Web-технологии 3

Преобразования XML
(XSL Transformations, XSLT)

Введение

- Семейство рекомендаций Extensible Stylesheet Language (XSL, расширяемый язык таблиц стилей)
  - XSL трансформации (XSL Transformations, XSLT)
  - форматирующие объекты XSL (XSL Formatting Objects, XSL FO)
- Форматирование и преобразование XML документов
- XSL – Server-side технология
- Cascading Stylesheets (CSS) – client-side технология

XSLT

- XML приложение, определяющее правила преобразования одного XML в другой
- XSLT документ = таблица стилей XSLT = набор шаблонов
- Сравнение элементов дерева XML с шаблонами
- Выходное дерево – набор содержимого из шаблонов
- Использование XPath для навигации
- Работа с корректными XML документами
- Поддержка схем, DTD и пространств имен

Структура XSLT

- Таблица стилей XSLT - XML документ
- Корневой элемент **stylesheet** или **transform**
- Пространство имен http://www.w3.org/1999/XSL/Transform
- Рекомендуемый префикс **xsl**
- Пример

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

- Результат – текст XML документа без разметки

xml-stylesheet

- Подключение XSLT документа в XML документе
- Пример

```
<?xml version="1.0"?>
<?xml stylesheet type="text/xml"
href="http://www.oreilly.com/styles/people.xsl"?>
<people>
  …
</people>
```

Шаблоны

- Каждый шаблон представлен элементом xsl:template
- Атрибут **match** – выражение XPath к входным данным
- Тело элемента – шаблон для замены
- Пример:

  <xsl:template match="person">Человек</xsl:template>
- Результат:

  <?xml version="1.0" encoding="utf 8"?>

  Человек

  Человек
- Можно:

  <xsl:template match="person"> Человек </xsl:template>
- Нельзя:

  <xsl:template match="person"> Человек </xsl:template>

xsl:value-of

- Выборка содержимого из входных элементов и подстановка в шаблон
- Результат – строковое значение
- Атрибут **select** – выражение XPath исходного значения
- Пример:

```
<xsl:template match="person">
  <p> <xsl:value-of select="name"/> </p>
</xsl:template>
```

- Результат:

```
<p>
    Алан
    Тьюринг
</p>
```

xsl:apply-templates

- По умолчанию прямой порядок разбора
- Шаблон родителя активизируется до шаблона потомков
- Изменение порядка обхода
- Пример (перестановка фамилии и имени):

```
<xsl:template match="name">
  <xsl:value-of select="last_name"/>,
  <xsl:value-of select="first_name"/>
</xsl:template>
<xsl:template match="person">
  <xsl:apply-templates select="name"/>
</xsl:template>
```

- Результат:

```
Тьюринг,
  Алан
```

- Не имеет значения порядок шаблонов в таблице стилей
- Имеет значение порядок элементов во входном документе
- Пример (размещение дочерних элементов внутри родительского):

```
<xsl:template match="people">
  <html>
    <head><title>Знаменитые ученые</title></head>
    <body>
      <xsl:apply-templates select="person"/>
    </body>
  </html>
</xsl:template>
```

Встроенные шаблонные правила

- 7 видов узлов XML:
  - Корневой узел,
  - Узлы элементов,
  - узлы атрибутов,
  - текстовые узлы,
  - узлы комментариев,
  - узлы инструкций обработки
  - узлы пространств имен
- Каждый вид имеет встроенные шаблонные правила по умолчанию

Узлы атрибутов и текстовые узлы

- Шаблон по умолчанию:

```
<xsl:template match="text()|@*">
  <xsl:value of select="."/>
</xsl:template>
```

- Текстовый узел – текст
- Атрибут – значение атрибута **(по умолчанию не обрабатывается!)**
- Пример:

```
<xsl:template match="person">
  <dt><xsl:apply-templates select="name"/></dt>
  <dd><ul>
    <li>Родился: <xsl:apply-templates select="@born"/></li>
    <li>Умер: <xsl:apply-templates select="@died"/></li>
  </ul></dd>
</xsl:template>
```

Узлы элементов и корневой узел

- Шаблон по умолчанию

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

- XSLT применяет шаблоны ко всем узлам, кроме узлов атрибутов и пространств имен
- Если правило переопределяет шаблон для элемента, то дочерние элементы могут и не обрабатываться!

