:template match="osoby">
  <table> ...
    <xsl:apply-templates select="osoba" mode="tabela"/>
  </table>
</xsl:template>
```

```
<xsl:template match="osoba">
  <li><xsl:apply-templates select="imię | nazwisko"/></li>
</xsl:template>
```

```
<xsl:template match="osoba" mode="tabela">
  <tr><td><xsl:value-of select="imię"/></td>
    <td><xsl:value-of select="nazwisko"/></td></tr>
</xsl:template>
```

Możliwe wartości atrybutu mode:

- nazwa,
- #default,
- #current – przy wywołaniu,
- #all – w szablonie.

Szablony wbudowane

```
<xsl:template match="element()|document()" mode="#all">  
  <xsl:param .../> ...
```

```
  <xsl:apply-templates select="child::node()" mode="#current">  
    <xsl:with-param .../> ...  
  </xsl:apply-templates>  
</xsl:template>
```

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

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

Instrukcja for-each

```
<xsl:template match="osoby">
  <ul>
    <xsl:for-each select="osoba">
      <li><xsl:value-of select="imię"/>
        <xsl:value-of select="nazwisko"/></li>
    </xsl:for-each>
  </ul>
</xsl:template>
```

Instrukcja `if`

```
<xsl:template match="rozdział">
  <xsl:if test="tytuł">
    <h2><xsl:value-of select="tytuł"/></h2>
  </xsl:if>
  <xsl:apply-templates select="node()[local-name() != 'tytuł']"/>
</xsl:template>
```

Instrukcja choose

```
<xsl:template match="osoby">
  <xsl:choose>
    <xsl:when test="$x > 0"> Większe </xsl:when>
    <xsl:when test="$x < 0"> Mniejsze </xsl:when>
    <xsl:otherwise> Chyba równe... </xsl:otherwise>
  </xsl:choose>
</xsl:template>
```

- tylko pierwsza gałąź z prawdziwym dozorem,
- otherwise opcjonalne.

Tworzenie węzłów wynikowych

- W konstruktorze sekwencji:
 - instrukcje tworzące węzły,
 - bezpośrednia zawartość.

Instrukcje tworzące węzły

```
<xsl:element name="elem">
  <xsl:attribute name="atryb">wartość atrybutu</xsl:attribute>
  <xsl:text>Zawartość tekstowa</xsl:text>
  <xsl:comment>To będzie komentarz</xsl:comment>
</xsl:element>
<xsl:processing-instruction target="xml-stylesheet"
  type="text/css" href="styl.css"</xsl:processing-instruction>
```

Bezpośrednia zawartość

```
<elem atryb="wartość atrybutu">
  Zawartość tekstowa <xsl:comment>To będzie komentarz</xsl:comment>
</elem>
```

Umieszczanie wyniku wyrażenia XPath

- Instrukcje XSLT `sequence`, `copy-of` i `value-of`.
- Wyrażenie XPath w atrybucie `select`.
- Do wyniku wstawiane:
 - `sequence` – wyliczona sekwencja,
 - `copy-of` – (głęboka) kopia sekwencji,
 - `value-of` – węzeł tekstowy z reprezentacją tekstową sekwencji.

value-of – szczegóły

- Źródło sekwencji:
 - wyrażenie XPath w atrybucie `select`,
 - lub konstruktor sekwencji w zawartości.
- Sekwencja poddana atomizacji.
- Wynikowy tekst:
 - rzutowanie każdego atomu na `string`,
 - rozdzielone separatorem podanym w atr. `separator`.

```
<x><xsl:value-of select="1 to 4" separator="|" /></x>
```

```
<x>1|2|3|4</x>
```

- W XSLT 1.0 (i trybie zgodności z 1.0) tylko wartość pierwszego węzła.

Umieszczanie wyniku wyrażenia XPath w atrybutach

- Nawiasy klamrowe w wartości atrybutu (jak w XQuery).
- Wstawiana reprezentacja tekstowa wyliczonej sekwencji.

