XML 2010

Erik Tjong Kim Sang
erikt(at)xs4all.nl

27 september 2010

Vorige week

Presentatie van XML-bestanden: Cascading Style Sheets

- selectors, eigenschappen en waarden
- display; onderdelen van een blok: padding, border en margin
- kleuren, fonts, centreren
- toevoegen tekst; lijsten, tabellen, links en plaatjes
- geschiedenis, voordelen en nadelen van CSS

Deze week

Omzetten van XML-documenten naar andere formaten met XSLT

Wat is XSLT?

XSLT (*Extensible Style Language for Transformation*) is een programmeertaal die wordt gebruikt om XML-bestanden om te zetten in andere structuren (vaak HTML)

De taal bestaat uit instructies die voor XML-elementen bepaalde acties definiëren, bijvoorbeeld het veranderen van de tekstkleur:

```
<xsl:template match="toneelstuk">
   <font style="color: blue;"><!-- text --></font>
</xsl:template>
```

Deze instructie verandert de tekstkleur in `toneelstuk`-elementen

Werken met XSLT (1)

Een XSLT-programma ziet eruit als een XML-bestand (extensie: `.xsl`)

Maar een XSLT-bestand kan een gevarieerde mix van tags bevatten (bijvoorbeeld XML en HTML) en kan daarom niet worden gevalideerd

Op onze Linuxsystemen kan XSLT worden gebruikt via het commando `xsltproc`:

```
$ xsltproc programma.xsl invoer.xml > uitvoer.html
```

Het bestand `invoer.xml` wordt door het programma `programma.xsl` omgezet in ht bestand `uitvoer.html`

Werken met XSLT (2)

Het XSL-bestand mag ook in het XML-bestand worden gedefinieerd:

```
<?xml version="1.0" encoding="utf8"?>
<?xml-stylesheet type="text/xml" href="programma.xsl"?>
...
```

In dit geval hoeft het stijlbestand niet aan `xsltproc` worden doorgegeven:

```
$ xsltproc invoer.xml > uitvoer.html
```

Het XML-document kan nu ook in een webbrowser worden geladen

Een voorbeeld van een XSLT-programma

```
1: <xsl:stylesheet version="1.0"
2:  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
3: <xsl:template match="/">Hello world!</xsl:template>
4: </xsl:stylesheet>
```

Dit programma bevat 1 instructie voor de root (/) van een document

Het vervangt het XML-document door de regel *Hello world!* (zie regel 3)

Op regels 1 en 2 staat de verplichte definitie van het document als stylesheet (namespace `xsl`); op regel 4 wordt deze afgesloten

Hoe ziet een XML-bestand eruit voor XSLT?

XSLT ziet een XML-bestand als een boom, met de volgende aantekeningen:

Niet alleen elementen worden als knopen beschouwd maar ook attributen, tekst, commentaar, namespaces en processinginstructies

Boven alle knopen is een naamloze knoop root (/) geplaatst

Voorbeelddocument met XSLT-interpretatie

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!-- markup toegevoegd door: erikt -->
<toneelstuk>
<tekst spreker="ANTONIO">
  'k Weet waarlijk niet waarom 'k zoo somber ben,
```

De rootknoop bevat twee kinderen: commentaar en `<toneelstuk>`

De knoop `<toneelstuk>` heeft 1 kind: `<tekst>`

De knoop `<tekst>` heeft twee kinderen: attribuut `spreker` en tekst

Basis van XSLT-instructies

XSLT-instructies bewerken het element dat in het `match`-attribuut wordt genoemd, bijvoorbeeld:

```
<xsl:template match="beschrijving">...
```

Binnen de instructie moet iets worden gedaan met de inhoud van het element, anders is het niet meer aanwezig in de XSLT-uitvoer:

