

5 XML数据处理技术

XSLT

徐杨

xuyang@scut.edu.cn

主要内容

- XSLT 简介
- XSLT 的工作原理
- XSLT 的模板
- XSLT 模板中各种转换功能

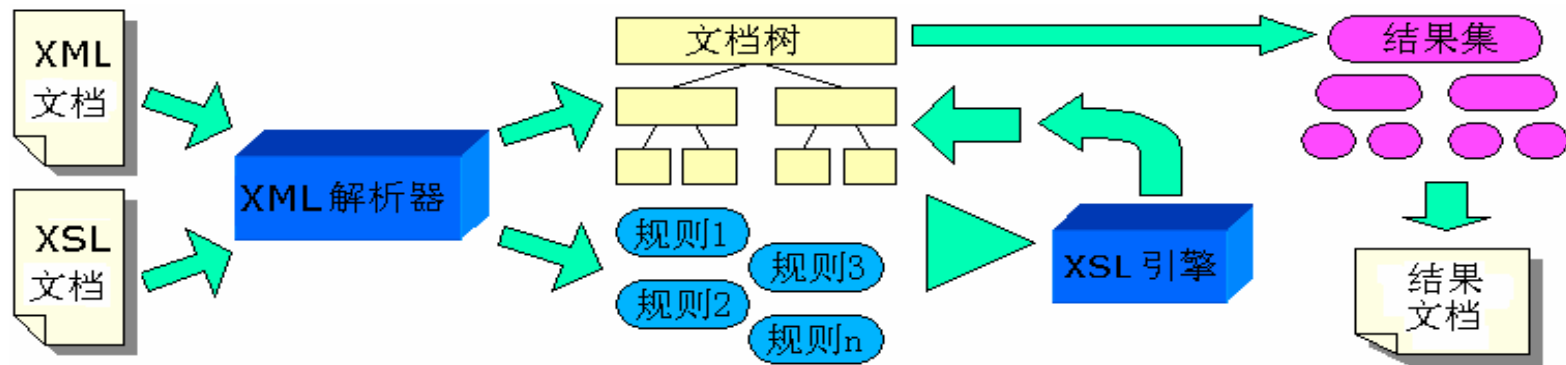
1. XSLT 简介

- XSLT (eXtensible Stylesheet Language Transformations) 中的字母 “T” 表示 “转换”，它是 XSL 规范中的一部分，是用于对 XML 树型数据进行结构转换的有力工具。可以根据指定的**转换规则**将一个 XML 文档树或者其中的部分内容转换为另一种文档树形式。
- XSLT 所提供的转换功能非常有效，并不仅仅局限于结构上的重组，准确的说，可以将 XML 文档转换为任何形式，包括 XML、HTML 和普通文本，以致于很多人使用 XSL 作为一种在浏览器中显示 XML 数据的工具，也就是利用这种转换功能，将 XML 文档转换为 HTML（通常需要增加一些 HTML 中的有关显示的标记），然后在浏览器中进行显示。XSLT 的功能要比 CSS 强得多。

XSLT 简介

- XSLT 语言是一种声明性（Declarative）的编程语言，即 XSLT 程序本身只是包含了一些转换规则的 XML 文档。而这些规则可以被递归地应用到转换过程中。
- XSLT 处理程序（或称之为执行引擎）将首先确定 XSLT 规则，然后根据规则的匹配条件（通过 XPath 表达式指定）、以及优先顺序完成相应的转换操作。
- XSLT 本身也是一个 XML 文档，所以它也必须严格遵守 XML 规范。其根元素的命名空间为：<http://www.w3.org/1999/XSL/Transform>。

2. XSLT 的工作原理



- 在进行 XSLT 的转换任务时，通常需要两个输入文档，一个是包含源数据的 XML 文档，一个是包含转换任务规则的 XSLT 文档；
- 由 XML 解析器对这两个文档进行解析，将包含源数据的 XML 文档转换为所对应的文档树结构，将 xslt (xsl) 文档中定义的处理模块看作是一系列的转换规则。
- 由 XSLT 引擎调用这些规则，对文档树进行遍历，分别处理其中指定的数据节点，将其转换为所需的结果集，并序列化为结果文档。