Комментарии, инструкции и пространства имен

- Шаблон по умолчанию

  <xsl:template match="processing instruction()| comment()"/>
- В XPath не существует выражения для пространств имен
- Префикс конвертируется в абсолютный путь

Режимы

- Многократное использование входных элементов
- Атрибут **mode** для указания режима
- Элемент xsl:apply-templates активизирует только шаблоны с соответствующим значением атрибута
- Пример:

```
<xsl:template match="people">
<html>
<head><title>Знаменитые ученые</title></head>
<body>
  <ul><xsl:apply templates select="person" mode="toc"/></ul>
  <xsl:apply templates select="person"/>
</body>
</html>
</xsl:template>
```

```
<!-- Шаблоны режима оглавления -->
<xsl:template match="person" mode="toc">
  <xsl:apply-templates select="name" mode="toc"/>
</xsl:template>

<xsl:template match="name" mode="toc">
  <li><xsl:value-of select="last_name"/>,
  <xsl:value of select="first_name"/></li>
</xsl:template>

<!-- Шаблоны обычного режима -->
<xsl:template match="person">
  <p><xsl:apply-templates/></p>
</xsl:template>
</xsl:stylesheet>
```

```
<html>
<head><title>Знаменитые ученые</title></head>
<body>
<ul>
<li>Тьюринг, Алан</li>
<li>Фейнман, Ричард</li>
</ul>
<p>
Алан
Тьюринг
специалист по информатике
математик
криптограф
</p>
</body>
</html>
```

Шаблоны значений атрибутов

- Статическое определение атрибута

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

- Динамическое значение – использование XPath
- Пример:

```
<xsl:template match="name">
<name first="{first_name}"
initial="{middle_initial}"
last="{last_name}" />
</xsl:template>
```

- Результат:

```
<name first="Ричард" initial="Ф" last="Фейнман"/>
```

Пространства имен

- Анализ элементов в связке с пространствами имен

```
<people xmlns="http://namespaces.oreilly.com/people">
  <person born="1912" died="1954">
  …
  </person>
</people>
```

- Пример шаблона

```
<xsl:template match="name">
  <p><xsl:value-of select="last_name"/>,
  <xsl:value-of select="first_name"/></p>
</xsl:template>
```

- Шаблон не применим!

- Правильный шаблон

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
   xmlns:pe="http://namespaces.oreilly.com/people">
...
<xsl:template match="pe:name">
   <p><xsl:value-of select="pe:last_name"/>,
   <xsl:value-of select="pe:first_name"/></p>
</xsl:template>
```

Категории элементов XSLT

- Два корневых элемента
- 13 элементов верхнего уровня
- 20 элементов инструкций
- Элементы-расширения XSLT-процессоров
- Собственные расширения элементов

xsl:apply-templates

- Применение шаблона с наибольшим приоритетом
- Синтаксис

```
<xsl:apply-templates
select = "выражение набор узлов"
mode = "УточненноеИмя">
<!-- (xsl:sort | xsl:with-param)* -->
</xsl:apply templates>
```

- **select**, необязательный, выражение XPath
- **mode**, необязательный, фильтр шаблонов

xsl:attribute

- Добавление атрибута к результату
- Синтаксис

```
<xsl:attribute
name = "УточненноеИмя"
namespace = "URI">
<!-- шаблон для значения атрибута -->
</xsl:attribute>
```

- **name**, обязательный, Имя атрибута
- **namespace**, необязательный URI пространства имен

Пример

- xslt:

```
<IMG>
  <xsl:attribute name="src">
    <xsl:value-of
select="imagenames/imagename" />
  </xsl:attribute>
</IMG>
```

- Результат:

```
<IMG src="imagenames/imagename"/>
```

xsl:attribute-set

- Определение коллекции атрибутов
- Синтаксис

  <xsl:attribute-set

  name = "УточненноеИмя"

  use-attribute-sets = "УточненныеИмена">

  <!-- xsl:attribute* -->

  </xsl:attribute set>
- **name**, обязательный, имя набора
- **use-attribute-sets**, необязательный, добавление других наборов (через пробел)

Пример

- xslt:

