基于 XSLT 2.0 的升级策略分析①

文　展　(成都信息工程学院通信工程系　四川省成都市　610225)
黄小燕　(成都信息工程学院控制工程系　四川省成都市　610225)
曾晓辉　(成都信息工程学院通信工程系　四川省成都市　610225)

摘　要： 本文将基于 XSLT2.0 的新特性着重阐述升级 XSLT 的各种策略，这些策略将提供给你丰富的升级方式考虑，让你选择更加适合自己的方式来升级你的 XSLT 转换开发项目。

关键词： XSLT2.0　升级策略　孤岛

1　引言

W3C 发布的规范 XSLT2.0 系列是一种转换 XML 文档的语言。它包括很多新的特性，其中一部分是为了克服 XSLT 1.0 的不足而设计的，是对 XSL 转换语言一个非常成功的扩展。

XSLT 将 XML 内容转换到其他格式，包括其它 XML 格式。例如，我们可以使用 XSLT 将数据库的 XML 输出转换成 XHTML 网站或者一系列可作打印的 XSL－FO 文档。XSLT 2.0 标准化了许多前版本中只属扩展的部分，例如创建多个输出文档或创建用户定义的 XPath 函数。在有更多对国际化的支持和更丰富的开发工具供程序设计员使用下，XSLT 2.0 比起现时已被使用于大规模关键任务开展的 XSLT 1.0 更为合适[1]。

除了新功能之外，XSLT 2.0 引入了强大输入和支持选择性地使用 W3C XML Schema。强大输入是企业级编程语言的一个特点，如 Java、C＋＋ 和 C#，它可以减少程序错误，减少开发和维护大型系统的开销。

2　为什么要升级到 XSLT2.0

如果编写过 XSLT 样式表，您可能知道通过某些编码模式来获得超出代码表面含义之外的结果。看到 generate－id( ) 的时候您的脑子里会立刻想到什么呢？是 node－set( ) 呢？还是 substring－before( ) 呢？或是 disable－output－escaping 呢？XSLT 2.0 很快就将成为最终的推荐标准，可以用它使您的代码更容易阅读。新的 XSLT 特性允许您使用更接近所要表达的含义的术语。它包含了分组，隐含文档节点，用户定义函数，日期时间处理，模式感知以及输出方面的多方改进，这些令人激动的特性会让我们的 XML 转换工作更加容易和富有激情。

我们知道 XSLT 1.0 主要在两个 W3C 文档中定义：XSLT 和 XPath。而 2.0 版是为了适应 XQuery 的步伐而设计的，现在 XSLT 家族的核心包括六个规范：XSLT、XPath、Functions and Operators (F&O)、Data Model (XDM)、Formal Semantics 和 Serialization。与 1.0 类似，它也建立在其他几种基础规范之上(XML、名称空间等)，并且把 XML Schema 纳入了自己的轨道。此前的需求文档申明了要实现的新特性的目标，肯定了开发 2.0 规范的必要性[2]。

3　XSLT2.0 的新特性

XSLT 2.0 引入了许多新特性，在 XSLT1.0 的基础上有了诸多改进，尤其在分组、隐含文档节点、用户定义函数、日期时间处理、模式感知以及输出方面。

3.1　简化的分组

XSLT 1.0 没有解决元素分组的问题，虽然可以通过 for－each 循环来识别，但效率很低。因此对 XSLT 2.0 的要求之一就是必须要简化分组，2.0 中分组是通过新的 xsl:for－each－group 指令和两个支持函数实现的，即 current－group( ) 和 current－grouping－key

① 成都信息工程学院科研基金资助(CRF200508)

( )。和 for – each 一样,for – each – group 允许使用 xsl:sort 作为子元素来控制分组处理的顺序。

看下面的例子,把清单 1 的 XML 文档中的城市列表,转换为清单 2 中以国家进行分组的 HTML 表格:

清单 1. XML 文档中的城市列表

```
<cities > <city name = " milan" country = " italy" pop = " 5" / >
    <city name = " paris" country = " france" pop = " 7" / >
    <city name = " munich" country = " germany" pop = " 4" / >
    <city name = " lyon" country = " france" pop = " 2" / >
    <city name = " venice" country = " italy" pop = " 1" / >
</cities >
```

