Министерство образования и науки Российской Федерации
Федеральное государственное бюджетное образовательное
учреждение высшего профессионального образования
ПЕТРОЗАВОДСКИЙ ГОСУДАРСТВЕННЫЙ
УНИВЕРСИТЕТ

К. А. Кулаков, В. М. Димитров

ТЕХНОЛОГИИ XML

ЧАСТЬ II. Преобразование данных

Учебное пособие для студентов математического факультета

Петрозаводск
Издательство ПетрГУ
2015

ББК 32.973.2
УДК 004
К90

Печатается по решению редакционно-издательского совета Петрозаводского государственного университета

Издается в рамках реализации комплекса мероприятий Программы стратегического развития ПетрГУ на 2012–2016 гг.

Р е ц е н з е н т ы:
канд. тех. наук. Р. В. Воронов; канд. физ.-мат. наук. А. В. Бородина

Кулаков, К. А.

К90 Технологии XML : в 2 ч. : учебное пособие для студентов математического факультета / К. А. Кулаков, В. М. Димитров; М-во образования Рос. Федерации, Федер. гос. бюджет. образоват. учреждение высш. проф. образования Петрозавод. гос. ун-т. — Петрозаводск : Издательство ПетрГУ, 2015.

ISBN 978-5-8021-2139-9
Ч. 2. Преобразование данных. — 2015. — 72 с.
ISBN 978-5-8021-2138-2

В учебном пособии содержатся теоретические и практические сведения по преобразованию данных для языка XML и освещаются следующие темы: расширяемый язык таблиц стилей для трансформации (Extensible Stylesheet Language Transformation, XSLT), таблицы стилей (Cascading Stylesheets, CSS), форматирующие объекты (XSL Formatting Objects, XSL-FO). Представлены варианты практических заданий по рассматриваемым темам.

Пособие предназначено для студентов математического факультета направлений подготовки «Прикладная математика и информатика» и «Информационные системы и технологии».

ББК 32.973.2
УДК 004

ISBN 978-5-8021-2138-2 (Ч. 2.)
ISBN 978-5-8021-2139-9

Содержание

Введение

Первая часть учебного-методического пособия посвящена обзору языка разметки XML, правилам построения XML документов, наиболее распространенным моделям документов DTD и XSD, а также языкам запросов (XPath, XPointer и XQuery).

Важной частью практического использования XML [1] документов является их преобразование. Преобразования XML документа используются для представления данных с помощью другого языка (например, HTML), получения другой иерархии данных из исходной (например, для использования в другом приложении), получения заданного набора данных (например, определенной выборки в случае задания некоторого фильтра) и т.д.

Однако, посимвольный анализ или написание трансляторов для таких преобразований является достаточно трудоемкой и трудозатратной задачей, поэтому появилось семейство рекомендаций Extensible Stylesheet Language (XSL, расширяемый язык таблиц стилей), направленное на создание средств и инструментов для облегчения решения задачи преобразований XML документов.

Вторая часть учебного-методического пособия рассказывает о технологиях преобразования XML документов. В частности рассматриваются следующие темы: расширяемый язык таблиц стилей для трансформации (Extensible Stylesheet Language Transformation, XSLT), таблицы стилей (Cascading Stylesheets, CSS), форматирующие объекты (XSL Formatting Objects, XSL-FO).

Материалы пособия используются в Петрозаводском государственном университете на математическом факультете в следующих курсах:

- “Передовые Web-технологии” для бакалавров на 4 курсе по направлению “Прикладная математика и информатика” (010400);
- “Web-технологии” для бакалавров на 3 курсе по направлению “Информационные системы и технологии” (230400).

§ 1. Расширяемый язык таблиц стилей для трансформации (XSLT)

Программа на языке XSLT (Extensible Stylesheet Language Transformation) [2] представляет собой XML документ, определяющий набор правил для преобразования XML документа в другое его представление. Эти правила, также называемые шаблонами, сравнивают элементы дерева входного XML документа и формируют выходное дерево (выходной документ).

Общая схема применения XSLT представлена на рисунке 1. На вход XSLT процессору (специальной программной системе, реализующей обработку преобразований) подается описание шаблонов на языке XSLT и входной XML документ. После работы процессора, заключащееся в применении шаблонов для найденных элементов входного XML документа, формируется новый выходной документ.

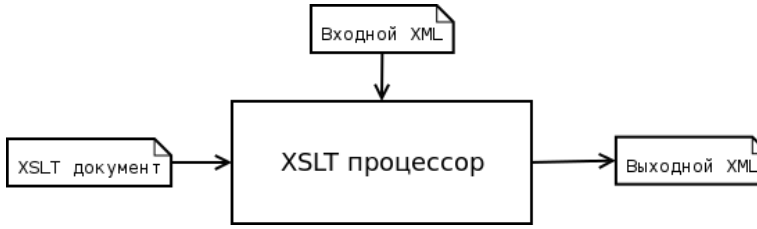

Рис. 1: Схема применения XSLT

Для нахождения узлов из входного XML документа язык XSLT использует язык XPath [3]. Также язык XSLT поддерживает схемы (в частности DTD [4]) для проверки корректности самого XSLT кода и пространство имен для возможности определения или заимствования правил в других XSLT документа.

1.1. Структура документа на языке XSLT

Так как документ на языке XSLT является XML документом, то его структура жестко следует всем правилам и ограничениям структуры XML документов. Однако, имеются некоторые особенности, характерные для XSLT документов.

Во-первых, корневым элементом для XSLT документа должен быть элемент `stylesheet` или `transform`. Можно использовать любой из них, так как они являются синонимами для друг друга. Обязательным атрибутом для этих элементов является атрибут `version`, значение которого определяет версию спецификации XSLT преобразований. Минимальная версия, которая на данный момент используется, имеет значение "1.0", включающая базовые и основные элементы XSLT. Отметим, что существует версия "1.1" с некоторыми расширениями. Помимо атрибута `version` корневой элемент может включать следующие необязательные атрибуты: `extension-element-prefixes` (разделенный пробелами набор пространств имен, из которых были использованы расширения), `exclude-result-prefixes` (разделенный пробелами набор пространств имен, который не должен быть включен в результирующий XML документ), `id` (уникальный идентификатор для документа XSLT).

Во-вторых, необходимо использовать пространство имен `http://www.w3.org/1999/XSL/Transform`, в котором даны определения элементов и атрибутов XSLT. Для этого пространства имен рекомендуется использовать префикс `xsl`. В листинге 1.1 приведен пример пустого XSLT документа без шаблонов. Результатом применения такого документа будет текст XML документа без разметки.

Листинг 1.1: Пустой XSLT документ

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

Также бывает полезным возможность подключения XSLT преобразований к XML документу. Это может пригодиться для приложений, которые по заданным XSLT преобразованиям построят новый документ, который в свою очередь будут дальше использовать. Одним из примеров такого приложения является веб-обозреватель, для которого обычно XML документ преобразуется в документ с гипертекстовой разметкой (HTML).

