Eine Schnelleinführung in XPath

Michael Dienert

21. Oktober 2010

1 xml-Dokumente in Baumdarstellung

Mit XPath lassen sich bestimmte Teile eines xml-Dokuments selektieren. Dabei wird das xml-Document als **Baum** betrachtet. Dabei tauchen die Elemente des xml-Dokuments als **Knoten** im Baum auf. Die wichtigsten **Knotentypen** sind

- Wurzelknoten
- Elementknoten
- Textknoten
- Attributknoten

Ferner gibt es noch:

- Namensraumknoten
- Processing-Instruction-Knoten
- Kommentarknoten

2 Lokalisierungspfade

Um nun bestimmte Knoten eines xml-Dokuments auszuwählen, gibt man einen **Lokalisierungspfad** an.
Die Teile eines Lokalisierungspfads werden **Lokalisierungsschritte** genannt. Die einzelnen Lokalisierungsschritte eines Pfads werden mit dem Zeichen "/" getrennt.
Wie auch bei Pfaden eines Dateisystems, gibt es **absolute** und **relative** Lokalisierungspfade.

3 Lokalisierungsschritte

Die Lokalisierungsschritte sind so aufgebaut:

```
Achsenname::Knotentest[Prädikat]
```

3.1 Achsennamen

Mit den Achsennamen kann man bestimmte Teile eines Baums auswählen. Insgesamt gibt es 13 Achsen.

- ancestor : alle Vorfahren des aktuellen Knotens.
- ancestor-or-self
- attribute : alle Attribute des aktuellen Knotens, wenn dieser ein Element ist.
- child : die wichtigste Achse. Wählt alle *direkten* Nachfahren des aktuellen Knotens aus.
- descendant : alle Nachfahren des aktuellen Knotens
- descendant-or-self
- following : alle Nachbarn, die *nach* dem aktuellen Knoten stehen
- following-sibling : alle nachfolgenden Geschwister des aktuellen Knotens
- namespace
- parent : der Elternknoten
- preceding : alle Nachbarn, die *vor* dem aktuellen Knoten stehen.
- preceding-sibling : alle vorausgehenden Geschwister
- self : der aktuelle Knoten selbst

3.2 Knotentest

Mit dem Knotentest wird überprüft ob:

1. ein Knoten einen bestimmten *Namen* hat
2. **und** ob ein Knoten mit dem Knotentyp der gewählten Achse übereinstimmt. Zur Erinnerung: die Knotentypen sind in Kap.1 aufgeführt. Verwendet man im Knotentest als Namen einen "*", werden alle Knoten ausgewählt, die dem Knotentyp der Achse entsprechen.
3. verwendet man statt des Namens eine *Knotenfunktion* kann man bestimmte Knotentypen selektieren. Wichtige Knotenfunktionen sind:
   - text() für Textknoten
   - node() für beliebige Knotentypen

3.3 Prädikat

Prädikate stehen in eckigen Klammmern und dienen als **zusätzliche** Filter, um die Knotenmenge weiter einzuengen.

3.4 Abgekürzte Schreibweise

Folgende Abkürzungen sind bei Xpath-Angaben erlaubt:

child-Achse : Da die child-Achse die am häufigsten verwendete Achse ist, kann `child::` einfach weggelassen werden. Aus `child::autor/child::titel` wird somit `autor/titel`

alle Elemente selektieren : Oft kommt es vor, dass man alle Elemente eines Dokuments selektieren möchte, egal auf welcher Baumebene sie stehen. Das kann man mit dem Ausdruck `/descendant-or-self::node()` lösen. Dieser lange Ausdruck kann durch `//` ersetzt werden. `//` darf am Anfang oder innerhalb eines Lokalisierungspfads einsetzen. Steht es am Anfang, werden alle Elemente des Dokuments selektiert, innerhalb eines Pfads werden alle Elemente ab dem Auftreten von `//` selektiert:

`/descendant-or-self::node()  =  //`

aktuellen Knoten selektieren : Wie im Dateisystem auch, selektiert der Punkt `(.)` den aktuellen Knoten:

`/self::node()  =  .`

die Elternknoten selektieren : Die Übereinstimmung mit Dateisystemen geht noch weiter: zwei Punkte `(..)` selektieren den Elternknoten also:

`/parent::node()  =  ..`

Attribute selektieren : Dieser Achsenausdruck für Attribute wird mit `@` abgekürzt:

`attribute::  =  @`

ausgewählt.

