XML-Praxis

Komplexe Transformationen mit XSLT

Jörn Clausen
joern@TechFak.Uni-Bielefeld.DE

Übersicht

- Sortieren
- XML erzeugen und weiterverarbeiten
- modes
- Daten aus mehreren XML-Dokumenten lesen

sorting

- Reihenfolge der Verarbeitung bestimmen:

```
<xsl:apply-templates select="item">
  <xsl:sort select="name"/>
</xsl:apply-templates>
```

- Reihenfolge umkehren:

```
  <xsl:sort select="name" order="descending"/>
```

- nummerische Sortierung:

```
  <xsl:sort select="quant" data-type="number"/>
```

- mehrere Sortierkriteren möglich

Aufgaben

- Die Datei `peanuts.xml` enthält einige Namen. Gib sie z.B. als HTML-Liste aus. Sortiere sie dabei nach folgenden Eigenschaften:
  - Vorname
  - Nachname
  - „Geburtsjahr“
  - umgekehrt nach Vorname
  - umgekehrte Dokumentreihenfolge
  - Nachname und Vorname

```
<body>
  <ul>
    <xsl:apply-templates match="character">
      <xsl:sort select="surname"/>
    </xsl:apply-templates>
  </ul>
</body>

<xsl:template match="character">
  <li>
    <xsl:value-of select="name"/>
    <xsl:text> </xsl:text>
    <xsl:value-of select="surname"/>
  </li>
</xsl:template>
```

```
<xsl:sort select="name"/>

<xsl:sort select="@intro" data-type="number"/>

<xsl:sort select="name" order="descending"/>

<xsl:sort select="position()" order="descending" data-type="number"/>

<xsl:sort select="surname"/>
<xsl:sort select="name"/>
```

sorting, cont.

- auch in Schleife verwendbar:

```
<xsl:for-each select="item">
  <xsl:sort select="quant"/>
  ...
</xsl:for-each>
```

- nur am Anfang des Schleifenrumpfs

result tree fragments

- bereits gesehen: node set in Variable

```
<xsl:variable name="tocsl" select="slide[title/@toc='yes']"/>
<xsl:for-each select="$tocsl">
```

- statischen/generierten XML-Code in Variable ablegen:

```
<xsl:variable name="footer">
  <hr/>
  <address>generated with pres2html</address>
</xsl:variable>
```

- erzeugt kein node set sondern *result tree fragment*
- einzige Möglichkeit in XSLT 1.0:

```
<xsl:copy-of select="$footer"/>
```

Aufgaben

- Definiere zwei globale Variablen `header` und `footer`, die den HTML-Code für eine Kopf- und Fußzeile enthalten. Füge sie mit `xsl:copy-of` in die Ausgabe ein.
- Definiere einen globalen Parameter `date`, der beim Aufruf des XSLT-Prozessors übergeben wird:

  ```
  $ xsltproc --param date '"2003-08-08"' ...
  ```

  Das Datum soll in der Fußzeile erscheinen.

```
<xsl:param name="date"/>
<xsl:variable name="header">
  <address>Peanuts Characters</address>
  <hr/>
  <h1>Sorted List of Peanuts Characters</h1>
</xsl:variable>
<xsl:variable name="footer">
  <hr/>
  <address>generated <xsl:value-of select="$date"/></address>
</xsl:variable>

<xsl:copy-of select="$header"/>
...
<xsl:copy-of select="$footer"/>
```

result tree fragments verarbeiten

- result tree fragment in node set umwandeln:

```
<xsl:variable name="list">
  <name>Charlie Brown</name>
  <name>Lucy van Pelt</name>
  <name>Snoopy</name>
</xsl:variable>
<xsl:for-each select="exslt:node-set($list)/name">
```

- vorher name space `exslt` deklarieren:

```
xmlns:exslt="http://exslt.org/common"
```

- in den meisten XSLT-Prozessoren implementiert
- in XSLT 2.0: *temporary trees*

dictionaries

- übersetze ccTLDs in Ländernamen:

```
<xsl:variable name="ccdict">
  <cc suffix="de">Germany</cc>
  <cc suffix="uk">United Kingdom</cc>
  <cc suffix="fr">France</cc>
</xsl:variable>

<xsl:template match="domain">
  <xsl:variable name="tld"
    select="substring(.,string-length(.)-1)"/>
  <xsl:value-of select="."/> is in
  <xsl:value-of
    select="exslt:node-set($ccdict)/cc[@suffix=$tld]"/>
</xsl:template>
```

Aufgaben

- Die Datei `cd-collection2.xml` enthält eine weitere CD-Sammlung. Das CD-Element wurde um ein Attribut `genre` mit den folgenden Belegungen ergänzt:

`jz`	Jazz
`fs`	Fusion
`st`	Sound Track
`rb`	Rythm&Blues

  Erweitere das bisherige Stylesheet zur Formatierung so, daß der ausgeschriebene Name des Genres angezeigt wird.

```
<xsl:variable name="genre">
  <genre abbrv="rb">Rythm&amp;Blues</genre>
    <genre abbrv="jz">Jazz</genre>
      <genre abbrv="fs">Fusion</genre>
        <genre abbrv="st">Sound Track</genre>
</xsl:variable>

<i>
<xsl:variable name="thisgenre" select="@genre"/>
<i>
  <xsl:value-of select="exslt:node-set($genre)/genre[@abbrv=$thisgenre]"/>
</i>
```

mehrstufige Verarbeitung

- Zwischenergebnis in Variable ablegen und weiterverarbeiten
- ermöglicht komplexe Transformationen:
  - Sortierkriterien, die nicht mit XPath berechnet werden können
  - Gruppierungen
  - Filterung
  - ...

mehrstufige Verarbeitung, cont.

