XSL-преобразования

XML Bible (второе издание), Глава 17: XSL-преобразования

Данная серия документов подготовлена на основе материалов сайта Школы Консорциума W3C. Этот сайт является экспериментальным сервером, на котором содержание документов хранится в формате XML. Пользователям сайта эти документы доступны в виде HTML (преобразование на стороне клиента с помощью таблицы стилей XSLT) и в виде PDF (преобразование тех же документов в XSL-FO, а затем в формат PDF).

Содержание и вступление

1: http://xml.nsu.ru/extra/xslt_1.xml
2: http://xml.nsu.ru/extra/xslt_2.xml
3: http://xml.nsu.ru/extra/xslt_3.xml
4: http://xml.nsu.ru/extra/xslt_4.xml
5: http://xml.nsu.ru/extra/xslt_5.xml
6: http://xml.nsu.ru/extra/xslt_6.xml
7: http://xml.nsu.ru/extra/xslt_7.xml

- Типы выражений
- Наборы узлов
- Булевские
- Числа
- Строки
- Фрагменты результирующего дерева

- **Дефолтные шаблонные правила**[8]
  - Дефолтные правила для элементов
  - Дефолтное правило для текстовых узлов и атрибутов
  - Дефолтные правила для процессуальных инструкций и комментариев
  - Реализация дефолтных правил
- **Формирование выходного потока**[9]
  - Шаблоны значений атрибутов
  - Введение элементов в выходной поток с помощью xsl:element
  - Введение атрибутов в выходной поток с помощью xsl:attribute
  - Задание набора атрибутов
  - Создание процессуальных инструкций с помощью xsl:processing-instruction
  - Создание комментариев с помощью xsl:comment
  - Создание текста с помощью xsl:text
- **Копирование контекстного узла с помощью элемента xsl:copy**[10]
- **Нумерация узлов с помощью элемента xsl:number**[11]
  - Дефолтные номера
  - Атрибут level
  - Атрибут count
  - Атрибут from
  - Конвертирование числа в строку
  - Атрибут format
  - Атрибут letter-value
  - Атрибуты группировки
- **Сортировка выходных элементов**[12]
- **Моды**[13]
- **Задание констант с помощью элемента xsl:variable**[14]
- **Именованные шаблоны**[15]
- **Передача параметров шаблонам**[16]
- **Удаление и сохранение пробелов**[17]
- **Организация ветвления**[18]

8: http://xml.nsu.ru/extra/xslt_8.xml
9: http://xml.nsu.ru/extra/xslt_9.xml
10: http://xml.nsu.ru/extra/xslt_10.xml
11: http://xml.nsu.ru/extra/xslt_11.xml
12: http://xml.nsu.ru/extra/xslt_12.xml
13: http://xml.nsu.ru/extra/xslt_13.xml
14: http://xml.nsu.ru/extra/xslt_14.xml
15: http://xml.nsu.ru/extra/xslt_15.xml
16: http://xml.nsu.ru/extra/xslt_16.xml
17: http://xml.nsu.ru/extra/xslt_17.xml

- Элемент xsl:if
- Элемент xsl:choose

- **Смешивание нескольких таблиц стилей**[19]
  - Импорт с помощью элемента xsl:import
  - Включение с помощью элемента xsl:include
  - Встраивание с помощью элемента xsl:stylesheet
- **Методы выведения и заключение**[20]
  - Элемент xsl:output
  - XML-декларация
  - Определение типов документа DTD
  - Форматирование результирующего кода
  - Секции CDATA
  - Медиа-тип
  - Заключение

Расширяемый язык таблиц стилей XSL (Extensible Stylesheet Language) включает в себя **язык преобразований** и **язык форматирования**. Каждый из них, по сути, является самостоятельным XML-приложением. Язык преобразований описывает элементы, задающие правила, по которым один XML-документ преобразуется в другой XML-документ. Полученный в результате XML-документ может содержать разметку и DTD оригинального документа, а может состоять из совершенно другого набора элементов. В частности, он может содержать элементы, описанные во второй части XSL, форматирующие объекты (formatting objects). В этой главе рассказывается о первом компоненте XSL, о языке преобразований.

Что такое XSL?

Трансформационная и форматирующая части языка XSL могут функционировать независимо друг от друга. Например, язык преобразований (трансформаций) может преобразовывать XML-документ в правильный HTML-файл, никак не используя форматирующие объекты XSL. Это и есть тот стиль XSL, о котором мы начали говорить в главе 5 и особое внимание уделим в этой. Более того, нет никакой необходимости в том, чтобы документ на основе языка форматирующих объектов XSL был сгенерирован действием трансформационной компоненты XSL на некоторый XML-документ. Например, несложно вообразить написанный на Java конвертор, который читает TeX или PDF-файлы и транслирует их в язык форматирующих объектов XSL (хотя на начало 2001 года таких конверторов не существовало).

По сути, XSL представляет собой не один язык, а два. Первый язык - язык трансформаций, второй - язык форматирования. Язык трансформаций полезен независимо от языка форматирования. Его способность перемещать данные из одного XML-представления в другое делает его важной частью основанных на XML систем электронной коммерции, систем обмена данными и мета-данными, важной составляющей любого приложения, которое должно конвертировать данные из одного XML-представления в другое. Эти области объединяет то, что они не имеют отношения к представлению данных на дисплее для восприятия человеком - они

18: http://xml.nsu.ru/extra/xslt_18.xml
19: http://xml.nsu.ru/extra/xslt_19.xml
20: http://xml.nsu.ru/extra/xslt_20.xml

предназначены только для перемещения данных из одной компьютерной системы или программы в другую.

В результате первые реализации XSL сосредотачивались на трансформационной части и игнорировали форматирующую. Такие реализации не полны, но полезны даже в этом случае. В конце концов, не все данные предназначены для выведения на монитор или печать.

Перекрестная ссылка

Языку форматирующих объектов XSL посвящена глава 18.

Несколько замечаний относительно XSL

Язык XSL продолжает развиваться. Он уже радикально менялся и почти наверняка изменится в будущем. Эта глава основывается на рекомендации XSLT 1.0 (16 ноября 1999 года). Поскольку сейчас XSLT является официальной рекомендацией консорциума W3C, я надеюсь, что любые изменения лишь что-то добавят к существующему синтаксису, но не сделают невалидными документы, подготовленные в соответствии с рекомендацией XSLT 1.0. W3C уже начал работу над рекомендациями XSLT 1.1 и 2.0, но похоже, что все пригодные документы по рекомендации XSLT 1.0 останутся пригодными и по новым рекомендациям XSLT 1.1 и 2.0.

Однако не все программное обеспечение поддерживает рекомендацию 1.0. В частности, версия 5.5 (и более ранние) браузера Internet Explorer поддерживают лишь очень старую рабочую версию XSLT, которая почти ничто по сравнению с окончательной рекомендацией. Не рассчитывайте, что все примеры этой главы будут работать в IE, хотя бы и после существенной правки. Язык, который реализован в IE не является языком XSLT; любая книга или человек, который говорит иначе, говорит неправду. Компания Microsoft и документы, которые она публикует на своем вебсайте, несут пропаганду нестандартной версии XSLT (и других языков), отсутствует четкое разделение - что является настоящим XSLT, а что является расширением стандарта компанией Microsoft (можно сказать "извращением").

В ноябре 2000-го года Microsoft выпустил MSXML 3.0 - XML-парсер и XSLT-процессор для IE, который гораздо точнее поддерживает XSLT 1.0. Вы можете скачать его с сайта Microsoft[21]. Тем не менее, и в нем имеются некоторые ошибки и целые области, в которых Microsoft не следует спецификации, так что нельзя говорить о полной реализации XSLT 1.0. Важно и то, что MSXML 3.0 не связан с IE5.5; даже если вы инсталлируете этот парсер, он не заменит автоматически старый, нестандартный. Чтобы заменить старую версию, вам нужно скачать и запустить отдельную программу, которая называется xmlinst.exe, вы ее можете найти на той же странице, что и сам парсер MSXML 3.0.

Общий взгляд на XSL-преобразования

Во время XSL-преобразования XSLT-процессор считывает XML-документ и таблицу стилей XSLT. На основе инструкций, которые процессор находит в таблице стилей XSLT, он вырабатывает новый XML-документ или его фрагмент. Кроме того, имеется специальная поддержка генерации HTML. С некоторыми усилиями большинство XSLT-процессоров можно применять для генерации текста в произвольном формате, хотя язык XSLT главным образом предназначен для преобразований XML в XML и XML в HTML.

21: http://msdn.microsoft.com/xml/general/xmlparser.asp

Деревья

Как вы уже знаете, каждый правильный XML-документ представляет собой дерево. Дерево - это структура данных, образованная соединенными между собой узлами, ветвящимися от верхнего узла, он называется корнем. Корень соединен со своими дочерними узлами, каждый из них может в свою очередь соединяться со своими дочерними узлами и так далее. Узлы, у которых нет собственных дочерних узлов, называются листьями. Схема этого дерева очень похожа на гинеологическое древо, в котором перечисляются все потомки некоего общего предка. Самое замечательное свойство дерева состоит в том, что каждый его узел вместе со своими дочерними узлами тоже образует дерево. Таким образом, дерево является иерархической структурой деревьев, в которой каждое дерево образовано деревьями меньшего размера.

В XSLT элементы, атрибуты, пространства имен, процессуальные инструкции и комментарии считаются узлами. Кроме того, корень документа отличается от корневого элемента. Таким образом, XSLT-процессоры моделируют XML-документ как дерево, образованное семью видами узлов:

- Корень
- Элементы
- Текст
- Атрибуты
- Пространства имен
- Процессуальные инструкции
- Комментарии

Определение типа документа (DTD) и декларация типа документа специально не включаются в это дерево. Однако, DTD может назначить некоторым атрибутам элементов значения по умолчанию и эти атрибуты становятся дополнительными атрибутными узлами дерева.

Например, рассмотрим XML-документ на листинге 17-1. Он содержит начало периодической таблицы химических элементов. В этой главе мы постоянно будем использовать этот пример.

Документ 17-1[22]

Корневой элемент PERIODIC_TABLE содержит дочерние элементы ATOM. Каждый элемент ATOM содержит несколько элементов, несущих информацию об атомном номере, атомном весе, обозначающем символе, температуре кипения и так далее. Атрибут UNITS задает единицы измерения.

Кстати

ELEMENT в данном случае было бы более подходящим именем элемента, чем ATOM. Однако, рассказ об элементах ELEMENT и попытки различить химические элементы и XML-элементы привели бы к путанице. Поэтому, по крайней мере для этой главы, имя ATOM кажется более верным.

Листинг 17-1:
Периодическая таблица элементов в XML-формате: два химических элемента - водород и гелий

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xml" href="17-2.xsl"?>
```

22: http://xml.nsu.ru/extra/samples/17-1.xml

```
<PERIODIC_TABLE>
  <ATOM STATE="GAS">
    <NAME>Hydrogen</NAME>
    <SYMBOL>H</SYMBOL>
    <ATOMIC_NUMBER>1</ATOMIC_NUMBER>
    <ATOMIC_WEIGHT>1.00794</ATOMIC_WEIGHT>
    <BOILING_POINT UNITS="Kelvin">20.28</BOILING_POINT>
    <MELTING_POINT UNITS="Kelvin">13.81</MELTING_POINT>
    <DENSITY UNITS="grams/cubic centimeter">
      <!-- At 300K, 1 atm -->
      0.0000899
    </DENSITY>
  </ATOM>
  <ATOM STATE="GAS">
    <NAME>Helium</NAME>
    <SYMBOL>He</SYMBOL>
    <ATOMIC_NUMBER>2</ATOMIC_NUMBER>
    <ATOMIC_WEIGHT>4.0026</ATOMIC_WEIGHT>
    <BOILING_POINT UNITS="Kelvin">4.216</BOILING_POINT>
    <MELTING_POINT UNITS="Kelvin">0.95</MELTING_POINT>
    <DENSITY UNITS="grams/cubic centimeter"><!-- At 300K -->
      0.0001785
    </DENSITY>
  </ATOM>
</PERIODIC_TABLE>
```

Рассмотрим этот документ как дерево. Оно начинается сверху с корневого узла (это не то же самое, что корневой элемент), который содержит два дочерних узла: процессуальную инструкцию xml-stylesheet и корневой элемент PERIODIC_TABLE. (XML-декларация невидима для XSLT-процессора и не является частью дерева, которое обрабатывает XSLT-процессор.) Элемент PERIODIC_TABLE содержит два дочерних узла, это элементы ATOM. Каждый элемент ATOM содержит атрибутный узел, соответствующий атрибуту STATE, и несколько дочерних элементных узлов. Каждый дочерний элемент содержит, в свою очередь, узел для своего содержания, а также узлы для атрибутов, комментариев и процессуальных инструкций, которые в нем имеются. Заметьте, что многие узлы не являются элементами. Есть текстовые узлы, атрибутные узлы, узлы комментариев, узлы пространств имен, узлы процессуальных инструкций. В отличие от CSS, XSL не ограничен только работой с целыми элементами, здесь гораздо более тонкий подход к документу, позволяющий базировать стиль на комментариях, атрибутах, процессуальных инструкциях, содержании элементов и так далее.

Кстати

Как и XML-декларация, внутреннее подмножество DTD или декларация DOCTYPE не являются частью дерева. Однако, может наблюдаться эффект добавления атрибутных узлов к некоторым элементам через декларации типа <!ATTLIST>, в которых объявляются фиксированные (#FIXED) значения атрибутов или значения по умолчанию.

XSLT трансформирует одно дерево XML в другое. Говоря точнее, XSLT-процессор принимает на входе представленное XML-документом дерево и генерирует новое дерево, тоже представленное XML-документом. Поэтому трансформационную часть XSL иногда называют перобразованием деревьев. Язык трансформаций XSL содержит операторы выбора отдельных узлов дерева, переупорядочивания узлов и выведения узлов. Узлы могут быть элементными узлами, а могут быть и целыми деревьями. Помните о том, что все эти операторы - для отбора улов и для их вывода - предназначены для операций над деревьями.

На входе должен быть XML-документ. Нельзя применить XSLT для трансформации не XML-форматов, таких, как PDF, TeX, Microsoft Word, PostScript, MIDI и т.д. HTML и SGML являются пограничными случаями, поскольку они очень близки к XML. XSLT может работать с теми HTML и SGML-документами, которые удовлетворяют условиям правильности XML. Однако, XSLT ничего не может поделать с огромным количеством неправильных HTML и SGML-документов, которые находятся на большей части веб-сайтов и в системах генерации документов. XSLT не является языком регулярных выражений общего назначения для трансформации произвольных данных.

В большинстве случаев в результате XSLT-трансформации мы тоже получаем XML-документ. Однако, мы можем получить на выходе фрагмент дерева, который может использоваться как парсируемая внешняя сущность в другом XML-документе. (То есть, в выходном документе может отсутствовать корневой элемент) Другими словами, на выходе мы можем получить не только правильный XML-документ, но и часть правильного документа. XSLT-трансформация не может выдать текст, который нарушает правила XML, например:

```
<B><I>Неверное вложение тэгов!</B></I>
```

На заметку

Элемент xsl:output и атрибут disable-output-escaping, которые будут обсуждаться ниже, несколько сглаживают это ограничение.

Большинство XSLT-процессоров поддерживают выведение в виде HTML и/или простого текста, хотя стандарт не требует от них такой способности. До некоторой степени это позволяет трансформировать такие не XML-форматы, как TeX, RTF или PostScript. Однако, XSLT не предназначен для преобразований такого рода. Он предназначен для преобразований XML в XML. Если вам на выходе нужен не XML-формат, вероятно, лучше воспользоваться XSLT для преобразования XML в некоторый промежуточный формат, например, TeXML[23], а затем воспользоваться дополнительным, не-XSLT программным обеспечением, для преобразования промежуточного формата в тот, который вам нужен.

Документы таблиц стилей XSLT

XSLT-документ содержит правила-шаблоны. Правило-шаблон содержит паттерн, который определяет соответствующие ему узлы дерева. Когда XSLT-процессор трансформирует XML-документ, он двигается через дерево XML-документа и по очереди проверяет каждый узел. Если процессор обнаруживает, что некоторый узел соответствует паттерну одного из правил-шаблонов таблицы стилей, он выводит этот шаблон. Обычно шаблоны содержат разметку, какие-то новые данные, другие данные копируются из исходного XML-документа.

Для описания правил, шаблонов и паттернов в XSLT используется XML. Корневым элементом XSLT-документа является либо элемент stylesheet, либо transform, которые находятся в пространстве имен http://www.w3.org/1999/XSL/Transform. По договоренности, это пространство имен отображается в префиксе xsl, но можно выбрать и префикс на свое усмотрение. В этой главе я всегда буду использовать префикс xsl. Начиная с этого места будет подразумеваться, что префикс xsl соответствует пространству имен http://www.w3.org/1999/XSL/Transform.

На заметку

Если вы определили не верное URI пространства имен, либо использовали URI, соответствующее

23: http://www.alphaworks.ibm.com/tech/texml

рабочему варианту спецификации, например, http://www.w3.org/TR/WD-xsl, либо просто допустили ошибку в написании нормального URI, XSLT-процессор возвратит сам документ таблицы стилей, а не трансформированный входной документ. Это результат пересечений некоторых темных разделов спецификации XSLT 1.0. Детали не важны. Важно лишь то, что это очень странное поведение выглядит ошибкой, если вы не знаете, что происходит. Но если вы знаете, исправить ситуацию просто. Просто используйте правильный URI: http://www.w3.org/1999/XSL/Transform.

Каждое правило-шаблон задается элементом xsl:template. Паттерн правила размещается в атрибуте match элемента xsl:template. Выходной шаблон представляется содержимым элемента xsl:template. Все инструкции шаблона, предназначенные, например, для выбора определенных частей входного дерева для включения их в выходное дерево, выполняются тем или иным XSLT- элементом. Эти элементы имеют в своем имени префикс xsl:. Элементы, которые не имеют префикса xsl:, являются частью выходного дерева.

В листинге 17-2 приводится пример простой таблицы стилей XSLT, содержащей два правила-шаблона. Первое правило-шаблон относится к корневому элементу PERIODIC_TABLE. Оно заменяет этот элемент элементом html. Содержимое элемента html будет результатом применения других шаблонов в таблице стилей к содержимому элемента PERIODIC_TABLE.

Документ 17-2[24]

Второй шаблон относится к элементам ATOM. Он заменяет каждый элемент ATOM в исходном документе на элемент P в выходном документе. Правило xsl:apply-templates вставляет текст выбранной части исходного документа в выходной документ. Таким образом, содержимое элемента P будет текстом (но не разметкой), который находится внутри соответствующего элемента ATOM.

Корневой элемент xsl:stylesheet имеет два необходимых атрибута, version и xmlns:xsl, каждый из этих атрибутов должен иметь указанное здесь значение: (1.0 для атрибута version и http://www.w3.org/1999/XSL/Transform для атрибута xmlns:xsl). С точным синтаксисом всех этих элементов мы познакомимся далее.

Листинг 17-2:
Таблица стилей XSLT для периодической таблицы химических элементов с двумя правилами-шаблонами

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

Вместо элемента xsl:stylesheet можно использовать элемент xsl:transform. Это точный синоним, имеющий тот же синтаксис, семантику и атрибуты. Например:

```
<?xml version="1.0"?>
<xsl:transform version="1.0"
```

24: http://xml.nsu.ru/extra/samples/17-2.xml