```
<xsl:template match="beschrijving"/>
```

Deze instructie gooit alle elementen `beschrijving` met hun inhoud weg

Verwerking van niet-gespecificeerde elementen

Als XSLT een XML-element tegenkomt waarvoor geen instructie is gedefinieerd dan voert het de *default*-acties uit:

- kindelementen worden verwerkt met beschikbare instructies
- tekstinhoud wordt zichtbaar gemaakt
- attribuutwaarden worden zichtbaar gemaakt
- alle andere informatie wordt weggelaten: elementnamen, commentaar en processing-instructies

Default-regels in XSLT

Verwerk kindelementen:

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

Laat tekstinhoud zien:

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

Laat attribuutwaarden zien: (niet reproduceerbaar met `xsltproc`)

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


Voorbeelden van paden in het `match`-attribuut

`match="element"`
→ alle elementen met naam `element`

`match="ouder/element"`
→ alle elementen met naam `element` en ouder `ouder`

`match="voorouder//element"`
→ alle elementen met naam `element` en voorouder `voorouder`

`match="element[@attribuut='waarde']"`
→ elementen `element` met attribuut `attribuut` met waarde `waarde`

Verwerking van de inhoud van een element

`<xsl:apply-templates/>`
→ verwerk de inhoud van het huidige element

`<xsl:value-of select="pad"/>`
→ laat de inhoud zien van het element dat aan het pad `pad` voldoet

Opmerkingen:

Bij de tweede methode worden de beschikbare instructies voor de kindelementen niet verwerkt

Naar het huidige element kan worden verwezen met `select="."`

Case-study 1: een stijlbestand voor onze toneelstukken

We gaan nu een stijlbestand maken voor de toneelstukdocumenten van de eerste prakticumopdracht

We beginnen met het toevoegen van een verwijzing naar een leeg stijlbestand:

```
<?xml-stylesheet type="text/xml" href="toneelstuk.xsl"?>
```

Wat is het effect hiervan?

Alle tekst van het document wordt in 1 paragraaf geplaatst

Het definiëren van paragrafen

We beginnen met het plaatsen van de inhoud van de elementen `<beschrijving>` en `<tekst>` op aparte paragrafen

We willen een HTML-bestand maken en dus gebruiken we HTML-elementen:

```
<xsl:template match="beschrijving | tekst">
   <p><xsl:apply-templates/></p>
</xsl:template>
```

Helaas zien we geen effect als we het document met een browser bekijken

Controle van het bestand

Om te controleren of het XSLT-bestand wel de gewenste acties heeft uitgevoerd, bekijken we de broncode van de pagina met de browser

Daar zien we het XML-bestand. Dit maakt ons dus niet wijzer

We gaan terug naar de command-line en halen het document door `xsltproc`:

```
$ xsltproc --novalid toneelstuk.xml
```

De tags `<p>` en `</p>` blijken correct te zijn toegevoegd

Waarom zien we geen paragrafen in de browser?

Een aanwijzing over de oorzaak van dit probleem zit in de eerste regel van de uitvoer van `xsltproc`: `<?xml version="1.0"?>`

Het document is een XML-document en de browser weet niet hoe de `<p>`-elementen in XML moeten worden gerepresenteerd

We moeten de XSLT-uitvoer expliciet als HMTL definieren in het stijlbestand:

```
<xsl:output method="html"/>
```

Nu worden de paragraafgrenzen wel zichtbaar gemaakt door de browser

Context-gevoelige definities

Nu beginnen ook de beschrijvingen met voetnoten op een aparte regel en dat willen we corrigeren

XSLT laat ook context-gevoelige instructies toe:

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

Deze instructie beïnvloed alleen elementen `beschrijving` die een element `tekst` als ouder hebben

Ook XSLT gebruikt voor elementen de meest specifieke instructies

Stijlinstructies

Stijldefinities zoals kleur en type font kunnen we in apart stijlbestand opslaan of bij de elementen zelf:

```
<xsl:template match="beschrijving">
  <p style="padding:5px; color:white; background-color: black;">
    <xsl:apply-templates/>
  </p>
</xsl:template>