- Aufgabe: Wer ist wie häufig Deutscher Meister geworden?

```
<soccerchamps>
  <champ year="2003">FC Bayern München</champ>
  <champ year="2002">BV 09 Borussia Dortmund</champ>
  <champ year="2001">FC Bayern München</champ>
  ...
</soccerchamps>
```

- Ansatz: Zählergebnis in result tree fragment speichern:

```
<team number="17">FC Bayern München</team>
<team number="3">BV 09 Borussia Dortmund</team>
<team number="5">Borussia Mönchengladbach</team>
...
```

- mit `xsl:for-each` und `xsl:sort` leicht zu verarbeiten

mehrstufige Verarbeitung, cont.

```
<xsl:template match="/soccerchamps">
  <xsl:variable name="countlist">
    <xsl:call-template name="countchamps"/>
  </xsl:variable>
  <xsl:for-each select="exslt:node-set($countlist)/team">
    <xsl:sort select="@number" data-type="number" order="descending"/>
    <tr>
      <td><xsl:value-of select="."/></td>
      <td><xsl:value-of select="@number"/> championships</td>
    </tr>
  </xsl:for-each>
</xsl:template>
```

mehrstufige Verarbeitung, cont.

```
<xsl:template name="countchamps">
  <xsl:for-each select="/soccerchamps/champ[not(.=preceding::champ)]">
    <xsl:variable name="team" select="."/>
    <team>
      <xsl:attribute name="number">
        <xsl:value-of select="count(/soccerchamps/champ[.=$team])"/>
      </xsl:attribute>
      <xsl:value-of select="$team"/>
    </team>
  </xsl:for-each>
</xsl:template>
```

Der XPath-Ausdruck in der `for-each`-Schleife muß die gezeigt Form haben. Folgender Ausdruck funktioniert nicht:

```
/soccerchamps/champ[.!=preceding::champ]
```

Wenn node sets verglichen werden, sind

```
$x!=3
```

und

```
not($x=3)
```

nicht identisch. Der Vergleich

```
$x=3
```

ist dann wahr, wenn wenigstens ein Knoten in `$x` zu 3 evaluiert. Dementsprechend ist der Ausdruck `not(..=..)` wahr, wenn kein Knoten den Wert 3 hat. Der erste Ausdruck (`..!=..`) ist wahr, wenn wenigstens ein Knoten ungleich 3 ist. Ein node set kann also gleichzeitig gleich (=) und ungleich (!=) zu einem bestimmten Wert sein.

Aufgaben

- Verwende das Stylesheet aus der letzten Stunde oder `procorder.xsl` aus dem Übungs-Archiv.
- Ändere das Stylesheet so ab, daß die einzelnen Posten nach ihrem jeweiligen Gesamtwert sortiert ausgegeben werden:

```
<td>Narcotics</td><td>114.8</td><td>12</td><td>1377.6</td>
<td>Luxuries</td><td>91.2</td><td>7</td><td>638.4</td>
<td>Food</td><td>4.4</td><td>10</td><td>44</td>
<td>Textiles</td><td>6.4</td><td>5</td><td>32</td>
```

Erzeuge durch das named template `total` folgende XML-Ausgabe:

```
<line>
  <name>Food</name><price>4.4</price><quant>10</quant><sum>44</sum>
</line>
...
<total>2092</total>
```

Speichere sie als result tree fragment in einer Variable. Iteriere über die sortierten `line`-Elemente:

```
<xsl:for-each select="exslt:node-set($result)/line">
  <xsl:sort select="sum" data-type="number" order="descending"/>
  <tr>
    <td><xsl:value-of select="name"/></td>
    <td><xsl:value-of select="price"/></td>
    <td><xsl:value-of select="quant"/></td>
    <td><xsl:value-of select="sum"/></td>
  </tr>
</xsl:for-each>
```

Gib die Gesamtsumme aus:

```
<tr>
  <td><b>TOTAL</b></td><td> </td><td> </td>
  <td><xsl:value-of select="exslt:node-set($result)/total"/></td>
</tr>
```

modes

- Elemente mehrfach verarbeiten
- dabei unterschiedliche Formatierung
- Überschrift und Inhaltsverzeichnis:

```
<xsl:template match="heading" mode="chapter">
  <h2><a name="..."><xsl:apply-templates/></a></h2>
</xsl:template>

<xsl:template match="heading" mode="toc">
  <tr><td><a href="..."><xsl:apply-templates/></a><td></tr>
</xsl:template>
```