```
        xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <!-- здесь размещаются шаблоны -->
</xsl:transform>
```

В этой книге мы всегда будем использовать элемент xsl:stylesheet.

Где трансформировать XML?

Существует три основных способа преобразования XML-документов с помощью XSLT в другие форматы, например, в HTML:

- XML-документ и связанная с ним таблица стилей отправляются клиенту (веб-браузеру), который преобразует документ как указано в таблице стилей, и после этого представляет результат пользователю.
- Сервер применяет таблицу стилей XSLT к XML-документу и преобразует его в другой формат (обычно, в HTML). После этого результат отправляется клиенту (веб-браузеру).
- Какая-то программа преобразует оригинальный XML-документ в другой формат (обычно, в HTML), затем результат помещается на сервер. И сервер и клиент имеет дело с уже преобразованным документом.

Каждый из этих подходов требует различного программного обеспечения, хотя во всех используются одни и те же XML-документы и таблицы стилей XSLT. Обычный веб-сервер отправляет XML-документы браузеру Internet Explorer - пример первого подхода. Веб-сервер с установленным на нем IBM alphaWorks' XML Enabler[25] - пример второго подхода. Программа SAXON[26], предназначенная для использования из командной строки (ее разработал Michael Kay), преобразует XML-документы в HTML-документы. Затем эти документы размещаются на веб-сервере - это пример третьего подхода. И во всех этих случаях (по крайней мере, теоретически), используется тот же самый язык XSLT.

В этой главе мы будем в основном использовать третий подход, главным образом потому, что к моменту ее написания специализированные конвертирующие программы, такие, как SAXON и Xalan[27] (проект XML Apache Project) предоставляют наиболее полную и точную реализацию спецификации XSLT. Кроме того, данный подход обеспечивает полную совместимость с распространенными веб-браузерами и серверами, в то время как первый подход требует использования самых современных браузеров, а они есть далеко не у всех, а второй подход требует специального серверного программного обеспечения. И все же на практике легче модифицировать сервер, чем клиентов: вы сами можете установить специальное серверное программное обеспечение, но нельзя рассчитывать, что визитеры вашего сайта начнут специально устанавливать что-то себе.

Как применять Xalan

Xalan - это Java 1.1-приложение, работающее в символьной моде. Для его использования нужна совместимая с Java 1.1 виртуальная машина, например, Sun's Java Development Kit (JDK), Java Runtime Environment (JRE), Apple's Macintosh Runtime for Java 2.2 (MRJ), или виртуальная машина Microsoft. Вам нужно будет изменить переменную среды CLASSPATH и включить в нее файлы xalan.jar и xerces.jar (они входят в комплект дистрибутива Xalan). На Unix/Linux вы можете сделать это в файле .cshrc, если вы используете csh или tcsh, или в файле .profile, если вы используете sh, ksh или bash. В системе Windows 95/98 вы можете из-

25: http://www.alphaworks.ibm.com/tech/xmlenabler
26: http://users.iclway.co.uk/mhkay/saxon/
27: http://xml.apache.org/xalan-j/index.html

менить ее в файле AUTOEXEC.BAT. В Windows NT/2000, измените ее во вкладке System Control Panel Environment.

Кстати

Если вы используете JRE 1.2 или более позднюю версию, вы может просто поместить файлы xalan.jar и xerces.jar в директорию jre/lib/ext, а не возиться с переменной среды CLASSPATH. Если вместе с JRE вы установили в Windows JDK, у вас может быть две директории jre/lib/ext, одна, например, по адресу C:\jdk1.3\jre\lib\ext, а вторая, например, C:\Program Files\Javasoft\jre\1.3\lib\ext. Вам будет нужно скопировать архив jar в обе директории ext. Если поместить архив в одну директорию, а в другую - только ярлык, работать это не будет. Необходимо в каждой директории ext разместить полноценные копии.

На заметку

В этом разделе мы будем пользоваться в основном приложением Xalan, но все примеры должны работать с SAXON или с любым другим XSLT-процессором, который реализует рекомендацию по XSLT 1.0, датированную 16 ноября 1999 года.

Java-класс, содержащий основной метод Xalan: org.apache.xalan.xslt.Process. Вы можете запустить Xalan, набрав в командной строке или в DOS-окне:

```
C:\> java org.apache.xalan.xslt.Process
-in 17-1.xml -xsl 17-2.xsl -out 17-3.html
```

Эта строка запустит интерпретатор java для Java-класса, содержащего метод Xalan main(), org.apache.xalan.xslt.Process. Исходный XML-документ прописывается за флажком -in, в данном случае это 17-1.xml. Таблица стилей XSLT прописывается за флажком -xsl, у нас это 17-2.xsl; выходной HTML-файл прописывается за аргументом -out, у нас это 17-3.html. Если аргумент -out опущен, трансформированный документ выводится на консоль. Если опущен аргумент -xsl, Xalan попытается использовать таблицу стилей, прописанную в процессуальной инструкции xml-stylesheet в заголовке исходного XML-документа.

Таблица стилей на листинге 17-2 преобразует исходные документы в правильные HTML-файлы (мы обсуждали это в главе 6). Однако, можно проводить преобразования любого XML-документа в любой другой, но для этого нужно написать таблицу стилей, обеспечивающую преобразование. Например, можно создать таблицу стилей, которая преобразует документы векторного языка разметки (VML) в документы масштабируемой векторной графики (SVG):

```
% java org.apache.xalan.xslt.Process -in pinktriangle.vml
  -xsl VmlToSVG.xsl -out pinktriangle.svg
```

Большинство других работающих из командной строки XSLT-процессоров действуют аналогично, хотя, разумеется, у них могут отличаться опции и аргументы командной строки. Возможно, что с ними будет работать еще легче, если они не написаны на Java, поскольку не нужно будет конфигурировать CLASSPATH.

Кстати

Если вы используете Windows, можно работать с отдельно выполняемой версией SAXON, она называется Instant SAXON[28]. Это несколько проще, поскольку не нужно думать о переменных среды CLASSPATH. Для того, чтобы трансформировать документы с помощью этой программы, просто укажите в командной строке путь к файлу saxon.exe и сделайте нечто вроде:

C:\> saxon -o 17-3.html 17-1.xml 17-2.xsl

28: http://users.iclway.co.uk/mhkay/saxon/instant.html

На листинге 17-3 приведен результат трансформирования документа 17-1 таблицей стилей 17-2 с помощью Xalan. Обратите внимание, что Xalan не пытается "причесать" HTML-код, который получается в результате преобразования (в нем много лишних пробелов и форматирования). В конечном итоге это не важно, если вы будете смотреть этот документ в веб-браузерах, они игнорируют лишние пробелы. Этот код можно смотреть в любом браузере, не обязательно оснащенном средствами отображения XML, поскольку это обычный HTML.

Листинг 17-3:
HTML, сгенерированный таблицей стилей 17-2, приложенной к XML-документу 17-1

```
<html>
  <P>
    Hydrogen
    H
    1
    1.00794
    20.28
    13.81
      0.0000899
  </P>
  <P>
    Helium
    He
    2
    4.0026
    4.216
    0.95
      0.0001785
  </P>
</html>
```

Документ 17-3[29]

Прямое отображение XML-файлов с помощью таблиц стилей XSLT

Можно действовать иначе: не заниматься предварительным преобразованием XML-файла, а отправлять клиенту и XML-файл и содержащий описание отображения XSLT-файл. В этом случае за применение таблицы стилей и соответствующее отображение XML-документа отвечает клиентская сторона. Это несколько прибавляет работы клиенту, но значительно снижает нагрузку на сервер. При таком подходе таблица стилей XSLT должна преобразовать документ в понятный клиенту XML-формат. Обычно это HTML, хотя в будущем многие браузеры будут понимать и язык форматирующих объектов XSL.

Привязать таблицу стилей XSLT к XML-документу несложно. Для этого нужно просто вставить процессуальную инструкцию xml-stylesheet в заголовок документа, сразу после XML-декларации. В этой процессуальной инструкции должен иметься атрибут type со значением text/xml, и атрибут href, значение которого указывает на URL таблицы стилей, например:

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xml" href="17-2.xsl"?>
```

Таким же образом вы можете привязать к XML-документу таблицу стилей CSS. Единственная разница состоит в том, что атрибут type должен иметь значение не text/xml, а text/css.

29: http://xml.nsu.ru/extra/samples/17-3.htm

На заметку

В будущем будет доступен более специфичный медиа-тип MIME application/xslt+xml, который позволит отличать XSLT-документы от других XML-документов. Когда XSLT-процессоры будут его поддерживать, можно будет писать процессуальную инструкцию xml-stylesheet следующим образом:

```
<?xml-stylesheet type="application/xslt+xml" href="17-2.xsl"?>
```

Поддержка XSLT в Internet Explorer 5.0 и 5.5 отличается по нескольким пунктам от официальной рекомендации, датированной 16 ноября 1999 года. Во-первых, элементы XSLT относятся к пространству имен http://www.w3.org/TR/WD-xsl, а не к пространству имен http://www.w3.org/1999/XSL/Transform, хотя используется тот же префикс xsl. Во-вторых, в процессуальной инструкции xml-stylesheet используется нестандартный MIME-тип text/xsl, а не text/xml. И, наконец, не реализованы дефолтные правила для элементов, которым не соответствует ни один шаблон. В результате необходимо предусматривать шаблон для каждого элемента в иерархии, начиная с самого корня, а уж затем пытаться смотреть документ в Internet Explorer. На листинге 17-4 приведен пример: в этой таблице стилей три правила, относящиеся к корневому узлу, корневому элементу PERIODIC_TABLE и элементам ATOM соответственно.

Листинг 17-4:
Таблица стилей листинга 17-2, приспособленная для работы с Internet Explorer 5.0 и 5.5

```
<?xml version="1.0"?>
<!-- Это не-стандартная таблица стилей, написанная
только для Internet Explorer. Она не будет работать
ни в одном полноценном XSLT-процессоре. -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/TR/WD-xsl">
  <xsl:template match="/">
    <html>
      <xsl:apply-templates/>
    </html>
  </xsl:template>
  <xsl:template match="PERIODIC_TABLE">
    <xsl:apply-templates/>
  </xsl:template>
  <xsl:template match="ATOM">
    <P>
      <xsl:value-of select="."/>
    </P>
  </xsl:template>
</xsl:stylesheet>
```

Документ 17-1, отображаемый нестандартной таблицей стилей[30]

Комментарий

В идеале можно было бы использовать одни и те же XML-документы для прямого отображения и для предварительного преобразования в HTML. К сожалению, для этого нужно, чтобы Microsoft поддерживал настоящую спецификацию XSLT. Microsoft постоянно это обещает и постоянно нарушает свои обещания.

30: http://xml.nsu.ru/extra/samples/17-1old.xml

От переводчика

Internet Explorer 6 гораздо лучше соответствует официальной спецификации XSLT 1.0, так что примеры вполне можно смотреть в этом браузере. Кроме того, поддерживается и нестандартный XSLT с пространством имен http://www.w3.org/TR/WD-xsl. В зависимости от указанного в процессуальной инструкции пространства имен, Internet Explorer 6 будет пользоваться либо официальной спецификацией, либо старой версией. Примеры можно смотреть и в новом браузере Mozilla, но в таблице стилей должен присутствовать элемент <xsl:output method="html" />

Internet Explorer не реализует и многие другие части стандарта XSLT, в то же время предлагая несколько нестандартных расширений. Если вы успешно установили себе MSXML3 в режиме замены, то IE5 сможет поддерживать основную часть XSLT 1.0, включая пространство имен http://www.w3.org/1999/XSL/Transform. Однако, даже в этой версии есть несколько ошибок, например, использование в XML-документах процессуальной инструкции, указывающей на таблицу стилей MIME-типа text/xsl, а не text/xml. Далее в этой главе я буду использовать только стандартный XSLT и предварительно преобразовывать документы в HTML, и уже потом демонстрировать их в веб-браузере. Если вы обнаружите нечто такое, что не работает в Internet Explorer, пеняйте на Microsoft, не на меня.

XSL-шаблоны

Правила-шаблоны, задаваемые элементами xsl:template - наиболее важная часть таблиц стилей XSLT. Именно они связывают определенный выход с определенным входом. Каждый элемент xsl:template имеет атрибут match, он выделяет набор узлов исходного документа, на которые действует данный шаблон.

Содержимое элемента xsl:template - это задействуемый для преобразования выделенного набора узлов шаблон. Шаблон может содержать текст, который в неизменном виде выводится в выходной документ, а также XSLT-инструкции, копирующие данные из исходного XML-документа в выходной. Поскольку все XSLT-инструкции находятся в пространстве имен http://www.w3.org/1999/XSL/Transform, их легко отличить от элементов, которые буквально копируются в выходной документ. Например, вот шаблон, который действует на корневой узел исходного дерева:

```
<xsl:template match="/">
  <html>
    <head>
    </head>
    <body>
    </body>
  </html>
</xsl:template>
```

Когда XSLT-процессор начинает считывать исходный документ, первый узел, с которым он сталкивается - это корневой узел. Приведенное правило относится к корневому узлу и в нем XSLT-процессору дается указание вывести следующий текст:

```
<html>
  <head>
  </head>
  <body>
  </body>
</html>
```

Этот текст является правильным HTML. Поскольку XSLT-документ сам является XML-документом, его содержимое - шаблоны - тоже должны быть правильным XML.

Если вы оставите в таблице стилей XSLT только приведенный шаблон, на выходе вы получите только эти шесть тэгов. Так произойдет потому, что в этом шаблоне отсутствуют инструкции, заставляющие преобразователь двигаться далее по дереву и искать другие соответствия с шаблонами таблицы стилей.

Элемент xsl:apply-templates

Чтобы зайти в дерево глубже его корня, нужно дать указание преобразователю обрабатывать дочерние узлы корневого узла. Говоря в целом, чтобы задействовать содержимое дочерних узлов, нужно рекурсивно обрабатывать узлы всего XML-документа. Элемент, предназначенный для этого - xsl:apply-templates. Включая элемент xsl:apply-templates в выходной шаблон, вы даете указание преобразователю сравнивать каждый дочерний элемент исходного элемента (который и соответствует данному шаблону) с другими шаблонами в таблице стилей и, если соответствие обнаружено, выводить шаблон для соответствующего узла. Шаблон узла, соответствие с которым обнаружено, тоже может содержать элементы xsl:apply-templates для поиска соответствий с дочерними узлами уже этого узла. Когда процессор обрабатывает узел, он рассматривается им как целое дерево. Это преимущество древовидной структуры. Каждая ее часть может обрабатываться так же, как и дерево целиком. Например, на листинге 17-5 приводится образец таблицы стилей XSLT, в которой для обработки дочерних узлов применяется элемент xsl:apply templates.

Листинг 17-5:
Таблица стилей XSLT, которая рекурсивно обрабатывает дочерние узлы корневого узла

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <html>
      <xsl:apply-templates/>
    </html>
  </xsl:template>
  <xsl:template match="PERIODIC_TABLE">
    <body>
      <xsl:apply-templates/>
    </body>
  </xsl:template>
  <xsl:template match="ATOM">
    An Atom
  </xsl:template>
</xsl:stylesheet>
```

Документ 17-5[31]

Вот что происходит, когда эта таблица стилей применяется к документу на листинге 17-1:

- Корневой узел сравнивается со всеми правилами-шаблонами таблицы стилей. Он находится в соответствии с первым шаблоном.

31: http://xml.nsu.ru/extra/samples/17-5.xml

- На выход передается тэг <html>.
- Элемент xsl:apply-templates дает указание преобразователю обрабатывать дочерние узлы корневого узла исходного документа.
- Первый дочерний узел корня документа, процессуальная инструкция xml-stylesheet, сравнивается с правилами-шаблонами. Она не соответствует ни одному из них, так что на выход ничего не передается.
- Второй дочерний узел корневого узла исходного документа, корневой элемент PERIODIC_TABLE, сравнивается с правилами-шаблонами. Он соответствует второму правилу-шаблону.
- В выходной поток вписывается тэг <body>.
- Элемент xsl:apply-templates внутри элемента body заставляет преобразователь обрабатывать дочерние узлы элемента PERIODIC_TABLE.
- Первый дочерний элемент элемента PERIODIC_TABLE, элемент ATOM, описывающий атом водорода, сравнивается с правилами-шаблонами. Он соответствует третьему правилу-шаблону.
- На выход передается текст "An Atom".
- Второй дочерний элемент элемента PERIODIC_TABLE, элемент ATOM, описывающий атом гелия, сравнивается с правилами-шаблонами. Он соответствует третьему правилу-шаблону.
- На выход передается текст "An Atom".
- На выход передается тэг </body>.
- На выход передается тэг </html>.
- Обработка завершена.

Результат таков:

```
<html>
<body>
    An Atom
    An Atom
</body>
</html>
```

Атрибут select

Чтобы поменять текст "An Atom" на имя элемента ATOM, которое задается его дочерним элементом NAME, нужно задать шаблоны, которые должны применяться к дочерним элементам NAME элемента ATOM. Чтобы отобрать не все дочение узлы данного узла, а лишь некоторый их набор, нужно снабдить xsl:apply-templates атрибутом select, выделяющим нужные дочерние узлы, например:

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

В атрибуте select используется тот же вид паттернов, как и в атрибуте match элемента xsl:template. Я пока ограничиваюсь в примерах простыми именами элементов, но в разделе, посвященном паттернам соответствия и отбора (см. далее), вы познакомитесь с массой возможностей, которые можно использовать в атрибутах select и match. Если атрибут select отсутствует, выбираются все дочерние элементы, текстовые узлы, узлы комментариев и процессуальных инструкций. (Атрибуты и узлы пространств имен не выбираются.)

Если теперь, добавив в таблицу стилей на листинге 17-5 еще одно шаблонное правило, применить ее к документу 17-1, мы получим следующий результат:

```
<html>
<body>
  Hydrogen
  Helium
</body>
</html>
```

Документ 17-1, отображаемый с помощью таблицы стилей 17-5[32]

Вычисление значения узла с помощью элемента xsl:value-of

Элемент xsl:value-of вычисляет некоторое значение (обычно, хотя и не всегда, на основе данных исходного документа) и копирует полученный результат в выходной документ. Атрибут select элемента xsl:value-of определяет, какое именно значение вычисляется.

Допустим, вы хотите заменить текст An Atom именем элемента ATOM, заданным его дочерним элементом NAME. Вы можете заменить An Atom на <xsl:value-of select="NAME"/>, вот так:

```
<xsl:template match="ATOM">
  <xsl:value-of select="NAME"/>
</xsl:template>
```

Теперь при применении этой таблицы стилей к документу 17-1 будет сгенерирован следующий текст:

```
<html>
<body>
  Hydrogen
  Helium
</body>
</html>
```

Выбранный пункт, в данном случае элемент NAME, соотносится с текущим узлом. Текущий узел - это пункт, соответствующий шаблону, в данном случае это некоторый элемент ATOM. Таким образом, когда элемент ATOM, описывающий атом водорода вступает в соответствие с <xsl:template match="ATOM">, элемент NAME, содержащийся в этом элементе ATOM, выбирается элементом xsl:value-of. Аналогично, когда элемент ATOM, описывающий атом гелия вступает в соответствие с <xsl:template match="ATOM">, элемент NAME, содержащийся в этом элементе ATOM, выбирается элементом xsl:value-of.

Значение узла - это всегда строка, она может быть и пустой. Точное содержание этой строки зависит от типа узла. Наиболее распространенным типом узлов являются элементы, значение элементного узла очень просто: это конкатенация (объединение) всех символьных данных (но не разметки!) между начальным и конечным тэгом данного элемента. Например, первый элемент ATOM в документе 17-1 выглядит так:

```
<ATOM STATE="GAS">
  <NAME>Hydrogen</NAME>
  <SYMBOL>H</SYMBOL>
  <ATOMIC_NUMBER>1</ATOMIC_NUMBER>
  <ATOMIC_WEIGHT>1.00794</ATOMIC_WEIGHT>
```

32: http://xml.nsu.ru/extra/samples/17-1full.xml

```
    <BOILING_POINT UNITS="Kelvin">20.28</BOILING_POINT>
    <MELTING_POINT UNITS="Kelvin">13.81</MELTING_POINT>
    <DENSITY UNITS="grams/cubic centimeter">
      <!-- At 300K, 1 atm -->
      0.0000899
    </DENSITY>
  </ATOM>
```

А значение этого элемента таково:

```
Hydrogen
H
1
1.00794
1
20.28
13.81
  0.0000899
```

Я вычислил значение этого элемента удалив все тэги и комментарии. Все остальное, включая пробелы и переносы, осталось. Значения остальных шести типов узлов вычисляются схожим образом, обычно очевидными способами, см. таблицу 17-1.

Таблица 17-1: Значения узлов

Тип узла	Значение
Корень	Значение корневого элемента
Элемент	Конкатенация всех парсируемых символьных данных, содержащихся в элементе, включая символьные данные всех потомков данного элемента
Текст	Текст узла; по сути, сам узел
Атрибут	Нормализованное значение атрибута, как описано в разделе 3.3.3 рекомендации XML 1.0; обычно ссылки на сущности раскрываются, а пробелы и символы переноса удаляются; в значение узла не включаются имя атрибута, знак равенства и кавычки
Пространство имен	URI пространства имен
Процессуальная инструкция	Данные, представленные в процессуальной инструкции; в значение не включаются разграничители процессуальной инструкции: <? и ?>
Комментарий	Текст комментария, <!-- и --> в значение не включаются

Обработка множественных элементов с помощью xsl:for-each

Элемент xsl:value-of следует использовать только в тех случаях, когда очевидно, значение какого именно узла берется. Если может быть выбрано множество пунктов, тогда будет выбран лишь первый из них. Например, следующее правило не очень подходит, поскольку элемент PERIODIC_TABLE содержит более одного элемента ATOM:

```
<xsl:template match="PERIODIC_TABLE">
  <xsl:value-of select="ATOM"/>
```

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

Существует два способа по очереди обработать последовательность элементов. Первый метод вы уже видели: просто применяется xsl:apply-templates с атрибутом select, который отбирает нужные элементы, вот так:

```
<xsl:template match="PERIODIC_TABLE">
  <xsl:apply-templates select="ATOM"/>
</xsl:template>
<xsl:template match="ATOM">
  <xsl:value-of select="."/>
</xsl:template>
```

Конструкция select="." во втором шаблоне указывает процессору, что следует взять значение выбранного элемента, в данном случае это ATOM.

Второй вариант - это использование xsl:for-each. Элемент xsl:for-each обрабатывает по очереди каждый элемент, выбранный атрибутом select. Однако, дополнительные шаблоны в данном случае не требуются. Например:

```
<xsl:template match="PERIODIC_TABLE">
  <xsl:for-each select="ATOM">
    <xsl:value-of select="."/>
  </xsl:for-each>
</xsl:template>
```

Паттерны соответствия узлов

Атрибут match элемента xsl:template использует сложный синтаксис, позволяющий указывать точно на узел или множество узлов, с которым требуется установить соответствие. Атрибут select элементов xsl:apply-templates, xsl:value-of, xsl:for-each, xsl:copy-of и xsl:sort использует еще более мощное расширение этого синтаксиса, которое называется Xpath. Xpath позволяет не только выделять нужные узлы, но и отвергать не нужные. Далее мы обсудим различные паттерны соответствия и выделения узлов.

Соответствие с корневым узлом

Для того, чтобы выходной документ был правильным, первое, что нужно вывести в процессе XSL-преобразования, это корневой элемент выходного документа. Поэтому обычно таблица стилей XSLT начинается с правила, которое применяется к корневому узлу. Для задания в правиле корневого узла, атрибут match должен иметь значение "/". Например:

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

Это правило применяется только к корневому узлу исходного дерева. Когда корневой узел считывается, на выход передается тэг <DOCUMENT> и начинается обработка дочерних узлов. После того, как дочерние узлы корневого узла обработаны, на выход передается тэг </DOCUMENT>. Это правило переигрывает дефолтное правило для корневого узла. На лис-

тинге 17-6 приведена таблица стилей с единственным правилом, которое применяется к корневому узлу.

Листинг 17-6:
Таблица стилей XSLT с одним правилом для корневого узла

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:template match="/">
      <html>
        <head>
          <title>Atomic Number vs. Atomic Weight</title>
        </head>
        <body>
          <table>
            Данные атома размещаются здесь
          </table>
        </body>
      </html>
    </xsl:template>
</xsl:stylesheet>
```

Поскольку эта таблица стилей содержит правило только для корневого узла и в этом правиле-шаблоне не задана никакая дальнейшая обработка дочерних узлов, в результирующем документе окажется только буквальное содержимое, включенное в шаблон. Другими словами, результат применения таблицы стилей 17-6 к документу 17-1 (или к любому другому правильному XML-документу) будет такой:

```
<html>
<head>
<title>Atomic Number vs. Atomic Weight</title>
</head>
<body>
<table>
            Данные атомов размещаются здесь
          </table>
</body>
</html>
```

Соответствие именам элементов

Как указывалось ранее, большая часть паттернов содержит имя элемента, указывающее на все элементы с таким именем. Например, следующий шаблон соответствует элементам ATOM и выводит содержимое их под-элементов ATOMIC_NUMBER полужирным шрифтом:

```
<xsl:template match="ATOM">
  <b><xsl:value-of select="ATOMIC_NUMBER"/></b>
</xsl:template>
```

На листинге 17-7 приведена таблица стилей, которая расширяет таблицу 17-6. Во-первых, элемент xsl:apply-templates включен в правило-шаблон для корневого узла. В этом правиле используется атрибут select, который обеспечивает обработку только элементов PERIODIC_TABLE.