清单 2. 以国家进行分组的 HTML 表格

```
<table > <tr > <th > Country </th > <th > City List </th > <th >
Population </th > </tr >
    <tr > <td > Italy </td > <td > milan, venice </td > <td > 6 </
td > </tr >
    <tr > <td > france </td > <td > paris, lyon </td > <td > 9 </
td > </tr >
    <tr > <td > germany </td > <td > munich </td > <td > 4 </
td > </tr >
</table >
```

XSLT 1.0 在处理时,通过 for – each 循环,对每个特定的国家,我们首先要用下面的 XPath 表达式找出它的第一个城市:cities/city[not (@ country = preceding: : * /@ country)]。然后,对每个分组,为了获得每个国家的城市名字列表以及该国的总人口,需要再回头引用该组的所有其他成员。显然这不是个理想的局面,因为这种额外的代码效率低,而且往往是错误之源。

XSLT 2.0 使用 xsl:for – each – group,简化了分组问题,代码如清单 3 所示:

清单 3. XSLT 2.0 的转换方法

```
<xsl:for – each – group select = " cities/city" group – by = " @ coun-
try" >
<tr > <td > <xsl:value – of select = " @ country" / > </td >
    <td > <xsl:value – of select = " current – group ( ) /@ name"
separator = " , " / > </td >
    <td > <xsl:value – of select = " sum ( current – group ( ) /@
pop) " / > </td >
</tr >
</xsl:for – each – group >
```

在上例中,xsl:for – each – group 作为 XPath 的取值上下文的一部分,对" current group" 进行初始化,当前的分组是一个简单序列。一旦我们使用 group – by 属性设置好分组,以后就可以使用 current – group ( ) 函数引用当前的分组。这就完全消除了 XSLT1.0 方案中的额外开销。

3.2 输出的改进

一个 2.0 样式表可以生成多棵结果树,在不同的位置进行序列化。xslt 2.0 使用 xsl:result – document 元素提供了多个输出文档的一个标准方法。一个"主要结果文件"和几个"次级结果文件"。主要结果文件以 xhtml 存储,次级结果文件以纯文本方式存储。

3.3 隐含节点树

XSLT 1.0 受到挫折的主要原因之一是结果树片段(Result Tree Fragment,RTF)的局限性。虽然表示树的变量可以复制到结果集或者将其看作一个字符串,但是不能使用标准 XSLT/XPath 工具像树那样导航,除非你使用一个供应商提供的扩展函数,通常是 node – set ( ) 之类,来把这个 RTF 转化成一类节点集(含有一个根节点)。这样增加了开发者的工作量,因此,把复杂的变换分解成一系列简单的变换变得非常必要。

XSLT 2.0 简化了这一变换,当你使用 xsl:variable 创建一个临时树时,这个变量的值就是个真正的节点集。事实上,用 XPath 2.0 的术语来讲,它是个真正的节点序列(true node sequence),包含一个 document node (这是一个 XPath 2.0 的名称,也就是 XPath 1.0 的" root node")。在这个序列上你可以使用 XPath 表达式深入到这棵树,对它应用模板(template)等等,就像使用其它源文件一样。有了 XLST 2.0 就不再需要 node – set( ) 之类的扩展函数。

3.4 用户定义函数

XSLT2.0 引入了一个非常强大的功能,就是允许定义他们自己的函数,并可以在 XPath 表达式中使用。样式函数(Stylesheet functions)是用 xsl:function 元素定义的。这个元素必须指定一个 name 属性。它包含 0 个或多个 xsl:param 元素,然后是 0 个或多个 xsl:variable 元素,后面是唯一的 xsl:result 元素。这种严格的内容模型看起来是个限制,但是你会发现 XSLT2.0 的真正强大之处就在于可定义 xsl:result 元素的 select 属性。你可能会想到,XPath2.0 具备有条件表达式

(if...then)和迭代(iterative)(列举?)表达式(for...return)。

3.5 模式感知

XSLT 2.0 中模式感知是一种可选特性。如果处理程序支持它,可以发现这是一种有用的特性。模式感知特性主要用于错误检查。它可以用 XML 模式验证输入和输出,允许根据 Schema 类型引用样式表中的源节点。