Пример подключения XSLT преобразований к XML документу представлен в листинге 1.2. Для этого используется специальная директива `stylesheet`, которая задает документ преобразований. Атри-

бут `type` задает MIME тип, в котором представлены преобразования (в примере это XML). В атрибуте `href` указывается ссылка на этот документ, которая может быть как локальной, так и глобальной.

Листинг 1.2: Подключение XSLT к XML

```
<?xml version="1.0"?>
<?xml stylesheet type="text/xml" href="http://cs.petrsu.ru/
    people.xsl"?>
<people>
</people>
```

Для дальнейших примеров с XSLT шаблонами будет использоваться XML документ приведенный в листинге 1.3.

Листинг 1.3: Входной XML документ

```
<?xml version="1.0" encoding="UTF-8"?>
<people>
    <person born="1903" died="1957">
        <name>Джон фон Нейман</name>
    </person>
    <person born="1941" died="2011">
        <complex_name>
            <first_name>Деннис</first_name>
            <second_name>Ритчи</second_name>
        </complex_name>
    </person>
    <person born="1943">
        <complex_name>
            <first_name>Кен</first_name>
            <second_name>Томпсон</second_name>
        </complex_name>
    </person>
</people>
```

1.2. Шаблоны

Любой шаблон описывается с помощью элемента `template`. Важным атрибутом данного элемента является атрибут `match`, значением которого является XPath выражение, находящее в исходном документе необходимые для преобразований узлы.

В листинге 1.4 приведен шаблон, который должен заменить (преобразовать) все элементы типа `person` на фразу “Человек”. Таким об-

разом выходный документ для листинга 1.3 будет выглядеть так, как это представлено в листинге 1.5.

Листинг 1.4: Преобразования для элементов person

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

Листинг 1.5: Выходной документ для листинга 1.4

```
Человек
Человек
Человек
```

1.3. Элементы XSLT

Все элементы XSLT можно поделить на пять групп: два корневых элемента (`stylesheet` или `transform`, о них было упомянуто выше), тринадцать элементов верхнего уровня, двадцать элементов инструкций, элементы расширения конкретных реализаций XSLT процессоров и расширения элементов, реализованные пользователем. Рассмотрим некоторые элементы XSLT более подробнее.

Элемент xsl:value-of. Элемент `xsl:value-of` позволяет выбрать содержимое из входных элементов и подставить в шаблон. Результатом работы данного элемента является строковое значение. Набор входных элеметов задается с помощью выражения XPath в атрибуте `select`. В листинге 1.6 приведен пример, где для каждого элемента `person` будет производиться поиск по пути `name` и найденное значение будет подставлено в результат. Вывод для XSLT документа из листинга 1.6 и входного XML документа из листинга 1.3 представлен в листинге 1.7.

Листинг 1.6: Использование элемента value-of

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

Листинг 1.7: Выходной документ для листинга 1.6

```
<?xml version="1.0" encoding="utf-8"?>
<p>Джон фон Нейман</p>
```

Элемент xsl:apply-templates. Обработчики XSLT активизируют родительский шаблон раньше шаблона потомка, игнорируя последних в том случае, если не предупредить их об этом специально. Предположим, что для предыдущего примера необходимо добавить вывод заголовка, который будет определен в шаблоне для элемента `people`.

Листинг 1.8: Игнорирование шаблона

```
<xsl:template match="people">
    <h1>Известные люди</h1>
</xsl:template>
<xsl:template match="person">
    <p> <xsl:value-of select="name"/> </p>
</xsl:template>
```

Применяя XSLT преобразования из листинга 1.8, будет получена лишь строка

```
<h1>Известные люди</h1>.
```

Чтобы заставить обработчика XSLT вывести всю информацию, необходимо воспользоваться элементом `xsl:apply-templates`, который позволяет применять шаблоны для дочерних элементов. Таким образом нужно код из листинга 1.8 переписать так, как это представлено в листинге 1.9.

Листинг 1.9: Использование элемента xsl:apply-templates

```
<xsl:template match="people">
    <h1>Известные люди</h1>
    <xsl:apply-templates />
</xsl:template>
<xsl:template match="person">
    <p> <xsl:value-of select="name"/> </p>
</xsl:template>
```

Элемент `xsl:apply-templates` позволяет задавать узлы, для которых далее необходимо применить шаблон (сделать это можно с помощью выражения XPath в атрибуте `select`). Таким образом очевидно, что XSLT позволяет менять порядок обхода элементов во входном документе. Например, преобразования из листинга 1.10 меняют местами вывод значений элементов `first_name` и `second_name`. Результат этих преобразований представлен в листинге 1.11.

Листинг 1.10: Изменение порядка обработки элементов

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org
    /1999/XSL/Transform">
    <xsl:template match="complex_name">
  <p><xsl:value-of select="second_name"/>, <xsl:value-of
    select="first_name"/></p>
    </xsl:template>
    <xsl:template match="person">
  <xsl:apply-templates select="complex_name"/>
    </xsl:template>

    <xsl:template match="people">
  <h2>Peoples</h2>
  <xsl:apply-templates/>
    </xsl:template>

</xsl:stylesheet>
```

Листинг 1.11: Выходной документ для листинга 1.10

```
<h2>Peoples</h2>
<p>Ритчи, Деннис</p>
<p>Томпсон, Кен</p>
```

Стоит отметить, важность порядка элементов во входном XML документе: XML документ последовательно обрабатывается сверху-вниз если не задано других условий.

Элементы вызова шаблона. XSLT позволяет явно вызывать шаблон, используя его имя. Сделать это можно с помощью элемента `xsl:call-template`, указав в атрибуте `name` имя вызываемого шаблона. В листинге 1.12 приведен простой пример вызова шаблона (`simple_print`), который распечатывает текст `Person.` при каждом своем вызове.

Листинг 1.12: Вызов шаблона

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

```
    <xsl:template match="person">
  <xsl:call-template name="simple_print" />
    </xsl:template>
</xsl:stylesheet>
```

Однако для более сложных применений вызова шаблона может потребоваться передача в него определенных данных. Сделать это можно через объявление параметров с помощью элемента `xsl:with-param`. В этом случае необходимо описать принимаемые параметры в вызываемом шаблоне с помощью элемента `xsl:param`. Обращаться к значению параметра необходимо через символ `$`. В листинге 1.13 приведен пример вызова шаблона `print_born_year`, который принимает параметр `born_year` и распечатывает его значение.

Листинг 1.13: Вызов шаблона с параметрами

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <!-- Вызываемый шаблон. -->
    <xsl:template name="print_born_year">
        <!-- Описываем параметр. -->
        <xsl:param name="born_year"  />
        <xsl:value-of select="$born_year"/>
    </xsl:template>
    <!-- Вызываем наш шаблон для каждого элемента person. -->
    <xsl:template match="person">
        <xsl:call-template name="print_born_year">
        <!-- Передаем параметр в вызываемый шаблон. -->
        <xsl:with-param name="born_year">
            <xsl:value-of select="@born"/>
        </xsl:with-param>
        </xsl:call-template>
    </xsl:template>
</xsl:stylesheet>
```

Стоит отметить разницу между элементами `xsl:call-template` и `xsl:apply-template`. Элемент `xsl:call-template` работает как вызов функции с параметрами в традиционных языках программирования. Элемент `xsl:apply-templates` получает на вход множество узлов и итеративно обрабатывает их, не заставляя пользователя заботиться об организации цикла. Также эти элементы различны в отношении “текущего узла”. Для `xsl:call-template` он остается одинаковым как для вызываемого шаблона, так и для вызывающего. В то

время как для `xsl:apply-templates` “текущий узел” с каждым свои итеративным вызовом изменяется.