Во-вторых, добавлено правило, которое применяется только к элементам PERIODIC_TABLE,

это обеспечивается атрибутом match="PERIODIC_TABLE". Это правило создает заголовок для таблицы, а затем применяет шаблоны для формирования тела таблицы из элементов ATOM.

И наконец, привило, относящееся к элементам ATOM, обрабатывает их под-элементы NAME, ATOMIC_NUMBER и ATOMIC_WEIGHT с помощью элементов <xsl:value-of select="NAME"/>, <xsl:value-of select="ATOMIC_NUMBER"/> и <xsl:value-of select="ATOMIC_WEIGHT"/>. Их содержимое заключается внутрь HTML-элементов tr и td, так что в результате мы получаем таблицу атомных номеров, сопоставленных с атомными весами.

Документ 17-1, обрабатываемый таблицей стилей 17-7[33]

Можно заметить здесь одну особенность: точный порядок следования элементов NAME, ATOMIC_NUMBER, и ATOMIC_WEIGHT в исходном документе не имеет значения. Они появляются в выходном документе в том порядке, в котором отбираются таблицей стилей, то есть сначала номер, потом вес. Отдельные же атомы оказываются отсортированными в алфавитном порядке - как они шли в исходном документе. Далее вы познакомитесь с тем, как с помощью элемента xsl:sort можно изменять порядок следования атомов в выходном документе (например, для того, чтобы они шли в порядке возрастания атомных номеров).

Листинг 17-7:
Шаблоны, применяемые к заданным классам элементов с помощью атрибута select

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:template match="/">
      <html>
        <head>
          <title>Atomic Number vs. Atomic Weight</title>
        </head>
        <body>
          <xsl:apply-templates select="PERIODIC_TABLE"/>
        </body>
      </html>
    </xsl:template>
    <xsl:template match="PERIODIC_TABLE">
      <h1>Atomic Number vs. Atomic Weight</h1>
      <table>
        <th>Element</th>
        <th>Atomic Number</th>
        <th>Atomic Weight</th>
          <xsl:apply-templates select="ATOM"/>
      </table>
    </xsl:template>
    <xsl:template match="ATOM">
      <tr>
        <td><xsl:value-of select="NAME"/></td>
        <td><xsl:value-of select="ATOMIC_NUMBER"/></td>
        <td><xsl:value-of select="ATOMIC_WEIGHT"/></td>
      </tr>
    </xsl:template>
</xsl:stylesheet>
```

33: http://xml.nsu.ru/extra/samples/17-1full1.xml

Соответствие всему звездочкой *

Иногда требуется применить шаблон одновременно к нескольким типам элементов. С помощью звездочки (*) вместо имени элемента в атрибуте match можно указать соответствие всем элементам. Вот, например, шаблон, который указывает, что все элементы должны заключаться в элемент P:

```
<xsl:template match="*">
  <P>
    <xsl:value-of select="."/>
  </P>
</xsl:template>
```

Конечно, такое редко бывает нужно. Лучше использовать правила-шаблоны, уже заданные для элементов PERIODIC_TABLE, ATOM и корневого узла, а это правило применять только для остальных элементов. И это действительно можно сделать: в случае, если одному узлу соответствует два различных правила, приоритет имеет более специфичный. В нашем случае элементы ATOM будут обрабатываться шаблоном, имеющим атрибут match="ATOM", а не тем, в котором указано match="*". Однако, элементы NAME, BOILING_POINT, ATOMIC_NUMBER и другие элементы, которые не соответствуют другим, более специфичным шаблонам, активируют шаблон match="*".

Перед звездочкой можно поместить префикс пространства имен, чтобы показать, что только элементы определенного пространства имен должны отбираться. Например, следующий шаблон соответствует всем SVG-элементам, при этом предполагается, что в таблице стилей префикс svg отображается в обычное пространство имен SVG с URI http://www.w3.org/2000/svg.

```
<xsl:template match="svg:*">
  <DIV>
    <xsl:value-of select="."/>
  </DIV>
</xsl:template>
```

В документе 17-1 отсутствуют элементы из этого пространства имен, так что этот шаблон ничего бы не передал в выходной документ. Тем не менее, его можно применять в тех случаях, когда в документах имеются какие-то элементы SVG.

Соответствие дочерним узлам с помощью /

При использовании атрибута match вы не ограничены только непосредственными дочерними узлами текущего узла. Для выбора заданных иерархий элементов можно использовать символ /. Взятый в одиночку, этот символ соответствует корневому узлу, но его можно размещать между двумя именами элементов, это означает, что второй элемент является дочерним элементом первого. Например, ATOM/NAME указывает на элементы NAME, которые являются дочерними элементами элементов ATOM.

При использовании этой конструкции в элементах xsl:template можно выбирать только некоторые элементы данного вида. Например, следующее правило-шаблон отмечает тэгом элементы SYMBOL, которые непосредственными являются дочерними элементами элементов ATOM, и оно не будет работать с элементами SYMBOL, которые не являются непосредственными дочерними элементами элементов ATOM.

```
<xsl:template match="ATOM/SYMBOL">
```

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

Примечание

Помните о том, что это правило выбирает элементы SYMBOL, которые являются дочерними элементами элементов ATOM, а не элементы ATOM, которые имеют дочерние элементы SYMBOL. Другими словами, символ . в элементе <xsl:value-of select="."/> относится к SYMBOL, а не к ATOM.

Можно задавать и более глубокие соответствия, объединяя паттерны в один, например, PERIODIC_TABLE/ATOM/NAME отбирает элементы NAME, у которых родительским элементом является элемент ATOM, а у тех, в свою очередь, родительским элементом является элемент PERIODIC_TABLE.

Кроме того, можно использовать и звездочку *, чтобы указать на произвольность имени элемента в иерархии. Например, следующее правило-шаблон будет применяться ко всем элементам SYMBOL, которые являются внуками элемента PERIODIC_TABLE.

```
<xsl:template match="PERIODIC_TABLE/*/SYMBOL">
  <strong><xsl:value-of select="."/></strong>
</xsl:template>
```

И, наконец, как вы уже знаете, символ / сам по себе отбирает корневой узел документа. Например, следующее правило будет применяться ко всем элементам PERIODIC_TABLE, которые являются корневыми элементами документа:

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

Поскольку символ / указывает на корневой узел, конструкция /* указывает на любой корневой элемент, какой бы он ни был. Например, следующий шаблон работает одинаково, вне зависимости от того, какой элемент является корневым: PERIODIC_TABLE, DOCUMENT или SCHENECTADY:

```
<xsl:template match="/*">
  <html>
    <head>
      <title>Atomic Number vs. Atomic Weight</title>
    </head>
    <body>
      <xsl:apply-templates/>
    </body>
  </html>
</xsl:template>
```

Соответствие потомкам с помощью //

Иногда, особенно при нестройной исходной иерархии, бывает легче пропустить промежуточные узлы и просто отбирать все элементы определенного типа вне зависимости от того, являются ли они непосредственными дочерними элементами, внуками или правнуками или еще более отдаленными потомками. Двойной слэш // указывает на всех потомков элемента, независимо от их глубины. Например, следующее правило-шаблон будет применяться ко всем элементам-потомкам NAME элемента PERIODIC_TABLE, независимо от глубины, на которой они находятся:

```
<xsl:template match="PERIODIC_TABLE//NAME">
  <i><xsl:value-of select="."/></i>
</xsl:template>
```

Пример с периодической таблицей элементов не отличается особой глубиной иерархии, но в некоторых случаях, особенно когда элементы могут содержать другие элементы своего типа (например, если элемент ATOM содержит другой элемент ATOM), такая конструкция оказывается полезной.

Оператор //, расположенный в начале паттерна выбирает всех потомков корневого узла. Например, следующее правило-шаблон обрабатывает все элементы ATOMIC_NUMBER, игнорируя их местоположение:

```
<xsl:template match="//ATOMIC_NUMBER">
  <i><xsl:value-of select="."/></i>
</xsl:template>
```

Соответствие по идентификатору ID

Иногда требуется применить определенный стиль к отдельному элементу, не затрагивая всех других элементов этого типа. Простейший способ сделать это в XSLT - это привязать стиль к атрибуту ID элемента. Это делается с помощью селектора id(), в котором значение ID заключено в одинарные кавычки. Например, следующее правило делает полужирным содержимое элемента, с ID, равным e47:

```
<xsl:template match="id('e47')">
  <b><xsl:value-of select="."/></b>
</xsl:template>
```

Конечно, здесь предполагается, что элемент, который вы выбираете таким способом, имеет атрибут, декларированный в DTD исходного документа как имеющий тип ID. Однако, так бывает не всегда. Например, многие документы вообще не имеют DTD - они просто правильные, но не валидные. И даже если имеется DTD, нет гарантии, что какой-то конкретный элемент имеет атрибут типа ID.

Перекрестная ссылка

Атрибуты типа ID - это не просто атрибуты с именем ID. Атрибуты типа ID описываются в главе 11.

Соответствие атрибутам с помощью @

Вы уже знаете, что знак @ указывает на атрибуты и отбирает узлы по именам атрибутов: нужно просто перед именем атрибута, который вы хотите отобрать, поставить знак @. Например, следующее правило-шаблон соответствует атрибутам UNITS, и выводит их значение, заключенное в элементы I.

```
<xsl:template match="@UNITS">
  <I><xsl:value-of select="."/></I>
</xsl:template>
```

Однако, если просто добавить это правило к таблице стилей, значения этих атрибутов в выходном документе не появятся автоматически, поскольку атрибуты не являются дочерними

узлами содержащих их элементов. Таким образом, по умолчанию, когда XSLT-процессор двигается по дереву, он не видит атрибутных узлов. Их нужно обрабатывать, ясно прописывая их в подходящем значении атрибута select элемента xsl:apply-templates. На листинге 17-8 приводится таблица стилей, которая выводит таблицу атомных номеров и соответствующих температур плавления. Выводится не только значение элемента MELTING_POINT, но и значение его атрибута UNITS. Он отбирается элементом <xsl:apply-templates select="@UNITS"/>, размещенным в правиле-шаблоне для элементов MELTING_POINT.

Листинг 17-8:
Таблица стилей XSLT, которая отбирает значения атрибутов UNITS с помощью знака @

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:template match="/PERIODIC_TABLE">
      <html>
        <body>
          <h1>Atomic Number vs. Melting Point</h1>
          <table>
            <th>Element</th>
            <th>Atomic Number</th>
            <th>Melting Point</th>
            <xsl:apply-templates/>
          </table>
        </body>
      </html>
    </xsl:template>
    <xsl:template match="ATOM">
       <tr>
        <td><xsl:value-of select="NAME"/></td>
        <td><xsl:value-of select="ATOMIC_NUMBER"/></td>
        <td><xsl:apply-templates select="MELTING_POINT"/></td>
      </tr>
    </xsl:template>
    <xsl:template match="MELTING_POINT">
      <xsl:value-of select="."/>
      <xsl:apply-templates select="@UNITS"/>
    </xsl:template>
    <xsl:template match="@UNITS">
      <I><xsl:value-of select="."/></I>
    </xsl:template>
</xsl:stylesheet>
```

Вспомним о том, что значением атрибутного узла является просто нормализованное строчное значение атрибута. Когда вы примените таблицу стилей 17-8 к элементам ATOM, получится следующий результат:

```
  <tr>
<td>Hydrogen</td><td>1</td><td>13.81<I>Kelvin</I></td>
</tr>
  <tr>
<td>Helium</td><td>2</td><td>0.95<I>Kelvin</I></td>
</tr>
```

Документ 17-1, обрабатываемый таблицей стилей 17-8[34]

34: http://xml.nsu.ru/extra/samples/17-1full2.xml

Можно комбинировать атрибуты с элементами, используя различные операторы, например, паттерн BOILING_POINT/@UNITS указывает на атрибут UNITS элемента BOILING_POINT. Паттерн ATOM/*/@UNITS соответствует любому атрибуту UNITS любого дочернего элемента элемента ATOM. Этот паттерн особенно полезен для указания в правилах-шаблонах на атрибуты. Нужно помнить, что соответствие устанавливается атрибутному узлу, а не элементу, который содержит этот атрибут. Очень часто допускают ошибку, путая атрибутный узел с элементным узлом, который его содержит. Например, рассмотрим следующее правило: в нем делается попытка применить шаблон ко всем дочерним элементам, которые имеют атрибут UNITS:

```
<xsl:template match="ATOM">
  <xsl:apply-templates select="@UNITS"/>
</xsl:template>
```

Но на самом деле шаблон относится к не-существующим атрибутам UNITS элементов ATOM.

Также можно применять конструкцию @* - она устанавливает соответствие всем атрибутам элемента, например, BOILING_POINT/@* указывает на все атрибуты элемента BOILING_POINT. Можно также добавить после знака @ префикс пространства имен, чтобы указать на все атрибуты из определенного пространства имен. Например, @xlink:* соответствует всем XLink-атрибутам, таким как xlink:show, xlink:type и xlink:href (предполагается, что префикс xlink отображается в пространство имен http://www.w3.org/1999/xlink - URI пространства имен XLink.)

Соответствие комментариям с помощью comment()

Обычно комментарии в XML-документах просто игнорируются. Делать комментарии существенной частью документа - очень плохая идея. Тем не менее, в XSLT имеются средства устанавливать соответствие комментариям, если это необходимо.

Чтобы указать на комментарий, используется паттерн comment(). Хотя в этом паттерне присутствуют скобки, что придает ему вид функции, в этих скобках никогда не указывается никаких аргументов. Например, следующее правило-шаблон выводит все комментарии, выделенные курсивом:

```
<xsl:template match="comment()">
  <i><xsl:value-of select="."/></i>
</xsl:template>
```

Чтобы различить различные комментарии, нужно учесть родительский элемент комментария и других его предков. Например, вспомним, что элемент DENSITY выглядит так:

```
<DENSITY UNITS="grams/cubic centimeter">
  <!-- At 300K, 1 atm -->
  0.0000899
</DENSITY>
```

Для отбора определенных комментариев используются уже знакомые нам операторы. Например, следующее правило указывает только на комментарии, которые находятся внутри элементов DENSITY:

```
<xsl:template match="DENSITY/comment()">
  <i><xsl:value-of select="."/></i>
</xsl:template>
```

Единственная причина, по которой в документе 17-1 для задания физических условий используется комментарий - это для данного примера. На практике никогда не следует поме-

щать важную информацию в комментарии. Реальная причина, по которой XSLT позволяет указывать на комментарии, состоит в том, что таблица стилей может преобразовывать один XML-документ - в другой, не теряя комментариев. Любое другое использование означает плохо разработанную структуру документа. Следующее правило соответствует всем комментариям и копирует их в результирующий документ с помощью элемента xsl:comment.

```
<xsl:template match="comment()">
  <xsl:comment><xsl:value-of select="."/></xsl:comment>
</xsl:template>
```

Соответствие процессуальным инструкциям с помощью processing-instruction()

Когда XML станет хорошо структурированным, интеллектуальным и гибким, процессуальные инструкции будут не важнее комментариев. Но пока в них существует необходимость, например, для связывания таблицы стилей с документом.

Функция processing-instruction() указывает на процессуальные инструкции. Аргумент функции processing-instruction() - заключенная в кавычки цель выбираемой процессуальной инструкции (например, xml-stylesheet). Если аргумент опускается, выбирается первый узел процессуальной инструкции текущего узла. Например, следующее правило указывает на дочерние процессуальные инструкции корневого узла (скорее всего это будет процессуальная инструкция xml-stylesheet). Элемент xsl:processing-instruction вставляет процессуальную инструкцию с заданным именем и значением в выходной документ.

```
<xsl:template match="/processing-instruction()">
  <xsl:processing-instruction name="xml-stylesheet">
    type="text/xml" value="auto.xsl"
  </xsl:processing-instruction>
</xsl:template>
```

А вот это правило тоже относится к процессуальной инструкции xml-stylesheet, но через ее имя:

```
<xsl:template
  match="processing-instruction('xml-stylesheet')">
  <xsl:processing-instruction name="xml-stylesheet">
    <xsl:value-of select="."/>
  </xsl:processing-instruction>
</xsl:template>
```

Фактически, основная причина различения корневого элемента и корневого узла состоит в том, что можно читать и обрабатывать процессуальные инструкции, находящиеся вне корневого элемента. Хотя в процессуальной инструкции xml-stylesheet используется синтаксис вида имя = значение, XSL не рассматривает эти конструкции как атрибуты, поскольку процессуальные инструкции не являются элементами. Значение процессуальной инструкции - это просто все, что заключено между пробелом после ее названия и закрывающим тэгом ?>.