<xsl:template match="tekst/beschrijving">
  <font style="color:red;"><xsl:apply-templates/></font>
</xsl:template>
```


Centreren van tekst

De tekst kan worden gecentreerd door HTML-elementen in een instructie voor het `<toneelstuk>-element`:

```
<xsl:template match="toneelstuk">
  <html>
    <body style="font-family: Verdana, sans-serif;">
      <div style="width:400px; margin:auto;">
        <xsl:apply-templates/>
      </div>
    </body>
  </html>
</xsl:template>
```


Toevoegen van extra tekst

Extra tekst in de instructies wordt zichtbaar gemaakt in het uitvoerdocument

Tekst uit het XML-document kunnen we zichtbaar maken via het XSL-element `value-of`, bijvoorbeeld voor het attribuut `spreker`:

```
<xsl:value-of select="@spreker"/>
```

Het `@`-teken geeft aan dat we hier niet verwijzen naar element maar naar een attribuut

Tellen in XSLT

XSLT houdt zelf al tellers bij waarvan de inhoud kan worden opgevraagd

Hiervoor gebruiken we de instructie `<xsl:number>`, bijvoorbeeld:

```
<xsl:number count="tekst"/>
```

Deze instructie laat zien het hoeveelste element `<tekst>` op dit moment wordt verwerkt

Lijsten, tabellen, plaatjes en links

Met XSLT is het eenvoudig om structuren die beschikbaar zijn in HTML te presenteren vanuit een XML-document:

- plaats de informatie in XML-elementen
- definieer hoe de elementen moeten worden omgezet naar HTML

Bijvoorbeeld een linkelement en een plaatjeselement in XML:

```
<link href="url">anchor</link>
<plaatje src="url" width="width" height="height"/>
```

Het resultaat van het XSLT-stijlbestand

vreemde kwanten nu en dan gevormd: De een tuurt voortdurend door zijn wimpers heen, Lacht, als een papegaai, bij 'n doedelzak, En de ander heeft zoo'n zuur azijngezicht, Dat hij zijn tanden nooit ten glimlach toont, Schoon Nestor[2] zwoere op de aardigheid der grap.

BASSANIO, LORENZO _en_ GRATIANO _komen op._

OLANIO[9]: Daar komt Bassanio, uw eed'le neef, Gratiano, en Lorenzo. Vaart gij wel: Veel beter is 't gezelschap dat u zoekt.

Programmeren met XSLT

Veel standaardinstructies uit imperatieve programmeertalen (zoals C, Java en Perl) zijn ook beschikbaar in XSLT:

- condities: `<xsl:if` en `<xsl:choose`
- loops: `<xsl:foreach`
- functies: `<xsl:template name="..."`

Voorbeeld van `<xsl:if`

```
<xsl:template match="tekst">
  <xsl:if test="@spreker='ANTONIO'">
    ...
  </xsl:if>
</xsl:template>
```

De acties bij ... worden alleen uitgevoerd voor de teksten met spreker ANTONIO

Bij dit commando is geen `else`-instructie beschikbaar

Voorbeeld van `<xsl:choose`

```
<xsl:choose>
  <xsl:when test="@spreker='ANTONIO'">
    ...
  </xsl:when>
  <xsl:otherwise>
    ...
  </xsl:otherwise>
 </xsl:choose>
```

Deze instructie laat meerdere voorwaarden toe (`<xsl:when`) en 1 algemeen alternatief (`<xsl:otherwise`)

Voorbeeld van `<xsl:foreach`

Deze instructie laat een lijst van sprekers zien gescheiden door komma's:

```
<xsl:for-each select="tekst">
  <xsl:value-of select="@spreker"/>,
</xsl:for-each>
```

Met `<xsl:for-each` kan een lijst met specifieke informatie uit een XML-document zichtbaar worden gemaakt

Voor het tellen van dit soort informatie worden XPath-instructies gebruikt (college van volgende week)

Voorbeeld van functies in XSLT

