In what order do templates in an XSLT document execute, and do they match on the source XML or the buffered output?

Asked 15 years, 2 months ago Modified 11 years, 9 months ago Viewed 31k times

84

Here is something that has always mystified me about XSLT:

1. In what order do the templates execute, and
2. When they execute, do they match on (a) the original source XML, or (b) the current output of the XSLT to that point?

Example:

```
<person>
  <firstName>Deane</firstName>
  <lastName>Barker</lastName>
</person>
```

Here is a fragment of XSLT:

```
<!-- Template #1 -->
<xsl:template match="/">
  <xsl:value-of select="firstName"/> <xsl:value-of select="lastName"/>
</xsl:template>

<!-- Template #2 -->
<xsl:template match="/person/firstName">
  First Name: <xsl:value-of select="firstName"/>
</xsl:template>
```

Two questions about this:

1. I am assuming that Template #1 will execute first. I don't know why I assume this -- is it just because it appears first in the document?
2. Will Template #2 execute? It matches a node in the source XML, but by the time the we get to this template (assuming it runs second), the "firstName" node will not be in the output tree.

So, are "later" templates beholden to what has occurred in "earlier" templates, or do they operate on the source document, oblivious to what has been transformed "prior" to them? (All those words are in quotes, because I find it hard to discuss time-based

issues when I really have little idea how template order is determined in the first place...)

In the above example, we have a template that matches on the root node ("/") that -- when it is done executing -- has essentially removed all nodes from the output. This being the case, would this pre-empt all other templates from executing since there is nothing to match on after that first template is complete?

To this point, I've been concerned with later templates not executing because the nodes they have operated on do not appear in the output, but what about the inverse? Can an "earlier" template create a node that a "later" template can do something with?

On the same XML as above, consider this XSL:

```
<!-- Template #1 -->
<xsl:template match="/">
  <fullName>
    <xsl:value-of select="firstName"/> <xsl:value-of select="lastName"/>
  </fullName>
</xsl:template>

<!-- Template #2 -->
<xsl:template match="//fullName">
  Full Name: <xsl:value-of select="."/>
</xsl:template>
```

Template #1 creates a new node called "fullName". Template #2 matches on that same node. Will Template #2 execute because the "fullName" node exists in the output by the time we get around to Template #2?

I realize that I'm deeply ignorant about the "zen" of XSLT. To date, my stylesheets have consisted of a template matching the root node, then are completely procedural from there. I'm tired of doing this. I would rather actually understand XSLT correctly, hence my question.

xslt

Share

Improve this question

Follow

edited Oct 7, 2009 at 18:20

asked Oct 7, 2009 at 13:30

Deane

8,717 ● 12 ● 61 ● 115

You've missed the / in the 2nd person tag in your sample xml. – Chris R Oct 7, 2009 at 14:19

4 The "zen" I have learned -- the execution process of an XSLT is XML-centric, not XSL-centric. *The structure of the XML drives the flow, not the structure of the XSL.* This was one of the big

pieces I have not understood all these years. – Deane Oct 7, 2009 at 18:20

1 match="//fullName" is the same as match="fullName". A *pattern* tests whether a given node matches it from any context, as opposed to an XPath *expression* which *selects* nodes from a *particular* context. – Evan Lenz Oct 9, 2009 at 9:21

4 Answers

Sorted by: Highest score (default)

105

I love your question. You're very articulate about what you do not yet understand. You just need something to tie things together. My recommendation is that you read "How XSLT Works", a chapter I wrote to address exactly the questions you're asking. I'd love to hear if it ties things together for you.

Less formally, I'll take a stab at answering each of your questions.

> 1. In what order do the templates execute, and
> 2. When they execute, do they match on (a) the original source XML, or (b) the current output of the XSLT to that point?

At any given point in XSLT processing, there are, in a sense, two contexts, which you identify as (a) and (b): where you are in the *source tree*, and where you are in the *result tree*. Where you are in the source tree is called the *current node*. It can change and jump all around the source tree, as you choose arbitrary sets of nodes to process using XPath. However, conceptually, you never "jump around" the result tree in the same way. The XSLT processor constructs it in an orderly fashion; first it creates the root node of the result tree; then it adds children, building the result in document order (depth-first). [Your post motivates me to pick up my software visualization for XSLT experiments again...]

The order of template rules in a stylesheet never matters. You can't tell, just by looking at the stylesheet, in what order the template rules will be instantiated, how many times a rule will be instantiated, or even whether it will be at all. (`match="/"` is an exception; you can always know that it will get triggered.)

> I am assuming that Template #1 will execute first. I don't know why I assume this -- is it just because it appears first in the document?

Nope. It would be called first even if you put it last in the document. Template rule order never matters (except under an error condition when you have more than one template rule with the same priority matching the same node; even then, it's optional for the implementor and you should never rely on such behavior). It gets called first because the first thing that **always** happens whenever you run an XSLT processor is

a virtual call to `<xsl:apply-templates select="/"/>`. The one virtual call constructs the entire result tree. Nothing happens outside it. You get to customize, or "configure", the behavior of that instruction by defining template rules.