Элементы форматирования чисел Элемент `decimal-format` позволяет задавать формат вывода чисел, который можно затем использовать в функции `format-number`. Данный шаблон может принимать следующие атрибуты:

- name — задает имя шаблона.
- decimal-separator — определяет символ отеделения целой части от дробной. По умолчанию используется символ ’.’.
- grouping-separator — определяет символ разделения групп цифр. По умолчанию используется символ ’,’.
- infinity — определяет строку бесконечности. По умолчанию используется символ ’Infinity’.
- minus-sign — определяет символ перед отрицательным числом. По умолчанию используется символ ’-’.
- NaN — определяет значение строки в том случае, если было передано нечисловое значение. По умолчанию используется символ ’NaN’.
- percent — определяет символ процента. По умолчанию используется символ ’%’.
- per-mille — определяет символ промилле. По умолчанию используется символ ’‰’.
- zero-digit — задает символ нуля. Остальные цифры являются следующиеми символами за указанным в качестве символа нуля. По умолчанию используется символ ’0.
- digit — определяет символ цифры в шаблоне. По умолчанию используется символ ’#’.
- pattern-separator — определяет разделитель между положительной и отрицательной частями подшаблонов. По умолчанию используется символ ’;’.

В листинге 1.14 приведены два шаблона для вывода чисел. Шаблон `us` использует стандартные средства для вывода и форматирования чисел, а шаблон `df` использует для раздения целой и дробной частей символ ',', а для шаблона требует указывать символ '?' на месте цифр.

Листинг 1.14: Использование элемента xsl:decimal-format

```
XSLT:
<xsl:decimal-format name="us"
    decimal-separator='.' grouping-separator=',' />
<xsl:value-of
    select="format-number(24535.2, '###,###.00', 'us')"/>

Output:
24,535.20

XSLT:
<xsl:decimal-format name="df"
    decimal-separator=',' grouping-separator='' digit='?' />
<xsl:value-of
    select="format-number(24535.2, '???,???.00', 'df')"/>

Output:
24535,20
```

Элемент `xsl:number` используется для вывода заданного числа в заданном формате. Если число не задано, то подставляется текущая позиция элемента, которой можно управлять для построения нужной иерархии. Данный элемент может принимать следующие атрибуты:

- value — задает число, которое необходимо вывести. В качестве значения может использоваться функция языка XPath, возвращающая число (например, `position`).
- count — XPath выражение, определяющие какие узлы должны быть подсчитаны. Уровень поиска данных узлов зависит от атрибута `level`.
- level — уровень поиска узлов для выражения из атрибута `count`. Может принимать следующие значения:
  - single — поиск осуществляется на уровне родителя и самого себя.

  - multiple — поиск осуществляется на уровне всех родителей, предшествующих элементу.
  - any — поиск осуществляется на уровне всех элементов, предшествующих элементу.

  По умолчанию используется значение `single`.

- from — шаблон для узла, с которого необходимо начинать поиск для выражения из атрибута `count`
- format — задает формат вывода нумерации списка. Может принимать следующие значения:
  - “1” с результатом `1 2 3 ...`.
  - “01” с результатом `01 02 03 ...`.
  - “a” с результатом `a b c ...`.
  - “A” с результатом `A B C ...`.
  - “i” с результатом `i ii iii iv ...`.
  - “I” с результатом `I II III IV ...`.
  - другие специальные маркеры языка, задаваемые с помощью символов Unicode в шестнадцеричной системе счисления. Например, для списка из маркеров строчных букв русского языка можно использовать символ `0x0430`.
- lang — задает язык, алфавит которого должен использоваться при нумерации. Призван устранить неоднозначность в том случае если разные языки имеют одинаковый начальный символ. Например, тайваньская и корейская системы счисления имеют одинаковый начальный символ `0x4e01`, однако затем начинают различаться. Поэтому необходимо четко указать какую из этих систем использовать при нумерации (`zh-tw` или `ko` соответственно).
- letter-value — может принимать два значения: `alphabetic` (алфавитное) и `traditional` (традиционное). Позволяет устранить неоднозначность в том случае, если начальный символ алфавита и начальный символ системы счисления, использующий данный алфавит, совпадают. Например, для латинского алфавита

и римской системы счисления это не так. Начальный символ алфавита `a`, а начальный символ римской системы счисления — `i`. Однако, для алфавита Иврита и еврейской системы счисления эти символы совпадают (только с одиннадцатой позиции они начинают различаться), поэтому необходимо при использовании Иврита в качестве нумерованного списка четко указать использовать алфавит или систему счисления. По умолчанию используется значение `alphabetic`.

- grouping-separator — задает символ разделения групп цифр. По умолчанию используется запятая.
- grouping-size — задает количество цифр в группе, которые будут разделены с помощью символа из значения `grouping-separator` атрибута. По умолчанию используется значение 3.

В листинге 1.15 представлен пример создания нумерованного списка для различных форматов маркера. Результат применения преобразований можно увидеть в листинге 1.16

Листинг 1.15: Элемент `number`

```
<?xml version='1.0'?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org
    /1999/XSL/Transform" >
<xsl:template match="people">
  <xsl:for-each select="person">
    <xsl:number value="position()" format="1.␣"/>
    <xsl:value-of select="."/>
    <xsl:number value="position()" format="a)␣"/>
    <xsl:value-of select="."/>
    <xsl:number value="position()" format="&#x0410;)␣"/>
    <xsl:value-of select="."/>
  </xsl:for-each>

  <xsl:number value="37000000" grouping-separator="." grouping
    -size="3"/>
  <xsl:number value="3700" grouping-separator="." grouping-
    size="2"/>
</xsl:template>
</xsl:stylesheet>
```

Листинг 1.16: Результат работы элемента xsl:number

```
1. Джон фон Нейман
```

```
 a) Джон фон Нейман
 a) Джон фон Нейман
 2. Деннис Ритчи
 b) Деннис Ритчи
 б) Деннис Ритчи
 3. Кен Томпсон
 c) Кен Томпсон
 в) Кен Томпсон

37.000.000
37.00
```

Элементы копирования. Элемент `xsl:copy` позволяет копировать узел в выходной документ, в соответствии с теми шаблонами, которые вызываются внутри его описания. В листинге 1.17 приведен пример копирования элемента `person`. Результат работы этого преобразования для документа из листинга 1.3 представлен в листинге 1.18. Так как для первого элемента `person` не описан шаблон, то применяется шаблон по умолчанию (выводится содержимое элемента), для остальных элементов применяется шаблон `complex_name`.