Соответствие текстовым узлам с помощью text()

Текстовые узлы обычно игнорируются как узлы, хотя их значения включаются как часть значения выбранного элемента. Однако, с помощью оператора text() можно специально выбрать текстовый узел элемента. Несмотря на наличие скобок, у этого оператора нет аргументов. Например, следующее правило делает весь текст документа полужирным:

```
<xsl:template match="text()">
  <b><xsl:value-of select="."/></b>
</xsl:template>
```

Основное применение этот оператор находит в дефолтных правилах. XSLT-процессоры должны предоставлять при обработке документов следующее правило как дефолтное, независимо от того, задает его сам автор таблицы стилей или нет:

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

В нем указывается, что всякий раз, когда шаблон применяется к текстовому узлу, текст передается на выход. Если вам не нужно это дефолтное поведение, вы можете его переиграть. Например, следующее пустое правило-шаблон, если его включить в таблицу стилей, не будет автоматически передавать на выход текстовые узлы, пока вы специально не укажете на них другим правилом.

```
<xsl:template match="text()">
</xsl:template>
```

Использование оператора |

Символ вертикальной линии (|) позволяет правилу-шаблону соответствовать нескольким паттернам одновременно. Шаблон будет активироваться, если соответствие достигнуто по одному или другому паттерну. Например, следующее правило соответствует и элементам ATOMIC_NUMBER и элементам ATOMIC_WEIGHT:

```
<xsl:template match="ATOMIC_NUMBER|ATOMIC_WEIGHT">
  <B><xsl:apply-templates/></B>
</xsl:template>
```

Вокруг оператора | можно поставить пробелы, если так запись будет вам понятнее. Например:

```
<xsl:template match="ATOMIC_NUMBER | ATOMIC_WEIGHT">
  <B><xsl:apply-templates/></B>
</xsl:template>
```

Таким образом можно последовательно записать и более двух паттернов. Например, следующее правило-шаблон применяется к элементам ATOMIC_NUMBER, ATOMIC_WEIGHT, и SYMBOL (то есть, оно соответствует элементам ATOMIC_NUMBER, ATOMIC_WEIGHT и SYMBOL):

```
<xsl:template match="ATOMIC_NUMBER | ATOMIC_WEIGHT | SYMBOL">
  <B><xsl:apply-templates/></B>
</xsl:template>
```

Сначала обрабатывается оператор /, а уж затем |. Таким образом, следующее правило-шаблон соответствует под-элементу ATOMIC_NUMBER элемента ATOM или элементу ATOMIC_WEIGHT, который может иметь произвольного родителя, а не под-элементу ATOMIC_NUMBER элемента ATOM или под-элементу ATOMIC_WEIGHT элемента ATOM.

```
<xsl:template match="ATOM/ATOMIC_NUMBER|ATOMIC_WEIGHT">
  <B><xsl:apply-templates/></B>
</xsl:template>
```

Проверка с помощью [ ]

Мы уже немного использовали проверку на наличие различных узлов. Однако, можно организовывать и более детальные проверки соответствия узлов с помощью []. Можно выполнять много различных проверок, включая:

- Содержит ли данный элемент определенный дочерний элемент, атрибут или узел другого типа
- Равно ли значение атрибута определенной строке
- Соответствует ли значение элемента определенной строке
- Какое положение данный узел занимает в иерархии

Например, 106-й элемент периодической таблицы, сиборий, был получен только в микроскопических количествах. Даже его самые устойчивые изотопы имеют период полураспада 30 секунд. Имея дело с таким редким и коротко-живущим элементом, практически невозможно измерить его плотность, температуру плавления и другие макро-характеристики. Поэтому в документе с периодической таблицей опущены элементы , описывающие макро-характеристики сибория и других похожих элементов - эти данные просто отсутствуют. Если нужно создать таблицу, в которой сопоставлялся бы атомный номер элемента и его температура плавления, нужно опустить те элементы, для которых температура плавления неизвестна. Чтобы это сделать, можно создать один шаблон для элементов ATOM, у которых имеется дочерний элемент MELTING_POINT, и другой шаблон для элементов, у которых нет такого дочернего элемента:

```
<!-- Шаблон, который ничего не делает ни с одним элементом atom -->
<xsl:template match="ATOM" />
<!-- Для элементов atom, которые содержат информацию о
температуре плавления. Это правило переигрывает предыдущее для
тех атомов, у которых есть температура плавления. -->
<xsl:template match="ATOM[MELTING_POINT]">
  <tr>
    <td><xsl:value-of select="NAME"/></td>
    <td><xsl:value-of select="MELTING_POINT"/></td>
  </tr>
</xsl:template>
```

Обратите внимание, что здесь устанавливается соответствие с элементом ATOM, а не с MELTING_POINT, как в случае с ATOM/MELTING_POINT.

Скобки проверки могут содержать больше, чем просто имя дочернего элемента. Фактически, в них может содержаться любое XPath-выражение. (XPath-выражения - это расширение паттернов соответствия и мы их обсудим в следующем разделе.) Если заданный элемент имеет дочерний элемент, который соответствует этому выражению, считается, что он соответствует и всему паттерну. Например, следующее правило соответствует элементам ATOM, имеющим под-элементы NAME или SYMBOL.

```
<xsl:template match="ATOM[NAME | SYMBOL]">
</xsl:template>
```

Это правило-шаблон соответствует элементу ATOM, у которого есть дочерний элемент DENSITY с атрибутом UNITS:

```
<xsl:template match="ATOM[DENSITY/@UNITS]">
</xsl:template>
```

Чтобы найти все дочерние элементы элемента ATOM, у которых есть атрибуты UNITS, можно

использовать звездочку * для выделения всех дочерних элементов и [@UNITS] для того, чтобы из всех них выбрать только те, у которых имеется атрибут UNITS:

```
<xsl:template match="ATOM">
  <xsl:apply-templates select="*[@UNITS]"/>
</xsl:template>
```

Один из очень полезных паттернов проверки - это проверка равенства строк. Для проверки, соответствует ли значение узла точно данной строке, применяется знак равенства (=). Например, следующий шаблон находит элементы ATOM, которые содержат элемент ATOMIC_NUMBER, содержание которого - строка 10 (элемент неон).

```
<xsl:template match="ATOM[ATOMIC_NUMBER='10']">
  Это неон!
</xsl:template>
```

Проверка содержимого элементов может показаться весьма хитроумной, поскольку нужно при этом иметь точные значения, включая пробелы. Может быть, легче проверять значения атрибутов, поскольку они с меньшей вероятностью будут содержать незначащие пробелы. Например, таблица стилей на листинге 17-9 применяет шаблоны только к тем элементам ATOM, чьи атрибуты STATE имеют значение GAS.

Листинг 17-9:
Таблица стилей XSLT, которая отбирает только те элементы ATOM, чей атрибут STATE имеет значение GAS

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="PERIODIC_TABLE">
    <html>
      <head><title>Gases</title></head>
      <body>
        <xsl:apply-templates/>
      </body>
    </html>
  </xsl:template>
  <xsl:template match="ATOM"/>
  <xsl:template match="ATOM[@STATE='GAS']">
    <P><xsl:value-of select="."/></P>
  </xsl:template>
</xsl:stylesheet>
```

Для более сложных соответствий применяются XPath-выражения. Например, можно выбрать все элементы, чьи имена начинаются с буквы "A" или все элементы с атомным номером меньше 100.

Выражения XPath для отбора узлов

Атрибут select используется в элементах xsl:apply-templates, xsl:value-of, xsl:for-each, xsl:copy-of, xsl:variable, xsl:param и xsl:sort для точного задания обрабатываемых узлов. Значением этого атрибута является выражение, написанное на языке XPath. Язык XPath предоставляет средства идентификации отдельных элементов, групп элементов, фрагментов текста и других частей XML-документа. Синтаксис XPath применяется в XSLT и в XPointer.

Перекрестная ссылка

XPointer описывается в главе 20. XPath описывается далее в этой главе.

Выражения - расширение паттернов соответствия, описанных в предыдущем разделе. То есть, все паттерны являются выражениями, но не все выражения являются паттернами соответствия. Вспомним, что паттерны соответствия позволяют указывать соответствие узлам по имени элементов, по потомкам элемента, его атрибутам, а также позволяет проводить простые проверки на наличие этих элементов. Выражения XPath позволяют отбирать узлы с помощью всех этих критериев, плюс через указание на узлы-предки, родительский узел, соседние узлы, предыдущие и следующие узлы. Более того, выражения не ограничиваются выделением списка узлов, но могут выдавать булевские значения, числа и строки.

Узловые оси

Выражения не ограничиваются выделением дочерних элементов или потомков текущего узла. XPath предоставляет несколько осей, которые можно использовать для выделения различных частей дерева относительно некоторого заданного узла дерева, который именуется контекстным узлом. В XSLT контекстный узел обычно связывается с текущим узлом, которому соответствует шаблон, хотя имеются возможности изменить ситуацию. В таблице 17-2 приводится список узлов и их значений.

Таблица 17-2: Оси выражений

Ось	Отбор
`ancestor`	Предки: родитель контекстного узла, родитель родителя контекстного узла и так далее, до самого корневого узла
`ancestor-or-self`	Предки контекстного узла и сам контекстный узел
`attribute`	Атрибуты контекстного узла
`child`	Непосредственные дети контекстного узла
`descendant`	Потомки: дети контекстного узла, дети детей контекстного узла и так далее
`descendant-or-self`	Потомки и сам контекстный узел
`following`	Все узлы, которые идут после контекстного узла, исключая атрибутные узлы и узлы пространств имен
`following-sibling`	Все узлы, которые идут после контекстного узла и имеют того же самого родителя, что и контекстный узел
`namespace`	Пространство имен контекстного узла
`parent`	Уникальный родительский узел контекстного узла
`preceding`	Все узлы, которые идут до контекстного узла, исключая атрибутные узлы и узлы пространств имен
`preceding-sibling`	Все узлы, которые идут до контекстного узла и имеют того же самого родителя, что и контекстный узел
`self`	Контекстный узел

Указание оси ограничивает выражение таким образом, что оно отбирает только из набора уз-

лов, описанного в соответствующей строке таблицы 17-2. После оси обычно записывается двойное двоеточие (::) и проверка узлов, которая еще более сужает набор выбираемых узлов. Например, проверка узлов может содержать имя отбираемого элемента, как в следующем правиле-шаблоне:

```
<xsl:template match="ATOM">
  <tr>
    <td>
      <xsl:value-of select="child::NAME"/>
    </td>
    <td>
      <xsl:value-of select="child::ATOMIC_NUMBER"/>
    </td>
    <td>
      <xsl:value-of select="child::ATOMIC_WEIGHT"/>
    </td>
  </tr>
</xsl:template>
```

Это правило-шаблон указывает на элементы ATOM. Когда соответствие с элементом ATOM установлено, он становится контекстным узлом. Элементы NAME, ATOMIC_NUMBER и ATOMIC_WEIGHT отбираются среди всех дочерних узлов выбранного элемента ATOM и выводятся в виде табличных ячеек. (Если какой-то из этих элементов присутствует более, чем в одном экземпляре, например, если есть три элемента NAME, тогда они все отбираются, но выводится только значение первого.)

Ось child не позволяет вам делать ничего такого, чего вы не могли бы делать с помощью только имен элементов. Фактически, запись select="ATOMIC_WEIGHT" является просто сокращенной записью select="child::ATOMIC_WEIGHT". Но другие оси интересней.

Указание на родительский элемент недопустимо в паттернах соответствия, но не в регулярных выражениях. Чтобы указать на родителя, нужно использовать ось parent. Например, следующее правило-шаблон соответствует элементам BOILING_POINT, но выводит значение родительского элемента ATOM:

```
<xsl:template match="BOILING_POINT">
  <P><xsl:value-of select="parent::ATOM"/></P>
</xsl:template>
```

Некоторые радиоактивные химические элементы, например, полоний, имеют такой короткий период полураспада, что такие макро-характеристики, как температура кипения и плавления, невозможно измерить. Поэтому не у всех элементов ATOM необходимо имеются дочерние элементы BOILING_POINT. Такого рода правила позволяют создавать шаблоны, которые выводят только те элементы, у которых есть под-элементы с информацией о температуре плавления. Разовьем этот пример. Таблица стилей на листинге 17-10 устанавливает соответствие с элементами MELTING_POINT, но выводит родительский элемент ATOM с помощью записи parent::ATOM.

Листинг 17-10:
Таблица стилей, которая выводит только те элементы, для которых известна температура плавления

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

```
            <xsl:apply-templates select="PERIODIC_TABLE"/>
          </body>
        </html>
      </xsl:template>
      <xsl:template match="PERIODIC_TABLE">
        <h1>Элементы с известной температурой плавления</h1>
        <xsl:apply-templates select=".//MELTING_POINT"/>
      </xsl:template>
      <xsl:template match="MELTING_POINT">
        <p>
          <xsl:value-of select="parent::ATOM"/>
        </p>
      </xsl:template>
  </xsl:stylesheet>
```

Иногда бывает нужно отобрать ближайшего предка определенного типа. Это делается осью ancestor. Например, следующее правило выводит значение ближайшего элемента PERIODIC_TABLE, который содержит элемент SYMBOL, с которым и установлено соответствие.

```
<xsl:template match="SYMBOL">
  <xsl:value-of select="ancestor::PERIODIC_TABLE"/>
</xsl:template>
```

Ось ancestor-or-self похожа на ось ancestor с одной разницей: если контекстный узел удовлетворяет проверке узлов, он тоже будет отобран. Например, следующее правило соответствует всем элементам. В частности, если соответствие установлено элементу PERIODIC_TABLE, именно он и будет отобран в xsl:value-of.

```
<xsl:template match="*">
  <xsl:value-of select="ancestor-or-self::PERIODIC_TABLE"/>
</xsl:template>
```

Проверки улов

За осью может следовать не имя узла, а одна из четырех функций типов узлов:

- comment()
- text()
- processing-instruction()
- node()

Функция comment() отбирает узел-комментарий. Функция text() отбирает текстовый узел. Функция processing-instruction() отбирает узел процессуальной инструкции, а функция node() отбирает все типы узлов. (Звездочка * отбирает только элементные узлы.) Функция processing-instruction() может иметь необязательный аргумент, конкретизирующий имя отбираемой процессуальной инструкции.

Иерархические операторы

Вместе со строковыми выражениями можно использовать операторы / и //. Например, таблица стилей на листинге 17-11 выведет таблицу имен химических элементов, их атомные номера и температуры плавления - только те элементы, у которых имеется информация о температуре плавления. Это делается так: сначала отбирается элемент MELTING_POINT, а затем

находится его родительский элемент NAME и дочерний элемент ATOMIC_NUMBER этого родительского элемента с помощью конструкции select="parent::*/child::NAME)".

Листинг 17-11:
Таблица стилей, выводящая таблицу температур плавления, сопоставленных с атомными номерами

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:template match="/PERIODIC_TABLE">
      <html>
        <body>
          <h1>Atomic Number vs. Melting Point</h1>
          <table>
            <th>Element</th>
            <th>Atomic Number</th>
            <th>Melting Point</th>
            <xsl:apply-templates select="child::ATOM"/>
          </table>
        </body>
      </html>
    </xsl:template>
    <xsl:template match="ATOM">
      <xsl:apply-templates
       select="child::MELTING_POINT"/>
    </xsl:template>
    <xsl:template match="MELTING_POINT">
       <tr>
        <td>
          <xsl:value-of select="parent::*/child::NAME"/>
        </td>
        <td>
          <xsl:value-of
         select="parent::*/child::ATOMIC_NUMBER"/>
        </td>
        <td>
          <xsl:value-of select="self::*"/>
          <xsl:value-of select="attribute::UNITS"/>
        </td>
      </tr>
   </xsl:template>
</xsl:stylesheet>
```

Это не единственный способ решения проблемы. Другой вариант - использовать оси preceding-sibling, following-sibling или обе, если относительное местоположение (preceding - предыдущий или following - следующий) неопределено. Необходимое в этом случае правило-шаблон для элемента MELTING_POINT выглядит так:

```
 <xsl:template match="MELTING_POINT">
    <tr>
     <td>
       <xsl:value-of
        select="preceding-sibling::NAME
              | following-sibling::NAME"/>
     </td>
     <td>
```

```
        <xsl:value-of
         select="preceding-sibling::ATOMIC_NUMBER
               | following-sibling::ATOMIC_NUMBER"/>
      </td>
      <td>
        <xsl:value-of select="self::*"/>
        <xsl:value-of select="attribute::UNITS"/>
      </td>
    </tr>
  </xsl:template>
```

Сокращенный синтаксис

Записи различных осей в таблице 17-2 слишком громоздки и ими неудобно пользоваться. XPath предоставляет сокращенный синтаксис, который заменяет полную запись наиболее распространенных осей, и на практике чаще пользуются именно им. В таблице 17-3 приведены полные и сокращенные записи осей.

Таблица 17-3: Сокращенный синтаксис выражений XPath

Сокращение	Полная запись
.	self::node()
..	parent::node()
name	child:: name
@ name	attribute:: name
//	/descendant-or-self::node()/

На листинге 17-12 приводится другой вариант таблицы стилей 17-11 с применением сокращенного синтаксиса - результат работы этих таблиц стилей совершенно одинаков.

Листинг 17-12:
Таблица стилей с использованием сокращенного синтаксиса, выводящая таблицу температур плавления, сопоставленных с атомными номерами

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:template match="/PERIODIC_TABLE">
      <html>
        <body>
          <h1>Atomic Number vs. Melting Point</h1>
          <table>
            <th>Element</th>
            <th>Atomic Number</th>
            <th>Melting Point</th>
            <xsl:apply-templates select="ATOM"/>
          </table>
        </body>
      </html>
    </xsl:template>
    <xsl:template match="ATOM">
```

```
    <xsl:apply-templates
     select="MELTING_POINT"/>
  </xsl:template>
  <xsl:template match="MELTING_POINT">
     <tr>
      <td>
        <xsl:value-of
         select="../NAME"/>
      </td>
      <td>
        <xsl:value-of
       select="../ATOMIC_NUMBER"/>
      </td>
      <td>
        <xsl:value-of select="."/>
        <xsl:value-of select="@UNITS"/>
      </td>
    </tr>
  </xsl:template>
</xsl:stylesheet>
```

В паттернах соответствия сокращенный синтаксис можно использовать только для осей child и attribute. Полный синтаксис, использующий оси из таблицы 17-2 предназначен только для выражений.

Типы выражений

Каждое выражение имеет одно-единственное значение. Например, выражение 3 + 2 имеет значение 5. Все выражения, с которыми мы уже имели дело, имели в качестве своего значения набор узлов. Но в XSLT есть пять типов выражений:

- Наборы узлов
- Булевские
- Числа
- Строки
- Фрагменты результирующего дерева

Наборы узлов

Набор узлов - это неупорядоченная группа узлов исходного документа. Все оси таблицы 17-2 возвращают наборы узлов, содержащие узлы, которым они соответствуют. Какие узлы входят в этот набор - зависит от контекстного узла, от проверки узлов и от оси.

Например, когда контекстным узлом является элемент PERIODIC_TABLE документа 17-1, XPath-выражение select="child::ATOM" возвращает набор узлов, содержащий оба элемента ATOM этого документа. XPath-выражение select="child::ATOM/child::NAME" возвращает набор узлов, содержащий два элементных узла <NAME>Hydrogen</NAME> и <NAME>Helium</NAME>, если контекстным узлом является элемент PERIODIC_TABLE документа 17-1.

Контекстный узел является представителем списка контекстных узлов. Список контекстных

узлов - это такая группа элементов, в которой они все соответствуют одному и тому же правилу, обычно в результате действия xsl:apply-templates или xsl:for-each. Например, когда таблица стилей 17-12 применяется к документу 17-1, шаблон для ATOM вызывается дважды, один раз для атома водорода, второй раз - для атома гелия. При первом вызове контекстным узлом является элемент ATOM, соответствующий водороду, второй раз - гелию. Но в обоих случаях список контекстных узлов - это набор, состоящий и из водородного и из гелиевого элемента ATOM.

В таблице 17-4 приводится несколько функций, предназначенных для обработки наборов узлов, в качестве аргументов или в качестве контекстных узлов.

Таблица 17-4: Функции, которые обрабатывают или возвращают наборы узлов

Функция	Тип результата	Возвращает
`position()`	число	Позиция контекстного узла в списке контекстных узлов; первый узел в списке имеет позицию 1
`last()`	число	Число узлов в списке контекстных узлов; совпадает с позицией последнего узла в списке
`count( набор узлов )`	число	Количество узлов в наборе узлов.
`id( строка1 строка2 строка3… )`	набор узлов	Набор узлов, содержащий все элементы документа, имеющие ID, перечисленный в списке аргументов; возвращает пустой набор, если элементы с заданным ID отсутствуют
`key( строка name , значение Object )`	набор узлов	Набор узлов, содержащий все узлы документа, у которых имеется ключ с заданным значением. Ключи устанавливаются элементом верхнего уровня xsl:key.
`document( строка URI, строка base )`	набор узлов	Набор узлов документа, на который указывает URI; узлы выбираются по именному якорю или указателю XPointer, имеющемуся в URI. Если именной якорь или указатель XPointer отсутствует, тогда набором узлов принимается корневой элемент названного документа. Относительные URI соотносятся с базовым URI, определенном вторым аргументом. Если второй аргумент опущен, тогда относительные URI соотносятся с URI таблицы стилей (не с исходным документом!).
`local-name( набор узлов )`	строка	Локальное имя (все, что идет после префикса пространства имен) первого узла в аргументе набор узлов; можно использовать без аргументов для получения локального имени контекстного узла
`namespace-uri( набор узлов )`	строка	URI пространства имен первого узла в наборе узлов; можно использовать без аргументов для получения URI пространства имен контекстного узла; возвращает пустую строку, если узел не имеет пространства имен

name(набор узлов)	строка	квалифицированное имя (и префикс и локальная часть) первого узла в аргументе набор узлов; может применяться без аргументов для получения квалифицированного имени контекстного узла
generate-id(набор узлов)	строка	уникальный идентификатор первого узла в аргументе набор узлов; может применяться без аргументов для генерации ID для контекстного узла.

Если одной из этих функций передается аргумент неправильного типа, XSLT пытается конвертировать аргумент в правильный тип; например, конвертируя число 12 в строку "12". Однако, в набор узлов нельзя конвертировать ни один аргумент.

Для определения положения элемента в наборе узлов можно использовать функцию position(). На листинге 17-13 приводится таблица стилей, которая перед именем каждого химического элемента пишет номер его позиции в документе, для этого применяется конструкция <xsl:value-of select="position()"/>.

Листинг 17-13:
Таблица стилей, которая нумерует описания химических элементов в том порядке, в каком они идут в документе

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/PERIODIC_TABLE">
    <HTML>
      <HEAD><TITLE>The Elements</TITLE></HEAD>
      <BODY>
        <xsl:apply-templates select="ATOM"/>
      </BODY>
    </HTML>
  </xsl:template>
  <xsl:template match="ATOM">
    <P>
      <xsl:value-of select="position()"/>.
      <xsl:value-of select="NAME"/>
    </P>
  </xsl:template>
</xsl:stylesheet>
```

Документ 17-1, к которому применена таблица стилей 17-13[35]

Когда эта таблица стилей применяется к документу 17-1, результат будет таков:

```
<HTML>
<HEAD>
<TITLE>The Elements</TITLE>
</HEAD>
<BODY>
<P>1.
      Hydrogen</P>
<P>2.
      Helium</P>
```

35: http://xml.nsu.ru/extra/samples/17-1full4.xml

```
</BODY>
</HTML>
```

Булевские

Булевские переменные могут иметь только два возможных значения: true или false. XSLT позволяет преобразовывать любой вид данных в булевские значения. Это часто происходит скрытно - когда строка, число или набор узлов используются там, где ожидается булевское значение, как например, в атрибуте test элемента xsl:if. Эти преобразования могут также выполняться функцией boolean(), которая конвертирует аргумент любого типа в булевское значение в соответствии с этими правилами:

- Число имеет значение false, если оно равно нулю или NaN (это специальный символ, означающий "Не число" (Not a Number), он применяется для вывода результата деления на нуль или других неправильных операций); иначе имеет значение true.
- Пустой набор узлов имеет значение false. Все другие наборы узлов имеют значение true.
- Пустой фрагмент результирующего дерева имеет значение false. Все другие фрагменты результирующих деревьев имеют значение true.
- Строка нулевой длины имеет значение false. Все другие строки имеют значение true.