> Will Template #2 execute? It matches a node in the source XML, but by the time the we get to this template (assuming it runs second), the "firstName" node will not be in the output tree.

Template #2 (nor any other template rules) will never get triggered unless you have an `<xsl:apply-templates/>` call somewhere in the `match="/"` rule. If you don't have any, then no template rules other than `match="/"` will get triggered. Think of it this way: for a template rule to get triggered, it can't just match a node in the input. It has to match a node that you elect to *process* (using `<xsl:apply-templates/>`). Conversely, it will continue to match the node as many times as you choose to process it.

> Would [the `match="/"` template] pre-empt all other templates from executing since there is nothing to match on after that first template is complete?

That rule preempts the rest by nowhere including `<xsl:apply-templates/>` in it. There are still plenty of nodes that *could* be processed in the source tree. They're always all there, ripe for the picking; process each one as many times as you want. But the only way to process them using template rules is to call `<xsl:apply-templates/>`.

> To this point, I've been concerned with later templates not executing because the nodes they have operated on do not appear in the output, but what about the inverse? Can an "earlier" template create a node that a "later" template can do something with?

It's not that an "earlier" template creates a new node to be processed; it's that an "earlier" template in turn processes more nodes from the source tree, using that same instruction (`<xsl:apply-templates`). You can think of it as calling the same "function" recursively, with different parameters each time (the nodes to process as determined by the context and the `select` attribute).

In the end, what you get is a tree-structured stack of recursive calls to the same "function" (`<xsl:apply-templates>`). And this tree structure is **isomorphic** to your actual result. Not everyone realizes this or has thought about it this way; that's because we don't have any effective visualization tools...yet.

> Template #1 creates a new node called "fullName". Template #2 matches on that same node. Will Template #2 execute because the "fullName" node

> exists in the output by the time we get around to Template #2?

Nope. The only way to do a chain of processing is to explicitly set it up that way. Create a variable, e.g., `$tempTree`, that contains the new `<fullName>` element and then process *it*, like this `<xsl:apply-templates select="$tempTree">`. To do this in XSLT 1.0, you need to wrap the variable reference with an extension function (e.g., `exsl:node-set()`), but in XSLT 2.0 it will work just as is.

Whether you're processing nodes from the original source tree or in a temporary tree that you construct, either way you need to explicitly say what nodes you want to process.

What we haven't covered is how XSLT gets all its implicit behavior. You must also understand the *built-in template rules*. I write stylesheets all the time that don't even include an explicit rule for the root node (`match="/"`). Instead, I rely on the built-in rule for root nodes (apply templates to children), which is the same as the built-in rule for element nodes. Thus I can ignore large parts of the input, let the XSLT processor automatically traverse it, and only when it comes across a node I'm interested in will I do something special. Or I could write a single rule that copies everything recursively (called the identity transform), overriding it only where necessary, to make incremental changes to the input. After you've read "How XSLT Works", your next assignment is to look up the "identity transform".

> I realize that I'm deeply ignorant about the "zen" of XSLT. To date, my stylesheets have consisted of a template matching the root node, then are completely procedural from there. I'm tired of doing this. I would rather actually understand XSLT correctly, hence my question.

I applaud you. Now it's time to take the "red pill": read "How XSLT Works"

Share Improve this answer Follow

answered Oct 9, 2009 at 9:13

Evan Lenz
4,116 ● 1 ● 20 ● 18

BTW, what you were trying to do (write template rules to process nodes created by other template rules) is actually cool, though as I said it doesn't work unless you explicitly set up the pipeline using variables and variable references. I've sometimes wondered about creating a processing context in which an XSLT processor would get repeatedly invoked on the result until all elements (e.g., in a particular macro namespace) get processed and no more appear in the result--kind of like a recursive inclusion mechanism. That'd be handy. Pipelines in XSLT are a bit syntactically clunky as it is. – Evan Lenz Oct 9, 2009 at 9:28

Evan, your answer and sample chapter were really fantastic. I think I have a grasp on it, save one question, which I will post separately. – Deane Oct 10, 2009 at 15:49

Excellent answer, really helped my understanding. I agree that visualization tools would be a huge help - especially being able to see precisely which operators select content in the source, which generate content in the output, and how the current node and context are managed. – chrispitude Dec 23, 2019 at 6:26

I did make some progress on a visualization tool in more recent years: github.com/evanlenz/xslt-visualizer – Evan Lenz May 11, 2020 at 22:44

8

Templates **always** match in the source XML. So the order doesn't really matter, unless 2 or more templates match the same node(s). In that case, somewhat counter-intuitively, the rule with the **last** matching template is triggered.