Листинг 1.17: Использование элемента `xsl:copy`

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="person">
  <!-- Копируем элемент. -->
  <xsl:copy>
    <xsl:apply-templates/>
  </xsl:copy>
</xsl:template>
<xsl:template match="complex_name">
  <h2><xsl:value-of select="second_name" /></h2>
</xsl:template>
</xsl:stylesheet>
```

Листинг 1.18: Результат работы элемента xsl:copy

```
<?xml version="1.0"?>
<person>
    Джон фон Нейман
</person>
<person>
```

```
    <h2>Ритчи</h2>
</person>
<person>
    <h2>Томпсон</h2>
</person>
```

В отличие от элемента `xsl:copy` элемент `xsl:copy-of` копирует элемент не в соответсвии с описанными для них шаблонами, а напрямую. Обязательным атрибутом данного элемента является атрибут `select`, в котором задается набор элементов для копирования. Стоит отметить, что `xsl:copy-of` копирует также все дочерние элементы, атрибуты, пространства имен и потомки этих узлов.

В листинге 1.19 представлен код, который для элемента `person` копирует либо элемент `name`, либо элемент `first_name` из элемента `complex_name`. Результат можно посмотреть в листинге 1.20.

Листинг 1.19: Использование элемента `xsl:copy-of`

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="person">
  <xsl:copy-of select="complex_name/first_name|name" />
</xsl:template>
</xsl:stylesheet>
```

Листинг 1.20: Результат работы элемента xsl:copy-of

```
<?xml version="1.0"?>
<name>Джон фон Нейман</name>
<first_name>Деннис</first_name>
<first_name>Кен</first_name>
```

Условные элементы. XSLT позволяет ветвить выполнение шаблона в зависимости от поставленного условия. Для этого предназначены следующие элементы `xsl:if`, `xsl:choose`, `xsl:when` и `xsl:otherwise`.

Для элемента `xsl:if` обязательным атрибутом является атрибут `test`, который принимает логическое условие. Если данное условие истинно, то выполняются действия описанные в этом шаблоне. Если в качестве условия используется выражение XPath, то оно будет считаться ложным в том случае, если результат выполнения XPath не

содержит в себе элементов. В листинге 1.21 выводится значение атрибута `died` только в том случае, если этот атрибут существует. Результат выполнения данных преобразований представлен в листинге 1.22.

Листинг 1.21: Использование элемента `xsl:if`

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

  <xsl:template match="person">
    <b>
    <xsl:apply-templates />,
    <xsl:value-of select="@born" />
    <xsl:if test="@died">
      - <xsl:value-of select="@died" />
    </xsl:if>
    </b>
    </xsl:template>
</xsl:stylesheet>
```

Листинг 1.22: Результат работы элемента xsl:if

```
<?xml version="1.0"?>
<b>Джон фон Нейман, 1903 - 1957</b>
<b>Деннис Ритчи, 1941 - 2011</b>
<b>Кен Томпсон, 1943</b>
```

Если необходимо последовательно проверить несколько условий, то это можно организовать с помощью элемента `xsl:choose`, который состоит из элементов `xsl:when`. Каждый элемент `xsl:when` должен определять атрибут `test` с логическим условием. Если логическое условие истинно, то выполняются действия описанные в данном шаблоне, при этом оставшиеся условия не проверяются. Если необходимо выполнить некоторые действия в том случае, если никакое условие не выполнено, то для этого необходимо определить эти действия в элементе `xsl:otherwise`.

В листинге 1.23 используется элемент `xsl:choose` для проверки даты рождения и вывода соответсвующего сообщения (листинг 1.24).

Листинг 1.23: Использование элемента `xsl:choose`

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

```
  <xsl:template match="person">
      <u>
      <xsl:choose>
  <xsl:when test="@born &lt; 1925">
      Начало XX века.
  </xsl:when>
  <xsl:when test="@born &lt; 1975">
      Середина XX века.
  </xsl:when>
  <xsl:otherwise>
      Конец XX века.
  </xsl:otherwise>
    </xsl:choose>
    </u>
  </xsl:template>
</xsl:stylesheet>
```

Листинг 1.24: Результат работы элемента xsl:choose

```
<?xml version="1.0"?>
<u>Начало XX века.</u>
<u>Середина XX века.</u>
<u>Середина XX века.</u>
```

Элементы подключения других таблиц стилей. Использовать таблицы стилей XSLT из других документов можно двумя способами: либо импортировать их с помощью элемента `xsl:import`, либо подключить с помощью элемента `include`. Для каждого из этих элементов обязательным атрибутом является атрибут `href`, который должен содержать ссылку (URI) на подключаемый документ.

Для элемента `xsl:import` важен порядок указания подключения. Исходя из этого порядка будут применяться соответствующие шаблоны. Пусть пользователь использует следующую схему:

- XSLT-файл A импортирует XSLT-файлы B и C, именно в таком порядке.
- XSLT-файл B импортирует XSLT-файл D.
- XSLT-файл C импортирует XSLT-файл E.

Тогда приоритеты применения шаблонов будут следующими (в порядке возрастания): D, B, E, C, A. Т.е. в случае одинаковости шаблонов

будет применяться тот шаблон, документ которого имеет более высокий приоритет.

Важным последствием использования `xsl:import` является возможность явного применения шаблонов из импортированных документов. Для этого необходимо воспользоваться элементом `xsl:apply-imports`, который вызывает переопределеное правило шаблона, заданное в импортированной таблице стилей. Элемент `xsl:apply-imports` необходимо использовать тогда, когда необходимо применить шаблон из импортированной таблицы стилей в обход шаблонов из текущего документа.

Рассмотрим пример использования импорта таблиц стилей. Для таблицы стилей из листинга 1.25 подключаются таблицы стилей из листингов 1.26 и 1.27. Если мы применим данную таблицу стилей для входного документа из листинга 1.28, то получим следующий результат:

```
<?xml version="1.0"?>
<div>Some binary operations</div>
  1+2=12<br/>
  1-2=-1<br/>
  1*2=21<br/>
```

Вначале для каждого элемента `op` распечатывается первый операнд, значение атрибута `symbol`, второй операнд и знак “=” согласно шаблону из главного документа. Затем вызывается явное применение шаблонов из импортированных документов. Для первого и третьего выражения получаем значения `12` и `21` соответственно, так как шаблон для элемента `op` из документа `str.xsl` имеет более высокий приоритет по сравнению с шаблоном из документа `arith.xsl`. Но для второго выражения получаем значение `-1`, которое является результатом работы шаблона из документа `arith.xsl`, так как для условия со значением атрибута `symbol` “-” шаблон имеется только в этом документе.

Если поменять порядок импортирование документов `arith.xsl` и `str.xsl`, то получим следующий результат:

```
<?xml version="1.0"?>
<div>Some binary operations</div>
```

```
  1+2=3<br/>
  1-2=-1<br/>
  1*2=2<br/>
```

Для всех трех выражений после знака “=” получаем значение работы шаблонов из документа `arith.xsl`, так как эти шаблоны стали более приоритетные по сравнению с шаблонами из документа `str.xsl`.

Если явно не вызывать элемент `apply-imports`, то получим следующий результат:

```
<?xml version="1.0"?>
<div>Some binary operations</div>
  1+2=<br/>
  1-2=<br/>
  1*2=<br/>
```

Шаблоны из документов `str.xsl` и `arith.xsl` не применяются, так как для элемента `op` шаблон определен в главном документе, который имеет самый высокий приоритет.

Листинг 1.25: Использование элемента `xsl:apply_imports`

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org
    /1999/XSL/Transform">
  <xsl:import href="arith.xsl"/>
  <xsl:import href="str.xsl"/>
  <xsl:template match="op">
    <xsl:value-of select="operand[1]"/>
    <xsl:value-of select="@symbol"/>
    <xsl:value-of select="operand[2]"/>
    =
    <xsl:apply-imports/>
  <br/>
  </xsl:template>
</xsl:stylesheet>
```

Листинг 1.26: Документ `arith.xsl`

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

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

Листинг 1.27: Документ `str.xsl`

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org
    /1999/XSL/Transform">
  <xsl:template match="desc">
    <div><xsl:value-of select="."/></div>
  </xsl:template>
  <xsl:template match="op[@name='add']">
    <xsl:value-of select="operand[1]"/>
    <xsl:value-of select="operand[2]"/>
  </xsl:template>
  <xsl:template match="op[@name='mul']">
    <xsl:value-of select="operand[2]"/>
    <xsl:value-of select="operand[1]"/>
  </xsl:template>
</xsl:stylesheet>
```

Листинг 1.28: Входной документа для примера из листинга 1.25

```
<?xml version="1.0" encoding="UTF-8"?>
<ops>
  <desc>Some binary operations</desc>
  <op name="add" symbol="+">
    <operand>1</operand>
    <operand>2</operand>
  </op>
  <op name="sub" symbol="-">
    <operand>1</operand>
    <operand>2</operand>
  </op>
  <op name="mul" symbol="*">
    <operand>1</operand>
    <operand>2</operand>
```

```
  </op>
</ops>
```

Для элемента `xsl:include` не существует приоритетов. В случае одинаковости шаблонов будет вызван шаблон из последней подключенной таблицы стилей. Стоит особо отметить, что при наличии одинаковых шаблонов в главном и подключенном документах, шаблон из главного документа будет проигнорирован. Если в листинге 1.25 использовать элемент `xsl:include` вместо `xsl:import`, то будет получен следующий результат для входного документа из листинга 1.28.

```
<?xml version="1.0"?>
<div>Some binary operations</div>
  12<br/>
  -1<br/>
  21<br/>
```

Шаблон из главного документа игнорируется. Для первого и третьего выражения применяется шаблон из документа `str.xsl`, так как он подключается последним, для второго выражения применяется шаблон из документа `arith.xsl`, так как для условия со значение атрибута `symbol` "-" шаблон в документе `str.xsl` отсутствует.

Элемент xsl:key. Элемент `xsl:key` позволяет создать описание ключа, на которое затем можно сослаться. Обязательными атрибутами этого элемента являются атрибут `name`, определяющий имя ключа, атрибут `match`, задающий список элементов для ключа и атрибут `use`, задающий значение ключа. Сослаться на значение ключа можно с помощью функции `key`.

В листинге 1.29 приведен пример создания ключа с именем `people-search`, работающий с элементами `person` и использующий атрибут `born` в качестве своего значения. Вызов функции `key('people-search', '1903')` формирует список элементов `person` со значением атрибута `born` 1903.

Листинг 1.29: Использование элемента `xsl:key`

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

```
  <xsl:key name="people-search" match="person" use="@born" />
  <xsl:template match="/">
    <xsl:for-each select="key('people-search','1903')">
       <div><xsl:value-of select="@born" /></div>
    </xsl:for-each>
  </xsl:template>
</xsl:stylesheet>
```

Элементы работы с пробельными символами. Язык XSLT позволяет управлять пробельными символами в выходном тексте. С точки зрения спецификации XSLT пробельными символами являются символ переноса каретки, используемый для возвращения позиции курсора к началу строки, символы перевода строки, пробела и табуляции.

Элементы `xsl:preserve-space` и `xsl:strip-space` позволяют соответственно оставить в неизменном виде или удалить пробельные символы из выходного текста. Единственным и обязательным атрибутом этих элементов является атрибут `elements`, в котором необходимо определить список элементов для выполнения соответствующего действия. Стоит отметить, что при указании списка можно использовать символ `*`, как отдельно, так и в качестве префикса, обозначающего все элементы или элементы с определенным суффиксом.

Элемент `xsl:preserve-space` не оказывает влияния на пробелы внутри текстовых элементов. Элемент `xsl:strip-space` не обрабатывает элемент, если либо он включен в список для обработки элементом `xsl:preserve-space`, либо содержит хотя бы один непробельный символ.

Вспомогательные элементы. В данном параграфе будут описаны вспомогательные XSLT элементы. Элемент `xsl:comment` позволяет вставлять в выходной элемент комментарий. В листинге 1.30 приведен пример использования данного элемента.

Листинг 1.30: Использование элемента xsl:comment

```
 XSLT:
<xsl:comment>Insert top news story</xsl:comment>
 Выход:
<!--Insert top news story-->
```

Элемент `xsl:element` позволяет добавить в результирующий документ новый XML элемент. Обязательным аттрибутом этого элемента является атрибут `name`, задающий имя элемента. Также есть возможность указать пространство имен (атрибут `namespace`) и список атрибутов. Добавить атрибуты можно либо указав определенный список атрибутов (используя атрибут `use-attribute-sets`, в значении которого через пробел указываются имена списков), либо прямо в описании элемента (описание атрибутов прямо в элементе стоит делать до описания значения элемента, иначе некоторые реализации XLST преобразований могут выдать ошибку о невозможности добавления атрибута к уже добавленному элементу).

В листинге 1.31 определяется элемент с именем `scientist`. К нему добавляется список атрибутов под именем `scientist_common_attrs`, также в самом элементе определен атрибут `born`, со значением атрибута `born` из оригинального элемента. Значением элемента является значение элемента `name`. В листинге 1.32 приведен результат применения этих XSLT преобразований к входному элементу из 1.3. Отметим, что два последних добавленых в результат элемента являются пустыми, т.к. оригинальные элементы не содержат элемента с именем `name`.

Листинг 1.31: Добавление элемента

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org
    /1999/XSL/Transform">
  <xsl:template match="person">
    <xsl:element name="scientist"
      use-attribute-sets="scientist_common_attrs">
      <xsl:attribute name="born">
        <xsl:value-of select="@born" />
      </xsl:attribute>
      <xsl:value-of select="name" />
    </xsl:element>
  </xsl:template>
  <xsl:attribute-set name="scientist_common_attrs">
    <xsl:attribute name="field">Информатика</xsl:attribute>
    <xsl:attribute name="country">США</xsl:attribute>
  </xsl:attribute-set>
</xsl:stylesheet>
```

Листинг 1.32: Результат добавления элемента

```
<?xml version="1.0"?>
```

```
<scientist field="Информатика" country="США" born="1903">
    Джон фон Нейман
</scientist>
<scientist field="Информатика" country="США" born="1941"/>
<scientist field="Информатика" country="США" born="1943"/>
```

Элемент `xsl:attribute` позволяет добавлять атрибут к родительскому элементу. Обязательным атрибутом данного элемента является атрибут `name`, который задает имя атрибута. В листинге 1.33 представлен пример кода XSLT, который добавляет атрибут `src` к элементу `img`, выбирая его значение из элементов входного документа (`imagename`). Результатом работы данного шаблона будет элемент `img` с атрибутом `src` и значением текстового элемента `imagename`.