Булевские значения также производятся в качестве результата выражений, в которых задействованы следующие операторы:

- = равно
- != не равно
- < меньше, чем (на самом деле <)
- > больше, чем
- <= меньше или равно (на самом деле <=)
- >= больше или равно

Замечание

Знак < недопустим в значениях атрибутов. Поэтому его нужно заменять на <, даже в тех случаях, когда он используется как оператор меньше-чем.

Чаще всего эти операторы используются в предикатных проверках, которые определяют, должно ли быть вызвано правило. Выражения XPath могут содержать не только паттерны, отбирающие определенные узлы, но и предикаты, которые дополнительно фильтруют отобранный набор узлов. Например, child::ATOM отбирает все дочерние элементы ATOM контекстного узла. Однако, child::ATOM[position()=1] отбирает только первый дочерний элемент ATOM контекстного узла. Конструкция [position()=1] - это предикат проверки узла ATOM, который возвращает булевский результат: True, если позиция элемента ATOM равна единице, а иначе false. Каждая проверка узла может содержать любое число предикатов, но обычно используется не более одного.

Например, следующее правило-шаблон применяется только к первому элементу ATOM в периодической таблице - это достигается проверкой, равна ли позиция элемента единице.

```
<xsl:template match="PERIODIC_TABLE/ATOM[position()=1]">
  <xsl:value-of select="."/>
</xsl:template>
```

А вот это правило-шаблон применяется ко всем элементам ATOM, которые не являются пер-

вым дочерним элементом элемента PERIODIC_TABLE - это достигается проверкой, является ли номер позиции больше, чем единица:

```
<xsl:template match="PERIODIC_TABLE/ATOM[position()>1]">
  <xsl:value-of select="."/>
</xsl:template>
```

Ключевые слова and и or логически комбинируют булевские выражения в соответствии с обычными логическими правилами. Например, допустим, вам нужен шаблон, который соответствует элементу ATOMIC_NUMBER, который является первым и последним дочерним элементом своего родительского элемента; то есть, является единственным дочерним элементом. Чтобы этого достичь, в правиле-шаблоне нужно применить ключевое слово and:

```
<xsl:template
 match="ATOMIC_NUMBER[position()=1 and position()=last()]">
  <xsl:value-of select="."/>
</xsl:template>
```

Если первое условие не выполняется и имеет значение false, тогда все выражение and имеет значение false, поэтому второе условие даже не будет проверяться.

Следующий шаблон соответствует и первому и последнему элементу ATOM среди дочерних элементов - это достигается проверкой, является ли номер позиции элемента равным единице или равным количеству элементов в наборе:

```
<xsl:template match="ATOM[position()=1 or position()=last()]">
  <xsl:value-of select="."/>
</xsl:template>
```

Тут используется логическое или or, так что этот шаблон будет соответствовать и если выполняются оба условия. То сеть, он будет соответствовать элементу ATOM, который является первым или последним дочерним элементом своего родителя. Если верно первое, то верно все выражение or, а второе условие даже не будет проверяться.

Функция not() переворачивает результат операции. Например, следующее правило-шаблон соответствует всем элементам ATOM, которые не являются первыми дочерними элементами своих родителей:

```
<xsl:template match="ATOM[not(position()=1)]">
  <xsl:value-of select="."/>
</xsl:template>
```

То же самое правило-шаблон может быть записано с использованием оператора "не равно" !=:

```
<xsl:template match="ATOM[position()!=1]">
  <xsl:value-of select="."/>
</xsl:template>
```

Следующее правило-шаблон соответствует всем элементам ATOM, которые не являются ни первыми, ни последними дочерними элементами своих родителей:

```
<xsl:template match =
 "ATOM[not(position()=1 or position()=last())]">
  <xsl:value-of select="."/>
</xsl:template>
```

В XSLT отсутствует оператор "исключающего или". Однако, его можно сформировать из конъюнкции not(), and и or. Например, следующее правило-шаблон отбирает те элементы ATOM, которые являются либо первыми либо последними дочерними элементами, но не обоими сразу:

```
<xsl:template
 match="ATOM[(position()=1 or position()=last())
             and not(position()=1 and position()=last())]">
  <xsl:value-of select="."/>
</xsl:template>
```

Есть еще три функции, возвращающих булевские значения:

- true() всегда возвращает булевское true
- false() всегда возвращает булевское false
- lang(code) возвращает true, если текущий узел имеет тот же самый язык (заданный в атрибуте xml:lang), как и аргумент code

Числа

Числа в XPath представляются как 64-битные числа с плавающей запятой двойной точности. Даже числа 42 или -7000, которые выглядят как целые числа, хранятся также. Не-числовые значения, например, строки или булевские значения, автоматически преобразуются, когда необходимо в числа или по требованию пользователя с помощью функции number(), которая действует в соответствии со следующими правилами:

- Булевское значение преобразуются в 1, если оно равно true; и 0, если оно равно false.
- Из строки удаляются начальные и завершающие пробелы, а затем она преобразуется в число по вполне понятным правилам; например, строка "12" конвертируется в число 12. Если строка не может быть проинтерпретирована как число, она конвертируется в специальный символ NaN ("Не число").
- Наборы узлов и фрагменты дерева-результата конвертируются в строки; строки конвертируются в числа.

Например, следующий шаблон выводит только искусственные трансурановые элементы, то есть, те химические элементы, атомный номер которых превышает 92 (атомный номер урана). Набор узлов, полученный из ATOMIC_NUMBER, автоматически конвертируется в строчное значение текущего узла ATOMIC_NUMBER. Затем эта строка конвертируется в число.

```
<xsl:template match="/PERIODIC_TABLE">
  <HTML>
    <HEAD><TITLE>Трансурановые элементы</TITLE></HEAD>
    <BODY>
      <xsl:apply-templates select="ATOM[ATOMIC_NUMBER>92]"/>
    </BODY>
  </HTML>
</xsl:template>
```

XPath предоставляет четыре стандартных арифметических операции:

- + прибавить
- - вычесть
- * умножить
- div делить (более распространенный в этих случаях символ / в XPath используется в других целях)

Например, <xsl:value-of select="2+2"/> вставляет в выходной документ строку "4". Эти операции обычно используются в проверках. Например, следующее правило отбирает те элементы, чей атомный вес более, чем в два раза превышает их атомный номер:

```
<xsl:template match="/PERIODIC_TABLE">
  <HTML>
    <BODY>
      <H1>High Atomic Weight to Atomic Number Ratios</H1>
      <xsl:apply-templates
        select="ATOM[ATOMIC_WEIGHT > 2 * ATOMIC_NUMBER]"/>
    </BODY>
  </HTML>
</xsl:template>
```

А вот этот шаблон выводит отношения атомного веса химических элементов к их атомному номеру:

```
<xsl:template match="ATOM">
  <p>
    <xsl:value-of select="NAME"/>
    <xsl:value-of select="ATOMIC_WEIGHT div ATOMIC_NUMBER"/>
  </p>
</xsl:template>
```

В XPath также имеется менее распространенный бинарный оператор mod, который вычисляет остаток деления двух чисел. При использовании вместе с функцией position(), этот оператор позволяет, например, выводить каждый второй элемент ATOM измененным цветом. Для этого нужно просто задать шаблоны, которые применяют различные стили, в зависимости от того, чему равно выражение position() mod 2 - единице или нулю. Например, в следующих двух правилах в двух различных стилях используются различные цвета ячеек таблицы:

```
<xsl:template match="ATOM[position() mod 2 = 1]">
    <tr>
     <td><xsl:value-of select="NAME"/></td>
     <td><xsl:value-of select="ATOMIC_NUMBER"/></td>
     <td><xsl:apply-templates select="MELTING_POINT"/></td>
    </tr>
</xsl:template>
<xsl:template match="ATOM[position() mod 2 = 0]">
    <tr style="color: #666666">
     <td><xsl:value-of select="NAME"/></td>
     <td><xsl:value-of select="ATOMIC_NUMBER"/></td>
     <td><xsl:apply-templates select="MELTING_POINT"/></td>
    </tr>
</xsl:template>
```

Вы можете изменить делитель на 3 и применять особый стиль к каждому третьему элементу, и так далее.

И наконец, в XPath имеется четыре функции для работы над числами:

- floor() возвращает самое большое целое число, которое меньше или равно данному числу
- ceiling() возвращает самое маленькое целое число, которое больше или равно данному числу
- round() округляет число до ближайшего целого
- sum() возвращает сумму своих аргументов

Например, следующее правило-шаблон оценивает количество нейтронов в атоме, вычитая из атомного веса (средний вес нейтронов и протонов в естественном распределении изотопов) атомный номер элемента (количество протонов), и округляя результат до ближайшего целого числа:

```
<xsl:template match="ATOM">
  <p>
    <xsl:value-of select="NAME"/>
    <xsl:value-of
     select="round(ATOMIC_WEIGHT - ATOMIC_NUMBER)"/>
  </p>
</xsl:template>
```

Следующее правило вычисляет средний атомный вес всех атомов в таблице, складывая атомные весы, а затем разделяя результат на количество атомов:

```
<xsl:template match="/PERIODIC_TABLE">
  <HTML>
    <BODY>
    <H1>Средний атомный вес</H1>
      <xsl:value-of
       select="sum(descendant::ATOMIC_WEIGHT)
               div count(descendant::ATOMIC_WEIGHT)"/>
    </BODY>
  </HTML>
</xsl:template>
```

Строки

Строка - это последовательность символов Unicode. Другие типы данных могут конвертироваться в строки с использованием функции string(), которая действует в соответствии со следующими правилами:

- Наборы узлов конвертируются в строки на основе значения первого узла в наборе, как идет вычисление в элементе xsl:value-of в соответствии с правилами таблицы 17-1.
- Фрагмент результирующего дерева конвертируется в строку таким образом, будто фрагмент заключен в элемент. Берется значение этого воображаемого элемента. И снова, значение этого элемента вычисляется так же, как в элементе xsl:value-of в соответствии с правилами таблицы 17-1. То есть, объединяются все текстовые части фрагмента дерева (но не разметка).
- Числа конвертируются в строки, при этом считается, что числа записаны в европейском стиле, например, -12 или 3.1415292.
- Булевское значение false конвертируется в английское слово false. Булевское значение true конвертируется в английское слово true.

Кроме функции string(), в XSLT имеется еще 10 функций для работы со строками. Их список приводится в таблице 17-5.

Таблица 17-5: Функции XPath для работы со строками

Функция	Тип результата	Возвращает
starts-with(основная строка , префиксная строка)	Булевское значение	True, если основная строка начинается с префиксной строки, иначе - false
contains(содержащая строка , содержимая строка)	Булевское значение	True, если содержимая строка является частью содержащей строки, иначе false
substring (строка , офсет, длина)	Строка	Под-строка строки заданной длины, начиная с заданного аргументом офсет; или все символы, начиная от заданного аргументом офсет до конца

		строки, если аргумент длина опущен; длина и офсет округляются при необходимости до ближайшего целого значения
substring-before (строка , строка-маркер)	Строка	Часть строки от первого символа до (не включая) первого появления строки-маркера
substring-after (строка , строка-маркер)	Строка	Часть строки от конца первого появления строки-маркера до конца строки; первый символ строки имеет офсет 1
string-length(строка)	Число	Число символов в строке
normalize-space(строка)	Строка	Удаляются пробелы до и после строки, несколько подряд идущих пробелов внутри строки заменяются на один; если аргумент опущен, нормализуется строчное значение контекстного узла
translate(строка , замещаемый текст , замещающий текст)	Строка	Возвращает строку, в которой символы замещаемой строки заменены на символы замещающей строки
concat (строка1, строка2, ...)	Строка	Возвращает конкатенацию нескольких строк, которые передаются в качестве аргументов, в том порядке, в каком они идут
format-number(число , форматная строка , локальная строка)	Строка	Возвращает строчную форму числа, отформатированную в соответствии с заданной форматной строкой, как в Java 1.1 (класс java.text.DecimalFormat) (см. здесь[36]); локальная строка - это необязательный аргумент, предоставляющий имя элемента xsl:decimal-format, который используется для интерпретации форматной строки

Фрагменты результирующего дерева

Фрагмент результирующего дерева - это кусок XML-документа, который не является законченным узлом или набором узлов. Например, использование функции document() с URI, который указывает на середину элемента, может привести к получению фрагмента результирующего дерева. Фрагменты результирующего дерева также возвращаются некоторыми функциями-расширениями (функциями, уникальными для каждой конкретной реализации XSLT).

Поскольку фрагменты результирующих деревьев не являются правильным XML, с ними мало что можно сделать. Фактически, единственной допустимой операцией является их конвертация в строку или булевское значение с применением функций string() и boolean() соответственно.

Дефолтные шаблонные правила

36: http://java.sun.com/products/jdk/1.1/docs/api/java.text.DecimalFormat.html

Необходимость тщательно отслеживать в таблице стилей XSLT всю иерархию XML-документа может показаться неудобной. Это особенно заметно, если документ не следует какой-то стабильной, предсказуемой схеме, вроде периодической таблицы химических элементов, а представляет собой разнородный набор текста и разметки, как многие веб-страницы. В этих случаях нужно иметь общие правила, которые могут найти элемент и применить к нему шаблон вне зависимости от того, где он находится в исходном документе.

Чтобы облегчить эту работу, в XSLT имеется несколько дефолтных правил-шаблонов, которые имплицитно включаются во все таблицы стилей. Первое дефолтное правило соответствует корню и элементным узлам и применяет шаблон ко всем дочерним узлам. Второе дефолтное правило соответствует текстовым узлам и атрибутам, копируя их значения в выходной поток. В сумме эти два правила приводят к тому, что даже пустая таблица стилей XSLT, в которой есть только один пустой элемент xsl:stylesheet, прямо выдаст символьные данные исходного XML-документа.

Дефолтные правила для элементов

Первое дефолтное правило применяется к элементным узлам и корневому узлу:

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

Запись *|/ - это сокращение XPath, означающее "любой элементный узел или корневой узел." Задача этого правила - обеспечить рекурсивную обработку всех элементов, даже если к ним не применяется ни одно другое явно заданное правило. То есть, пока какое-то другое правило не переиграет это (особенно это касается корневого элемента), будут обрабатываться все элементные узлы.

Однако, если имеется явно заданное правило для любого родительского узла, дефолтное правило больше не будет активизироваться для его дочерних элементов, если правило-шаблон для родительского узла не содержит элемент xsl:apply-templates. Например, можно прекратить всю обработку, соответствующую корневому элементу, не применять шаблоны к дочерним узлам и не использовать xsl:for-each, если воспользоваться следующим правилом:

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

Дефолтное правило для текстовых узлов и атрибутов

Исключительно наблюдательные читатели могли заметить несколько примеров, которые, кажется, выводят содержание некоторых элементов, но при этом на самом деле не используют значения выводимых элементов! Это содержимое обрабатывается дефолтным правилом для текстовых и атрибутных узлов. Правило таково:

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

Это правило соответствует всем текстовым и атрибутным узлам (match="text()|@*") и выводит их значение (<xsl:value-of select="."/>). Другими словами, правило копирует текст из исходного документа в выходной. Это правило гарантирует, что по крайней мере текст элементов будет выводиться, даже если это не указано ни одним правилом. Это правило может быть

переиграно другим, определенным для конкретного элемента - если вы хотите получить на выходе не просто его текстовое содержимое.

Кроме того, это правило копирует значения атрибутов (но не их имена). Однако, происходит преобразование из атрибутов в исходном документе в простой текст на выходе. Поскольку не существует дефолтного правила, которое бы применяло к атрибутам, это правило не будет активировано до тех пор, пока какое-то другое не-дефолтное правило в таблице стилей не применит шаблоны к атрибутам одного или нескольких элементов.

Дефолтные правила для процессуальных инструкций и комментариев

Существует также дефолтное правило для процессуальных инструкций и комментариев. Оно говорит: с ними ничего не делать, то есть попросту убирать из выходного документа комментарии и процессуальные инструкции, будто их не существует. Правило выглядит не так:

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

Конечно, вы можете заменить это правило своими собственными правилами для обработки процессуальных инструкций и комментариев, если нужно.

Реализация дефолтных правил

В совокупности дефолтные правила означают: если применить к XML-документу пустую таблицу стилей, в которой нет ничего, кроме пустого элемента xsl:stylesheet или xsl:transform (как на листинге 17-14), из исходного документа в выходной копируется все секции #PCDATA, содержащиеся в элементах исходного документа. Но в выходном документе будет отсутствовать разметка. Это правила чрезвычайно низкого приоритета, поэтому правила с любыми другими соответствиями будут иметь приоритет над дефолтными правилами.

Листинг 17-14: Пустая таблица стилей XSLT

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

Замечание

Одна из основных причин трудностей и ошибок в связи с реализацией XSLT в Internet Explorer 5.0 и 5.5, заключается в том, что IE не предоставляет ни одного из этих дефолтных правил. Вам нужно заботиться о том, чтобы каждому узлу, содержание которого вы хотите вывести (включая его потомков), имелось явное соответствие в одном из шаблонов.

Формирование выходного потока

Часто бывает необходимо передать на выход различную разметку в зависимости от каких-то условий. Например, может понадобиться переместить содержание элемента FILENAME в атрибут HREF элемента A, или заменить элемент одного типа на элементы различных типов в зависимости от значения атрибута. Это достигается использованием элементов xsl:element,

xsl:attribute, xsl:processing-instruction, xsl:comment и xsl:text. XSLT-инструкции, которые используются внутри этих элементов и шаблонов значений атрибутов, применяются в значениях атрибутов этих элементов для варьирования их выходного результата.

Шаблоны значений атрибутов

Шаблоны значений атрибутов копируют данные из исходного документа в значения атрибутов выходного документа. Например, допустим, вам нужно конвертировать периодическую таблицу элементов в список пустых элементов ATOM, в которых информация размещается в атрибутах:

```
<ATOM NAME="Vanadium"
  ATOMIC_WEIGHT="50.9415"
  ATOMIC_NUMBER="23"
/>
```

Чтобы это сделать, нужно извлечь содержание элементов исходного документа и поместить его в атрибуты выходного документа. Первое, что, скорее всего, вы попытаетесь попробовать, будет выглядеть так:

```
<xsl:template match="ATOM">
  <ATOM NAME="<xsl:value-of select='NAME'/>"
    ATOMIC_WEIGHT="<xsl:value-of select='ATOMIC_WEIGHT'/>"
    ATOMIC_NUMBER="<xsl:value-of select='ATOMIC_NUMBER'/>"
  />
</xsl:template>
```

Но это не правильный XML. Нельзя использовать символ < внутри значения атрибута. Более того, чрезвычайно трудно создать программное обеспечение, которое могло бы парсить такие конструкции в самом общем случае.

Вместо этого следует внутри значений атрибутов данные заключить в фигурные скобки {}, которые заменяют элемент xsl:value-of. Правильный способ записи приведенного выше примера выглядит так:

```
<xsl:template match="ATOM">
  <ATOM NAME="{NAME}"/>
    ATOMIC_WEIGHT="{ATOMIC_WEIGHT}"
    ATOMIC_NUMBER="{ATOMIC_NUMBER}"
  />
</xsl:template>
```

В выходном документе {NAME} заменяется на значение дочернего элемента NAME указанного элемента ATOM. {ATOMIC_WEIGHT} заменяется на значение его дочернего элемента ATOMIC_WEIGHT, {ATOMIC_NUMBER} заменяется на значение его дочернего элемента ATOMIC_NUMBER и так далее.

