XSLT beyond <for-each>

Codegarden '10

Chriztian Steinmeier

Me

- Frontend Developer at Heyday in Aarhus, DK
- 10+ years of XSLT experience

Me

- Frontend Developer at Heyday in Aarhus, DK
- 10+ years of XSLT experience
- Almost a full year of Umbraco experience (!)

You ?

Why are You even here?

Why are You even here?

According to Twitter...

Why are You even here?

"XSLT is too verbose..."

Why are You even here?

“Using XSLT kills your soul....”

Why are You even here?

“There’s too much typing!”

Why are You even here?

"You need 30 lines of XSLT to accomplish one line of

[INSERT_ANY_OTHER_LANGUAGE_HERE]"

Why are You even here?

"I hate XSLT..."

Why are You even here?

"Yipes, this is horrible."

Why are You even here?

"LOL, yeah... XSLT makes your eyes bleed :-)"

Guess what?

Guess what?

“I love XSLT!”

Guess what?

“XSLT freakin‘ rocks!”

Demo

Templates

Templates

- Only the most important concept of XSLT (d'oh!)

Templates

- Create a template for the element to output

XSLT

```xsl
<xsl:template match="person">
    <!-- do stuff -->
</xsl:template>
```

Templates

- Fill-in the desired output

XSLT

```
<xsl:template match="person">
    <p class="person">
      <a href="mailto:jane@doe.com"> Jane Doe </a>
    </p>
</xsl:template>
```

Templates

- Grab the dynamic parts from the surrounding XML

XSLT

```
<xsl:template match="person">
    <p class="person">
        <a href="mailto:{email}">
            <xsl:value-of select="name" />
        </a>
    </p>
</xsl:template>
```

Templates

“[magic happens...]”

Templates

```
-<p class="person">
   <a href="mailto:john@doe.com">John Doe</a>
 </p>
-<p class="person">
   <a href="mailto:jane@doe.com">Jane Doe</a>
 </p>
-<p class="person">
   <a href="mailto:lotsof@doe.com">Lotsof Doe</a>
 </p>
```

<xsl:apply-templates />

<xsl:apply-templates />

- Built-in templates helps you

XSLT

```
<!-- Elements and rootnode: Process childnodes -->
  <xsl:template match="* | /">
      <xsl:apply-templates />
  </xsl:template>
<!-- Text and attributes: Output content -->
  <xsl:template match="text() | @*">
      <xsl:value-of select="." />
  </xsl:template>
```

“Template rules and xsl:apply-templates are not an advanced feature to be used only by advanced users. […] If you aren't using them, you are making your life unnecessarily difficult.”

#1 — Selecting specific stuff

XSLT

```
<xsl:template match="/">
  <h2>Yahoo people</h2>
    <xsl:apply-templates select="people/person[contains(email, '@yahoo.com')]" />
  <!-- … -->
  <h2>Google people</h2>
    <xsl:apply-templates select="people/person[contains(email, '@gmail.com')]" />
</xsl:template>
```

#2 — Excluding specific stuff

XSLT

```
<!-- Empty template(s) to suppress output -->
  <xsl:template match="person[name = 'Jar-Jar Binks']" />
  <xsl:template match="person[contains(email, '@example.com')]" />
```

The XPath Beast

XPath

- A tricky part of XSLT.
- People have trouble with XPath

XPath

- Navigating/locating nodes in XML
- Very much like navigating a filesystem
- Used in `select`, `test` and `match` attributes

XPath

XSLT

```
<!-- Select nodes for output -->
  <xsl:apply-templates select="movies/movie" />

<!-- Test for specific attribute -->
  <xsl:if test="movie[@imdb]" />

<!-- Match a specific director's movies -->
  <xsl:template match="movie[director[name = 'David Fincher']]" />
```

XPath

- 'Generic' XML yields complex XPaths

XPath

- 'Generic' XML yields complex XPaths

XML

```
<node id="1890" nodeTypeAlias="CalendarEvent" >
  <data alias="eventTitle" >MVC Bootcamp</data>
</node>
```

XSLT

```
<xsl:template match="node[@nodeTypeAlias = 'CalendarEvent']" >
    <xsl:apply-templates select="data[@alias = 'eventTitle']" />
</xsl:template>
```

XPath

- 'Generic' XML yields complex XPaths

XML

```
<CalendarEvent id="1890">
    <eventTitle>MVC Bootcamp</eventTitle>
</CalendarEvent>
```

XSLT

```
<xsl:template match="CalendarEvent">
    <xsl:apply-templates select="eventTitle" />
</xsl:template>
```

News Flash

“They just fixed that!”

Axes

Axes

- `self::` Name of element
- `parent::` ../
- `ancestor::`
- `ancestor-or-self::`

Axes

- `child::` /
- `descendant::` //
- `descendant-or-self::`

Axes

- `following::`
- `following-sibling::`
- `preceding::`
- `preceding-sibling::`

Axes

“There’s an app for that!”

Predicates

Predicates

Filters the selection (think “where...” or “having...”)

XSLT

```
<!-- Filter by attribute -->
  <xsl:value-of select="movie[@title = 'Star Wars']/synopsis" />
<!-- Filter by position -->
  <xsl:apply-templates select="person[position() = 2]" />
  <xsl:apply-templates select="person[2]" />
<!-- Filter by child node having an attribute with a specific value -->
  <xsl:value-of select="node[data[@alias = 'umbracoNaviHide'] = 1]" />
```

Predicates

Can be chained to filter the previous result

- `*[position() <= 5][@votes < 4]`

XML

```
<answer votes="1" rating="3" />
<answer votes="4" rating="3" />
<answer votes="8" rating="2" />
<answer votes="2" rating="4" />
<answer votes="18" rating="9" />
...
```

Be careful, though...