Листинг 1.33: Добавление атрибута

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

Также XSLT позволяет добавить множество атрибутов с помощью элемента `xsl:attributes-set`.

Элемент `xsl:message` позволяет отправить XSLT процессору сообщение. Обычно он используется для вывода сообщений об ошибке. Необязательным атрибутом этого элемента является атрибут `terminate`, который может принимать значение либо “yes”, что означает прервать работу по преобразованиям, либо “no”, что означает продолжить работу. По умолчанию используется значение “no”.

В листинге 1.34 приведен пример использования элемента `xsl:message` для вывода сообщения о том, что должен быть определен атрибут `died` в случае если атрибут `born` меньше значения 1850.

Листинг 1.34: Использование элемента `xsl:message`

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org
    /1999/XSL/Transform">
  <xsl:template match="person">
    <xsl:if test="@born &lt; 1850 and not(@died)">
      <xsl:message terminate="yes">
        Атрибут died должен быть определен.
      </xsl:message>
```

```
    </xsl:if>
  </xsl:template>
</xsl:stylesheet>
```

Элемент `xsl:namespace-alias` позволяет переопределить пространство имен в выходном документе. Это может быть полезно при генерации другого XSLT документа из исходного. Обязательными атрибутами этого элемента являются атрибут `stylesheet-prefix`, задающий пространство имен, которое необходимо заменить в исходном документе, и `result-prefix`, задающий пространство имен, на которое необходимо заменить в выходном документе. Стоит отметить, что элемент `xsl:namespace-alias` является элементом верхнего уровня и может находиться только в элементе `xsl:stylesheet` (или `xsl:transform`.

В листинге 1.35 пространство имен `alt` в исходном документе изменяется на пространство имен `xsl` в результирующем документе (листинг 1.36).

Листинг 1.35: Использование элемента `xsl:namespace-alias`

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

Листинг 1.36: Выходной документ для листинга 1.35

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

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

Элемент `xsl:variable` позволяет определить константу с определенным именем (атрибут `name`) и значением, которое задается в значении атрибута `select` или самом содержимом элемента (если значение будет задано там и там, то это считается ошибкой, задание значения должно осуществляться в одном месте). Стоит отметить, что значение данной константы изменить нельзя. Константа глобальна, то есть доступна в любом месте документа, в том случае, если она определена как элемент верхнего уровня. Если константа определена внутри шаблона, то она считается локальной и доступна только внутри этого элемента. Обращаться к значению элемента необходимо по имени с подстановкой перед именем символа `$`.

В листинге 1.37 определены две константы с именами `delimiter` и `limit-year`. Первая константа является глобальной и определяет свое содержимое внутри элемента, а вторая — локальной и задает свое значение в атрибуте `select`. Использование константы `limit-year` за пределами шаблона для элементов `person` считается ошибкой. Результат выполнения данных преобразований представлен в листинге 1.38.

Листинг 1.37: Использование элемента `xsl:variable`

```
<?xml version='1.0'?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org
    /1999/XSL/Transform" >
  <!-- Глобальная переменная, распечатывающаю строку -->
  <xsl:variable name="delimiter">
    <xsl:text>==========</xsl:text>
  </xsl:variable>

  <!-- Шаблон для элемента person -->
  <xsl:template match="person">
    <!-- Локальная переменная, задающая ограничение по году. -->
    <xsl:variable name="year-limit" select="'1930'" />

    <!-- Вывод элемента person и строки разделителя. -->
    <xsl:if test="@born > $year-limit">
      <xsl:value-of select="." />
      <xsl:copy-of select="$delimiter" />
    </xsl:if>
  </xsl:template>
</xsl:stylesheet>
```

Листинг 1.38: Выходной документ для листинга 1.37

```
Деннис Ритчи
==========
Кен Томпсон
==========
```

Элемент `xsl:processing-instruction` позволяет вставлять в результирующее дерево инструкции по обработке. Данный элемент имеет один обязательный атрибут `name`, задающий имя инструкции. Код в листинге 1.39 показывает как с помощью элемента `xsl:processing-instruction` вставить инструкцию таблицей стилей.

Листинг 1.39: Пример использования элемента processing-instruction

```
XSLT:
<xsl:processing-instruction name="xml-stylesheet">
href="style.css" type="text/css"
</xsl:processing-instruction>

Выходной документ:
<?xml-stylesheet href="style.css" type="text/css"?>
```

1.4. Встроенные шаблонные правила

Стоит напомнить, что XML документ может содержать семь видов узлов (корневой узел, узлы элементов, узлы атрибутов, текстовые узлы, узлы комментариев, узлы инструкций обработки и узлы пространства имен). Каждый вид узла имеет встроенные шаблонные правила по умолчанию. Рассмотрим каждое из них.

Шаблон по умолчанию для текстовых узлов (выражение `text()`) и атрибутов (выражение `@*`) представлен в листинге 1.40. При обработке текстового узла или узла атрибута их значения извлекаются и подставляются в шаблон.

Листинг 1.40: Шаблон по умолчанию для текстовых узлов

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

Для узлов элементов, в том числе корневого узла, шаблон по умолчанию представлен в листинге 1.41. Как видно из шаблона для элемен-

тов применяются определенные пользователем шаблоны в том случае если они имеются.

Листинг 1.41: Шаблон по умолчанию для текстовых узлов

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

1.5. Режимы

В некоторых случаях для выполнения задачи получения выходного документа необходимо использовать разные шаблоны для одних и тех же элементов. Для решения этой проблемы XSLT использует понятие режимов (задается с помощью атрибута `mode`), которые позволяют выбрать из множества шаблонов один необходимый.

Предположим, что для документа из листинга 1.3 нам необходимо создать оглавление, а затем уже параграфы о людях описанных в этом документе. Листинг 1.42 задает два шаблона для элементов `person`. Первый шаблон — это шаблон, который выполняется по умолчанию, а второй шаблон — это шаблон режима **“toc”**, который будет выполняться при указании данного режима в элементе `apply-templates`.

Листинг 1.42: Использование режимов

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/
    Transform">
<xsl:template match="people">
    <html>
  <head><title>Знаменитые ученые</title></head>
  <body>
      <ul><xsl:apply-templates select="person" mode="toc"/></ul>
      <xsl:apply-templates select="person"/>
  </body>
    </html>
</xsl:template>

<!-- Шаблон оглавления -->
<xsl:template match="person" mode="toc">
    <li><xsl:apply-templates select="complex_name/second_name" mode="toc
    "/></li>
</xsl:template>

<!-- Обычный шаблон -->
<xsl:template match="person">
  <p>
    <b><xsl:apply-templates select="complex_name/second_name"/></b>,
       <xsl:apply-templates select="complex_name/first_name"/>
```

```
      (<xsl:apply-templates select="@born" /> - <xsl:apply-templates
    select="@died" />)
  </p>
</xsl:template>

</xsl:stylesheet>
```