```
<kwiatek nazwa="{@nazwa-kwiatka}"/>  
<xsl:element name="h{@poziom}"> ...</xsl:element>
```

Zmienne lokalne

- Zmienne deklarytywne – brak instrukcji przypisania.
- Cykliczne referencje zabronione.

Przykład

```
<xsl:template match="costam">
  <xsl:variable name="jaki_x">
    <xsl:choose>
      <xsl:when test="$x > 0">dodatni</xsl:when>
      <xsl:when test="$x = 0">równy zero</xsl:when>
      <xsl:when test="$x < 0">ujemny</xsl:when>
    </xsl:choose>
  </xsl:variable>

  ...<xsl:value-of select="$jaki_x"/>...
</xsl:template>
```

Zmienne i parametry – globalne

- Wartość **parametru** przekazana "z zewnątrz" podczas wywołania (można podać wartość domyślną).
- Zmienne wyliczane raz dla całego arkusza.

Przykład

```
<xsl:param name="nazwa"/>
<xsl:variable name="ile-elementow"
  select="count(//element()[name() = $nazwa)"/>

<xsl:variable name="tekst">
  <p>Dokument ma <xsl:value-of select="$ile-elementow"/>
    elementów.</p>
</xsl:variable>

<xsl:template match="/">
  ... <xsl:sequence select="$tekst"/> ...
</xsl:template>
```

Parametry szablonów

- W szablonie `param`.
- W wywołaniu `with-param`.

Przykład

```
<xsl:template match="pracownicy">
  <ul>
    <xsl:apply-templates select="osoba">
      <xsl:with-param name="prefix" select="'Pracownik: '"/>
    </xsl:apply-templates>
  </ul>
</xsl:template>

<xsl:template match="osoba">
  <xsl:param name="prefix"/>
  <li><xsl:value-of select="$prefix"/><xsl:apply-templates /></li>
</xsl:template>
```

Szablony nazwane

- W szablonie atrybut `name`.
- `call-template` uruchamia.
- Bez zmiany węzła bieżącego (inaczej niż `apply-templates`).
- Możliwa rekursja.

Przykład

```
<xsl:template name="opisz-element">
  <p>Element o nazwie <xsl:value-of select="name()" />.</p>
</xsl:template>
```

```
<xsl:template match="/">
  <html><body>
    <h1>Wszystkie elementy:</h1>
    <xsl:for-each select="//*">
      <xsl:call-template name="opisz-element"/>
    </xsl:for-each>
  </body></html>
</xsl:template>
```

Parametry i rekursja w szablonach nazwanych

- Pozwalają na "programowanie" w XSLT (nawet 1.0).

Silnia (z akumulatorem)

```
<xsl:template name="silnia">
  <xsl:param name="n"/>
  <xsl:param name="res" select="1"/>
  <xsl:choose>
    <xsl:when test="$n > 1">
      <xsl:call-template name="silnia">
        <xsl:with-param name="n" select="$n - 1"/>
        <xsl:with-param name="res"
          select="$n * $res"/>
      </xsl:call-template>
    </xsl:when>
    <xsl:otherwise> <xsl:value-of select="$res"/> </xsl:otherwise>
  </xsl:choose>
</xsl:template>
```

Definiowanie własnych funkcji

- Tylko w XSLT 2.0.

Silnia (bez akumulatora)

```
<xsl:function name="loc:silnia">
  <xsl:param name="n"/>
  <xsl:sequence select="if($n &lt;= 1)
    then 1
    else $n * loc:silnia($n - 1)"/>
</xsl:function>
```

Sortowanie podczas przetwarzania

- Instrukcja `sort` w `for-each`, `for-each-group` i `apply-templates`.
- Opcje sortowania w atrybutach:
 - `select` – klucz,
 - `data-type`, `lang` – rodzaj danych,
 - `order`, `case-order`, `stable`.

Prosty przykład

```
<xsl:template match="osoby">
  <ul>
    <xsl:apply-templates select="osoba">
      <xsl:sort select="nazwisko"/>
      <xsl:sort select="imię"/>
    </xsl:apply-templates>
  </ul>
</xsl:template>
```

Sortowanie sekwencji

- Instrukcja `perform-sort`.

Przykład z rekomendacji

```
<xsl:function name="bib:books-by-price" as="schema-element (bib:book) *">
  <xsl:param name="in" as="schema-element (bib:book) *"/>
  <xsl:perform-sort select="$in">
    <xsl:sort select="xs:decimal (bib:price)"/>
  </xsl:perform-sort>
</xsl:function>

...
<xsl:copy-of select="bib:books-by-price (//bib:book)
  [position() = 1 to 5]"/>
```

Grupowanie

Przykład z rekomendacji