Share Improve this answer Follow

answered Oct 7, 2009 at 14:06

mirod
16.1k ● 3 ● 48 ● 65

2 My understanding was if 2 or more templates match the same node then the one with the most specific match will run. I assume you mean 2 or more with exactly the same match condition? – Chris R Oct 7, 2009 at 14:39

3 sorta, the rules are given in the spec: w3.org/TR/xslt#conflict. Unless you use the priority attribute, then a lot of patterns end up with the same priority. Note the last paragraph of the section in the spec though: "an XSLT processor may signal the error". – mirod Oct 7, 2009 at 15:10

I didn't know the rule about same priority either giving an error or using the last depending on the processor implementation, or that you could set priority yourself with the attribute, thanks. – Chris R Oct 7, 2009 at 15:44

3

In your 1st example Template #1 runs because when you start processing the input xml it begins at the root and that is the only template in your stylesheet that matches the root element. Even if it was 2nd in the stylesheet it would still run 1st.

In this example template 2 will not run as you have already processed the root element using template 1 and there are no more elements to process after the root. If you did want to process other elements using additional templates you should change it to.

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

This then allows you to define a template for each element you are interested in and process the xml in a more logical way, rather than doing it procedurally.

Also note that this example will not output anything as at the current context (the root) there is no firstName element, only a person element so it should be:

```
<xsl:template match="/">
  <xsl:value-of select="person/firstName"/> <xsl:value-of
select="person/lastName"/>
</xsl:template>
```

I find it easier to think that you are stepping through the xml, starting at the root and looking for the template that matches that element then following those instructions to generate teh output. The XSLT transforms the input document to the output so the output doucument is empty at the start of the transformation. The output is not used as part of the transformation it is just the output from it.

In your 2nd example Template #2 will not execute because the template is run against the input xml not the output.

Share

Improve this answer

Follow

edited Oct 7, 2009 at 14:53

answered Oct 7, 2009 at 14:15

Chris R

2,544 ● 3 ● 26 ● 32

2

Evan's answer is basically a good one.

However one thing which does seem to be lacking is the ability to "call" up chunks of code without doing any matching. This would - at least in some people's opinion - enable much better structuring.

I have made a small example in an attempt to show what I mean.

```
<xsl:template match="/" name="dotable">
<!-- Surely the common html part could be placed somewhere else -->
    <!-- the head and the opening body -->
<html>
<head><title>Salary table details</title></head>

    <body>
<!-- Comments are better than nothing -->
    <!-- but that part should really have been somewhere else ... -->

<!-- Now do what we really want here ... this really is making the table! -->

<h1>Salary Table</h1>
<table border = "3" width="80%">
<xsl:for-each select="//entry">
    <tr>
        <td><xsl:value-of select="name" /></td>
        <td><xsl:value-of select="firstname" /></td>
        <td><xsl:value-of select="age" /></td>
        <td><xsl:value-of select="salary" /></td>
```

```
    </tr>
</xsl:for-each>
</table>

<!-- Now close out the html -->
</body>
</html>
<!-- this should also really be somewhere else -->

<!-- This approach works, but leads to horribly monolithic code -->
    <!-- Further - it leads to templates including code which is strictly -->
    <!-- not relevant to them. I've not found a way round this yet -->
</xsl:template>
```

However, after fiddling around a bit, and at first making use of the hint that if there are two matching templates the last one in the code will be selected, and then restructuring my code (not all shown here), I achieved this which seems to work, and hopefully generates the correct code, as well as displaying the wanted data -

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<!-- <?xml version="1.0"?>-->

<xsl:template name="dohtml">
  <html>
      <xsl:call-template name="dohead" />
      <xsl:call-template name="dobody" />
  </html>
</xsl:template>

<xsl:template name="dohead">
<head>
    <title>Salary details</title>
</head>
</xsl:template>

<xsl:template name="dobody">
<body>
    <xsl:call-template name="dotable" />
</body>
</xsl:template>

<xsl:template match="/entries" name="dotable">

<h1>Salary Table</h1>
<table border = "3" width="80%">
<xsl:for-each select="//entry">
    <tr>
        <td><xsl:value-of select="name" /></td>
        <td><xsl:value-of select="firstname" /></td>
        <td><xsl:value-of select="age" /></td>
        <td><xsl:value-of select="salary" /></td>
    </tr>
</xsl:for-each>
</table>

</xsl:template>
```

```
<xsl:template  match="/" name="main">
            <xsl:call-template name="dohtml" />
</xsl:template>
```

[Scroll the code above up-down if you can't see it all]

The way this works is the main template always matches - matches on /

This has the chunks of code - templates - which are called.

This now means that it is not possible to match another template on / but it is possible to match explicitly on a named node, which in this case is the highest level node in the xml - called entries.

A small modification to the code produced the example given above.

Share

Improve this answer

Follow

edited Mar 9, 2013 at 13:12

answered Mar 9, 2013 at 12:19

user2151421

21 ● 2