XSLT 的一个简单示例

- pepole.xml+pepole.xslt
- XSLT 并不是一种专门用于将 XML 转换为 HTML 的工具，它的目的是为半结构化数据（树型模型）的转换、查询提供一种通用的实现机制，其输出结果并不仅局限于 HTML，可以是任何所需的文本格式（比如 XML）。

XSLT 文档基本结构

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

XSLT的根元素

**XSLT使用的命名空
间的声明**

XSLT版本号

3. XSLT 的模板

- 模板是 XSLT 转换工作的具体规则，所有的转换任务是通过一系列的模板体现出来的。因此，如何声明和调用模板，是 XSLT 转换任务的关键。
- 在 XSLT 文档中通常至少包含一个模板，即 `xsl:template match= "/"`，该模板用于处理文档节点（Document Node），相当于 Java 中的 `"main(...)"` 函数。

XSLT的模板

- 模板在很多方面都与函数非常类似。在使用上，必须首先声明（可以指定模板的名称、参数、返回类型等等），然后在合适的地方显式地调用该模板（在遍历文档树节点的过程中、或者直接调用），才能够执行相应的转换规则。

XSLT的模板

- 模板完整的声明语法：

```
<xsl:template match=Pattern
               name=QName
               mode=QName
               priority=Number
               as=Sequence-type>
  <!-- other xsl elements and literal result elements -->
  .....
</xsl:template>
```

- `<xsl:template>` 和 `</xsl:template>` 之间的内容相当于一个函数的函数体，表示在调用该模板时应该执行的具体操作。而 `xsl:template` 元素开始标记中的属性 `match`、`name`、`mode` 和 `priority` 则用于描述该模板的相关信息，下面对这些属性进行了详细的介绍。

3.1 模板的匹配路径属性 match

- 在 XSLT 中，模板的调用分为两种方式：
 - 根据模板的匹配路径（在遍历的过程中）进行调用，具体有两种情况：
 - 对于模板 `xsl:template match="/"`，XSLT 处理器将在碰到 XML 文档的文档节点时自动调用该模板；就好像作为程序执行的入口，Java 虚拟机自动调用主类的 `main(...)` 方法。
 - 对于其他的模板 `match=other-pattern`，将在模板 `xsl:template match= "/"` 的转换规则（函数体）中通过指出匹配路径的方式（使用 `xsl:apply-templates`）进行隐式地、或者显式地调用；
 - 根据模板名称属性（`name`），使用 `xsl:call-template name=template-name` 进行调用。

match 属性的示例

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0" >
<!-- transform the input root (/) and the Person element-->
  <xsl:template match="/">
    ...
  </xsl:template>
  <xsl:template match="Person">
    ...
  </xsl:template>
</xsl:stylesheet>
```

模板1

模板2

- match 属性值可以使用绝对路径表达式（比如 `"/Person"`），这表示在遍历文档树的过程中碰到 `/ Person` 元素时，需要执行该模板中的内容。如果使用相对路径表达式，模板 2 将应用于 `/ Person`、`/*/ Person` 等等（所有的 `Person` 元素）

3.2 模板的名称属性 name

- 前面的模板 `<xsl:template match= "/" >` 和 `<xsl:template match= "Person" >`，它们都没有具体的名称，因此将其称为无名模板。
- 这些模板之所以可以没有名称，是因为它们的调用是在遍历文档树的过程中自动进行的，根本不需要名称。

命名模板的示例

- 也可以使用 name 属性为模板指定一个名称，使其成为命名模板。