Шаблоны значений атрибутов могут обладать еще более сложным паттерном, чем просто имя элемента. Фактически, в шаблоне значения атрибута можно использовать любое выражение XPath. Например, следующее правило-шаблон отбирает элементы DENSITY в документе 17-1.

```
<xsl:template match="DENSITY">
  <BULK_PROPERTY
    NAME="DENSITY"
    ATOM="{../NAME}"
    VALUE="{normalize-space(.)}"
    UNITS="{@UNITS}"
```

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

Шаблон преобразует их в элементы BULK_PROPERTY, которые выглядят так:

```
<BULK_PROPERTY NAME="DENSITY" ATOM="Helium"
  VALUE="0.0001785" UNITS="grams/cubic centimeter"/>
```

Значения атрибутов можно формировать не только одним шаблоном значения атрибута. Можно комбинировать шаблон значения атрибута с буквальными данными или с другими шаблонами значений атрибутов. Например, следующее правило-шаблон соответствует элементам ATOM и заменяет их на их имена, отформатированные как ссылки на файлы в виде H.html, He.html и так далее. Имя файла создается с помощью шаблона значения атрибута {SYMBOL}, а буквальные данные добавляют точку и расширение файла.

```
<xsl:template match="ATOM">
  <A HREF="{SYMBOL}.html">
    <xsl:value-of select="NAME"/>
  </A>
</xsl:template>
```

В значение атрибута можно включать несколько шаблонов значения атрибута. Например, следующее правило-шаблон включает в значение атрибута VALUE единицы измерения плотности, а не размещает их в отдельном атрибуте:

```
<xsl:template match="DENSITY">
  <BULK_PROPERTY
    NAME="DENSITY"
    ATOM="{../NAME}"
    VALUE="{normalize-space(.)} {@UNITS}"
  />
</xsl:template>
```

Шаблоны значений атрибутов можно использовать во многих атрибутах таблицы стилей XSLT. Это особенно важно в элементах xsl:element, xsl:attribute и xsl:processing-instruction, где шаблоны значений атрибутов позволяют дизайнеру организовать выбор - какие именно элементы, атрибуты или процессуальные инструкции появятся в выходном документе еще до того, как исходный документ будет считан. Нельзя использовать шаблоны значений атрибутов в качестве значений атрибутов select или match, в атрибуте xmlns, в атрибутах, которые задают имена других XSLT-элементов или в атрибутах элементов верхнего уровня (которые являются непосредственными дочерними элементами элемента xsl:stylesheet).

Введение элементов в выходной поток с помощью xsl:element

Обычно элементы включаются в выходной документ просто с помощью буквального включения начального и конечного тэга в содержимое шаблона. Например, чтобы вставить элемент P, вы можете просто вписать <P> и </P> в подходящие места таблицы стилей. Однако, иногда возникает необходимость для определения, какой именно элемент поместить в выходной поток, на основе каких-то деталей исходного документа. Такая необходимость, например, может возникнуть, когда производится преобразование исходного словаря, в котором для хранения информации используются атрибуты, в выходной словарь, в котором для этого применяются элементы.

Элемент xsl:element вставляет в выходной документ элемент. Имя элемента задается шаблоном значения атрибута в атрибуте name элемента xsl:element. Содержимое элемента выводится из содержимого элемента xsl:element, которое может в себя включать дополнительные

инструкции xsl:attribute, xsl:processing-instruction, и xsl:comment (все они будут обсуждаться ниже) для вставки соответствующих конструкций.

Предположим, например, вам нужно заменить элементы ATOM элементами GAS, LIQUID, и SOLID в зависимости от значения атрибута STATE. Используя xsl:element, это можно сделать с помощью единственного правила, которое конвертирует значение атрибута STATE в имя элемента. Вот как выглядит такое правило:

```
<xsl:template match="ATOM">
  <xsl:element name="{@STATE}">
    <NAME><xsl:value-of select="NAME"/></NAME>
    <!-- rules for other children -->
  </xsl:element>
</xsl:template>
```

Применяя более сложные шаблоны значений атрибутов, можно выполнить все нужные вам преобразования и вычисления.

Введение атрибутов в выходной поток с помощью xsl:attribute

Можно помещать атрибуты в выходной документ, просто вписывая в шаблон атрибуты буквально, например, чтобы вставить элемент DIV с атрибутом ALIGN, имеющим значение CENTER, можно просто вписать <DIV ALIGN="CENTER"> и </DIV> в нужные места таблицы стилей. Однако, часто для определения значения атрибута и даже его имени приходится опираться на какие-то данные исходного документа.

Допустим, например, вам нужна таблица стилей, которая отбирает имена атомов и форматирует их как ссылки на файлы H.html, He.html, Li.html и так далее:

```
<LI><A HREF="H.html">Hydrogen</A></LI>
<LI><A HREF="He.html">Helium</A></LI>
<LI><A HREF="Li.html">Lithium</A></LI>
```

Каждый элемент в исходном документе будет выдавать различное значение атрибута HREF. Элемент xsl:attribute вычисляет имя и значение атрибута и вставляет их в выходной поток. Каждый элемент xsl:attribute является дочерним элементом элемента xsl:element или буквального элемента. Атрибут, вычисленный элементом xsl:attribute, будет привязываться к элементу, вычисленному его родительским элементом и вставляться в выходной документ. Имя атрибута определяется атрибутом name элемента xsl:attribute. Значение атрибута задается содержимым элемента xsl:attribute. Например, следующее правило-шаблон выдает приведенный выше результат:

```
<xsl:template match="ATOM">
  <LI><A>
    <xsl:attribute name="HREF">
      <xsl:value-of select="SYMBOL"/>.html
    </xsl:attribute>
    <xsl:value-of select="NAME"/>
  </A></LI>
</xsl:template>
```

Все элементы xsl:attribute должны идти впереди всего остального содержимого их родительских элементов. Нельзя добавлять элементу атрибут уже после того, как вы начали вписывать его содержимое. Например, следующий шаблон не правильный:

```
<xsl:template match="ATOM">
  <LI><A>
```

```
      <xsl:value-of select="NAME"/>
      <xsl:attribute name="HREF">
        <xsl:value-of select="SYMBOL"/>.html
      </xsl:attribute>
    </A></LI>
  </xsl:template>
```

Задание набора атрибутов

Часто бывает нужно применить одну и ту же группу атрибутов к многим различным элементам - одного и того же типа или к разным. Например, может понадобиться применить атрибут style к каждой ячейке HTML-таблицы. Чтобы сделать это просто, можно задать один или несколько атрибутов как члены набора атрибутов - это делается на верхнем уровне таблицы стилей с помощью xsl:attribute-set, а затем можно включать в элементы этот набор атрибутов с помощью атрибута xsl:use-attribute-sets.

Например, этот элемент xsl:attribute-set задает набор атрибутов cellstyle, который содержит атрибут font-family со значением New York, Times New Roman, Times, serif и атрибут font-size со значением 12pt.

```
<xsl:attribute-set name="cellstyle">
  <xsl:attribute name="font-family">
    New York, Times New Roman, Times, serif
  </xsl:attribute>
  <xsl:attribute name="font-size">12pt</xsl:attribute>
</xsl:attribute-set>
```

Следующее правило-шаблон применяет этот набор атрибутов к элементам td выходного документа:

```
<xsl:template match="ATOM">
  <tr>
    <td xsl:use-attribute-sets="cellstyle">
      <xsl:value-of select="NAME"/>
    </td>
    <td xsl:use-attribute-sets="cellstyle">
      <xsl:value-of select="ATOMIC_NUMBER"/>
    </td>
  </tr>
</xsl:template>
```

Элемент может задействовать несколько наборов атрибутов - для этого нужно в значении атрибута xsl:use-attribute-sets перечислить все названия наборов атрибутов, разделяя их пробелами. Все атрибуты из всех наборов будут в этом случае применены к данному элементу. Например, этот элемент td будет иметь атрибуты и из набора атрибутов cellstyle и из набора numberstyle:

```
      <td xsl:use-attribute-sets="cellstyle numberstyle">
        <xsl:value-of select="ATOMIC_NUMBER"/>
      </td>
```

Если один и тот же атрибут задается в нескольких различных наборах, тогда используется определение из последнего перечисленного набора. Если имеется несколько наборов атрибутов с одним и тем же именем (это может случиться, когда одна таблица стилей импортирует другую), тогда атрибуты двух наборов смешиваются. Если одинаково называющиеся наборы атрибутов определяют один и тот же атрибут, тогда выбирается значение атрибута из

набора, который имеет более высокую важность. Ошибкой является наличие в таблице стилей одноименных наборов атрибутов одинаковой важности, если в них определяется один и тот же атрибут.

Можно включать наборы атрибутов в определенные элементы, добавляя элемент use-attribute-sets в элементы xsl:element, xsl:copy или xsl:attribute-set. Пример:

```
<xsl:element name="td" use-attribute-sets="cellstyle">
  <xsl:value-of select="ATOMIC_NUMBER"/>
</xsl:element>
```

Префикс xsl: не является необходимым (и даже нежелателен), когда use-attribute-sets находится в атрибуте XSLT-элемента, а не в элементе результирующего набора.

Создание процессуальных инструкций с помощью xsl:processing-instruction

Элемент xsl:processing-instruction помещает в выходном документе процессуальную инструкцию. Цель (target) процессуальной инструкции задается необходимым атрибутом name. Содержимое элемента xsl:processing-instruction преобразуется в содержание процессуальной инструкции. Например, следующее правило заменяет элемент PROGRAM на процессуальную инструкцию gcc:

```
<xsl:template match="PROGRAM">
  <xsl:processing-instruction name="gcc"> -O4
  </xsl:processing-instruction>
</xsl:template>
```

Элемент PROGRAM в исходном документе заменяется на процессуальную инструкцию в выходном:

```
<?gcc -O4
  ?>
```

Содержимое элемента xsl:processing-instruction может включать в себя элементы xsl:value-of и xsl:apply-templates, поскольку они выдают чистый текст. Например,

```
<xsl:template match="PROGRAM">
  <xsl:processing-instruction name="gcc">-O4
    <xsl:value-of select="NAME"/>
  </xsl:processing-instruction>
</xsl:template>
```

Элемент xsl:processing-instruction не может содержать инструкции xsl:element и другие, которые вставляют элементы или атрибуты. Кроме того, элемент xsl:processing-instruction не должен содержать никаких других инструкций или текста, которые вставляют в выходной поток конструкцию ?>, поскольку она трактуется как конец процессуальной инструкции.

Создание комментариев с помощью xsl:comment

Элемент xsl:comment вставляет в выходной документ комментарий. У него нет атрибутов, а содержит он текст нужного комментария. Пример:

```
<xsl:template match="ATOM">
  <xsl:comment>Здесь был атом.</xsl:comment>
</xsl:template>
```

Это правило заменяет узлы ATOM на комментарий:

```
<!--Здесь был атом.-->
```

Содержимое элемента xsl:comment может включать в себя элементы xsl:value-of и элементы xsl:apply-templates, поскольку результатом действия этих инструкций является текст. Нельзя встраивать внутрь элемента xsl:comment инструкции xsl:element и другие, которые создают элементы или атрибуты. Кроме того, элемент xsl:comment не может содержать никаких инструкций или буквального текста, которые могут вставить в комментарий двойное тире - это может привести к неправильному комментарию.

Создание текста с помощью xsl:text

Элемент xsl:text вставляет в выходной поток свое содержимое в качестве буквального текста. Например, следующее правило заменяет каждый элемент ATOM строкой "Здесь был атом."

```
<xsl:template match="ATOM">
  <xsl:text>Здесь был атом.</xsl:text>
</xsl:template>
```

Элемент xsl:text используется не очень часто, поскольку обычно можно просто вписать в шаблон любой текст. Однако, элемент xsl:text имеет несколько преимуществ. Во-первых, он сохраняет пробелы, переносы и табуляцию, даже если узел не содержит ничего, кроме этих символов. По умолчанию, XSLT-процессоры удаляют все текстовые узлы, которые содержат только эти символы. Поэтому такое свойство элемента xsl:text полезно при работе с текстами стихов, листингами кода или другими текстами, в которых пробелы и символы форматирования важны.

Во вторых, этот элемент позволяет прямо вставлять в выходной документ символы < и & без предварительного преобразования их в < и &. Чтобы это сделать, поместите в элемент xsl:text общие ссылки на сущности для этих символов (< или &), а затем установите значение атрибута disable-output-escaping элемента xsl:text в yes. Это бывает полезно, когда нужно в код выходного документа включить JavaScript, например:

```
<xsl:template match="SCRIPT">
  <script language="javascript">
    <xsl:text disable-output-escaping="yes">
      &lt;!-- if (
         location.host.tolowercase().indexof("ibiblio")
         &lt; 0) {
           location.href="http://www.ibiblio.org/xml/";
        }
      } // --&gt;
    </xsl:text>
  </script>
</xsl:template>
```

Так можно на выходе получить неправильный XML. (Здесь именно такой случай.) Но если вы хотите получить не-XML формат, например, HTML или TeX, это то, что нужно. Однако и в этом случае таблица стилей и исходный документ остаются правильными XML-документами.

Копирование контекстного узла с помощью элемента xsl:copy

Элемент xsl:copy копирует исходный узел в выходное дерево. При этом дочерние элементы, атрибуты и другое содержимое автоматически тоже не копируются. Однако, содержимое элемента xsl:copy, расположенного внутри элемента xsl:template, который отбирает дочерние узлы копируемого элемента, может также скопировать и их. Это бывает очень полезно в тех случаях, когда ведется трансформация документа, имеющего один словарь разметки в другой, обладающий схожим словарем. Например, следующее правило-шаблон удаляет все атрибуты и дочерние элементы из элементов ATOM и меняет их на их содержимое, заключенное в элемент b:

```
<xsl:template match="ATOM">
   <xsl:copy>
     <b><xsl:value-of select="."/></b>
   </xsl:copy>
</xsl:template>
```

Следующий полезный шаблон на основе элемента xsl:copy может провести идентичное преобразование, то есть преобразовать документ в него самого. Этот шаблон выглядит так:

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

Вы можете немного изменить идентичное преобразование, чтобы получить похожий, а не идентичный документ. Например, таблица на листинге 17-15 оставляет документ неизменным, только удаляет из него комментарии. Для этого из атрибутов match и select удалено соответствие comment().

Листинг 17-15:
Таблица стилей XSLT, которая удаляет из документа комментарии

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template
     match="*|@*|processing-instruction()|text()">
    <xsl:copy>
      <xsl:apply-templates
        select="*|@*|processing-instruction()|text()"/>
    </xsl:copy>
  </xsl:template>
</xsl:stylesheet>
```

Элемент xsl:copy копирует только исходный узел. Можно копировать и другие узлы, может быть, несколько, применяя элемент xsl:copy-of. Атрибут select этого элемента отбирает копируемые узлы. Например, на листинге 17-16 приведена таблица стилей, которая использует элемент xsl:copy-of для удаления из периодической таблицы элементов тех из них, в которых не содержится информации о точке плавления - в ней происходит копирование на выход

только тех элементов ATOM, у которых есть дочерний элемент MELTING_POINT.

Листинг 17-16:
Таблица стилей, которая копирует только те элементы ATOM, у которых есть дочерний элемент MELTING_POINT

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:template match="/PERIODIC_TABLE">
      <PERIODIC_TABLE>
        <xsl:apply-templates select="ATOM"/>
      </PERIODIC_TABLE>
    </xsl:template>
    <xsl:template match="ATOM">
      <xsl:apply-templates select="MELTING_POINT"/>
    </xsl:template>
   <xsl:template match="MELTING_POINT">
     <xsl:copy-of select=".."/>
  </xsl:template>
</xsl:stylesheet>
```

Замечание

На листингах 17-15 и 17-16 приведены примеры XSL-преобразований из исходного словаря в тот же самый словарь. В отличие от других примеров этой главы, результатом преобразований в них не является правильный HTML.

Нумерация узлов с помощью элемента xsl:number

Элемент xsl:number вставляет в выходной документ форматированное целое число. Значение целого числа задается атрибутом value. Атрибут содержит число, которое округляется до ближайшего целого числа, а затем оно форматируется в соответствии со значением атрибута format. У каждого из этих атрибутов имеются разумные значения по умолчанию, например, взгляните на таблицу стилей для элемента ATOM на листинге 17-17.

Листинг 17-17:
Таблица стилей XSLT, которая нумерует атомы

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:template match="PERIODIC_TABLE">
      <html>
        <head><title>The Elements</title></head>
        <body>
          <table>
            <tr><xsl:apply-templates select="ATOM"/></tr>
          </table>
        </body>
      </html>
    </xsl:template>
    <xsl:template match="ATOM">
      <td><xsl:number value="ATOMIC_NUMBER"/></td>
```

```
        <td><xsl:value-of select="NAME"/></td>
      </xsl:template>
</xsl:stylesheet>
```

Когда эта таблица стилей применяется к документу 17-1, в результате мы получим следующее:

```
<html>
<head>
<title>The Elements</title>
</head>
<body>
<table>
<tr>
<td>1</td><td>Hydrogen</td><td>2</td><td>Helium</td>
</tr>
</table>
</body>
</html>
```

Документ 17-1, преобразованный таблицей стилей 17-7[37]

Каждому элементу соответствует его атомный номер. Атрибут value может содержать любые данные, которые можно преобразовать в число с помощью правил XPath. В данном случае преобразуется дочерний элемент ATOMIC_NUMBER указанного элемента ATOM.

Дефолтные номера

Если для вычисления числа вы используете атрибут value, больше ничего и не нужно. Но если вы опускаете атрибут value, тогда выводимое число будет номером позиции текущего узла в исходном дереве. Например, рассмотрим таблицу стилей на листинге 17-18, она выводит таблицу химических элементов, у которых температура кипения ниже или равна температуре кипения азота.

Листинг 17-18:
Таблица стилей XSLT, которая нумерует атомы

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:template match="PERIODIC_TABLE">
      <html>
        <head><title>The Elements</title></head>
        <body>
          <table>
            <tr>
              <td>Name</td>
              <td>Position</td>
              <td>Default Number</td>
              <td>Boiling Point</td>
            </tr>
            <xsl:apply-templates
              select="ATOM[BOILING_POINT &lt;= 77.344]"/>
          </table>
```

37: http://xml.nsu.ru/extra/samples/17-1full6.xml

```
        </body>
      </html>
    </xsl:template>
    <xsl:template match="ATOM">
      <tr>
        <td><xsl:value-of select="NAME"/></td>
        <td><xsl:number value="position()"/></td>
        <td><xsl:number/></td>
        <td><xsl:number value="BOILING_POINT"/></td>
      </tr>
    </xsl:template>
</xsl:stylesheet>
```

Дефолтное значение, вычисленное элементом xsl:number - это позиция узла среди других соседних узлов того же типа (в данном случае среди других элементов ATOM). Оно отличается от числа, возвращаемого функцией position(), которая подсчитывает номер позиции относительно других узлов в списке контекстного узла (среди других узлов, которым соответствует тот же шаблон - в данном примере hydrogen, helium, nitrogen и neon). Можно изменять то, что вычисляет элемент xsl:number с помощью трех атрибутов:

- level
- count
- from

Атрибут level

По умолчанию, если атрибут value отсутствует, элемент xsl:number выдает номер исходного узла среди соседних узлов того же типа. Например, если элемент xsl:number входит в элементы ATOMIC_NUMBER, а не в элементы ATOM, ни у одного не будет номера выше, чем 1, потому что у всех элементов ATOM не больше одного дочернего элемента ATOMIC_NUMBER. Хотя весь документ содержит несколько элементов ATOMIC_NUMBER, они не являются соседними друг другу.

Если атрибут level элемента xsl:number имеет значение any, нумерация идет на основе общего количества элементов того же типа во всем документе - подсчитываться будут не только те узлы, которые входят в список текущего узла, но и все узлы того же типа. Например, даже если вы отберете только атомные номера газов, будут нумероваться также и элементы ATOMIC_NUMBER, соответствующие жидкостям и твердым химическим элементам, даже если данный шаблон не передает эти элементы на выход. Рассмотрим следующие правила:

```
<xsl:template match="ATOM">
  <tr><xsl:apply-templates select="NAME"/></tr>
</xsl:template>
<xsl:template match="NAME">
  <td><xsl:number level="any"/></td>
  <td><xsl:value-of select="."/></td>
</xsl:template>
```

Поскольку атрибут level имеет значение any, эти шаблоны выдадут результат, в котором нумерация не начинается с 1 для каждого нового элемента NAME:

```
<tr>
<td>1</td><td>Hydrogen</td>
</tr>
```

```
<tr>
<td>2</td><td>Helium</td>
</tr>
```

Если вы уберете атрибут level или установите его значение в дефолтное значение single, тогда результат будет выглядеть так:

```
<tr>
<td>1</td><td>Hydrogen</td>
</tr>
<tr>
<td>1</td><td>Helium</td>
</tr>
```

Несколько менее полезный вариант - установить значение атрибута level элемента xsl:number в multiple - тогда нумерация будет вестись и по соседям текущего узла и по его предкам (но не по его дочерним узлам, которые не являются соседями текущего узла).

Атрибут count

По умолчанию, если атрибут value отсутствует, нумеруются только элементы того же типа, что и текущий узел. Но можно воспользоваться атрибутом count элемента xsl:number и установить его значением выражение, описывающее, что нужно нумеровать. Например, следующее правило соответствует всем дочерним элементам элемента ATOM. Правило помещает перед каждым дочерним элементом номер его позиции среди других дочерних элементов.

```
<xsl:template match="ATOM/*">
  <td><xsl:number count="*"/></td>
  <td><xsl:value-of select="."/></td>
</xsl:template>
```

Результат действия этого правила будет выглядеть так:

```
<td>1</td><td>Hydrogen</td>
<td>2</td><td>H</td>
<td>3</td><td>1</td>
<td>4</td><td>1.00794</td>
<td>5</td><td>20.28</td>
<td>6</td><td>13.81</td>
<td>7</td><td>
  0.0000899
</td>
<td>1</td><td>Helium</td>
<td>2</td><td>He</td>
<td>3</td><td>2</td>
<td>4</td><td>4.0026</td>
<td>5</td><td>4.216</td>
<td>6</td><td>0.95</td>
<td>7</td><td>
  0.0001785
</td>
```

Атрибут from

Атрибут from содержит XPath-выражение, которое определяет, с какого элемента в исходном дереве ведется нумерация. Тем не менее, нумерация все равно начинается с 1, а не с 2 или 10 или еще какого-нибудь числа. Атрибут from только изменяет то, какой элемент считается первым. Этот атрибут имеет значение только тогда, когда level="any". В других случаях он не имеет никакого действия.

