**Применение языка преобразований XSLT**

Ссылка на справочник по основным элементам языка XSLT: <https://xsltdev.ru/xslt/tr/>

**Как работает XSLT**

Модель XSLT включает в себя такие части как:

* документы XML,
* стили XSLT,
* процессор XSLT,
* выходные документы.

Документы XML являются входными данными, которые нужно преобразовать в другие документы. Документ стиля XSLT является корректным (well formed) документом XML и содержит набор правил для выполнения преобразования. Иными словами, документ стиля является шаблоном.

Процессор XSLT является приложением, которое принимает в качестве входных данных документы XML и стили XSLT. Он выполняет трансформацию, то есть применение набора правил в стилях XSLT к документам XML. Результатом этой работы являются выходные документы.

Процессоры XSLT имеют множество реализаций и встроены во многие браузеры вроде Internet Explorer, Firefox, Mozilla, Opera и другие. В Internet Explorer используется инструмент MSXML, разработанный Microsoft. XSLT-процессор встроен в Internet Explorer, начиная с версии 4.5.

**XPath**

Другой частью технологии XSL является язык XPath, предназначенный для доступа к узлам документа XML путем задания путей и выражений. Язык Xpath используется в файлах стилей для навигации внутри XML-документов, определения частей исходного XML-документа, которые совпадают с одним или более заранее заданными шаблонами. При нахождении совпадения процессор XSLT применит к нему соответствующие правила из файла стиля и преобразует его в часть результирующего документа. В файлах стилей XSLT выражения XPath используются весьма интенсивно.

**Применение XSLT**

Язык XSLT состоит из множества инструкций, записанных в виде тегов. Имя каждой инструкции обычно начинается с символов xsl. Для выполнения трансформации документ стиля XSLT должен являться корректным документом XML.

Для преобразования документа XML необходимо добавить в начало документа инструкцию, подобную следующей:

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

XSL-файл стилей обычно содержит множество элементов, самым главным из которых является элемент xsl:stylesheet. Именно он указывает, что данный XML-файл является файлом стилей. Кроме него могут содержаться другие элементы, например xsl:template, xsl:value-of. Документ XML и файл стиля передается в XSLT-процессор, который обрабатывает данные файлы, выполняет преобразование и выводит результат трансформации.

Ниже представлен документ XML, содержащий список языков программирования.

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

<languages>

<language>C#</language>

<language>Visual Basic</language>

<language>Delphi</language>

<language>Prolog</language>

</languages>

Необходимо вывести этот список в формате HTML. Для этой цели используем инструкцию xsl:for-each, которая будет применять часть шаблона к секциям документа, наименование которых указано в атрибуте select. В нашем случае укажем select="languages/language".

Файл стилей будет применяться следующий:

<?xml version="1.0" encoding="UTF-8"?>

<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform" xmlns:fo="http://www.w3.org/1999/XSL/Format">

<xsl:output method="html"/>

<xsl:template match="/">

<H4>My favorite languages:</H4>

<ul>

<xsl:for-each select="/languages/language">

<li>

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

</li>

</xsl:for-each>

</ul>

</xsl:template>

</xsl:stylesheet>

Шаблон внутри xsl:for-each выводит содержимое каждого элемента language из languages. Для этой цели используется инструкция xsl:value-of и задаваемый атрибут select=".". Это означает, что процессор должен выбирать текст содержимого текущего элемента в документе. Кроме отражения содержимого можно задавать имена конкретных тегов, а также атрибутов для выборки значений, хранящихся в них. Примеры будут рассмотрены далее.

Откроем XML файл через Internet Explorer или другой браузер. Будет выведен такой текст:

**My favorite languages:**

* C#
* Visual Basic
* Delphi
* Prolog

Одним из самых главных элементов в стиле является xsl:template. Служит для определения повторно используемого шаблона и содержит правила, по которым будет преобразован документ XML. В атрибуте match содержится выражение для отбора узлов, к которым будет применен шаблон. Также может присутствовать атрибут name. В этом случае есть возможность вызывать шаблон по имени инструкцией xsl:apply-templates.

Для повторения вывода шаблона для каждого элемента документа применяется инструкция xsl:for-each. Шаблон выполняется для каждого элемента, соответствующего условию, указанному в атрибуте select.