4 升级到 XSLT2.0 的策略

XSLT 2.0 基本兼容 XSLT 1.0,并增加了很多新特性。您的 1.0 样式表也许只用到了 1.0 的一部分,可能将用到 2.0 特性集合更小的一部分。准备升级的时候,要把精力集中到您使用的旧特性和吸引您的新特性上。本文提出了五种选择,代表五种经过提炼的升级观点(也许不能说五种升级观点,因为也包括不升级的选择)。选择哪种策略有两类决策因素需要考虑:组织的能力因素和吸引您的 2.0 特性的推动因素[3]。

如果要把自己的 XSLT1.0 版本升级到新的 2.0 版本,我们必须根据自己的实际情况选择合理的策略。首先必须了解一个基本概念:样式表模块。它是用来表示能够导入或包括到称为样式表的集合实体的单位(通常是文件)。模块提供了比模板划分更强的划分,主要因为 XSLT 声明通常局限于单个模块。阅读完本文并且考虑了哪种方法最好之后,如果准备采用渐进式的方法,可以使用模块把旧的 XSLT 代码从新代码中划分出来,除了已有的模块划分之外。

为了便于从 1.0 样式表迁移到 2.0,厂商可以根据自己的意愿提供支持向后兼容的 2.0 处理程序。在样式表中,该特性由 version 属性控制,它可以出现在任何元素中。而在 1.0 中,version 属性只能出现在顶层 xsl:stylesheet 元素中。如果一个元素包含该属性,并且值为 1.0,则对该元素本身及其中的子树启用向后兼容特性。从概念上说类似于能够像在 1.0 版本中那样执行的代码孤岛,除了极少数例外情况。要记住,该特性是根据样式表的文档结构而不是运行转换时的执行流来启用的。比如,如果 xsl:call-template 指令包含属性 version="1.0",但执行中调用的命名模板包含属性 version="2.0",则执行那个被调用的模板时不会启用向后兼容。

类似的,被导入或者被包括的样式表模块如果包含属性 version="1.0",则对那个模块启用向后兼容特性,即使主样式表是 2.0 样式表也是如此。如果执行由 2.0 处理程序在向后兼容模式下执行,结果可能和 1.0 处理程序生成的结果不完全一样。关于 1.0 处理程序和 2.0 处理程序向后兼容模式的差异列表。

以上讨论了升级到 XSLT2.0 的必要性和 XSLT2.0 的新特性,下面将具体分析从 XSLT 1.0 升级到 XSLT2.0 的五种策略各自的优点和缺点,以便用户选择适合自己的方式来升级你的 XSLT 转换开发项目。

4.1 按新版本重写

如果选择按新版本重写这种方式,那么就类似于一切从头开始。为了充分利用 XSLT 2.0 的语法和特性,必须为可能需要修改整个设计和重写大部分样式表模块作好准备。如果当前的样式表有很好的模块化结构,可以选择保留原来的结构。更常见的情况是,如果您的 1.0 样式表已经有清晰的架构,您就可以节省一些计划时间,这取决于您希望利用哪些 2.0 特性。

这种升级方式主要有下面一些优点:因为基本上从零开始,所以我们可以充分利用 2.0 的语法和特性的优势。不需要寻找支持向后兼容的 2.0 处理程序。使用 2.0 特性可以让代码更容易阅读。因为不需要向后兼容,代码会更简单直接。结果是跨 2.0 处理程序的可移植性。

这种方式也存在以下缺点:你必须学习有关的所有 2.0 特性。同时我们将会面临着很多分析和准备工作要做。

4.2 转化大部门样式表,保留 1.0 孤岛

如果全部用 2.0 重写过于激进,那么次佳的选项就是将大部分样式表模块转换到 2.0 并重用一些 1.0 模块。可以在单个模板或者更深的层次上保留 1.0 孤岛。该选项仍然需要一些规划,特别是在什么地方使用新的 2.0 特性,并调查在新的样式表结构中旧模块的可重用性。

这种升级方式优点如下:如果希望最终避免使用向后兼容的话,该选项是渐进式迁移的最佳方法。该选项可以将注意力集中到向后兼容和喜欢的特性上。可以增强代码的可读性。可以增强代码的模块化和可重用性。可以像过去那样应用本地标准调整或代码清理。

这种升级方式缺点也是明显的:本地版本控制和调整中容易出错。该选项需要支持向后兼容的处理程

序。和孤岛有关的代码可能更难懂(虽然其他地方改进了)。马上需要很多规划工作以便切实地改进代码。如果避免触及薄弱的部分,这种方法可能延长混合使用多种版本的时间。选择不同版本的分支代码可能需要更多的编码时间。