```
<xsl:attribute-set name="title-style">
  <xsl:attribute name="font-size">12pt</xsl:attribute>
  <xsl:attribute name="font-weight">bold</xsl:attribute>
</xsl:attribute-set>
```

xsl:call-template

- Вызов шаблона по имени, в т.ч. циклический
- Синтаксис

  <xsl:call-template

  name = "УточненноеИмя">

  <!-- xsl:with-param* -->

  </xsl:call-template>
- **name**, обязательный, имя шаблона

xsl:comment

- Вставка комментария в результат
- Синтаксис

  <xsl:comment>

  <!-- шаблон -->

  </xsl:comment>
- Пример:

  <xsl:comment>insert top news story</xsl:comment>
- Результат:
- <!--insert top news story-->

xsl:decimal-format

- Шаблон для преобразования числа в строку
- Синтаксис

```
<xsl:decimal-format
name = "имя"
decimal-separator = "символ"
grouping-separator = "символ"
infinity = "строка"
minus-sign = "символ"
NaN = "строка"
percent = "символ"
per-mille = "символ"
zero-digit = "символ"
digit = "символ"
pattern-separator = "символ"/>
```

- name – имя
- decimal-separator – отеделение целой от дробной
- grouping-separator – разделение групп цифр
- infinity - бесконечность
- minus-sign – перед отрицательным числом
- NaN – нечисловое значение
- percent – процент
- per-mille – промилле
- zero-digit – символ нуля, остальные цифры – следующие символы
- digit – цифра в шаблоне
- pattern-separator – разделитель положительного и отрицательного подшаблонов

Пример

- xslt:

  <xsl:decimal-format name="us" decimal-separator='.' grouping-separator=',' />

  …

  <td><xsl:value-of select="format-number(24535.2, '###,###.00', 'us')"/></td>

  …

- Результат:

  <td>24,535.20</td>

xsl:element

- Вставка элемента в результат
- Спецификация

```
<xsl:element
name = "имя"
namespace = "URI"
use-attribute-sets = "Имя1 Имя2...">
<!-- шаблон -->
</xsl:element>
```

Пример

- xslt:

```
<xsl:template match="item">
  <xsl:element name="xsl:template">
    <xsl:attribute name="match">cost</xsl:attribute>
    <xsl:attribute name="xml:space">preserve</xsl:attribute>
    <xsl:apply-templates/>
  </xsl:element>
</xsl:template>
```

- Результат:

```
<xsl:template match="cost"
    xml:space="preserve"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
My Item</xsl:template>
```

xsl:copy

- Копирует текущий узел из источника в вывод
- Синтаксис:

```
<xsl:copy
  use-attribute-sets = QNames>
<!-- шаблон -->
</xsl:copy>
```

- **use-attribute-sets** – список атрибутов через пробел

Пример (копирование дерева)

- xslt:

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

- Результат:

```
<catalog><book id="bk101"><author>Gambardella, Matthew</author><title>XML Developer's Guide</title><genre>Computer</genre><price>44.95</price><publish_date>2000 -10-01</publish_date><description>An in-depth look at creating applications with XML.</description></book><book id="bk102">...</book>
...</catalog>
```

xsl:copy-of

- Вставляет в результирующее дерево поддеревья и фрагменты результирующего дерева
- Синтаксис:
- <xsl:copy-of
- select = "выражение" / >
- Копирует также все дочерние элементы, атрибуты, пространства имен и потомки этих узлов
-

Пример

- xslt:

```
<xsl:template match="person">
  <p>
    <xsl:copy-of select="given-name"/>
    <xsl:text> </xsl:text>
    <xsl:copy-of select="family-name"/>
  </p>
</xsl:template>
```

- Результат:

```
<p><given-name age="10">
<name>Fred</name>
<nick-name>Freddy</nick-name>
</given-name>
```

xsl:if

- условные фрагменты шаблонов
- Синтаксис

```
<xsl:if
test = "логическое выражение">
<!-- шаблон -->
</xsl:if>
```

- **test**, обязательный, выражение XPath

Пример

- xslt:

```
<xsl:template match="namelist/name">
  <xsl:apply-templates/>
  <xsl:if test="position()!=last()">, </xsl:if>
</xsl:template>
```

- Результат:

  Albert, Terrance, Will, Sylvia, Timothy, Gordon, James, Robert, Dan, Sasha

xsl:for-each

- обходит все узлы, указанные атрибутом **select**, и применяет шаблоны к каждому из них
- Синтаксис

  <xsl:for-each

  select = "выражение набор узлов">

  <!-- (xsl:sort*, шаблон) -->

  </xsl:for-each>
- **select**, обязательный, выражение XPath

Пример

- xslt:

```
<xsl:for-each select="customers/customer">
  <TR>
    <TD><xsl:value-of select="name" /></TD>
    <TD><xsl:value-of select="address" /></TD>
  </TR>
</xsl:for-each>
```

- Результат:

```
<TR>
<TD>Albert Aikens</TD>
<TD>368 Elm St.</TD>
</TR>
<TR>
…
```

xsl:choose, xsl:when и xsl:otherwise

- Проверка нескольких условий
- Синтаксис:

```
<xsl:choose>
<!-- (xsl:when+, xsl:otherwise?) -->
</xsl:choose>
<xsl:when
test = "логическое выражение">
<!-- шаблон -->
</xsl:when>
<xsl:otherwise>
<!-- шаблон -->
</xsl:otherwise>
```

Пример

```
<xsl:template match="order">
  <xsl:choose>
    <xsl:when test="total &lt; 10">
      (small)
    </xsl:when>
    <xsl:when test="total &lt; 20">
      (medium)
    </xsl:when>
    <xsl:otherwise>
      (large)
    </xsl:otherwise>
  </xsl:choose>
  <xsl:apply-templates />
  <BR/>
</xsl:template>
```

Результат примера

(small) 9

(medium) 19

(large) 29

xsl:fallback

- обработка элементов XSLT, которые не могут быть обработаны синтаксическим анализатором: например, элементов, являющихся частью новой версии или нераспознанного расширения
- Синтаксис:

```
<xsl:fallback>
<!-- шаблон -->
</xsl:fallback>
```

Пример

```
<xsl:import-table href="blah.html" name="sample">
  <xsl:fallback>
  <xsl:comment>This version of the parser does not support the creation of a
     table with the 'xsl:import-table' element, so the following
     table has been generated using the 'fallback' element.</xsl:comment>
     <xsl:for-each select='records/record'>
       <tr>
         <td><xsl:value-of select='name'/></td>
         <td><xsl:value-of select='address'/></td>
         <td><xsl:value-of select='phone'/></td>
       </tr>
     </xsl:for-each>
  </xsl:fallback>
</xsl:import-table>
```

xsl:import

- Импорт таблицы стилей XSLT
- Синтаксис

  <xsl:import href = "URI" />
- **href** – Обязательный, URI-ссылка
- Допустим, выполняются следующие условия:
  - XSLT-файл A импортирует XSLT-файлы B и C, именно в таком порядке.
  - XSLT-файл B импортирует XSLT-файл D.
  - XSLT-файл C импортирует XSLT-файл E.
- Приоритеты импорта (в порядке возрастания): D, B, E, C, A.
- Нельзя импортировать самого себя!

xsl:apply-imports

- Вызывает переопределенное правило шаблона, заданное в импортированной таблице стилей
- Синтаксис:

  <xsl:apply-imports />
- обрабатывает текущий узел, используя только те шаблоны, которые были импортированы в таблицу стилей с помощью xsl:import

Пример

- xslt:

```
<xsl:import href="arith.xsl"/>
<xsl:import href="str.xsl"/>
<xsl:template match="op">
  <xsl:value-of select="operand[1]"/>
  <xsl:value-of select="@symbol"/>
  <xsl:value-of select="operand[2]"/>
  = <xsl:apply-imports/>
  <br/>
</xsl:template>
```

- arith.xsl:

```
<xsl:template match="op[@symbol='+']">
  <xsl:value-of select="sum(operand)"/> (from arith.xsl)
</xsl:template>
<xsl:template match="op[@symbol='-']">
  <xsl:value-of select="number(operand[1])-number(operand[2])"/>
 (from arith.xsl)
</xsl:template>
<xsl:template match="op[@symbol='*']">
  <xsl:value-of select="number(operand[1])*number(operand[2])"/>
  (from arith.xsl)
</xsl:template>
```

- str.xsl:

```
<xsl:template match="desc">
  <DIV><xsl:value-of select="."/></DIV>
</xsl:template>
<xsl:template match="op[@name='add']">
  <xsl:value-of select="operand[1]"/>
  <xsl:value-of select="operand[2]"/> (from str.xsl)
</xsl:template>
<xsl:template match="op[@name='mul']">
  <xsl:value-of select="operand[2]"/>
  <xsl:value-of select="operand[1]"/> (from str.xsl)
</xsl:template>
```

- xml:

```
<ops>
  <desc>Some binary operations</desc>
  <op name="add" symbol="+">
    <operand>1</operand>
    <operand>2</operand>
  </op>
  <op name="sub" symbol="-">
    <operand>1</operand>
    <operand>2</operand>
  </op>
  <op name="mul" symbol="*">
    <operand>1</operand>
    <operand>2</operand>
  </op>
</ops>
```

Результат

Some binary operations

1+2 = 12 (from str.xsl)

1-2 = -1 (from arith.xsl)

1*2 = 21 (from str.xsl)

- Если поменять порядок импорта:

  Some binary operations

  1+2 = 3 (from arith.xsl)

  1-2 = -1 (from arith.xsl)

  1*2 = 2 (from arith.xsl)

- Если не использовать xsl:apply-imports

  Some binary operations

  1+2 =

  1-2 =

  1*2 =

xsl:include

- Добавление содержимого таблицы стилей
- Нельзя использовать рекурсивно (A-B-A)
- Нет приоритета в включенной и подключаемой таблицах стилей
- Синтаксис

  <xsl:include href = "URI" / >
- **href** – Обязательный, URI-ссылка

Пример

```
<xsl:stylesheet>
<xsl:template match="TITLE">
  <DIV STYLE="color:blue">
    Title: <xsl:value-of select="."/>
  </DIV>
</xsl:template>
<xsl:include href="xslincludefile.xsl" />
</xsl:stylesheet>

<xsl:stylesheet>
<xsl:template match="TITLE">
   Title - <xsl:value-of select="."/><BR/>
</xsl:template>
</xsl:stylesheet>
```

Результат

Title - Lover Birds

Title - Catwings

Title - Splish Splash

- Если шаблоны с одинаковым приоритетом, то берется последний

xsl:key

- Определение ключа, на который можно сослаться
- Синтаксис:

  <xsl:key

  name = "УточненноеИмя"

  match = "маска"

  use = "выражение" / >
- **name**, обязательный, имя ключа
- **match**, обязательный, шаблон соответствия
- **use**, обязательный, Выражение XPath, значение ключа
- Ссылка на ключ с помощью key()

Пример

- xml:

```
<?xml version='1.0'?>
<?xml-stylesheet type="text/xsl" href="key_sample.xsl" ?>
<titles>
   <book title="XML Today" author="David Perry"/>
   <book title="XML and Microsoft" author="David Perry"/>
   <book title="XML Productivity" author="Jim Kim"/>
</titles>
```

- xsl:

```
<?xml version='1.0'?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform" >
<xsl:key name="title-search" match="book" use="@author"/>
<xsl:template match="/">
  <HTML> <BODY>
      <xsl:for-each select="key('title-search', 'David Perry')">
       <div> <xsl:value-of select="@title"/> </div>
     </xsl:for-each>
  </BODY> </HTML>
</xsl:template>
</xsl:stylesheet>
```

- Результат:

<HTML> <BODY>

XML Today

XML and Microsoft

</BODY> </HTML>

xsl:message

- Отправка сообщения XSLT процессору
- Обычное использование – отладочная информация
- Синтаксис:

```
<xsl:message
terminate = "yes" | "no">
<!-- шаблон -->
</xsl:message>
```

- **terminate**, необязательный, прерывание работы
- Содержимое - шаблон

xsl:namespace-alias

- Заменяет префикс, связанный с данным пространством имен, другим префиксом.
- Пример использования: генерация xslt документа (проблема идентичных имен пространств)
- Синтаксис

```
<xsl:namespace-alias
stylesheet-prefix = "префикс"
result-prefix = "префикс" />
```

Пример xsl:

```
<?xml version='1.0'?>
<xsl:stylesheet version="1.0"
      xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
      xmlns:alt="http://www.w3.org/1999/XSL/Transform-alternate">
<xsl:namespace-alias stylesheet-prefix="alt" result-prefix="xsl"/>
<xsl:template match="/">
  <alt:stylesheet>
 <alt:import href="IERoutines.xsl"/>
            <alt:template match="/">
               <div> <alt:call-template name="showTable"/> </div>
            </alt:template>
</alt:stylesheet>
</xsl:template>
</xsl:stylesheet>
```

Результат

```
<?xml version="1.0" encoding="UTF-16"?>
<xsl:stylesheet
     xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:import href="IERoutines.xsl" />
<xsl:template match="/">
  <div> <xsl:call-template name="showTable" /> </div>
</xsl:template>
</xsl:stylesheet>
```

xsl:number

- Вставляет форматированное число в результирующее дерево.
- Синтаксис

```
<xsl:number
value = "числовое выражение"
count = "маска"
from = "маска"
level = "single" | "multiple" | "any"
format = "буква или цифра"
lang = "код_языка"
letter-value = "alphabetic" | "traditional"
grouping-separator = "символ"
grouping-size = "число" / >
```

- **value**, необязательный, выражение XPath, возвращающее число, которое требуется отформатировать.
- **level**, необязательный, какие уровни исходного дерева следует учитывать
- **count**, необязательный, определяет, какие узлы должны вычисляться
- **from**, необязательный, шаблон начального узла
- **format**, необязательный, формат нумерации
- **lang**, необязательный, код языка
- **letter-value**, необязательный, По умолчанию traditional (I, II, III, IV, ...) или alphabetic (I, J, K, L, ...)
- **grouping-separator**, необязательный, разделитель групп цифр ( )
- **grouping-size**, необязательный, размер группы (3)

Форматы

- **1:** 1, 2, 3, 4, 5, 6, . . .
- **5:** 5, 6, 7, 8, 9, . . .
- **01:** 01, 02, 03, 04, 05, 06, 07, 08, 09, 10, . . .
- **A:** A, B, C, D, . . . , Z, AA, AB, AC, . . .
- **a:** a, b, c, d, . . ., z, aa, ab, ac, . . .
- **I:** i, ii, iii, iv, v, vi, vii, viii, ix, x, xi, . . .
- **I:** I, II, III, IV, V, VI, VII, VIII, IX, X, XI, . . .

- Можно использовать специальные маркеры языка

Пример

- xml:

```
<?xml version='1.0'?>
<?xml-stylesheet type="text/xsl" href="numelem.xsl" ?>
<items>
   <item>Car</item>
   <item>Pen</item>
   <item>LP Record</item>
   <item>Wisdom</item>
   <item>Cell phone</item>
   <item>Film projector</item>
   <item>Hole</item>
</items>
```

- xsl:

```
<?xml version='1.0'?>
<xsl:stylesheet version="1.0"
      xmlns:xsl="http://www.w3.org/1999/XSL/Transform" >
<xsl:template match="items">
   <xsl:for-each select="item">
      <xsl:sort select="."/>
      <xsl:number value="position()" format="1. "/>
      <xsl:value-of select="."/>,
      <xsl:number value="position()" format="&#x0069;) "/>
      <xsl:value-of select="."/>
      <br/>
   </xsl:for-each>
</xsl:template>
</xsl:stylesheet>
```

Результат

<?xml version="1.0" encoding="UTF-16"?>1. Canopy,

i) Canopy
2. Car,

ii) Car
3. Cell phone,

...

xi) Wisdom

xsl:output

- Задает параметры вывода.
- Синтаксис