Test auf Position : Das Prädikat

`autor[position()=5]`

kann durch

`autor[5] abgekürzt werden.`

4 Ein Beispiel

Selektieren des Passworts des Benutzers 'michael':

```
<?xml version="1.0" encoding="UTF-8"?>
<passwd>
    <user>
        <username>oliver</username>
        <password>becker</password>
    </user>
    <user>
        <username>michael</username>
        <password>neu</password>
    </user>
    <user>
        <username>alfred</username>
        <password>mad</password>
    </user>
</passwd>
```

Eine von vielen Möglichkeiten wäre:

```
/child::passwd/
   child::user/
      child::username[text()='michael']/
         following-sibling::password/
            child::text()
```

A Übungsaufgaben

1. (a) Wähle das Wurzelelement aus.

   (b) Wähle alle Kinder des Wurzelelements aus, die ccc-Elemente sind (`beispiel1.xml`).

   ```
   <aaa>
      <bbb/>
      <ccc/>
      <bbb/>
      <bbb/>
      <ddd>
         <bbb/>
      </ddd>
      <ccc/>
   </aaa>
   ```

2. Wähle alle Kinder von ddd-Elementen aus, die bbb-Elemente sind. Die ddd-Elemente sollen ihrerseits Kinder des Wurzelelements sein (`beispiel2.xml`).

```
<aaa>
   <bbb/>
   <ccc>
      <bbb/>
   </ccc>
   <bbb/>
   <bbb/>
   <ddd>
      <bbb/>
      <aaa/>
   </ddd>
   <ccc/>
</aaa>
```

3. Wähle im Dokument **alle** bbb-Elemente aus. Der Lokalisierungspfad soll dabei mit ' // ' beginnen. Welchem Ausdruck entspricht ' // ' (`beispiel3.xml`)?

```
<aaa>
   <bbb/>
   <ccc/>
   <bbb/>
   <ddd>
      <bbb/>
   </ddd>
   <ccc>
      <ddd>
         <bbb/>
         <bbb/>
      </ddd>
   </ccc>
</aaa>
```

4. ***** - selektiert Elementknoten

   text() - selektiert Textknoten

   node() - selektiert Text- und Elementknoten

   (a) Wähle alle Elementknoten (keine Textknoten!) aus, die Kinder des durch den Pfad `/aaa/ccc/ddd` gegebenen Kontexts sind.

   (b) Nun sollen nur alle Textknoten ausgewählt werden, die Kinder des Kontexts sind.

   (c) Schliesslich sollen **alle** Kindsknoten des Kontexts gewählt werden.

   (d) Wähle alle bbb-Elemente aus, die drei Vorfahren haben.

   (e) Wähle alle Elemente des Dokuments aus.

   (f) Wähle den gesamten Text des Dokuments aus.

   (g) Wähle das gesamte Dokument aus.

   (h) Wähle alle aaa- und eee-Elemente aus

   Datei: `beispiel4.xml` .

```
<aaa>
   <ggg>
      <ddd>
         <bbb/>
         <bbb/>
         <eee/>
         <fff/>
      </ddd>
   </ggg>
   <ccc>
      <ddd>
         <bbb/>
         <bbb/>
         <eee>
            <zzz/>
         </eee>
         <fff/>
      </ddd>
   </ccc>
   <ccc>
      <bbb>
         <bbb>
            <bbb/>
         </bbb>
      </bbb>
   </ccc>
</aaa>
```

5. (a) Wähle das erste bbb-Kind des aaa-Elements aus.
   (b) Wähle das letzte bbb-Kind des aaa-Elements aus.

   (`beispiel5.xml`).

```
<aaa>
   <bbb/>
   <bbb/>
   <bbb/>
   <bbb/>
</aaa>
```

6. (a) Wähle alle ’ pid ’-Attribute aus.
   (b) Wähle alle Elemente aus, die ein ’ aid ’-Attribut haben.
   (c) Wähle den Text aller Elemente aus, die ein ’ aid ’-Attribut haben.
   (d) Wähle den Text aller Elemente aus, deren ’ pid ’-Attribut den Wert 2 hat.
   (e) Wähle alle Elemente aus, die irgendein Attribut haben.
   (f) Wähle alle Elemente aus, die überhaupt kein ein Attribut haben.
   (g) Wähle alle bbb-Elemente aus, deren Attribut ’ pid ’ den Wert ’2’ hat.
   (h) Nun wähle den Text innerhalb des Elements aus.
   (i) Wähle alle Artikel der Warengruppe ’io’ aus. Leerzeichen vor oder nach dem Attributswert sollen dabei ignoriert werden.