Инструкция xsl:value-of служит для вычисления выражения, записанного в атрибуте select с последующим выводом результата в том месте, где расположен сам элемент.

**Фильтрация**

Мы рассмотрели случай, когда считываются значения каждого узла. Однако часто возникает необходимость выбирать только часть данных, то есть их надо фильтровать. Шаблоны XSLT поддерживают два способа фильтрации.

Один из них - это применение атрибута select инструкции xsl:for-each, а второй - применение атрибута match элемента xsl:template. Применение match мы рассмотрим позже, а сейчас рассмотрим select.

Изменим немного файл с данными: добавим атрибут high, обозначающий, является ли язык из нашего списка языком высокого уровня. Также расширим сам список языков.

<?xml-stylesheet type="text/xsl" href="languages4.xsl"?>

<languages>

<language high="true">C#</language>

<language high="true">Visual Basic</language>

<language high="true">Delphi</language>

<language high="true">Prolog</language>

<language high="false">Assembler</language>

<language high="true">Java</language>

<language high="true">Perl</language>

</languages>

Заметим, что значение false для атрибута high стоит только для значения "Assembler". Изменим немного файл таблицы стилей:

<?xml version="1.0" encoding="UTF-8"?>

<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform" xmlns:fo="http://www.w3.org/1999/XSL/Format">

<xsl:output method="html"/>

<xsl:template match="/">

<H4>My favorite languages:</H4>

<ul>

<xsl:for-each select="/languages/language[@high='true']">

<li>

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

</li>

</xsl:for-each>

</ul>

</xsl:template>

</xsl:stylesheet>

В секции [@high='true'] мы указываем, что выбирать следует только те узлы документа, у которых атрибут high имеет значение 'true'. Знак @ является символом, указывающим на то, что после него стоит имя атрибута.

Посмотрим на результат:

**My favorite languages:**

* C#
* Visual Basic
* Delphi
* Prolog
* Java
* Perl

Как видим, значение "Assembler" не отображается в списке языков, то есть процессор XSLT отфильтровал данные согласно заданным условиям.

**Сортировка**

Кроме фильтрации другой часто применяемой операцией при генерации кода является сортировка. Одним из способов сортировки является применение элемента xsl:sort путем вложения в xsl:for-each или в xsl:apply-templates. Элемент xsl:sort имеет атрибуты select и order, которые указывают на элементы, по которым должна осуществляться сортировка и на порядок сортировки соответственно.

Рассмотрим немного измененный вариант документа - вместо атрибута high будем использовать элемент level, принимающий значения high или low. А имя языка запишем в элемент name.

?xml version="1.0" encoding="UTF-8"?>

<languages>

<language>

<name>C#</name>

<level>high</level>

</language>

<language>

<name>Visual Basic</name>

<level>high</level>

</language>

<language>

<name>Delphi</name>

<level>high</level>

</language>

<language>

<name>Prolog</name>

<level>high</level>

</language>

<language>

<name>Assembler</name>

<level>low</level>

</language>

<language>

<name>Java</name>

<level>high</level>

</language>

<language>

<name>Perl</name>

<level>high</level>

</language>

</languages>

В следующей таблице стилей в инструкцию xsl:for-each вложим элемент xsl:sort с атрибутами select="name" и order="ascending". В рамках указанных значений применение тела шаблона к набору элементов language осуществляется последовательно в соответствии с возрастанием значения элемента name.

<?xml version="1.0" encoding="UTF-8"?>

<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform" xmlns:fo="http://www.w3.org/1999/XSL/Format">

<xsl:output method="html"/>

<xsl:template match="/">

<H4>My favorite languages:</H4>

<ul>

<xsl:for-each select="/languages/language[./level='high']">

<xsl:sort select="name" order="ascending"/>

<li>

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

</li>

</xsl:for-each>

</ul>

</xsl:template>

</xsl:stylesheet>

В атрибуте select мы фильтруем по значению элемента level. Также в атрибуте select инструкции xsl:value-of указываем непосредственно имя элемента.

Получается такой результат.

**My favorite languages:**

* C#
* Delphi
* Java
* Perl
* Prolog
* Visual Basic

Названия языков отсортированы в алфавитном порядке и значение "Assembler" не отображается в списке.

**Вложенные шаблоны**

В сложных случаях возникает необходимость применения вложенных друг в друга шаблонов. Рассмотрим документ, в котором содержатся таблицы и их поля вместе с названиями.