```
<xsl:template name="menu">
  <p>
    <a href="item1">Item 1</a> |
    <a href="item2">Item 2</a>
  </p>
</xsl:template>
```

Functies zijn handig om stukken code die vaker voorkomen in het document te benoemen

Naar de code kan worden verwezen met de naam, bijvoorbeeld

```
<xsl:call-template name="menu"/>
```


Case-study 2: combineren van data

In deze case-study combineren we resultaten van sportwedstrijden in XML tot een resultatentabel in HTML

De invoer van het programma ziet er als volgt uit:

```
<results>
  <result>
    <hometeam>Roda JC Kerkrade</hometeam>
    <awayteam>FC Twente</awayteam>
    <homegoals>0</homegoals>
    <awaygoals>0</awaygoals>
  </result>
  <result>
```


Isoleren van teamresultaten

De wedsrijdresultaten kunnen in 1 keer worden omgezet in een tabel maar dat is ingewikkeld omdat teams zowel onder `<hometeam>` als `<awayteam>` kunnen staan

Daarom plaatsen we eerst elk resultaat van een team in een apart element dat er als volgt uitziet:

```
<result team="FC Twente" scored="0" against="0" points="1"/>
<result team="NEC" scored="1" against="0" points="3"/>
<result team="VVV-Venlo" scored="0" against="1" points="0"/>
<result team="sc Heerenveen" scored="1" against="3" points="0"/>
<result team="PSV" scored="3" against="1" points="3"/>
```

XSLT-code voor de eerste transformatie

```
<xsl:template match="result">
  <xsl:choose>
    <xsl:when test="homegoals/text() = awaygoals/text()">
      <result>
        <xsl:attribute name="team">
          <xsl:value-of select="hometeam"/></xsl:attribute>
        <xsl:attribute name="scored">
          <xsl:value-of select="homegoals"/></xsl:attribute>
        <xsl:attribute name="against">
          <xsl:value-of select="awaygoals"/></xsl:attribute>
        <xsl:attribute name="points">1</xsl:attribute>
      </result>
      <result>
```


XSLT-code voor de tweede transformatie (1)

```
<xsl:for-each
     select="//result[not(@team =
       preceding-sibling::result/@team)]">
    <xsl:sort select="@team" data-type="text" order="ascending"/>
    <xsl:call-template name="process">
      <xsl:with-param name="team">
        <xsl:value-of select="@team"/>
      </xsl:with-param>
    </xsl:call-template>
</xsl:for-each>
```

Hiermee wordt de functie `process` aangeroepen voor elk team

XSLT-code voor de tweede transformatie (2)

```
<xsl:template name="process">
  <xsl:param name="team">empty</xsl:param>
  <xsl:variable name="scored"
    select="sum(//result[@team=$team]/@scored)"/>
  <xsl:variable name="against"
    select="sum(//result[@team=$team]/@against)"/>
  <xsl:variable name="points"
    select="sum(//result[@team=$team]/@points)"/>
  <tr>
    <xsl:attribute name="points"><xsl:value-of
      select="$points"/></xsl:attribute>
    <td align="left"><xsl:value-of select="$team"/></td>
    <td align="right"><xsl:value-of select="$points"/></td>
```


Resultaten

Het tweede stijlbestand genereert een alfabetisch gesorteerde tabel met teamnamen punten en doelsaldo's

Voor het sorteren op punten zou nog een derde stijlbestand nodig zijn

Dit voorbeeld laat zien dat XSLT ook getallen kan verwerken en daarbij complexe transformaties kan uitvoeren

Achtergrondliteratuur

David Hunter, Jeff Rafter, Joe Fawcett and Eric van der Vlist, *Beginning XML (Programmer to Programmer)*. John Wiley & Sons, 2004 (chapter 8: XSLT)

Erik T. Ray, *Learning XML*, O'Reilly, 2001 (chapter 6: Transformation: Repurposing Documents)

W3schools, *CSS Tutorial*, http://www.w3schools.com/xsl/

EINDE