

XML Vorlesung FH Aargau, SS 2006

Ausblick XPath 2.0 und XSLT 2.0

Erik Wilde

10.7.2006

<http://dret.net/lectures/xml-fhnw-ss06/>

10.7.2006

XML Vorlesung FHA SS 2006

1

Wichtige Änderungen ggü. Version 1.0

- starke Erweiterung des Sprachumfangs
- Result Tree Fragments gestrichen
- statt dessen: Jeder XPath-Ausdruck gibt eine Sequence zurück
- auf Sequences kann mit XPath operiert werden
- Unterstützung der XML Schema Datentypen
- Sequences können eine Mischung von Objekten dieser Datentypen enthalten

10.7.2006

XML Vorlesung FHA SS 2006

2

Wichtige Erweiterungen

- Gruppieren von Knoten
- benutzerdefinierte Funktionen
- Zeichenkettenverarbeitung mit regulären Ausdrücken
- Schreiben mehrerer Ausgabedokumente und Formate in einem Durchgang

Gruppieren

- Gruppieren ist eine häufige Anwendung
 - Iteration über eine Menge von Knoten
 - gewisse Eigenschaften sind gleich
 - Gleichbehandlung der gruppierten Knoten
- in XSLT 1.0 nur umständlich möglich
 - durch *Muenchian Grouping* effizient machbar
- `xs1:for-each-group` erlaubt Gruppierung
- Zugriff auf aktuelle Gruppe mit `current-group()` Funktion

Beispiel Sortieren und Gruppieren

Ausgabe aller Elementnamen eines Dokuments

```
<xsl:variable name="allnodes">
  <xsl:for-each-group select="//*" group-by="name()">
    <xsl:sort select="name()"/>
    <xsl:value-of select="name(current-group()[1])"/>
    <xsl:if test="position() != last()">, </xsl:if>
  </xsl:for-each-group>
</xsl:variable>
```

10.7.2006

XML Vorlesung FHA SS 2006

5

Gruppieren mit group-by

- Postleitzahlen in einer Adressliste
 - Iteration über alle Adressen
 - Gruppierung nach der Postleitzahl
 - je nach Anwendungsfall
 - Iteration über alle Adressen (Liste von Adressen)
 - Ausgabe der Postleitzahl (Liste von Postleitzahlen)
- Gruppierung über die gesamte Population
 - Postleitzahlen sind nicht sortiert
 - aufwendige Operation bei grossen Dokumenten

10.7.2006

XML Vorlesung FHA SS 2006

6

Gruppieren mit group-adjacent

- Gruppierung von benachbarten Knoten
 - effizienter als die Gruppierung durch group-by
- zwei Anwendungsfälle
 1. vorsortierte Knotenliste (sortiert nach Postleitzahl)
 2. keine Zusammenfassung erwünscht
 - z.B. eine Messkurve, wo jede Wertänderung signifikant ist
- Gruppierung innerhalb der Sequenz

10.7.2006

XML Vorlesung FHA SS 2006

7

Benutzerdefinierte Funktionen

- neues Top Level Element `xs1:function`
- Funktionsname muss durch eigenen Namensraum abgegrenzt werden
- Aufruf innerhalb XPath-Ausdruck möglich
 - wesentlich schlanker als `xs1:call-template`
- deklarierte Parameter müssen übergeben werden, Vorbelegung nicht möglich
- Funktionswert muss via `xs1:result` zurückgegeben werden

10.7.2006

XML Vorlesung FHA SS 2006

8

Beispiel benutzerdefinierte Funktion

- Namespace Declaration:
`xmlns:cr="http://www.xml-web.de/ChangeRequests"`
- Funktion:

```
<xsl:function name="cr:href-fragment">
  <xsl:param name="teststring"/>
  <xsl:result>
    <xsl:sequence select="$teststring"/>
  </xsl:result>
</xsl:function>
```
- Aufruf:

```
<xsl:variable name="test"
  select="cr:href-fragment(@xlink:href)"/>
```