<?xml version="1.0" encoding="UTF-8"?>

<database>

<table name="book">

<field>id</field>

<field>title</field>

<field>author\_id</field>

</table>

<table name="author">

<field>id</field>

<field>first\_name</field>

<field>last\_name</field>

</table>

</database>

В данном примере у нас будут два шаблона. Один будет основной, для таблиц, а другой будет вызываемым из него, для полей таблиц. Для вызова шаблона используется инструкция xsl:apply-templates.

<?xml version="1.0" encoding="UTF-8"?>

<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:output method="html"/>

<xsl:template match="/">

<H4>Database tables</H4>

<xsl:for-each select="database/table">

<table border="1" width="150">

<tr>

<td>

<b>

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

</b>

</td>

</tr>

<xsl:apply-templates select="field"/>

</table>

<br/>

</xsl:for-each>

</xsl:template>

<xsl:template match="field">

<tr>

<td>

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

</td>

</tr>

</xsl:template>

</xsl:stylesheet>

XSLT-процессором будет сформирован следующий HTML-код:

<H4>Database tables</H4>

<table border="1" width="150">

<tr>

<td><b>book</b></td>

</tr>

<tr>

<td>id</td>

</tr>

<tr>

<td>title</td>

</tr>

<tr>

<td>author\_id</td>

</tr>

</table><br><table border="1" width="150">

<tr>

<td><b>author</b></td>

</tr>

<tr>

<td>id</td>

</tr>

<tr>

<td>first\_name</td>

</tr>

<tr>

<td>last\_name</td>

</tr>

</table><br>

При открытии сгенерированного результата в браузере будет показано следующее:

#### Database tables

|  |
| --- |
| **book** |
| id |
| title |
| author\_id |

|  |
| --- |
| **author** |
| id |
| first\_name |
| last\_name |

Инструкция xsl:apply-templates позволяет вызывать определенный шаблон и выполнять его много раз. Кроме того она является более наглядной и удобной альтернативой xsl:for-each.

**Обработка с условием**

Для применения условий к обработке шаблонов используется одна из инструкций: xsl:if или xsl:choose. Инструкция xsl:if дает простую функциональность if-then, а инструкция xsl:choose поддерживает выбор одного из нескольких возможных вариантов. Сейчас рассмотрим пример с использованием xsl:if, а пример с использованием xsl:choose рассмотрим позже.

В следующем примере необходимо создать код объявления процедур на языке PL/SQL такого вида:

procedure prc\_save\_record (id in out number, name in varchar2, title varchar2, save\_date out date);

procedure prc\_delete\_record (id number);

Объявление состоит из ключевого слова procedure, имени процедуры, имен и типов параметров. Для каждого параметра может указываться его вид: in, out, in out или без указания его. Представим наше объявление процедур в иерархическом виде в следующем документе:

<?xml version="1.0" encoding="UTF-8"?>

<package>

<procedure name="prc\_save\_record">

<parameter name="id" type="number" in="true" out="true"/>

<parameter name="name" type="varchar2" in="true"/>

<parameter name="title" type="varchar2"/>

<parameter name="save\_date" type="date" out="true"/>

</procedure>

<procedure name="prc\_delete\_record">

<parameter name="id" type="number"/>

</procedure>

</package>

Создадим следующий файл стиля:

<?xml version="1.0" encoding="UTF-8"?>

<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:output method="html"/>

<xsl:template match="/">

<xsl:for-each select="package/procedure">

procedure <xsl:value-of select="@name"/>

(<xsl:for-each select="parameter">

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

<xsl:text> </xsl:text>

<xsl:if test="@in='true'">in </xsl:if>

<xsl:if test="@out='true'">out </xsl:if>

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

<xsl:if test="not(position()=last())">, </xsl:if>

</xsl:for-each>);

<xsl:if test="not(position()=last())">

<BR/>

<BR/>

</xsl:if>

</xsl:for-each>

</xsl:template>

</xsl:stylesheet>

Инструкция xsl:for-each здесь используется дважды, причем одна вложена в другую. В первой инструкции просматриваются все процедуры, а во второй выполняется работа над каждым параметром процедуры. В атрибуте test инструкции xsl:if делается проверка на необходимость добавления ключевых слов in, out. Кроме того, инструкция проверки используется для определения того, нужно ли ставить запятую после параметра. Если параметр не последний, что проверяется условием not(position()=last()), то ставится запятая. Функция position() возвращает номер позиции данного контекста в процессе обхода ветвей. Функция last() возвращает общее число позиций. Аналогично проверяется необходимость перехода на следующую строку в конце каждой процедуры.