Выходной документ после применения XSLT из листинга 1.42 представлен в листинге 1.43:

Листинг 1.43: Результат работы режимов

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=
    UTF-8">
<title>Знаменитые ученые</title>
</head>
<body>
<ul>
<li>Ритчи</li>
<li>Томпсон</li>
</ul>
<p><b>Ритчи</b>, Деннис (1941 - 2011)</p>
<p><b>Томпсон</b>, Кен (1943 - )</p>
</body>
</html>
```

1.6. Пространства имен

Если во входном XML документе были использованы элементы из сторонних пространств имен, то для обработки с помощью XSLT нужно это также учитывать. Предположим, что элементы входного документа из листинга 1.3 описаны в пространстве имен `http://cs.petrsu.ru/people`. Тогда в XSLT коде нужно это явно указать и для доступа к элементам использовать префикс данного пространства имен. Пример такого XSLT документа приведен в листинге 1.44.

Листинг 1.44: Использование пространств имен

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

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

§ 2. Каскадные таблицы стилей для XML (CSS XML)

Язык разметки XML предназначен для описания данных. Однако, помимо задачи описания данных существует задача их представления пользователю. Одним из способов описания стилей оформления для конечного документа являются каскадный таблицы стилей (Cascading Style Sheets, CSS [5]). Одним из примеров приложений, умеющий применять стили CSS, являются современные веб-обозреватели. Стоит особо отметить, что язык CSS не является XML разметкой. В данной главе речь пойдет о стилях CSS.

Для примеров будет использоваться XML файл из листинга 2.1.

Листинг 2.1: Пример XML файла

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/css" href="example.css"
  media="all" charset="UTF-8"?>
<people>
    <person born="1903" died="1957" class="person-item"
            place-born="Будапешт, Венгрия" id="1">
        <name>Джон фон Нейман</name>
    </person>
    <person born="1941" died="2011" class="person-item"
            place-born="НьюЙорк-, США" id="2">
        <name>Деннис Ритчи</name>
    </person>
    <person born="1943" class="person-item"
            place-born="Новый Орлеан, США" id="3">
        <name>Кен Томпсон</name>
        <field>Математика</field>
        <field>Информатика</field>
    </person>
</people>
```

2.1. Подключение таблицы стилей CSS к документу XML

Для того чтобы подключить таблицу стилей CSS к документу XML необходимо воспользоваться инструкцией `xml-stylesheet`. Пример ее вызова приведен в листинге 2.2.

Листинг 2.2: Подключение таблицы стилей CSS

```
<?xml-stylesheet type="text/css" href="example.css" media="
    screen"
      title="Для вебобозревателя-" charset="UTF-8"?>
```

Вызов данной инструкции поддерживает указание следующих атрибутов:

- `type` — MIME тип таблицы стилей. Для таблицы стилей CSS указывается значение `text/css`.
- `href` — ссылка на документ, содержащий описание таблицы стилей CSS. Может быть как относительной, так и абсолютной. В листинге 2.2 указана относительная ссылка на документ `example.css`, находящийся в той же директории, что и сам XML документ.
- `charset` — указание кодировки таблицы стилей.
- `title` — обычно представляет собой краткое описание таблицы стилей.
- `media` — тип носителя для отображения. На данный момент определены следующие типы носителей:
  - `all` — предназначен для всех видов устройств.
  - `braille` — предназначен для воспроизведения на устройствах с поддержкой шрифта Брайля.
  - `embossed` — предназначен для печати на принтерах с поддержкой шрифта Брайля.
  - `handheld` — предназначен для вывода на маленькие экраны с ограниченными характиристиками.
  - `print` — предназначен для печати на материальных носителях (например, бумаге).
  - `projection` — предназначен для
  - `screen` — предназначен для вывода на цветные экраны компьютеров.
  - `speech` — предназначен для устройств или программных средств, синтезирующих речь.

  - `tty` — предназначен для вывода на моноширинные экраны (например, терминалы).
  - `tv` — предназначен для вывода на телевизионные экраны (цветные экраны с высоким разрешением и возможностью воспроизведения звука, но ограниченными возможностями по прокрутке).

- `alternate` — флаг для обозначения существования альтернативной таблицы стилей. Может принимать значения “yes” или “no” (по умолчанию используется значение “no”). Если флаг установлен в значение “yes”, то программная система, применяющая таблицы стилей, может предгалагать пользователю выбрать один из стилей.

В листинге 2.3 приведен пример подключения нескольких таблиц стилей CSS для разных видов носителей для отображения.

Листинг 2.3: Подключение нескольких таблиц стилей CSS

```
<?xml-stylesheet type="text/css" href="example.css" media="screen"
      title="Для вебобозревателя-" charset="UTF-8"? alternate="yes">
<?xml-stylesheet type="text/css" href="example_light_theme.css"
      media="screen" title="Для вебобразователя-" charset="UTF-8"?>
<?xml-stylesheet type="text/css" href="example_projection.css"
      media="projection" title="Для презентаций" charset="UTF-8"?>
<?xml-stylesheet type="text/css" href="example_print.css" media="print"
      title="Для печати" charset="UTF-8"?>
```

2.2. Синтаксис таблиц стилей CSS

Таблица стилей CSS состоит из правил, которые определяют свойства для элементов XML. Данный набор свойст заключается в фигурные скобки и представляет собой блок из пар, разделенных знаком точка с запятой. Каждая пара состоит из имени свойства и его значения. Набор правил задается для набора элементов, который определяется с помощью правил выбора элементов (или селектора, более подробно конструкции выбора см. в 2.3.). В листинге 2.4 в качестве селектора использовалось имя элемента `name` и заданы значения свойств ширины (`30 px`) и цвета (`red`).

Листинг 2.4: Пример определения правил CSS

```
name {
```

```
    width: 30px;
    color: red;
}
```

Также отметим, что для комментариев в таблицах стилей CSS используются символы `/* комментарий */`.

2.3. Конструкции выбора элементов

Для выбора набора элементов таблица стилей CCS предусматривает возможность применения сокращенного варианта XPath с использованием некоторых своих особенностей. Перечислим основные конструкции для выбора элементов, разбив их на несколько категорий (при написании `элемент` в конструкции выбора подразумевается, что речь идет об имени элемента).