```
<xsl:output
method = "xml" | "html" | "text"
version = "NMTOKEN"
encoding = "имя_кодировки"
omit-xml-declaration = "yes" | "no"
standalone = "yes" | "no"
doctype-public = "PUBLIC_ID"
doctype-system = "SYSTEM_ID"
cdata-section-elements = "имя_элемента_1 имя_элемента_2..."
indent = "yes" | "no"
media-type = "строка" />
```

- **method**, необязательный, по умолчанию xml, может быть html, text и др. (если поддерживается)
- **version**, необязательный, версия метода вывода
- **encoding**, необязательный, наименование кодировки
- omit-xml-declaration, необязательный, если yes, то xml объявление не включается
- **standalone**, необязательный, значение атрибута в xml
- **doctype-public**, необязательный, открытый идентификатор,
- **doctype-system**, необязательный, системный идентификатор
- **cdata-section-elements**, необязательный, список имен элементов через пробел содержимое которых идет в CDATA
- **indent**, необязательный, если yes то украшение дерева пробелами
- **media-type**, необязательный, MIME тип для вывода, напр. text/html или text/xml.

Пример

```
<?xml version='1.0'?>
<xsl:stylesheet version="1.0"
      xmlns:xsl="http://www.w3.org/1999/XSL/Transform" >
<xsl:output method="html" version="4.0"/>
<xsl:template match="/">
  <html>
    <xsl:apply-templates/>
  </html>
</xsl:template>
</xsl:stylesheet>
```

- Результат:

```
<html>Gambardella, Matthew XML Developer's Guide... </html>
```

xsl:param

- Определяет именованный параметр
- Синтаксис

```
<xsl:param
name = "УточненноеИмя"
select = "выражение">
<!-- шаблон -->
</xsl:param>
```

- **name**, обязательный, имя параметра.
- **select**, необязательный, выражение XPath, задающее значение атрибута.
- Содержимое – шаблон
- Может быть или содержимое или select
- Должен быть прямым дочерним элементом <xsl:template>

- Неправильно:

```
<xsl:template name="getcount">
  <xsl:element name="strong">
    <xsl:param name="counted">
      <xsl:value-of select="count(//book)"/>
    </xsl:param>
    Total Book Count: <xsl:value-of select="$counted"/>
  </xsl:element>
</xsl:template>
```

- Правильно:

```
<xsl:template name="getcount">
  <xsl:param name="counted">
    <xsl:value-of select="count(//book)"/>
  </xsl:param>
  <xsl:element name="strong">
    Total Book Count: <xsl:value-of select="$counted"/>
  </xsl:element>
```

Пример

```
<?xml version='1.0'?>
<?xml-stylesheet type="text/xsl" href="paramelem.xsl"?>
<xsl:stylesheet version="1.0"
      xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="ol/li">
   <br/>
   <xsl:call-template name="numbered-block"/>
</xsl:template>
```

```
<xsl:template match="ol//ol/li">
  <xsl:call-template name="numbered-block">
    <xsl:with-param name="format">a. </xsl:with-param>
  </xsl:call-template>
</xsl:template>

<xsl:template name="numbered-block">
  <xsl:param name="format">1. </xsl:param>
    <xsl:number format="{$format}"/>
    <xsl:apply-templates/>
</xsl:template>
</xsl:stylesheet>
```

- Результат

1. the 2. cat a. sat b. on c. the 3. mat

xsl:with-param

- передача именованного параметра шаблону
- Шаблон получает параметр через <xsl:param>
- Синтаксис

```
<xsl:with-param
name = "УточненноеИмя"
select = "выражение">
<!-- шаблон -->
</xsl:with-param>
```

```
<xsl:param name="lang">en</xsl:param>
<xsl:variable name="messages"
      select="document(concat('resources/', $lang, '.xml'))/messages"/>

<xsl:template name="msg23" match="msg23">
  <xsl:call-template name="localized-message">
    <xsl:with-param name="msgcode">msg23</xsl:with-param>
  </xsl:call-template>
</xsl:template>

<xsl:template name="localized-message">
  <xsl:param name="msgcode"/>
  <!-- Show message string. -->
  <xsl:message terminate="yes">
    <xsl:value-of select="$messages/message[@name=$msgcode]"/>
  </xsl:message>
</xsl:template>
```

xsl:variable

- Определение переменной-константы
- Значение нельзя изменять
- Синтаксис

```
<xsl:variable
name = "УточненноеИмя"
select = "выражение">
<!-- шаблон -->
</xsl:variable>
```

- Использование: $имя

xsl:preserve-space

- У каких элементов не следует удалять пробельные символы
- Синтаксис

<xsl:preserve-space

elements="УточненноеИмя_1 УточненноеИмя_2..." / >

- **elements**, обязательный, список разделенных пробельными символами элементов
- Не оказывает влияния на пробелы внутри текстовых элементов

xsl:strip-space

- Удаляет пробелы из документа.
- Синтаксис