- `node[position() > 3][position() < 6]`

XML

```
<node id="1201" />
<node id="1202" />
<node id="1203" />
<node id="1204" />
<node id="1205" />
<node id="1206" />
<node id="1207" />
<node id="1208" />
```

Probably didn't expect this:

- `node[position() > 3][position() < 6]`

XML

```
<node id="1201" />
<node id="1202" />
<node id="1203" />
<node id="1204" />
<node id="1205" />
<node id="1206" />
<node id="1207" />
<node id="1208" />
```

Combine in same predicate using the *and* operator:

- `node[position() > 3 and position() < 6]`

XML

```
<node id="1201" />
<node id="1202" />
<node id="1203" />
<node id="1204" />
<node id="1205" />
<node id="1206" />
<node id="1207" />
<node id="1208" />
```

"The Unnecessaries (2010)"

Starring (in alphabetical order):

- `count(`*expr*`)`
- `string(`*expr*`)`

"A tale of two friends that frequently found themselves paying unnecessary visits to the attributes of lady XSLT."

Who's counting?

XSLT

```
<xsl:if test="count($currentPage/node) &gt; 0">
    <!-- do stuff -->
</xsl:if>
```

same as:

XSLT

```
<xsl:if test="$currentPage/node">
    <!-- do stuff -->
</xsl:if>
```

Unless, of course...

You're the funny (and possibly annoying) guy on the team :-)

XSLT

```
<xsl:if test="Castle[count(Dracula)]">
    <xsl:call-template name="RUN" />
</xsl:if>
```

String-noise

XSLT

```
<xsl:if test="node[string(data[@alias = 'umbracoNaviHide']) = '1']" >
    <!-- do stuff -->
</xsl:if>
```

Same as:

XSLT

```
<xsl:if test="node[data[@alias = 'umbracoNaviHide'] = 1]" >
    <!-- do stuff -->
</xsl:if>
```

String-noise

```
<xsl:variable name="newsContainer" select="string('News')" />
```

XSLT

again, same as:

```
<xsl:variable name="newsContainer" select=" 'News' " />
```

XSLT

How-To...

Conditional attributes

XSLT

```
<xsl:template match="movie">
    <p>
        <xsl:attribute name="class">
            <xsl:choose>
                <xsl:when test="position() mod 2 = 0">even</xsl:when>
                <xsl:otherwise>odd</xsl:otherwise>
            </xsl:choose>
        </xsl:attribute>
        <!-- do stuff -->
    </p>
</xsl:template>
```

Conditional attributes

XSLT

```
<xsl:template match="movie">
    <p class="odd">
        <xsl:if test="position() mod 2 = 0">
            <xsl:attribute name="class">even</xsl:attribute>
        </xsl:if>
        <!-- do stuff -->
    </p>
</xsl:template>
```

Conditional attributes

XSLT

```
<xsl:template match="movie">
  <p class="substring('even|odd', (position() mod 2) * 5 + 1, 4)">
    <!-- do stuff -->
  </p>
</xsl:template>
```

Conditional attributes

XSLT

```
<xsl:attribute-set name="odd-or-even">
    <xsl:attribute name="class">
        <xsl:value-of select="substring('even|odd', (position() mod 2) * 5 + 1, 4)" />
    </xsl:attribute>
</xsl:attribute-set>
<!-- ... -->
<xsl:template match="movie">
    <p xsl:use-attribute-sets="odd-or-even">
        <!-- do stuff -->
    </p>
</xsl:template>
```

Entity Tricks

Entity Tricks

Probably noticed this in the built-in Umbraco XSLT template files:

DTD

```
<!DOCTYPE xsl:stylesheet [ <!ENTITY nbsp " "> ]>
```

Enables HTML-style hard spaces in your XSLT, e.g.:

XSLT

```
<xsl:value-of select="data[@alias = 'dateOfBirth']" />
 
<img src="{data[@alias = 'icon']}" width="16" height="16" />
```

Entity Tricks

But we can do more than that...

Entity Tricks

Back to the 'generic' XML thing

XML

```
<node id="1890" nodeTypeAlias="CalendarEvent" >
  <data alias="eventTitle" >MVC Bootcamp</data>
</node>
```

XSLT

```
<xsl:template match="node[@nodeTypeAlias = 'CalendarEvent']" >
    <xsl:apply-templates select="data[@alias = 'eventTitle']" />
</xsl:template>
```

Entity Tricks

We can create custom entities for the complex stuff:

DTD

```
<!ENTITY CalendarEvent "node[@nodeTypeAlias = 'CalendarEvent']">
<!ENTITY eventTitle "data[@alias = 'eventTitle']">
```

- and use those to bridge the mind gap:

XSLT

```
<xsl:template match="&CalendarEvent;">
    <xsl:apply-templates select="&eventTitle;" />
</xsl:template>
```

Entity Tricks

- and of course, the bonus:

DTD

```
<!ENTITY CalendarEvent "CalendarEvent">
<!ENTITY eventTitle "eventTitle">
```

- all set and done for ~~Umbraco 4.1~~ Umbraco 4.5

XSLT

```
<xsl:template match="&CalendarEvent;">
    <xsl:apply-templates select="&eventTitle;" />
</xsl:template>
```

“No changes to the XSLT???”

Demo

Questions ?

THANKS!

- Twitter: `@greystate`
- Blog: `http://greystate.dk`
- XSLT 'pimpage': `http://pimpmyxslt.com`
- Work: `http://www.heyday.dk`

Credits

- Awesomeness: our.umbraco.org, Umbraco + Hartvig & Co.
- Custom font: “planet e-style” by Mads Rydahl