Инструкция xsl:if используется для выяснения необходимости включения какого-либо шаблона в результирующий вывод. Шаблон, находящийся внутри инструкции xsl:if выполняется только в том случае, когда выражение XPath, находящееся в условии test возвращает истину. При этом аналога команды else языков программирования в XSLT не существует. Для выполнения подобной функциональности надо использовать инструкцию xsl:choose.

Элемент xsl:text служит для вывода текста в результат трансформации. Чаще всего он используется для вывода пробелов, так как без применения xsl:text пробелы будут схлопываться. Атрибут disable-output-escaping позволяет включать и выключать преобразование системных символов вроде знаков больше, меньше, амперсанта.

Инструкция xsl:choose применяется, когда есть необходимость выбора шаблона из нескольких возможных согласно каким-либо условиям. Она содержит одну или более команд xsl:when и может содержать xsl:otherwise. Каждая команда xsl:when содержит в условии test выражение. Если оно является истиной, то выполняется содержимое xsl:when. Порядок условий xsl:when имеет значение, так как результатом всей конструкции будет шаблон под первым истинным выражением. Если ни одно из условий в xsl:when не вернуло истину, в результате будет обработан шаблон под xsl:otherwise.

Например:

<xsl:choose>

<xsl:when test="$a=l">Истина</xsl:when>

<xsl:when test="$a=0">Ложь</xsl:when>

<xsl:otherwise>Некорректное значение</xsl:otherwise>

</xsl:choose>

**Переменные**

Инструкция xsl:variable служит для объявления объектов наподобие переменной. Строго говоря, эти элементы являются не переменными, а скорее инициализируемыми во время запуска обработки шаблона константами. Но в контексте нашего курса мы будем называть их переменными.

Они позволяют определить заменяемое значение, которое можно использовать в таблице стилей при трансформации.

<xsl:variable name="Имя переменной" select="выражение" as="тип" />

Значение переменной может быть также получено другими путями, например:

<xsl: variable name="a">8</xsl:variable>

Или так:

<xsl:variable name="str">

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

<xsl:variable>

Для подстановки значения переменной надо предварить имя переменной знаком $. Процессор XSLT в ходе преобразования присваивает ей значение. Рассмотрим XML-документ из предыдущего примера.

<?xml version="1.0" encoding="UTF-8"?>

<package>

<procedure name="prc\_save\_record">

<parameter name="id" type="number" in="true" out="true"/>

<parameter name="name" type="varchar2" in="true"/>

<parameter name="title" type="varchar2"/>

<parameter name="save\_date" type="date" out="true"/>

</procedure>

<procedure name="prc\_delete\_record">

<parameter name="id" type="number"/>

</procedure>

</package>

Теперь поставим задачу вывести для каждой процедуры число параметров внутри объявления процедуры. Для этой цели объявим в инструкции xsl:variable переменную amount и присвоим значение count(parameter) - для каждого узла procedure будет считать число узлов parameter. Кроме того при выводе мы собираемся выводить слово параметр в правильном числе (выводим одно из трех: параметр, параметра, параметров). Для этого мы в переменную modulo10 будем записывать остаток от деления amount на 10. А инструкция xsl:choose добавлена для выборки одного из вариантов написания слова параметр.

<?xml version="1.0" encoding="UTF-8"?>

<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:output method="html" encoding="windows-1251"/>

<xsl:template match="/">

<xsl:for-each select="package/procedure">

<xsl:variable name="amount">

<xsl:value-of select="count(parameter)"/>

</xsl:variable>

<xsl:variable name="modulo10" select="$amount mod 10"/>

Процедура

<xsl:value-of select="@name"/> имеет

<xsl:value-of select="$amount"/>

<xsl:choose>

<xsl:when test="$amount &gt; 10 and $amount &lt; 20"> параметров</xsl:when>

<xsl:when test="$modulo10=1"> параметр</xsl:when>

<xsl:when test="($modulo10 &gt; 1 and $modulo10 &lt; 5)"> параметра</xsl:when>

<xsl:otherwise> параметров</xsl:otherwise>

</xsl:choose>

<BR/>

</xsl:for-each>

</xsl:template>

</xsl:stylesheet>

Результат

Процедура prc\_save\_record имеет 4 параметра  
 Процедура prc\_delete\_record имеет 1 параметр