4.3 建立新版本孤岛

如果需要特定的 2.0 特性,可以尝试在 1.0 样式表中补上 2.0 版的孤岛,使用支持向后兼容的 2.0 处理程序运行它。如果处理程序生成意料之外的结果,则调整 1.0 模块直到满意为止。该选项把目标集中到需要的 2.0 特性上,规划和重新设计的工作量很小。自然,如果需要的新特性很容易隔离出来,那么这种方法最有效。

采用这种升级方式,可逐步进行修改,是否进行修改由特殊的需求或者某一特定增强是否能带来很大好处来决定。同时,可以逐步学习新特性,把精力集中到性能瓶颈上。可以集中提高模块化和可重用性,并可以像过去那样应用本地标准调整和代码清理。如果必须保持和纯 1.0 处理程序的兼容性,这种方法更容易。

这种升级方式同时也存在不少缺点:如选择该选项通常会减缓采用新的 2.0 特性的进度,也不能很快享受到相应的好处。而且这种打补丁的方法可能由于使用错误的版本而造成错误。该选项需要支持向后兼容的处理程序。同时,使用孤岛可能损害整体的可读性(虽然孤岛本身可能改进了)。确定需要修改的地方时可能很难预测孤岛的数量和分布。更难利用具有广泛影响的新特性,如模式感知。支持向后兼容的 2.0 处理器运行 1.0 代码仍然面临同样的问题,调试的时候需要了解两个版本。

4.4 全面转化并调试

如果需要将已有的 1.0 样式表快速转化成 2.0 样式表,为何不将样式表版本属性设置为 2.0 看看能否从 2.0 处理得到预期结果呢? 如果不成功,就开始调试,逆向跟踪每个模板的结果并进行修改直到满意为止。调整可能涉及到使用适当的 2.0 指令,或者(如果支持向后兼容特性)将版本属性设为 1.0 看看会发生什么。当然,该选项不太关心利用新的 2.0 特性。

这种方式的优点是:可以采用渐进式方法进行转换。如果 1.0 样式表没有出现兼容性异常,那么这种转换方法可能最快。好的测试集可以保证成功转化。如果幸运的话,不需要寻找支持向后兼容的 2.0 处理程序。

缺点是:如果出现问题时选择将版本设置为 1.0,该选项需要支持向后兼容的处理程序。本地版本控制和调整中很容易出现错误,因而延长了调试周期。由于向后兼容和调整,得到的样式表可能很难读懂。

4.5 继续使用老版本

继续使用老版本,虽然有一些好处,比如你什么也不用做,也不需要新的处理程序(直到您再也雇不到人来编写纯 1.0 代码)。而且 1.0 版本处理程序已经成熟,经过了仔细的调试。

但是,这种方式同样存在明显的缺点:不能利用新的 2.0 特性。处理程序 bug 修正的周期可能延长,随着时间的变化,知道存在一种可读性更强的 2.0 替代品,1.0 中的老技术会让你觉得不可接受。

5 结束语

在本文中,我们主要讨论了 XSLT2.0 的新特点和从 XSLT1.0 版本升级到 2.0 版本的五种策略,虽然这五种策略看上去各有优劣,难以选择,但是最终的决定还在采取任何一种方式升级你的 XSLT 时,都要想清楚自己为什么要这么做,企业该如何管理开发,资源都是在哪些阶段可用,对企业的负担有多重,跨 XSLT 时的互操作性。你甚至可以将上面几种策略混合起来使用以达到你的目的,比方说,可以采用 4.1 和 4.2 的方法制定宏大的计划,但采用 4.4 试试看的办法来实施。

参考文献

1 Jovanovic, J., and Gasevic, D. Achieving knowledge interoperability: An XML/XSLT approach. Expert Systems with Applications, Volume 29, Issue 3, October 2005, Pages 535 - 553.

2 Kay, M. (2005). Comparing XSLT and XQuery. Proceedings of the XTech 2005 Conference on XML, the Web and beyond ; software available from : http://saxon. sourceforge. net/, last accessed on March 29, 2006.

3 Dmitry Kirsanov ,XSLT 2.0 Web Development, Prentice Hall,2004 - 03 - 01.

4 (美) Steven Holzner, 闻道工作室 译, XSLT 技术内幕, 机械工业出版社, 2002 - 1 - 1.