```
<xsl:for-each-group select="cities/city" group-by="@country">
  <tr>
    <td><xsl:value-of select="position()" /></td>
    <td><xsl:value-of select="@country" /></td>
    <td>
      <xsl:value-of select="current-group() /@name" separator=", " />
    </td>
    <td><xsl:value-of select="sum(current-group() /@pop) " /></td>
  </tr>
</xsl:for-each-group>
```

Plan

- 1 XSLT
 - Budowa arkusza
 - Wywoływanie szablonów
 - Instrukcje sterujące
 - Tworzenie wyniku
 - Zmienne i parametry
 - Szablony nazwane i funkcje
 - Sortowanie i grupowanie
- 2 Tekst i serializacja
 - Analiza tekstu
 - Metody serializacji
 - XSLT 1.0 a XSLT 2.0
 - Sposoby przetwarzania

Analiza tekstu

- Instrukcja `analyze-string`.
- Wynik ustalany na podstawie dopasowania do wyrażenia regularnego.

Przykład z rekomendacji

```
<xsl:analyze-string select="body" regex="\[(.*)\]">
  <xsl:matching-substring>
    <cite><xsl:value-of select="regex-group(1)" /></cite>
  </xsl:matching-substring>
  <xsl:non-matching-substring>
    <xsl:value-of select="." />
  </xsl:non-matching-substring>
</xsl:analyze-string>
```

Serializacja wyniku

- Wynikiem przekształcenia drzewo XPath.
- Serializacja – zapisanie wyniku jako sekwencji bajtów.
- Metody serializacji:
 - XML,
 - HTML,
 - XHTML (tylko XSLT 2.0),
 - text.

Specyfikacja w arkuszu

```
<xsl:output method="html" encoding="iso-8859-2"/>
```

Dodatkowe parametry serializacji

- `encoding` – kodowanie znaków,
- `version` – wersja XML lub HTML,
- `doctype-public`, `doctype-system` – deklaracja DOCTYPE.

Specyfikacja w arkuszu

```
<xsl:output method="xhtml" version="1.0" encoding="utf-8"  
  doctype-public "-//W3C//DTD XHTML 1.1//EN"  
  doctype-system="http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd"/>
```

Zapisywanie dodatkowych plików

- Instrukcja `result-document`.
- Tworzy i zapisuje do pliku dodatkowe drzewo wynikowe.

Przykład z rekomendacji

```
<xsl:output name="section-format" method="xhtml" indent="no"/>

<xsl:for-each-group select="*/xhtml:body/*"
                  group-starting-with="xhtml:h1">
  <xsl:result-document href="section{position()}.html"
                    format="section-format" validation="strip">
    <html xmlns="http://www.w3.org/1999/xhtml">
      <head><title><xsl:value-of select="."/></title></head>
      <body>
        <xsl:copy-of select="current-group()" />
      </body> </html>
    </xsl:result-document>
  </xsl:for-each-group>
```

Czego nie ma w XSLT 1.0

- XPath 2.0 z sekwencjami, typami, `if`-em itd.
- Definiowania funkcji.
- Grupowania.
- Zapisywania dodatkowych plików z wynikiem.
- `analyze-string`.
- ...

Tymczasowe fragmenty drzewa

- XSLT 1.0 – osobne typy node-set i result-tree-fragment.
 - nie wolno mieszać,
 - r-t-f nie wolno przetwarzać.
- XSLT 2.0 brak takiego podziału.

XSLT 2.0, ale nie XSLT 1.0

```
<xsl:variable name="tmp">  
  <xsl:apply-templates select="cośtam"/>  
</xsl:variable>
```

```
<xsl:apply-templates select="$tmp" mode="blabla"/>
```

Sposoby przetwarzania – push

- Przetwarzanie sterowane strukturą dokumentu źródłowego (ang. *push*):
 - przechodzimy po strukturze dokumentu źródłowego,
 - generujemy fragmenty struktury dokumentu wyjściowego.

Przykład

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

Sposoby przetwarzania – pull

- Przetwarzanie sterowane strukturą dokumentu wyjściowego (ang. *pull*):
 - jedna duża reguła dla węzła root,
 - generujemy strukturę dokumentu docelowego,
 - wyciągamy odpowiednie wartości z dokumentu źródłowego.

Przykład

```
<xsl:template match="/">
  <html><head><title>Expense Report Summary</title></head>
  <body>
    <h1>Company: <xsl:value-of select="company/name"/></h1>
    <p>Total Amount:
    <xsl:value-of select="expense-report/total"/></p>
  </body>
</html>
</xsl:template>
```