**Включение внешнего содержимого**

Инструкция xsl:import используется для подстановки содержимого другого файла стилей. Ее синтаксис выглядит так:

<хsl:import href="адрес URI" />

Она должна быть первым вложенным элементом в <xsl:stylesheet> или <xsl:transform>. В случае наличия конфликтов в стилях, применяются правила, которые записаны позже других.

Аналогом этой инструкции является инструкция xsl:include. Она также должна быть вложена в <xsl:stylesheet> или <xsl:transform>, однако отличие заключается в том, может подставляться в любое место внутри этих элементов и порядок расположения инструкции xsl:include не играет роли. Ее синтаксис выглядит аналогично:

<xsl:include href="адрес URI" />

**Улучшение функциональности**

Есть возможность расширять существующие способности XSLT, например, добавлять скрипты, которые понимает процессор XSLT.

Элемент xsl:function позволяет описывать самые настоящие пользовательские функции. Имя функции записывается в обязательном атрибуте nаmе, (аргументы функции задаются элементами xsl:param, а тело функции - это конструктор последовательности, записанный в содержимом элемента xsl:function. Результатом работы функции будет последовательность, созданная конструктором. Функция может вызываться рекурсивно.

**Генерация в C#**

Для выполнения преобразования XML-файлов средствами .Net используется класс XslCompiledTransform, являющийся процессором XSLT и поддерживающий XSLT версии 1.0. Метод Load класса загружает и компилирует стиль. Метод Transform выполняет трансформацию.

using System;

using System.Xml;

using System.Xml.Xsl;

using System.Xml.XPath;

namespace LectionXML

{

class Program

{

static void Main(string[] args)

{

XPathDocument xpd = new XPathDocument(@"variable.xml");

XslCompiledTransform xt = new XslCompiledTransform();

XmlTextWriter xtw = new XmlTextWriter(@"variable.txt",null);

xt.Load(@"variable.xsl");

xt.Transform(xpd, null, xtw);

xtw.Close();

}

}

}

Подлежащий преобразованию документ XML загружается в объект класса XPathDocument. А класс XslCompiledTransform используется для выполнения преобразования. Объект класса XmlTextWriter применяется для вывода результата в выходной файл.

Однако существует более короткий и простой альтернативный вариант с применением одного только класса XslCompiledTransform. Вместо всех приведенных выше объявлений и вызовов нужно написать всего три строчки кода.

XslCompiledTransform xt = new XslCompiledTransform();

xt.Load("variable.xsl");

xt.Transform("variable.xml", "variable.txt");

**Пример преобразования XML в XML**

Исходный XML-файл:

<?xml version="1.0"?>

<persons>

<person username="MP123456">

<name>Иван</name>

<surname>Иванов</surname>

</person>

<person username="PK123456">

<name>Пётр</name>

<surname>Петров</surname>

</person>

</persons>

Таблица стилей XSLT:

<?xml version="1.0"?>

<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">

<xsl:output method="xml" indent="yes" encoding="windows-1251"/>

<xsl:template match="persons">

<transform>

<xsl:apply-templates/>

</transform>

</xsl:template>

<xsl:template match="person">

<record>

<xsl:apply-templates select="@\*|\*"/>

</record>

</xsl:template>

<xsl:template match="@username">

<username>

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

</username>

</xsl:template>

<xsl:template match="name">

<fullname>

<xsl:apply-templates/>

<xsl:apply-templates select="following-sibling::surname" mode="fullname"/>

</fullname>

</xsl:template>

<xsl:template match="surname"/>

<xsl:template match="surname" mode="fullname">

<xsl:text> </xsl:text>

<xsl:apply-templates/>

</xsl:template>

</xsl:stylesheet>

Выходной XML-документ:

<?xml version="1.0" encoding="windows-1251"?>

<transform>

<record>

<username>MP123456</username>

<fullname>Иван Иванов</fullname>

</record>

<record>

<username>PK123456</username>

<fullname>Пётр Петров</fullname>

</record>

</transform>