```
<xsl:stylesheet version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/" name="one">                                命名模板 one
    <xsl:call-template name="another"/>
  </xsl:template>
  <xsl:template name="another">                                       命名模板 another
    <output>Simple output</output>
  </xsl:template>
</xsl:stylesheet>
```

输出结果为：`<output>Simple output</output>`

- 对于模板 one 来说，使用 name 属性指定名称 one 是有作用的，因为当调用模板 one 时，XSLT 引擎会根据路径匹配模板 one，然后使用 xsl:call-template 直接调用名为 another 的模板。在模板 another 执行结束之后，继续执行模板 one 的后续指令，这个过程非常类似于函数的调用。

3.3 模板的模式属性 mode

- 模板的 mode 属性可用于对模板进行进一步的标识。

```
<xsl:template match="Name" mode="C"> 模板 1  
    ...Do sth for Company Name...  
</xsl:template>  
<xsl:template match="Name" mode="P"> 模板 2  
    ...Do sth for Person Name...  
</xsl:template>
```

```
<Company>  
  <Name>Acme</Name>  
  <Person>  
    <Name>Dave</Name>  
    <Phone>123</Phone>  
  </Person>  
</Company>
```

- 文档中有两种 Name 元素，一种表示 Company 的名称，一种表示 Person 的名称。而对于不同的 Name 元素，如果希望使用不同的模板来进行处理，为了进一步地区分模板 1 和模板 2，使用了 mode 属性（分别为 "C" 和 "P"）。

3.4 模板的优先级属性 priority

- priority属性是用来表示模板的优先级。

```
<xsl:template match="Name" priority="2" >      模板 1  
    ...Do sth ...
```

```
</xsl:template>  
<xsl:template match="Name" priority="1">      模板 2  
    ...Do other things...  
</xsl:template>
```

- 碰到 Name 元素时将使用 priority 取值较大的模板。

3.5 模板的返回类型属性 as

- 模板返回类型属性 as 的取值，表示该模板应该返回的数据类型。

```
<xsl:template match="Name" as="element()">
    ...Do sth ...
</xsl:template>
```

```
<Person>
  <Name>Dave</Name>
  <Phone>123</Phone>
</Person>
```

- 表示该模板将会返回一个 XML 元素作为结果，当前其中可能包含子元素和文本内容。比如：
- 如果该模板输出多个 XML 元素组成的序列、非 XML 元素的内容、或者文本内容，那么 XSLT 处理器在执行模板的过程中将会报错。
- 模板的返回类型属性 as 是一个可选的参数，如果不指定该参数，则模块可以输出任意的文本内容。

3.6 模板的调用

1. 在遍历（广度优先遍历）的过程中匹配调用。
2. 通过名称直接调用。

1)使用 xsl:apply-templates 在广度优先、逐层向下的遍历过程中调用模板

- XSLT 中 apply-templates 元素的完整语法形式如下所示：

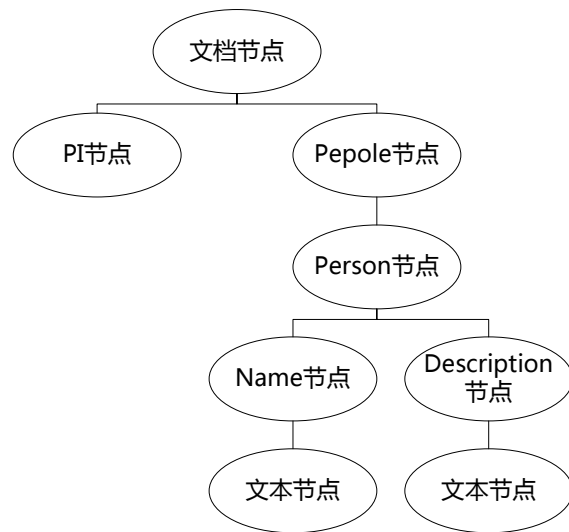
`<xsl:apply-templates select=Expression mode=QName>`

.....

`</xsl:apply-templates>`