Конвертирование числа в строку

До настоящего момента мы неявно предполагали, что числа выглядят как 1, 2, 3, и так далее, то есть, европейская нумерация начинается с 1 и прирастает на 1. Однако, это не единственная возможность. Например, номера страниц в предисловиях и другие начальные страницы книг часто нумеруются римскими цифрами i, ii, iii, iv, и так далее. В различных странах используются различные способы группировки цифр, различные способы отделения целой части от дробной и различные символы для различных цифр. Все эти форматы доступны с помощью использования четырех атрибутов элемента xsl:number:

- format
- letter-value
- grouping-separator
- grouping-size

Атрибут format

Можно выбирать стиль нумерации с помощью атрибута format элемента xsl:number. Этот атрибут имеет обычно одно из четырех значений:

- i: нумерация строчными римскими цифрами: i, ii, iii, iv, v, vi, . . .
- I: нумерация заглавными римскими цифрами: I, II, III, IV, V, VI, . . .
- a: нумерация строчными буквами: a, b, c, d, e, f, . . .
- A: нумерация заглавными буквами: A, B, C, D, E, F, . . .

Например, следующее правило нумерует атомы заглавными римскими цифрами:

```
<xsl:template match="ATOM">
  <P>
    <xsl:number value="position()" format="I"/>
    <xsl:value-of select="."/>
  </P>
</xsl:template>
```

Можно задать нумерацию обычными цифрами, впереди которых идут нули, если включить необходимое количество нулей в атрибут format. Например, установка format="01" выдаст нумерацию 01, 02, 03, 04, 05, 06, 07, 08, 09, 10, 11, 12, ... Это может быть полезно при выравнивании цифр в колонке таблицы.

Атрибут letter-value

Атрибут letter-value различает, какие буквы интерпретируются как числа, а какие - как буквы. Например, если вы хотите использовать format="I" для задания нумерации I, J, K, L,

M, N, ..., а не I, II, III, IV, V, VI, ..., нужно установить атрибут letter-value в alphabetic. Другое ключевое слово, traditional задает числовую последовательность, например:

```
<xsl:template match="ATOM">
  <P>
   <xsl:number value="position()"
               format="I" letter-value="alphabetic"/>
   <xsl:value-of select="."/>
  </P>
</xsl:template>
```

Атрибуты группировки

В США большие числа обычно пишутся разделенными запятыми в группы по три цифры, например 4,567,302,000. Но во многих языках и странах разделяют не запятыми, а пробелами или точками, например, 4.567.302.000 или 4 567 302 000. Более того, в некоторых странах разделяют не по три цифры, а по четыре, например, 4,5673,0000. Если у вас есть какой-либо длинный список, в котором тысяча или более пунктов, приходится задумываться об этих вещах.

Атрибут grouping-separator задает разделитель при группировке цифр. Атрибут grouping-size задает количество цифр в каждой группе:

```
<xsl:number grouping-separator=" " grouping-size="3"/>
```

Сортировка выходных элементов

Элемент xsl:sort сортирует выходные элементы в порядке, отличном от того, в каком они идут в исходном документе. Элемент xsl:sort должен быть дочерним элементом элементов xsl:apply-templates или xsl:for-each. Атрибут select элемента xsl:sort задает ключ, используемый для сортировки элементов, выходящих в результате работы инструкций xsl:apply-templates или xsl:for-each.

По умолчанию сортировка ведется в алфавитном порядке ключей. Если в данном элементе xsl:apply-templates или xsl:for-each присутствует несколько элементов xsl:sort, тогда элементы сортируются сначала по первому ключу, затем по второму ключу, и так далее. Если два элемента при этих обстоятельствах оказываются эквивалентными, они выводятся в том порядке, в каком идут в исходном документе.

Пусть, например, у вас есть файл, в котором элементы ATOM идут в алфавитном порядке. Чтобы отсортировать их по атомным номерам, можно использовать таблицу стилей, приведенную на листинге 17-19:

Листинг 17-19:
Таблица стилей XSLT, которая сортирует элементы по их атомным номерам

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:template match="PERIODIC_TABLE">
      <html>
        <head>
          <title>Atomic Number vs. Atomic Weight</title>
```

```
        </head>
        <body>
          <h1>Atomic Number vs. Atomic Weight</h1>
          <table>
            <th>Element</th>
            <th>Atomic Number</th>
            <th>Atomic Weight</th>
            <xsl:apply-templates>
              <xsl:sort select="ATOMIC_NUMBER"/>
            </xsl:apply-templates>
          </table>
        </body>
      </html>
    </xsl:template>
    <xsl:template match="ATOM">
      <tr>
        <td><xsl:apply-templates select="NAME"/></td>
        <td><xsl:apply-templates select="ATOMIC_NUMBER"/></td>
        <td><xsl:apply-templates select="ATOMIC_WEIGHT"/></td>
      </tr>
    </xsl:template>
</xsl:stylesheet>
```

Сортировка по алфавиту довольно ограничена по возможностям. Например, если отсортировать атомные номера по алфавиту, перывм будет идти водород, его атомный номер 1, но вторым будет идти не гелий с атомным номером 2, а неон с атомным номером 10. Хотя 10 идет по номерам после 9, по алфавиту 10 идет после 1.

Но можно назначить другой порядок сортировки, установив значение необязательного атрибута data-type в number, например:

```
<xsl:sort data-type="number" select="ATOMIC_NUMBER"/>
```

Теперь элементы будут правильно рассортированы по атомным номерам.

Можно изменить порядок сортировки от порядка убывания в порядок возрастания. Для этого нужно установить значение атрибута order в descending, вот так:

```
<xsl:sort order="descending"
          data-type="number"
          select="ATOMIC_NUMBER"/>
```

При этом сортировка будет вестись от химического элемента с максимальным атомным номером до элемента с минимальным атомным номером, так что водород окажется в конце списка.

Сортировка по алфавиту естественно зависит от алфавита. В атрибуте lang можно задать язык ключа сортировки. Значением этого атрибута должен быть код языка по стандарту ISO 639, например, en для английского языка. Но процессоры не обязаны знать, как проводить сортировку на всех языках, какие только могут встретиться в XML-документах. В то время, как сортировка по английскому алфавиту идет просто от первой буквы алфавита к последней, в других языках нужно использовать более сложные алгоритмы. А в некоторых языках есть несколько различных методов сортировки в зависимости от различных критериев. Атрибут lang игнорируется, если атрибут data-type имеет значение number.

Перекрестная ссылка

Здесь в атрибуте xml:lang используются те же самые значения, что и в атрибуте xml:lang, который описывался в главе 11.

И наконец, вы можете установить атрибут case-order в одно из двух значений upper-first илиlower-first, чтобы определить - должны ли строчные буквы при сортировке идти до заглавных или наоборот. Значение по умолчанию зависит от языка сортировки.

Моды

Иногда бывает нужно включить одно и то же содержимое исходного документа в выходной документ несколько раз. Это делается легко и просто многократным применением шаблонов - по одному разу для каждого места, где вы хотите увидеть повторяющиеся данные. Но что если вам нужны в разных местах данные, отформатированные по-разному? Тогда нужно действовать немного хитрее.

Пусть, например, вам нужен результат обработки периодической таблицы в виде серии из 100 ссылок на более детализированные описания отдельных химических элементов. В этом случае выходной документ мог бы начинаться примерно так:

```
<UL>
<LI><A HREF="#Ac">Actinium</A></LI>
<LI><A HREF="#Al">Aluminum</A></LI>
<LI><A HREF="#Am">Americium</A></LI>
<LI><A HREF="#Sb">Antimony</A></LI>
<LI><A HREF="#Ar">Argon</A></LI>
. . .
```

А дальше в этом документе могло бы идти описание какого-то одного химического элемента, отформатированное следующим образом:

```
<H3>
<A NAME="H">Hydrogen</A>
</H3>
<P>
    Hydrogen
    H
    1
    1.00794
    20.28
    13.81
      0.0000899
  </P>
```

Такого рода задачи часто возникают при необходимости автоматической генерации таблиц содержания. Здесь элемент NAME форматируется два раза и по-разному: в списке элементов и в теле документа, описывающем отдельный элемент. Нужно создать два правила, которые будут применяться к одному и тому же элементу ATOM в различных местах документа. Решение проблемы заключается в том, чтобы придать каждому правилу свой атрибут mode. Тогда вы сможете выбрать применяемый шаблон, устанавливая атрибут mode элемента xsl:apply-templates. На листинге 17-20 приводится пример.

Листинг 17-20:
Таблица стилей XSLT, в которой для размещения одних и тех же данных в двух различных местах используются моды

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
```

```
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:template match="/PERIODIC_TABLE">
      <HTML>
        <HEAD><TITLE>The Elements</TITLE></HEAD>
        <BODY>
          <H2>Table of Contents</H2>
          <UL>
            <xsl:apply-templates select="ATOM" mode="toc"/>
          </UL>
          <H2>The Elements</H2>
          <xsl:apply-templates select="ATOM" mode="full"/>
        </BODY>
      </HTML>
    </xsl:template>
    <xsl:template match="ATOM" mode="toc">
      <LI><A>
        <xsl:attribute name="HREF">#<xsl:value-of
          select="SYMBOL"/></xsl:attribute>
        <xsl:value-of select="NAME"/>
      </A></LI>
    </xsl:template>
    <xsl:template match="ATOM" mode="full">
      <H3><A>
        <xsl:attribute name="NAME">
          <xsl:value-of select="SYMBOL"/>
        </xsl:attribute>
        <xsl:value-of select="NAME"/>
      </A></H3>
        <P>
          <xsl:value-of select="."/>
        </P>
    </xsl:template>
  </xsl:stylesheet>
```

Дефолтные правила-шаблоны для узлов сохраняют моды, то есть, для каждой моды n, которую вы объявили в таблице стилей, XSLT-процессор добавляет одно правило-шаблон, которое применяется конкретно к этой моде и выглядит следующим образом:

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

Как обычно, вы можете переиграть дефолтное правило на свой собственный лад.

Задание констант с помощью элемента xsl:variable

Именованные константы помогают сделать код яснее. Вы можете заменить часто используемый кусок текста на ссылку на переменную с простым именем. Также можно изменять многократно появляющийся текст просто изменив определение константы.

Элемент xsl:variable задает именованную строку, которую можно использовать в любом месте таблицы стилей посредством шаблона значения атрибута. У этого элемента имеется единственный атрибут name, которое задает имя переменной, по которому к ней можно будет обра-

щаться. Содержимое элемента xsl:variable задает текстовую строку, которую заменяет переменная. Например, следующий элемент xsl:variable задает переменную с именем copy01, ее значение - Copyright 2001 Elliotte Rusty Harold:

```
<xsl:variable name="copy01">
  Copyright 2001 Elliotte Rusty Harold
</xsl:variable>
```

Чтобы получить доступ к значению этой переменной, нужно перед именем переменной записать в виде префикса знак доллара. Чтобы вставить ее в атрибут, используйте шаблон значения атрибута, например:

```
<BLOCK COPYRIGHT="{$copy01}">
</BLOCK>
```

Чтобы вставить значение переменной в выходной документ в виде текста, используется элемент xsl:value-of:

```
<xsl:value-of select="$copy01"/>
```

Содержимое элемента xsl:variable может включать в себя разметку, содержащую другие XSLT-инструкции. Это означает, что можно вычислять значение переменной на основе другой информации, включая и значения других переменных. Однако, переменная не может рекурсивно ссылаться сама на себя - ни прямо ни косвенно. Например, следующий пример содержит ошибку:

```
<xsl:variable name="GNU">
  <xsl:value-of select="$GNU"/>'s not Unix
</xsl:variable>
```

Подобным образом, переменные не могут ссылаться друг на друга циклически, например, так:

```
<xsl:variable name="Thing1">
  Thing1 loves <xsl:value-of select="$Thing2"/>
</xsl:variable>
<xsl:variable name="Thing2">
  Thing2 loves <xsl:value-of select="$Thing1"/>
</xsl:variable>
```

Элемент xsl:variable должен быть либо элементом верхнего уровня, то есть должен быть непосредственным дочерним элементом корневого элемента xsl:stylesheet, либо он может включаться в правила-шаблоны. Переменная, которая задается на верхнем уровне таблицы стилей, видна по всему документу, это глобальная переменная. В отличие от таких переменных, переменные, которые объявляются внутри правил-шаблонов, доступны только в следующих соседних элементах и их потомках (в области видимости переменной). Это локальные переменные. Локальные переменные имеют преимущество перед глобальными переменными с тем же именем. В случае конфликта между двумя переменными с одним и тем же именем, используется ближайшая локальная переменная.

Именованные шаблоны

Переменные могут содержать только обычный текст или разметку. В XSLT имеется и более мощное макро-средство, которое может размещать стандартную разметку и текст вокруг изменяющихся данных. Пусть, например, вам нужно, чтобы атомные номера, атомные веса и другие ключевые значения форматировались как ячейки таблицы с полужирным синим

шрифтом Times. Другими словами, нужно, чтобы результат выглядел так:

```
<td>
  <font face="Times, serif" color="blue" size="2">
    <b>52</b>
  </font>
</td>
```

Конечно, всего этого можно достигнуть с помощью примерно такого правила-шаблона:

```
<xsl:template match="ATOMIC_NUMBER">
  <td>
    <font face="Times, serif" color="blue" size="2">
      <b>
        <xsl:value-of select="."/>
      </b>
    </font>
  </td>
</xsl:template>
```

Ту же самую разметку можно повторить и в других правилах-шаблонах. Когда детализированная разметка становится более сложной и она повторяется в различных местах таблицы стилей, можно ее превратить в именованный шаблон. Именованные шаблоны напоминают переменные. Однако, они позволяют включать данные из того места, к которому применяется шаблон, а не просто вставлять фиксированный текст.

Элемент xsl:template может иметь атрибут name, по этому имени шаблон может вызываться, даже когда он не применяется прямо. Например, вот пример именованного шаблона для нашего куска разметки:

```
<xsl:template name="ATOM_CELL">
  <td>
    <font face="Times, serif" color="blue" size="2">
      <b>
        <xsl:value-of select="."/>
      </b>
    </font>
  </td>
</xsl:template>
```

Элемент <xsl:value-of select="."/> в середине именованного шаблона будет заменяться содержанием текущего узла из которого был вызван данный шаблон.

Элемент xsl:call-template входит в содержимое правила-шаблона. В нем необходим аргумент name, который определяет имя вызываемого шаблона. При обработке элемент xsl:call-template заменяется содержимым элемента xsl:template с тем же именем. Например, теперь можно переписать правило для элемента ATOMIC_NUMBER следующим образом, здесь элемент xsl:call-template вызывает именованный шаблон ATOM_CELL:

```
<xsl:template match="ATOMIC_NUMBER">
  <xsl:call-template name="ATOM_CELL"/>
</xsl:template>
```

Этот очень простой пример экономит лишь несколько строчек кода, но чем более сложен именованный шаблон, чем чаще он применяется, тем меньше и проще становится таблица стилей. Кроме того, именованные шаблоны, как и переменные, позволяют отредактировав только шаблон изменить работу таблицы стилей во многих местах. Например, если вы решите изменить цвет, которым выписывается атомный номер, атомный вес и другие ключевые значения на красный, вам нужно будет изменить только один шаблон. Вам не нужно будет

изменять каждое отдельное правило-шаблон. Это позволяет создавать более стройные стили.

Передача параметров шаблонам

При каждом вызове шаблона ему можно передавать параметры, влияющие на его результат. Это можно сделать как с именованным шаблоном, так и с безымянным. В элементе xsl:template параметры представляются дочерними элементами xsl:param. В элементах xsl:call-template или xsl:apply-templates параметры представляются дочерними элементами xsl:with-param.

Пусть, например, в каждую строчку таблицы элементов вам нужно добавить ссылку на определенный файл. Нужный результат может выглядеть примерно так:

```
<td>
  <font face="Times, serif" color="blue" size="2">
    <b>
      <a href="atomic_number.html">52</a>
    </b>
  </font>
</td>
```

Фокус в том, что значение атрибута href должно передаваться из той точки, откуда вызывается шаблон, поскольку он различный в каждом конкретном случае вызова шаблона. Например, атомные веса должны быть отформатированы следующим образом:

```
<td>
  <font face="Times, serif" color="blue" size="2">
    <b>
      <a href="atomic_weight.html">4.0026</a>
    </b>
  </font>
</td>
```

Соответствующий шаблон выглядит так:

```
<xsl:template name="ATOM_CELL">
  <xsl:param name="file">index.html</xsl:param>
  <td>
    <font face="Times, serif" color="blue" size="2">
      <b>
        <a href="{$file}"><xsl:value-of select="."/></a>
      </b>
    </font>
  </td>
</xsl:template>
```

Атрибут name элемента xsl:param предоставляет имя параметру (это важно, если их несколько), а содержание элемента xsl:param задает дефолтное значение этого параметра, которое используется, если при вызове шаблона данный параметр не конкретизируется. (Оно может быть задано и с помощью строкового выражения с применением атрибута select.)

При вызове шаблона дочерний элемент xsl:with-param элемента xsl:call-template предоставляет значение параметра, используя свой атрибут name для идентификации параметра, а его значение задается содержанием элемента xsl:with-param, например:

```
<xsl:template match="ATOMIC_NUMBER">
  <xsl:call-template name="ATOM_CELL">
    <xsl:with-param
                name="file">atomic_number.html</xsl:with-param>
  </xsl:call-template>
</xsl:template>
```

И снова, это очень простой пример. Однако, существуют и гораздо более сложные именованные шаблоны. Например, можно задать шаблоны заголовка и футера страниц веб-сайта, предназначенные для импортирования другими таблицами стилей, нужно было бы только менять несколько параметров, например, имя автора, титул страницы и дату публикации.

Удаление и сохранение пробелов

Возможно, вы заметили, что во многих примерах результирующих документов наблюдается несколько странное форматирование. Причина в том, что в исходных документах необходимо было разбивать длинные элементы на несколько строк, чтобы они умещались между полями данной книги. К сожалению, дополнительные пробелы и символы форматирования (переводы строки, табуляция и так далее) переносятся из исходного документа в выходной в неизменном виде. Для компьютера лишние пробелы и символы форматирования не имеют значения, но для человека они важны.

Дефолтное поведение при чтении текстовых узлов исходного документа, например, содержимого элементов ATOMIC_NUMBER или DENSITY - сохранение всех пробелов и символов форматирования. Типичный элемент DENSITY выглядит так:

```
<DENSITY UNITS="grams/cubic centimeter">
  <!-- At 300K, 1 atm -->
  0.0000899
</DENSITY>
```

Когда берется значение этого элемента, сохраняются все лишние пробелы до и после текста, несмотря на то, что они нужны только для того, чтобы текст влез на печатную страницу:

```
  0.0000899
```

Для удаления лишних пробелов из этой и других строк можно использовать функцию normalize-space(). Например, вместо <xsl:value-of select="DENSITY"/> следует написать <xsl:value-of select="normalize-space(DENSITY)"/>.

Можно автоматически удалять из исходного документа текстовые узлы, которые содержат только пробелы, используя элемент xsl:strip-space. Атрибут elements этого элемента верхнего уровня содержит список элементов, текстовые узлы которых, если они содержат только пробелы, будут удаляться. Например, следующая инструкция указывает, что из элементов DENSITY, NAME, SYMBOL и BOILING_POINT следует удалить все текстовые узлы, если они содержат только пробелы:

```
<xsl:strip-space elements="DENSITY NAME SYMBOL BOILING_POINT"/>
```

Используя звездочку *, можно удалить узлы, содержащие только пробелы с помощью следующей инструкции:

```
<xsl:strip-space elements="*"/>
```

Существует также и элемент xsl:preserve-space. Он имеет аналогичный синтаксис, но противоположное значение. Но, поскольку сохранение пробелов и символов форматирования - это

дефолтное поведение, этот элемент используется редко. Его главное назначение - переиграть элемент xsl:strip-space, импортированный из другой таблицы стилей или задать несколько элементов, в которых пробелы сохраняются, если дефолтное поведение было сброшено инструкцией <xsl:strip-space elements="*"/>.

Иное дело текстовые узлы, содержащие только пробелы в выходном документе. Они по умолчанию удаляются. Если вы хотите какой-то из них сохранить, нужно добавить атрибут xml:space со значением preserve его родительскому узлу или любому другому его узлу-предку.

Перекрестная ссылка

Атрибут xml:space обсуждался в главе 11.

Иногда самым простым способом включить в таблицу стилей важный пробел или символ форматирования - заключить его в элемент xsl:text. Пробелы внутри элемента xsl:text трактуются буквально и никогда не удаляются.