   (`beispiel6.xml`).

```
<aaa>
   <bbb pid = "1">alfred</bbb>
   <bbb pid = "2">hans</bbb>
   <bbb aid = "100">maus</bbb>
   <bbb aid = "101">hund</bbb>
   <bbb gruppe = "io">tastatur</bbb>
   <bbb gruppe = " io">maus</bbb>
   <bbb/>
</aaa>
```

7. (a) Wähle alle Elemente aus, die genau 2 bbb-Kinder haben.
   (b) Wähle alle Elemente aus, die genau 2 Kinder beliebigen Typs haben haben.

   (`beispiel7.xml`).

```
<aaa>
   <ccc>
      <bbb/>
      <bbb/>
      <bbb/>
   </ccc>
   <ddd>
      <bbb/>
      <bbb/>
   </ddd>
   <eee>
      <ccc/>
      <ddd/>
   </eee>
</aaa>
```

8. Aufgaben mit den Funktionen `name()` , `starts-with()` und `contains()`
   (a) Wähle alle Elemente mit dem Namen `bbb` aus.
   (b) Wähle alle Elemente aus, deren Namen mit dem Buchstaben 'b' beginnt.
   (c) Wähle alle Elemente aus, deren Namen den Buchstaben 'c' enthält.

   (`beispiel8.xml`).

```
<aaa>
   <bcc>
      <bbb/>
      <bbb/>
      <bbb/>
   </bcc>
   <ddb>
      <bbb/>
      <bbb/>
   </ddb>
   <bec>
      <ccc/>
      <dbd/>
   </bec>
</aaa>
```

9. Aufgaben mit der Funktion `string-length()`

(a) Wähle alle Elemente aus, deren Namen drei Buchstaben hat.

(b) Wähle alle Elemente aus, deren Namen weniger als drei Buchstaben hat.

(c) Wähle alle Elemente aus, deren Namen mehr als vier Buchstaben hat.

(`beispiel9.xml`).

```
<aaa>
   <b/>
   <cccc/>
   <dd/>
   <eee/>
   <ffffffff/>
   <gggg/>
</aaa>
```

10. Aufgaben mit `div, mod, floor(), ceiling()`

    (a) Wähle jedes dritte `bbb` -Element aus.

    (b) Wähle das mittlere oder die beiden mittleren `bbb` -Elemente aus

    (`beispiel10.xml`).

```
<aaa>
   <bbb/>
   <bbb/>
   <bbb/>
   <bbb/>
   <bbb/>
   <bbb/>
   <bbb/>
   <bbb/>
   <ccc/>
   <ccc/>
   <ccc/>
</aaa>
```

B Lösungen der Aufgaben

1. (a) `/aaa`

   (b) `/aaa/ccc`

2. `/aaa/ddd/bbb`

3. `//bbb`
   ' // ' entspricht der Achse `/descendant-or-self::node()`

4. (a) `/aaa/ccc/ddd/*`

   (b) `/aaa/ccc/ddd/text()`

   (c) `/aaa/ccc/ddd/node()`

   (d) `/*/*/*/bbb`

   (e) `//*`

(f) `//text()`

(g) `//node()`

(h) `//aaa | //eee`

5. (a) `/aaa/bbb[1]`

   (b) `/aaa/bbb[last()]`

6. (a) `//@pid`

   (b) `//*[@aid]`

   (c) `//*[@aid]/text()`

   (d) `//*[@pid=2]/text()`

   (e) `//*[@*]`

   (f) `//*[not(@*)]`

   (g) `//bbb[@pid=2]`

   (h) `//bbb[@pid=2]/text()`

   (i) `//bbb[normalize-space(@gruppe)='io']/text()`

7. (a) `//*[count(bbb)=2]`

   (b) `//*[count(*)=2]`

8. (a) `//*[name()='bbb']` ist dasselbe wie `//bbb`

   (b) `//*[starts-with(name(),'b')]`

   (c) `//*[contains(name(),'c')]`

9. (a) `//*[string-length(name()) = 3]`

   (b) `//*[string-length(name()) < 3]`

   (c) `//*[string-length(name()) > 4]`

10. (a) `//bbb[position() mod 3 = 0]`

    (b)
```
//bbb[position() = floor(last() div 2 +0.5)
     or position() = ceiling(last() div 2 +0.5)]
```

C Quellen

`http://www.zvon.org/xxl/XPathTutorial/General_ger/examples.html`