```
<xsl:template match="/">          模板 1  
  <xsl:apply-templates />  
</xsl:template>  
<xsl:template match="Person"> 模板 2  
  ....  
</xsl:template>
```

(a) pepole.xslt 中的两个模板



(b) pepole.xml 的文档树结构

select 和 mode 属性都是可选的

**在xsl:apply-templates元素中
可以传递调用参数**

select 属性

- `<xsl:apply-templates/>` 的作用就是指定继续遍历当前节点的所有子节点（以便根据实际路径和模板的match属性取值调用对应的模板），select 属性（取值为一个 XPath 表达式）允许指定仅遍历当前节点的哪些子节点（以调用相应的模板，如果存在）。
- 如果将 (a) 的模板 1 中的 `<xsl:apply-templates/>` 更改为 `<xsl:apply-templates select= "/People/Person " />`，那么将仅调用“People的Person子元素”所对应的模板（而不会调用“处理指令”所对应的模板）。这样一来，我们就可以根据具体的转换要求，仅遍历文档树中的部分内容。

mode 属性

- 而在 `xsl:apply-templates` 元素中，`mode` 属性则用于指定需要在 `match` 属性取值相同的模板中选择哪一个进行调用。当然，需要配合使用 `xsl:template` 元素和 `xsl:apply-templates` 元素的 `mode` 属性。

```
<Company>
  <Name>Acme</Name>
  <Person>
    <Name>Dave</Name>
    <Phone>123</Phone>
  </Person>
</Company>
```

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

模板 1

```
<xsl:template match="Company">
  <xsl:apply-templates select="Name" mode="C"/>
  <xsl:apply-templates select="Person/Name" mode="P"/>
</xsl:template>
```

模板 2

```
<xsl:template match="Name" mode="C">
  ...Do sth for Company Name...
</xsl:template>
```

模板 3

```
<xsl:template match="Name" mode="P">
  ...Do sth for Person Name...
</xsl:template>
```

模板 4

当然，可以干脆为模板 3 和模板 4 的 `match` 属性取不同的值

2) 使用 xsl:call-template 调用模板

```
<xsl:call-template name = qname>  
  <!-- Content: xsl:with-param* -->  
</xsl:call-template>
```

- 使用 xsl:with-param 元素，可以在 xsl:call-template 或者 xsl:apply-templates 中传递调用参数。
- 使用 xsl:template 声明模板时，可以通过 xsl:param 来声明参数。

模板参数的声明和传值

- 在 `xsl:template` 元素开始标记和结束标记之间，使用 `xsl:param` 元素为所在的模板声明相应的模板参数。

```
<xsl:template name="doSth">  
  <xsl:param name="param1"/>  
  <xsl:param name="param2"/>  
  ...模板正文...  
  ...使用 "$param1" 和 "$param2" 引用两个模板参数...  
</xsl:template>
```

模板参数的声明和传值

- 在 `xsl:call-template` 元素的开始标记和结束标记之间，可以使用 `xsl:with-param` 元素为所调用的模板传递所需的参数。

```
<xsl:call-template name="doSth">
  <xsl:with-param name="param1" select="'One'"/>
  <xsl:with-param name="param2" select="."/>
</xsl:call-template>
```

在使用 `xsl:with-param` 元素时，必须指明具体的模板参数名称，以便为其进行赋值，所以可以不按照声明时的顺序书写

可以使用 `xsl:with-param` 元素的 `as` 属性，为形式参数指定数据类型。


```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xsl:param name="person" />
  <xsl:template match="/">
    <html>
      <head>
        <title>Finding an age using an XSLT parameter</title>
      </head>
      <body>
        <xsl:apply-templates select="/Ages/Person[@name=$person]" />
      </body>
    </html>
  </xsl:template>
  <xsl:template match="Person">
    <p>The age of <xsl:value-of select="$person" /> is <xsl:value-of select="@age"/> </p>
  </xsl:template>
</xsl:stylesheet>
```

参数声明

参数传值

3.7 XSLT 中的内置模板

- 内置模板 (Built-in Templates) 是 XSLT 中的一个关键内容，对于理解 XSLT 对 XML 文档树结构的遍历方式、模板调用机制等内容来说，都是至关重要的。

对内置模板进行深入地分析

- 通过一个具体的示例来说明内置模板的存在，并观察和解释各种内置模板的含义、以及处理对象。
- empty.xslt