Организация ветвления

В XSLT имеется два элемента, которые позволяют изменять результат в зависимости от исходных данных. Элемент xsl:if позволяет выводить или не выводить определенный фрагмент XML в зависимости от того, какие паттерны представлены в исходном документе. Элемент xsl:choose берет один из нескольких возможных XML-фрагментов в зависимости от представленных в исходном документе паттернов. Большую часть того, что можно сделать с помощью элементов xsl:if и xsl:choose, можно сделать и правильным использованием шаблонов. Однако, иногда использовать элементы xsl:if или xsl:choose проще.

Элемент xsl:if

Элемент xsl:if предоставляет простой механизм для изменения выхода на основе паттерна. Атрибут test этого элемента содержит выражение, которое оценивается как булевское значение. Если выражение имеет значение true, выводится содержимое элемента xsl:if. Например, следующий шаблон выводит имена всех элементов ATOM. За каждым именем идет запятая, кроме последнего в списке:

```
<xsl:template match="ATOM">
  <xsl:value-of select="NAME"/>
  <xsl:if test="position()!=last()">, </xsl:if>
</xsl:template>
```

Это приведет к результату "Hydrogen, Helium", а не "Hydrogen, Helium, ".

Не существует элементов xsl:else или xsl:else-if. Этот механизм предоставляется элементом xsl:choose.

Элемент xsl:choose

Элемент xsl:choose выбирает один из нескольких возможных выходов в зависимости от опре-

деленных условий. Каждое условие и связанный с ним шаблон выхода задается дочерним элементом xsl:when. Атрибут test элементов xsl:when содержит XPath-выражение, имеющее булевское значение. Если выполняется много условий сразу, активируется только первый вариант, для которого условие выполняется. Если не выполняется ни одно условие, активируется шаблон в дочернем элементе xsl:otherwise. Например, следующее правило изменяет цвет результата в зависимости от того, какое значение имеет атрибут STATE элемента ATOM - SOLID, LIQUID или GAS:

```
<xsl:template match="ATOM">
  <xsl:choose>
    <xsl:when test="@STATE='SOLID'">
      <P style="color: black">
        <xsl:value-of select="."/>
      </P>
    </xsl:when>
    <xsl:when test="@STATE='LIQUID'">
      <P style="color: blue">
        <xsl:value-of select="."/>
      </P>
    </xsl:when>
    <xsl:when test="@STATE='GAS'">
      <P style="color: red">
        <xsl:value-of select="."/>
      </P>
    </xsl:when>
    <xsl:otherwise>
      <P style="color: green">
        <xsl:value-of select="."/>
      </P>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>
```

Смешивание нескольких таблиц стилей

В XML-документе может использоваться много различных словарей разметки, заданных многими различными определениями DTD. Для различных словарей обычно применяются различные таблицы стилей. Однако, для отдельных документов могут понадобиться свои собственные стилевые правила. Элементы xsl:import и xsl:include позволяют смешивать различные таблицы стилей, так что можно организовывать и совместно использовать таблицы стилей для различных словарей и задач.

Импорт с помощью элемента xsl:import

Элемент xsl:import - это элемент верхнего уровня, его атрибут href задает URI импортируемой таблицы стилей. Элементы xsl:import должны идти до любых других элементов верхнего уровня в корневом элементе xsl:stylesheet. Например, здесь элементы xsl:import импортируют таблицы стилей genealogy.xsl и standards.xsl:

```
<xsl:stylesheet version="1.0"
```

```
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:import href="genealogy.xsl"/>
    <xsl:import href="standards.xsl"/>
    <!-- other child elements follow -->
</xsl:stylesheet>
```

Правила в основной таблице стилей могут вступать в конфликт с правилами в импортируемой таблице стилей. Если случается конфликт, превосходство имеет импортированное правило. Если конфликтуют правила в двух импортированных таблицах стилей, тогда превосходство имеет правило из последней импортированной таблицы стилей (в данном случае standards.xsl).

Элемент xsl:apply-imports лишь немного отличается от элемента xsl:apply-templates - он применяет импортированные правила. Он не применяет никаких правил из основной таблицы стилей. Это позволяет получать доступ к импортированным правилам, которые иначе были бы переиграны правилами основной таблицы. Кроме похожего названия, элемент xsl:apply-imports имеет схожий с элементом xsl:apply-templates синтаксис. Единственная существенная разница заключается в том, что элемент xsl:apply-imports соответствует правилам импортированной таблицы стилей.

Включение с помощью элемента xsl:include

Элемент xsl:include - это элемент верхнего уровня, который копирует в текущую таблицу стилей другую таблицу стилей на то место, где он сам находится. (Точнее говоря, он копирует в текущий документ содержание элементов xsl-stylesheet или xsl:transform внешней таблицы стилей.) Его атрибут href задает URI включаемой таблицы стилей. Элемент xsl:include может располагаться на любом месте верхнего уровня после последнего имеющегося в таблице стилей элемента xsl:import.

В отличие от правил, импортируемых элементом xsl:import, правила, включаемые элементом xsl:include не имеют превосходства над правилами основной таблицы стилей. С точки зрения XSLT-процессора между правилами основной таблицы стилей и включенными правилами нет никакой разницы.

Встраивание с помощью элемента xsl:stylesheet

Можно прямо включить таблицу стилей XSLT в XML-документ, к которому она применяется. Я не рекомендую такой подход для использования на практике, а браузеры и XSLT-процессоры не обязаны его поддерживать. Тем не менее, иногда им пользуются. Для этого элемент xsl:stylesheet должен идти в качестве дочернего элемента корневого элемента XML-документа, а не являться сам корневым элементом. Он должен иметь атрибут id, задающий ему уникальное имя, этот id должен являться значением атрибута href в процессуальной инструкции xml-stylesheet, перед идентификатором нужно поставить символ #. Пример приведен на листинге 17-21.

Listing 17-21: An XSLT style sheet embedded in an XML document

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xml" href="#id(mystyle)"?>
<PERIODIC_TABLE>
  <xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
```

```
    id="mystyle">
    <xsl:template match="/">
      <html>
        <xsl:apply-templates/>
      </html>
    </xsl:template>
    <xsl:template match="PERIODIC_TABLE">
      <xsl:apply-templates/>
    </xsl:template>
    <xsl:template match="ATOM">
      <P>
        <xsl:value-of select="."/>
      </P>
    </xsl:template>
    <!--Не показывать саму таблицу стилей-->
    <xsl:template match="xsl:stylesheet"/>
  </xsl:stylesheet>
  <ATOM>
    <NAME>Actinium</NAME>
    <ATOMIC_WEIGHT>227</ATOMIC_WEIGHT>
    <ATOMIC_NUMBER>89</ATOMIC_NUMBER>
    <OXIDATION_STATES>3</OXIDATION_STATES>
    <BOILING_POINT UNITS="Kelvin">3470</BOILING_POINT>
    <MELTING_POINT UNITS="Kelvin">1324</MELTING_POINT>
    <SYMBOL>Ac</SYMBOL>
    <DENSITY UNITS="grams/cubic centimeter"><!-- At 300K -->
      10.07
    </DENSITY>
    <ELECTRONEGATIVITY>1.1</ELECTRONEGATIVITY>
    <ATOMIC_RADIUS UNITS="Angstroms">1.88</ATOMIC_RADIUS>
  </ATOM>
</PERIODIC_TABLE>
```

Методы выведения и заключение

В большей части примеров этой главы XML преобразовывался в правильный HTML. Но большинство XSLT-процессоров поддерживают три метода вывода:

- XML
- HTML
- text

XSLT-процессор ведет себя по-разному в зависимости от того, какой метод вывода он использует. Формат XML используется по умолчанию и во многих отношениях является самым простым методом - после обработки получается именно то, что вы запрашиваете в таблице стилей. Поскольку в правильном XML не разрешается прямо использовать знак меньше и амперсант, если использовать ссылку на символ < или ссылку на сущность <, чтобы вставить символ <, преобразователь выдаст < или, возможно, <. Если вы используете ссылку на символ & или ссылку на сущность &, чтобы вставить символ &, преобразователь выдаст & или &. Тем не менее, как вы увидите, есть несколько способов отключить их преобразование в ссылки на символы и сущности.

Метод выведения HTML предназначен для выведения стандартного HTML 4.0. Это не правильный HTML, который использовался в примерах этой главы, скорее обычный HTML, в котором пустые тэги выглядят как <HR> и <IMG>, а не <HR/> и <IMG/>, процессуальные инструкции завершаются конструкцией >, а не ?>, а символы <, которые используются в JavaScript, не преобразуются в <. Это облегчает вывод HTML, который может работать во многих браузерах и платформах, и не приводить к странным эффектам вроде двойных линеек-разделителей там, где должна быть одна и прочих эффектов, вызванных навязыванием HTML-коду XML-синтаксиса. Метод вывода HTML автоматически выбирается, если преобразователь замечает, что корневым элементов выходного документа является элемент html, HTML, HtMl или любая другая подобная комбинация букв, означающих Hypertext Markup Language.

Последним методом вывода является простой текст. Метод вывода text сначала формирует полное выходное дерево, как и при методе вывода XML, но затем возвращается только строчное значение этого дерева. Этот механизм полезен при преобразовании в такие не XML-форматы, как RTF или TeX. Главное преимущество текстового выходного формата состоит в том, что знак меньше не преобразуется в < или <, а амперсанты не конвертируются в & или &. Это позволяет эффективно выводить произвольный текст.

Элемент xsl:output

По умолчанию XSLT-процессор использует метод выведения XML (если он не обнаружит, что корневым элементом выходного документа является элемент HTML, в этом случае будет использоваться метод HTML.) Метод вывода можно изменить с помощью элемента верхнего уровня xsl:output. Атрибут method этого элемента xsl:output задает метод вывода и обычно имеет одно из трех значений:

- xml
- html
- text

Преобразователи могут поддерживать и другие значения, но пока поддерживаются только эти три. Например, чтобы на выходе получать чистый правильный HTML, в котором пустые тэги закрыты, знак меньше заменен ссылкой на сущность, и так далее, на верхнем уровне таблицы стилей нужно использовать следующий элемент xsl:output:

```
<xsl:output method="xml"/>
```

Чтобы получить на выходе обычный HTML, даже если вы не используете корневой элемент html, вам нужно поместить на верхний уровень таблицы стилей другой элемент xsl:output:

```
<xsl:output method="html"/>
```

Элемент xsl:output имеет и несколько других допустимых атрибутов, которые детализируют, как именно выводится XML. Они позволяют изменять вступительную часть документа, как код форматируется незначащими пробелами, и какие элементы используют секции CDATA, а не заменяют символы < и & соответствующими ссылками на сущности.

XML-декларация

Четыре атрибута элемента xsl:output форматируют XML-декларацию, которая будет использоваться в выходном документе. При этом предполагается, что методом вывода является xml.

Вот эти атрибуты:

- omit-xml-declaration
- version
- encoding
- standalone

Атрибут omit-xml-declaration может принимать значение yes или no. Если его значение yes, тогда в выходной документ XML-декларация не включается. Если его значение no, тогда включается. Чтобы, например, вставить в выходной документ самую простую XML-декларацию <?xml version="1.0"?>, на верхнем уровне таблицы стилей следует использовать такой элемент xsl:output:

```
<xsl:output method="xml" omit-xml-declaration="no"/>
```

Можно вставить и два отдельных элемента xsl:output:

```
<xsl:output method="xml"/>
<xsl:output omit-xml-declaration="no"/>
```

Дефолтное значение атрибута version XML-декларации - 1.0. В настоящее время это единственное разрешенное значение. Если в будущем ситуация изменится, тогда атрибут version элемента xsl:output позволит изменить версию в XML-декларации, например:

```
<xsl:output version="1.1"/>
```

Можно установить атрибут standalone XML-декларации в значение yes или no, используя атрибут standalone элемента xsl:output. Например, следующий элемент xsl:output вставит XML-декларацию <?xml version="1.0" standalone="yes"?>:

```
<xsl:output method="xml"
            omit-xml-declaration="no" standalone="yes"/>
```

Последняя допустимая часть XML-декларации - это декларация encoding. Как можно догадаться, она выводится с помощью атрибута encoding элемента xsl:output. Значением этого атрибута может быть любое допустимое имя кодировки, зарегистрированное через Internet Assigned Numbers Authority, как описывалось в главе 7. Например, чтобы вставить XML-декларацию <?xml version="1.0" encoding="ISO-8859-1"?>, нужно использовать следующий элемент xsl:output:

```
<xsl:output method="xml"
          omit-xml-declaration="no" encoding="ISO-8859-1"/>
```

Такая декларация изменит и кодировку, которую использует для выходного документа XSLT-процессор (по умолчанию используется UTF-8). Однако, не все процессоры поддерживают все возможные кодировки. Те, что написаны на Java, скорее всего, будут поддерживать больше кодировок, поскольку в Java имеется богатая библиотека классов, позволяющих почти тривиальными способами поддерживать несколько десятков наиболее популярных кодировок.

Определение типов документа DTD

В XSLT нет элементов, предназначенных для построения внутреннего подмножества DTD выходного документа с применением деклараций <!ELEMENT>, <!ATTLIST>, <!ENTITY> и <!NOTATION>. Однако, имеется два атрибута элемента xsl:output, которые можно применять для включения декларации DOCTYPE, указывающей на внешнее DTD. Это атрибуты doctype-system и doctype-public. Первый вставляет относящийся к DTD идентификатор SYSTEM, вто-

рой - идентификатор PUBLIC. Например, пусть в выходном документе необходима следующая декларация DOCTYPE:

```
<!DOCTYPE PERIODIC_TABLE SYSTEM "chemistry.dtd">
```

Для ее выведения на верхнем уровне таблицы стилей нужно использовать следующий элемент xsl:output:

```
<xsl:output doctype-system="chemistry.dtd"/>
```

Выделяя корневой элемент выходного дерева, XSLT-процессор определяет нужный корневой элемент, к которому относится декларация типов документа. Использовать полный URL вместо относительного URL тоже несложно:

```
<xsl:output
  doctype-system="http://www.mysite.com/chemistry.dtd"/>
```

С другой стороны, допустим, вам нужна следующая декларация DOCTYPE в выходном документе:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN"
         "http://www.w3.org/TR/REC-html40/loose.dtd">
```

Тогда нужно использовать оба атрибута doctype-system и doctype-public так, чтобы у декларации DOCTYPE имелись оба идентификатора - PUBLIC и SYSTEM. Например,

```
<xsl:output
  doctype-system="http://www.w3.org/TR/REC-html40/loose.dtd"
  doctype-public="-//W3C//DTD HTML 4.0 Transitional//EN"/>
```

Форматирование результирующего кода

Форматирование листингов в большей части примеров этой главы более, чем произвольно. Оно гораздо менее четкое, чем у вручную кодированных исходных документов. Однако, если пробелы и символы форматирования не имеют значения в выходном документе, вы можете дать указание преобразователю выводить XML "красиво", отмечая вложенность элементов различными отступами. Это делается с помощью атрибута indent элемента xsl:output. Если этот атрибут имеет значение yes (по умолчанию его значение no), тогда преобразователь позволяет (но не требует) вставлять (но не удалять) в выходном документе лишние пробелы и символы форматирования, для того, чтобы выходной документ выглядел "красиво". Это делается с помощью отступов и разбивок строк, например:

```
<xsl:output indent="yes"/>
```

Но при этом нельзя конкретизировать, на сколько увеличивается отступ каждого вложенного элемента (например, на два пробела или на один шаг табуляции). Вот все, что должен сделать преобразователь. В совокупности атрибуты xsl:strip-space и indent элемента xsl:output позволяют генерировать выходные документы почти настолько же аккуратные, что и вручную тщательно написанные XML-документы.

Секции CDATA

Стандарт XSLT не позволяет вставлять секции CDATA в произвольные места XML-документов, генерируемых XSL-перобразованиями. Однако, можно задать текстовое содержимое каждого конкретного элемента как размещенного в секции CDATA. В этом случае символы < и & не перекодируются в < и &, что обычно происходит. Для этого нужно поместить имя эле-

мента, чье текстовое содержимое должно быть заключено в рамки секции CDATA, в атрибут cdata-section-elements элемента xsl:output. Например, следующий элемент xsl:output сообщает, что содержимое элемента SCRIPT должно быть помещено в секцию CDATA:

```
<xsl:output cdata-section-elements="SCRIPT"/>
```

В атрибуте cdata-section-elements можно указать много элементов, чье текстовое содержимое должно быть размещено в секциях CDATA, для этого нужно их имена перечислить в атрибуте, разделяя пробелами. Например, следующий элемент xsl:output сообщает, что содержимое элементов SCRIPT и CODE должно быть заключено в секции CDATA:

```
<xsl:output cdata-section-elements="SCRIPT CODE"/>
```

Можно использовать для перечисления нескольких элементов и несколько отдельных элементов xsl:output, например:

```
<xsl:output cdata-section-elements="SCRIPT"/>
<xsl:output cdata-section-elements="CODE"/>
```

Медиа-тип

Последний атрибут элемента xsl:output определяет MIME медиа-тип выходного документа. Это атрибут media-type. Обычно он имеет значение text/xml, но может иметь и значение text/html при методе вывода HTML, значение text/plain для метода вывода text, или даже какое-то другое, например, text/rtf. Не следует задавать параметр кодировки charset для медиа-типа. Преобразователь должен определить кодировку из атрибута encoding элемента xsl:output. Например, следующий элемент xsl:output определяет, что выходная кодировка использует MIME-тип text/rtf:

```
<xsl:output media-type="text/rtf"/>
```

В зависимости от внешнего контекста, этот атрибут может определять расширение имени файла, значок файла, механизм обработки этого файла HTTP-сервером или что-то еще. И вновь, он может и не иметь никакого эффекта. XSLT-процессор может игнорировать это указание и выдавать один и тот же поток байтов или XML-дерево независимо от указанного медиа-типа - он важен для окружения, в котором существует XML-документ, но не для самого XML-документа.

Заключение

В этой главе вы познакомились с XSL-преобразованиями. В частности, вы узнали, что:

- Расширяемый язык таблиц стилей XSL (Extensible Stylesheet Language) объединяет два отдельных XML-приложения для преобразования и форматирования XML-документов.
- XSL-преобразование применяет правила к дереву, считанному из исходного XML-документа, и преобразует его в другое дерево, тоже оформленное как XML-документ.
- Правило-шаблон XSL - это элемент xsl:template, у которого есть атрибут match. Узлы исходного дерева сравниваются с паттернами атрибутов match различных шаблонов. Когда обнаруживается соответствие, на выход передается содержимое шаблона.
- Значением узла является чистая текстовая строка (не разметка), в которой заключено содержимое узла. Значение узла можно получить с помощью элемента xsl:value-of.
- Множественные элементы можно обрабатывать двумя способами: с помощью элемента xsl:apply-templates и с помощью элемента xsl:for each.

- Значение атрибута match элемента xsl:template - это паттерн соответствия, определяющий, какие узлы соответствуют данному шаблону.
- XPath-выражения (или просто выражения) - это расширение паттернов соответствия, они применяются в атрибутах select элементов xsl:apply-templates, xsl:value-of, xsl:for-each, xsl:copy-of, xsl:variable, xsl:param, xsl:with-param и xsl:sort.
- Дефолтные правила применяют шаблоны к элементным узлам и берут значение текстовых узлов и атрибутов.
- Элементы xsl:element, xsl:attribute, xsl:processing-instruction, xsl:comment и xsl:text выводят элементы, атрибуты, процессуальные инструкции, комментарии и текст, вычисляемые на основе данных в исходном документе.
- Элемент xsl:attribute-set задает общую группу атрибутов, которая может применяться ко многим элементам в различных шаблонах с помощью xsl:use-attribute-sets.
- Элемент xsl:copy копирует текущий узел из исходного документа на выход.
- Элемент xsl:number вставляет в выходной документ число, определяемое его атрибутом value используя заданный числовой формат, который определяется атрибутом format.
- Элемент xsl:sort может переупорядочить узлы исходного документ прежде, чем они копируются в выходной документ.
- Моды помогают применять различные шаблоны к одному и тому же элементу из различных мест таблицы стилей.
- Элемент xsl:variable определяет именованные константы, которые могут упростить код.
- Именованные шаблоны позволяют многократно применять одни и те же куски кода в различных местах таблицы стилей.
- Пробелы и символы форматирования сохраняются по умолчанию до тех пор, пока элемент xsl:strip-space или атрибут xml:space не изменят ситуацию.
- Элемент xsl:if задействует определенный код только в том случае, если выполняется условие в его атрибуте test.
- Элемент xsl:choose задействует шаблон своего первого дочернего элемента xsl:when у которого выполняется условие в атрибуте test, или шаблон внутри своего дочернего элемента xsl:otherwise, если не выполняется условие ни для одного элемента xsl:when.
- Элементы xsl:import и xsl:include позволяют смешивать правила из различных таблиц стилей.
- Элемент xsl:stylesheet позволяет включить таблицу стилей прямо в документ, к которому она применяется.
- Различные атрибуты элемента xsl:output позволяют задавать формат выходного документа, его XML-декларацию, определение типов документа, форматирование кода, кодировку и MIME-тип.

В следующей главе речь пойдет о второй части XSL: о словаре форматирующих объектов. Форматирующие объекты - чрезвычайно мощное средство точного задания чертежа, по которому формируются страницы. XSL-преобразования применяются для трансформации XML-документа в документ форматирующих объектов XSL.