- Aufruf:

```
<xsl:apply-templates select="heading" mode="chapter"/>
```

Beispiel: `procorder.xsl`

- `item`-Elemente aus Liste übernehmen und Summe hinzufügen
- zwei templates für `item`:
  - Summe berechnen
  - HTML-Ausgabe erzeugen
- Vorteile:
  - Code zum Formatieren kann übernommen werden
  - bessere Modularisierung des Stylesheets
  - bessere Erweiterbarkeit

procorder.xsl

```
<table width="100%">
  <xsl:copy-of select="$tablehead"/>
  <xsl:variable name="proclist">
    <xsl:call-template name="summarize">
      <xsl:with-param name="items" select="item"/>
    </xsl:call-template>
  </xsl:variable>
  <xsl:apply-templates select="exslt:node-set($proclist)/item"
    mode="table">
    <xsl:sort select="sum" data-type="number" order="descending"/>
  </xsl:apply-templates>
  <xsl:call-template name="listtotal">
    <xsl:with-param name="total">
      <xsl:value-of select="exslt:node-set($proclist)/total"/>
    </xsl:with-param>
  </xsl:call-template>
</table>
```

Die Variable `tablehead` enthält statischen HTML-Code, um die Kopfzeile der Tabelle zu erzeugen.

procorder.xsl, cont.

```
<xsl:template name="summarize">
  <xsl:param name="items"/>
  <xsl:param name="sum" select="0"/>
  <xsl:choose>
    <xsl:when test="$items">
      <xsl:apply-templates select="$items[1]" mode="sum"/>
      <xsl:call-template name="summarize">
        <xsl:with-param name="items" select="$items[position()!=1]"/>
        <xsl:with-param name="sum"
          select="$sum + $items[1]/price * $items[1]/quant"/>
      </xsl:call-template>
    </xsl:when>
    <xsl:otherwise>
      <total><xsl:value-of select="$sum"/></total>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>
```

procorder.xsl, cont.

```
<xsl:template match="item" mode="sum">
  <item>
    <xsl:copy-of select="*|@*"/>
    <sum><xsl:value-of select="price * quant"/></sum>
  </item>
</xsl:template>

<xsl:template match="item" mode="table">
  <tr>
    <xsl:for-each select="*">
      <td><xsl:value-of select="."/></td>
    </xsl:for-each>
  </tr>
</xsl:template>
```

In der `sum`-Variante werden mit Hilfe von `xsl:copy-of` alle Kind-Elemente und Attribute des `item`-Elements in das result tree fragment kopiert. Sie stehen somit einer späteren Verarbeitung, z.B. in der `table`-Variante, zur Verfügung.
Ein (kleiner) Nachteil dieses Stylesheets ist, daß das Produkt aus `price` und `quant` zweimal berechnet wird.

mehrere Quell-Dokumente

- Eingabe-Datei wird in Ausgabe-Datei transformiert
- weitere Dateien können gelesen werden

```
... select="document('literature.xml')/book" ...
```

- liefert node-set, das beliebig weiterverarbeitet werden kann
- Anwendungen:
  - dictionaries auslagern
  - statische HTML-Teile (Kopf-/Fußzeile)
- Web-Site:
  - Stylesheet transformiert Seiten-Layout
  - Inhalt wird per `document()` dazugelesen

Aufgaben

- Die Datei `tradegoods.xml` enthält eine Liste von Handelswaren und Preisen. `order2.xml` enthält eine Warenliste, die das `price`-Element nicht mehr enthält. Ändere das Stylesheets so ab, daß die Preis-Informationen aus der externen XML-Datei gelesen werden.
- Der Handel einiger Waren ist illegal, dies ist in der Preisliste durch das `status`-Attribut gekennzeichnet. Hebe in der HTML-Ausgabe alle illegalen Posten farblich hervor:

```
<tr bgcolor="red">
```

```
<xsl:variable name="pricelist" select="document('tradegoods.xml')/goods"/>

  <xsl:variable name="price"
    select="$pricelist/item[name=$items[1]/name]/price"/>
  <xsl:apply-templates select="$items[1]" mode="summarize"/>
  <xsl:call-template name="summarize">
    <xsl:with-param name="items" select="$items[position()!=1]"/>
    <xsl:with-param name="sum"
      select="$sum + $price * $items[1]/quant"/>
  </xsl:call-template>

  <xsl:variable name="name" select="name"/>
  <xsl:variable name="price"
    select="$pricelist/item[name=$name]/price"/>
  ...
    <price><xsl:value-of select="$price"/></price>
    <sum><xsl:value-of select="$price * quant"/></sum>

  <xsl:variable name="name" select="name"/>

  <tr>
    <xsl:if test="$pricelist/item[name=$name]/@status='illegal'">
      <xsl:attribute name="bgcolor">red</xsl:attribute>
    </xsl:if>
```