```
<xsl:stylesheet version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">  
</xsl:stylesheet>
```

内置模板的完整内容

```
<xsl:stylesheet version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```

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

模板 1

应用所有的元素节点和根节点

```
<xsl:apply-templates/>
```

```
</xsl:template>
```

```
<xsl:template match="text()|@*"> 模板 2
```

应用于文本和属性节点，在结果树中输出文本和属性节点的值

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

```
</xsl:template>
```

```
<xsl:template match="processing-instruction()|comment()"/> 模板 3
```

```
<xsl:template match="*/" mode="#all"> 模板 1'
```

应用于处理指令和注释

```
<xsl:apply-templates mode="#current"/>
```

```
</xsl:template>
```

```
<xsl:template match="text()|@*" mode="#all"> 模板 2'
```

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

```
</xsl:template>
```

```
<xsl:template match="processing-instruction()|comment()" mode="#all"/> 模板 3'
```

应用所有的元素节点和根节点，针对的是具有mode属性的 <xsl:apply-template/> 元素

```
</xsl:stylesheet>
```

内置模板的作用

```
<xsl:stylesheet version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <!-- transform the input root (/) and the message element-->
  <xsl:template match="message">
    .....
  </xsl:template>
</xsl:stylesheet>
```

- 内置模板的作用在于，能够让我们集中精力编写相关节点的处理模板，而无需过多地操心整个遍历过程中模板的逐层调用。比如在本示例中，关心的是如何处理 message 元素，那么只需要编写 match="message" 的模板即可，而该模板的调用，由内置模板来完成。
- 只有在用户没有自定义处理某个节点的模块规则时，才会调用内置模块中的规则，否则，用户自定义的模块规则将覆盖内置模块中的规则。

4. XSLT 模板中各种转换功能的实现

- XSLT提供了各种元素，比如 `xsl:value-of`、`xsl:text`、`xsl:for-each`、`xsl:sort` 等等可以完成各种转换功能。
- XSLT 实际上是一种基于 XML 的编程语言，在模板中灵活地使用这些转换功能，可以编写出各种各样的处理程序，甚至是递归的函数。

4.1 使用 xsl:value-of 提取文本内容

- 在对 XML 文档进行转换时，在很多情况下，需要提取其中的某些文本内容、或者根据自己的需要生成。

xsl:value-of 元素的完整语法形式为：**<xsl:value-of select = expression />**

其中，**expression** 中可以使用 XPath 表达式

```
<xsl:template match="text()|@" mode="#all">  
    <xsl:value-of select="."/>  
</xsl:template>
```

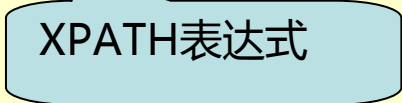
内置模板 2'

select="." 表示取当前节点的文本内容

- 对于文本节点和属性节点，xsl:value-of select="." 当然提取的是文本节点的内容和属性节点的取值；
- 对元素节点使用 <xsl:value-of select="." />，那么将得到元素节点的 String-Value。

使用 xsl:value-of 提取文本内容

```
<xsl:template match="/">
  <html>
    <body>
      <h3>Information about
        <xsl:value-of select="count(/People/Person)" /> people.</h3><br />
        <xsl:apply-templates select="/People/Person" />
      </body>
    </html>
  </xsl:template>
  <xsl:template match="Person">
    <h3><xsl:value-of select="Name" /></h3>
    <p><xsl:value-of select="Description" /></p><br />
  </xsl:template>
```