10.7.2006

XML Vorlesung FHA SS 2006

9

Reguläre Ausdrücke

- neue XPath-Funktionen `matches()`, `replace()` und `tokenize()`
 - `tokenize()` zerteilt einen String in Strings
- neues Element `xsl:analyze-string`
 - enthält `xsl:matching-substring`
 - und/oder `xsl:non-matching-substring`
 - `.` selektiert jeweils den substring
- Adressierung von Teilen eines Musters mit `regex-group()` Funktion

10.7.2006

XML Vorlesung FHA SS 2006

10

Beispiel Nutzung reguläre Ausdrücke

Konvertierung englische in deutsche Daten:

```
<xsl:analyze-string select="current-date()"
  regex="([0-9]+)-([0-9]+)-([0-9]+)">
  <xsl:matching-substring>
    <xsl:number value="regex-group(2)" format="01"/>
    <xsl:text>.</xsl:text>
    <xsl:number value="regex-group(1)" format="01"/>
    <xsl:text>.</xsl:text>
    <xsl:number value="regex-group(3)" format="0001"/>
  </xsl:matching-substring>
</xsl:analyze-string>
```

10.7.2006

XML Vorlesung FHA SS 2006

11

analyze-string Flags

- werden mit `flags="..."` angegeben
 - `i` selektiert *case-insensitive mode*
 - definiert durch die Unicode Character Database
 - `m` selektiert *multiline mode*
 - `^` und `$` selektieren Anfang und Ende einer Zeile
 - Normalfall: `^` und `$` selektieren Anfang und Ende des Strings
 - `s` selektiert *dot-all mode*
 - `.` selektiert alle Zeichen (auch `U+0A`, normalerweise nicht)
 - `x` selektiert *ignore-whitespace mode*
 - Whitespace Characters in der Regex sind insignifikant

10.7.2006

XML Vorlesung FHA SS 2006

12

Text → XML Transformation

- XSLT 2.0 Funktion `unparsed-text()`
 - Einlesen eines beliebigen Text Files
 - ergibt als Resultat einen String
- geschachteltes `xs1:analyze-string`
 1. Zeilenerkennung
 - `regex="\n"` und `xs1:non-matching-substring`
 2. Zeilenverarbeitung
 - `select="."` und Matching der Zeilenteile
- ergibt tabellenähnliches XML
 - falls nötig komplexere Transformation danach

10.7.2006

XML Vorlesung FHA SS 2006

13

Mehrere Ausgabedokumente/Formate

- Unterstützung mehrerer *Result Trees* neben dem *Principal Result Tree*
- Benennung von Ausgabeformaten in erweitertem `xs1:output` Element
- Referenz auf benanntes Ausgabeformat aus neuem Element `xs1:result-document`

10.7.2006

XML Vorlesung FHA SS 2006

14

Beispiel zusätzlicher *Result Tree*

- Deklaration Ausgabeformat

```
<xsl:output name="glossary-format" method="xml" />
```

- Serialisierung

```
<xsl:result-document href="{targetfn}"  
                    format="glossary-format">  
  <xsl:copy-of select="document($srcdoc)//*[@id=$destid]"/>  
</xsl:result-document>
```

XSLT 2.0 und XML Schema

- Integration von Typen in XSLT
 - Validierung und Typannotation des XML
 - Verwendung des getypten Baumes in XSLT
 - ermöglicht typbasierte Operationen
 - andernfalls muss von Hand gecastet werden
- Erzeugung von validierten Bäumen
 - Validierung des Resultats
 - Sicherstellung des Typs zur Laufzeit
 - bessere Fehlermeldungen und -lokalisierung

Zusammenfassung

- mächtigere Funktionalität ggü. Version 1.0
- neue Anwendungsfelder für XSLT
 - v.a. bei Stringverarbeitung
- tendenziell kürzerer Code
- höhere Komplexität durch grosse Anzahl von Funktionen
- experimentelle Implementierung liegt mit Saxon 8.x vor
 - Standard ist noch in Arbeit, *Working Draft*