```
<xsl:strip-space
elements="УточненноеИмя_1
УточненноеИмя_2..." />
```

- Пробельные символы сохраняются если элемент включен в <xml:preserve-space>
- Если элемент содержит хотя бы 1 непробельный символ, то он не обрабатывается

xsl:processing-instruction

- вставляет в результирующее дерево инструкцию обработки
- Синтаксис

```
<xsl:processing-instruction
name = "цель">
<!-- шаблон -->
</xsl:processing-instruction>
```

- **name**, обязательный, цель инструкции обработки
- Содержимое – шаблон для формирования инструкции

Пример

```
<?xml version='1.0'?>
<?xml-stylesheet type="text/xsl" href="pi.xsl" ?>
<customers>
<customer>
   <name>James Smith</name>
   <address>123 Elm St.</address>
   <phone>(123) 456-7890</phone>
</customer>
<customer>
   <name>Amy Jones</name>
   <address>456 Oak Ave.</address>
   <phone>(156) 789-0123</phone>
</customer>
</customers>
```

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

<xsl:output method='xml' version='1.0'/>
<xsl:template match="/">
  <xsl:processing-instruction name="xml-stylesheet">
  <xsl:text>type="text/xsl" href="style.xsl"</xsl:text>
  </xsl:processing-instruction>
  <xsl:apply-templates />
</xsl:template>

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

```
<?xml version="1.0" encoding="UTF-16"?>
<?xml-stylesheet type="text/xsl" href="style.xsl" ?>
<customers>
<customer>
<name>James Smith</name>
<address>123 Elm St.</address>
<phone>(123) 456-7890</phone>
</customer>
<customer>
<name>Amy Jones</name>
<address>456 Oak Ave.</address>
<phone>(156) 789-0123</phone>
</customer>
</customers>
```

xsl:sort

- Сортировка контекстных узлов
- Дочерний элемент для <xsl:apply-templates> или <xsl:for-each>
- Синтаксис

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

- select, обязательный, ключ сортировки
- data-type, необязательный, тип данных (text, number)
- lang, необязательный, используемый язык
- order, необязательный, порядок, в котором сортируются строки
- case-order, необязательный, сортировка букв (сначала верхний регистр или нижний)

xsl:text

- Вывод содержимого в виде текста
- Используется внутри шаблона
- Синтаксис

```
<xsl:text
disable-output-escaping = "yes" | "no">
<!-- #PCDATA -->
</xsl:text>
```

- disable-output-escaping, необязательный, если yes то спец.символы заменяются оригиналом
-

Вычисление значений

- **элемент** - Текст, содержащийся в элементе после разрешения всех ссылок на сущности и отбрасывания всех тегов.
- **текст** - Текст узла.
- **атрибут** - Нормализованное значение атрибута.
- **корневой узел** - Значение корневого элемента.
- **инструкция обработки** - Данные инструкции обработки (<?, ?> и цель не включаются).
- **комментарий** - Текст комментария (<!-- и --> не включаются).
- **пространство имен** - URI пространства имен.
-

Приоритет шаблона

- Шаблоны с масками поиска, состоящими только из имени атрибута или элемента (например, person или @profession), имеют приоритет 0.
- Шаблоны с масками поиска, состоящими только из функции processing-instruction() (например, processing-instruction() или processing-instruction('robots')), имеют приоритет 0.
- Шаблоны с масками поиска в форме prefix:* имеют приоритет –0,25.
- Шаблоны с масками поиска, состоящими только из критерия узла (*, @*, comment(), node(), text()), имеют приоритет –0,5. (т.е. встроенные шаблонные правила имеют приоритет –0,5. Однако они импортируются перед всеми другими шаблонными правилами и потому никогда не переопределяют явные шаблонные правила)
- Все остальные шаблоны (person[name='Фейнман'], people/person/@profession, person/text() и т. д.) имеют приоритет 0,5
- Если два или более шаблонов соответствуют узлу и имеют одинаковый приоритет, то как правило выбирается последний