XPath表达式

4.8 使用 xsl:copy 和 xsl:copy-of

| | xsl:copy | xsl:copy-of |
|----------|--|----------------------------------|
| 功能 | 将当前节点从源文档复制到结果树 | 复制任何节点集或结果树片段到结果树 |
| 对当前节点的处理 | 仅复制当前节点，不复制其子元素或者属性 | 复制节点及其所有的子孙节点，包括属性和子元素 |
| 节点内容 | 可以使用 XSLT 代码为新的节点创建内容，如果当前节点是元素节点或者根节点 | 所有的内容都来自 select 属性中指定的节点集和结果集树片段 |

示例

```
<?xml version="1.0"?>
<Companies>
  <Company>
    <Name>Acme</Name>
    <Person>
      <Name>Dave</Name>
      <Phone>123</Phone>
    </Person>
  </Company>
</Companies>
```

(a) 示例 XML 文档 company.xml

```
<xsl:stylesheet version="2.0" xmlns:xsl=".....">
  <xsl:template match="@*|node()">
    <xsl:copy>
      <xsl:apply-templates
        select="@*|node()"/>
    </xsl:copy>
  </xsl:template>
</xsl:stylesheet>
```

(b) 使用 xsl:copy 进行转换

```
<xsl:stylesheet version="2.0" xmlns:xsl=".....">
  <xsl:template match="/">
    <xsl:copy-of select="."/>
  </xsl:template>
</xsl:stylesheet>
```

(c) 使用 xsl:copy-of 进行转换

4.3 使用 xsl:for-each

- 可以直接使用 xsl:for-each 元素循环处理批量的节点。另外，可以使用 xsl:sort 指定批量处理节点时的顺序。

```
<xsl:for-each select = Expression>  
    ...循环处理代码...  
</xsl:for-each>
```

select 属性的取值是一个 XPath 表达式，将在当前上下文中计算，确定要循环访问的节点集。**xsl:for-each** 的开始标记和结束标记之间，是循环处理的过程体，相当于 Java 中 for 循环。

xsl:for-each 可以为循环访问建立上下文（将更改原有的上下文环境）。循环体中的 XSLT 转换指令将应用于 **select** 属性所选定的节点，每个选定的节点将成为新的上下文。

4.4 xsl:sort

```
<xsl:sort select = expression data-type = { "text" | "number" | QName }  
  order = { "ascending" | "descending" } case-order = { "upper-first" | "lower-first" } />
```

① select 属性的取值是一个 XPath 表达式，表示选择进行排序的字段；

② data-type 属性的取值是数据类型的名称，表示按照某种数据类型的排序方式进行排序，可以是 "text"、"number" 或者 XML Schema 数据类型及派生数据类型。

③ order 属性的取值表示排序的方向，可以为 "ascending" 或 "descending"。

④ case-order 属性的取值表示在排序时对大小写形式的处理。

xsl:for-each 和 xsl:sort 的使用示例

```
<?xml version="1.0"?>
<Companies>
  <Company>
    <Name>Acme</Name>
    <Person>
      <Name>Dave</Name>
      <Phone>123</Phone>
    </Person>
    <Person>
      <Name>Tina</Name>
      <Phone>234</Phone>
    </Person>
    <Person>
      <Name>Jerry</Name>
    </Person>
  </Company>
</Companies>
```

```
<xsl:stylesheet version="2.0" xmlns:xsl=".....">
  <xsl:template match="/"> 模板 1
    <xsl:apply-templates select="//Company"/>
  </xsl:template>
  <xsl:template match="Company"> 模板 2
    <xsl:for-each select="Person/Name">
      <xsl:sort select="."/>
      <li><xsl:value-of select="."/></li>
    </xsl:for-each>
  </xsl:template>
</xsl:stylesheet>
```

- Dave
- Jerry
- Tina

a) 使用 xsl:for-each 进行转换

```
<xsl:stylesheet version="2.0" xmlns:xsl=".....">
  <xsl:template match="/">                                模板 1'
    <xsl:apply-templates select="//Person/Name">
      <xsl:sort select="."/>
    </xsl:apply-templates>
  </xsl:template>
  <xsl:template match="Name">                                模板 2'
    <li><xsl:value-of select="."/></li>
  </xsl:template>
</xsl:stylesheet>
```

b) 不使用 xsl:for-each 进行转换

4.5 使用 xsl:if 进行条件判断

- 使用 xsl:if 元素，可以实现条件判断，其语法形式如下所示：

```
<xsl:if test = boolean-expression>
    ...条件满足时所执行的指令...
</xsl:if>
```

- test 属性是一个必选项，表示要测试的条件。如果在强制转换为布尔值时，此属性中的表达式计算为 True，那么将执行 xsl:if 元素开始标记和结束标记之间的操作。
- ```
<xsl:stylesheet version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
 <xsl:template match="/">
 <xsl:apply-templates select="//Person/Name"/>
 </xsl:template>
 <xsl:template match="Name">
 <xsl:if test="position() != last()">
 <xsl:value-of select="."/>
 </xsl:if>
 </xsl:template>
</xsl:stylesheet>
```
- 模板 2

## 4.6 使用 xsl:choose、xsl:when 和 xsl:otherwise 进行条件分支选择

- xsl:choose、xsl:otherwise 和 xsl:when 可以进行多个条件的测试，实际上就相当于 Java 中的 switch 和 case。

```
<xsl:choose>
 <xsl:when test=case1>
 ...
 </xsl:when>
 <xsl:when test=case2>
 ...
 </xsl:when>
 <xsl:otherwise>
 ...default...
 </xsl:otherwise>
</xsl:choose>
```

```
switch (...) {
case 1:
 ...
 break;
case 2:
 ...
 break;
default:
 break;
```



```

<orders>
 <order>
 <total>9</total>
 </order>
 <order>
 <total>19</total>
 </order>
 <order>
 <total>29</total>
 </order>
</orders>

```

(a) xslchoose.xml 文档

(small) 9  
 (medium) 19  
 (large) 29

(c) 转换所得的结果

```

<?xml version="1.0"?>
<xsl:stylesheet version="1.0" xmlns:xsl=".....">
 <xsl:template match="order">
 <xsl:choose>
 <xsl:when test="total < 10">
 (small)
 </xsl:when>
 <xsl:when test="total < 20">
 (medium)
 </xsl:when>
 <xsl:otherwise>
 (large)
 </xsl:otherwise>
 </xsl:choose>
 <xsl:apply-templates/>
 </xsl:template>
</xsl:stylesheet>

```

(b) xslchoose.xslt 文档

## 4.7 在输出结果中创建 XML 节点

- 要在 XSLT 的输出结果中创建 XML 节点实际上非常容易，只需要编写相应的 XML 标记即可。
- 有时我们需要根据所处理的内容动态地生成相应的 XML 节点，而不能采取硬编码的形式。

## 4.9 使用 xsl:element 生成 XML 元素

- xsl:copy 和 xsl:copy-of 可以将当前节点从源复制到输出，但是不能根据需要生成任意的 XML 节点，而使用 xsl:element 则可以完成这项任务。

```
<xsl:stylesheet version="2.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
 <xsl:template match="message">
 <xsl:element name="new-message">
 <xsl:value-of select="."/>
 </xsl:element>
 </xsl:template>
</xsl:stylesheet>
```

## 4.10 使用 xsl:attribute

`<message>Hello!</message>` 转换为 `<message content="Hello!"/>` :

```
<xsl:template match="message">
 <xsl:copy>
 <xsl:attribute name="content">
 <xsl:value-of select="."/>
 </xsl:attribute>
 </xsl:copy>
</xsl:template>
```

## 4.11 使用 xsl:processing-instruction 和 xsl:comment

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="2.0" xmlns:xsl="...">
 <xsl:template match="/">
 <xsl:comment>
 <xsl:text>Processing instruction should appear next</xsl:text>
 </xsl:comment>
 <xsl:processing-instruction name="xml-stylesheet">
 <xsl:text>type="text/xsl" href="some.xslt"</xsl:text>
 </xsl:processing-instruction>
 <xsl:copy-of select="."/>
 </xsl:template>
</xsl:stylesheet>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<!--Processing instruction should appear next-->
<?xml-stylesheet type="text/xsl" href="some.xslt"?>
<root-element>

.....
</ root-element >
```

# 5. XSLT 中变量的声明和使用

- 作为一种编程语言，自然离不开变量的声明和使用。
- 在 XSLT 中，可以使用 `xsl:variable` 元素声明变量，并在需要的位置进行使用。XSLT 中的变量与 Java 等高级程序设计语言中的变量有相同，也有不同。
- 相同之处就是变量的基本功能，即用于保存临时的计算结果，以便使用。
- 不同之处在于，XSLT 是一种处理半结构化树型数据的语言，所以 XSLT 的变量可以保存 XML 数据或者片段，即表示结构化的信息。另外，变量一旦赋值就不能改变。

# xsl:variable 元素

```
<xsl:variable name = QName select = Expression>
.....
</xsl:variable>
```

- name 属性表示变量的名称，以进行引用，这个属性是必须的。而 select 属性是可选的，如果使用该属性，那么变量值为计算该表达式得出的结果。

```
<xsl:variable name="city" select="'Wuhan'"/>
<xsl:variable name="city">Wuhan</xsl:variable>
```

# 变量的赋值

- `xsl:variable` 元素可以通过下列两种备选方法指定变量值：
- 如果元素具有 `select` 属性，属性值必须是表达式，变量值是计算该表达式得出的结果。在这种情况下，元素的内容必须是空的。
- 如果元素没有 `select` 属性，并且包含非空的内容，例如一个或多个子节点，内容将指定该值。

```
<xsl:variable name="person">
 <Person>
 <Name>Dave</Name>
 <Phone>123</Phone>
 </Person>
</xsl:variable>
```



## 6. 创建和引用模块化的 XSLT 文档

- 在 Java 语言中，可以将多个类组织在一个包中，以便进行管理和使用。
- 在 XSLT 中，也可以在一个 XSLT 文档中编写各种模板，而在另一个 XSLT 中进行调用，这样可以增强程序的模块化特性。XSLT 提供了两个元素 `xsl:include` 和 `xsl:import`，它们允许在一个 XSLT 文档中引用另一个 XSLT 文档中定义的内容。
- 两者的区别在于，`xsl:include` 将包含的内容作为在当前 XSLT 文档中声明的内容一样对待（不能重写其中的内容），而 `xsl:import` 则将导入的内容作为附加的内容对待（可以重写其中的内容）。

## utility.xslt

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
 <xsl:variable name="author" select="'U and Me'"/>
 <xsl:template name="AddComment">
 <xsl:comment>Author: <xsl:value-of select="$author"/>
 </xsl:comment>
 <xsl:text>
</xsl:text>
 </xsl:template>
</xsl:stylesheet>
```

## test\_include.xslt

```
<xsl:stylesheet version="1.0" xmlns:xsl="...">
 <xsl:include href="utility.xslt"/>
 <xsl:template match="/">
 <output>
 <xsl:call-template name="AddComment"/>
 </output>
 </xsl:template>
</xsl:stylesheet>
```

```
<output>
 <!--Author: U and Me-->
</output>
```

## utility.xslt

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
 <xsl:variable name="author" select="'U and Me'"/>
 <xsl:template name="AddComment">
 <xsl:comment>Author: <xsl:value-of select="$author"/>
 </xsl:comment>
 <xsl:text>
</xsl:text>
 </xsl:template>
</xsl:stylesheet>
```

## test\_include.xslt

```
<xsl:stylesheet version="1.0" xmlns:xsl="...">
 <xsl:import href="Utility.xsl"/>
 <xsl:variable name="author" select="'Tom and Jerry'"/>
 <xsl:template match="/">
 <output>
 <xsl:call-template name="AddComment"/>
 </output>
 </xsl:template>
</xsl:stylesheet>
```

```
<output>
 <!--Author: Tom and Jerry-->
</output>
```