Выбор потомков, дочерних и одноуровневых элементов

- `*` — выбор абсолютно всех элементов (так называемый универсальный селектор).
- `.` — выбор элементов с заданным значением атрибута `class`. Например, конструкция `.person-item` выберет все элементы, значением атрибута `class` которого является `person-item`. Для листинга 2.1 это будут три элемента с именем `person`.
- `#` — выбор элементов с заданным значением атрибута `id`. Например, конструкция `#1` выберет все элементы, значением атрибута `id` которого является `1`. Для листинга 2.1 это будет первый элемент с именем `person` (Джон фон Нейман).
- `элемент` — по имени элемента выберутся все элементы, соответствующие данному имени. Например, для конструкции `person` и примера из документа 2.1 будет выбрано три элемента с именем `person`.
- `первый элемент, второй элемент` — выборка и первых элементов, и вторых элементов. Имена элементов перечисляются через запятую в неограниченном количестве. Например, для конструкции `person, name` и примера из документа 2.1 будет выбрано три элемента с именем `person` и три элемента с именем `name`.

- `первый элемент второй элемент` — выборка всех вторых элементов, в родительской иерархии которых встречаются первые элементы, другими словами для первого элемента выбираются все потомки с именем второго элемента. Имена элементов разделяются пробельными символами. Например, для конструкции `people name` будут выбраны все элементы с именем `name`.

- `первый элемент > второй элемент` — выборка вторых элементов, родителями которых являются первые элементы, т.е. выборка всех элементов со вторым именем, которые являются прямыми потомками элементов с первым именим. Например, для конструкции `people > name` и примера из документа 2.1 не будет выбрано ни одного элемента, а для конструкции `person > name` будет выбрано три элемента с именем `name`.

- `первый элемент + второй элемент` — выборка второго элемента, который непосредственно стоит за первым элементом. Отличается от конструкции `>` тем, что работает на одном уровне иерархии. Например, конструкция `name + field` для третьего элемента `person` выберет один элемент `field` с содержимым `Математика`, так как этот элемент стоит непосредственно за элементом `name`, но не выберет элемент `field` с содержимым `Информатика`, так как этот элемент стоит за первым элементом `field`.

- `первый элемент` $\sim$ `второй элемент` — выборка всех вторых элементов, которым предшествует первый элемент. Отличается от конструкции `+` тем, что выбирает все вторые элементы независимо от того сколько и какие элементы между ними находятся и на каком уровне потомков первого элемента этот элемент располагается.

Выбор атрибута

- `[атрибут]` — выборка всех элементов с заданным атрибутом. Например, конструкция `[died]` для документа из примера 2.1 выберет два элемента с именем `person` (Джон фон Нейман (`died=1957`) и Деннис Ритчи (`died=2011`)).

- `[атрибут=значение]` — выборка всех элементов с заданным атрибутом и значением, строго соответствующее заданному. Например, конструкция `[died="2011"]` выберет один элемент с именем `person` (Деннис Ритчи (`died=2011`)).

- `[атрибут =слово]` — выборка всех элементов с заданным атрибутом и значением, содержащим заданное слово. Слово определяется как набор непробельных символов. Например, конструкция `[place-born∼="США"]` выберет два элемента с именем `person` (Деннис Ритчи (`place-born=‘‘Нью-Йорк, США’’`) и Кен Томпсон (`place-born=‘‘Новый Орлеан, США’’`)). Если бы между знаком запятой и словом “США” не стояло бы пробела, то данный элемент не попал бы в результат выборки.

- `[атрибут|=слово]` — выборка всех элементов с заданным атрибутом и значением, которое начинается с заданного слова. Слово определяется как набор непробельных символов. Например, конструкция `[place-born|="Будапешт"]` выберет один элемента с именем `person` (Джон фон Нейман (`place-born=‘‘Будапешт, Венгрия’’`)). Если задать значение образца как “Буд”, то в результат выборки не попадет ни один элемент.

- `[атрибутˆ=значение]` — выборка всех элементов с заданным атрибутом и значением, которое начинается с заданного. Например, конструкция `[bornˆ=‘‘194’’]` выберет два элемента с именем `person` (Деннис Ритчи (`born=‘‘1941’’`) и Кен Томпсон (`born=‘‘1943’’`)).

- `[атрибут$=значение]` — выборка всех элементов с заданным атрибутом и значением, которое заканчивается заданным. Например, конструкция `[born$=‘‘3’’]` выберет два элемента с именем `person` (Джон фон Нейман и Кен Томпсон).

- `[атрибут*=значение]` — выборка всех элементов с заданным атрибутом и значением, которое содержит заданное. Например, конструкция `[died*=‘‘1’’]` выберет два элемента с именем `person` (Джон фон Нейман и Деннис Ритчи).

Выбор псевдоклассов Под псевдоклассами подразумеваются классы, которые может добавлять система отображения на действие пользователя (например, отслеживание курсора мыши, посещенные/-непосещенные ссылки и т.д.). Перечислим некоторые из них:

- `link` — непосещенная ссылка.
- `visited` — посещенная ссылка.
- `active` — активный элемент.
- `hover` — элемент под указателем мыши.
- `focus` — выборка элементов, находящихся в фокусе.

Выбор псевдоэлементов Псевдоэлементы применяются к определенному фрагменту текста, например:

- `first-letter` — выбирает первую букву элемента.
- `first-line` — выбирает первую строку блочного элемента.
- `selection` — выбирает выделенный текст.
- `before` и `after` — выбирают точки перед и после указанного элемента.

2.4. Отображение элементов

После того как выбрана группа XML элементов с помощью описанных выше конструкций необходимо расположить их на экране. Сделать это можно с помощью задания свойства `display`. Если не задано никакое значение этого свойства, то по умолчанию используется значение `inline`. Существует несколько вариантов расположения. Рассмотрим некоторые из них:

- `inline` — помещение элемента в следующую доступную позицию. Фактически располагает элементы на одной строке. Например, код CSS в листинге 2.5 примененный к листингу 2.1 даст результат, описанный в листинге 2.6.

Листинг 2.5: Использование display:inline в CSS коде

```
name {
    display: inline;
}
```

Листинг 2.6: Результат применения display:inline

```
Джон фон Нейман Деннис Ритчи Кен Томпсон
```

- `block` — разделяет одноуровневые элементы на блоки. Фактически ставит перевод строки между элемента. Результат применения `display:block` к элементам `name` листинга 2.1 представлен в листинге 2.7.

Листинг 2.7: Результат применения display:block

```
Джон фон Нейман
Деннис Ритчи
Кен Томпсон
```

- `list-item` — разделяет одноуровневые элементы на блоки (как и в случае с `block`), а также для каждого блока добавляет маркер.
- `none` — скрывает элемент.
- `inline-block` — пока это возможно помещает элемент в следующую доступную позицию, как только это становится невозможным (закончилась ширина носителя, на который выводится результат), начинает новый блок.

Важным множеством элементов отображения является табличное отображение. Рассмотрим эти элементы (для их понимания приведены соответствующие этим элементам HTML теги):

- `table` — определение таблицы (`<table>`).
- `inline-table` — таблица внутри блока.
- `table-row-group` — группа строк таблицы (`<tbody>`).
- `table-header-group` — заголовок таблицы (`<thead>`).

Джон фон Нейман	1903	Будапешт, Венгрия
Деннис Ритчи	1941	Нью-Йорк, США
Кен Томпсон	1943	Новый Орлеан, США

Таблица 1: Результат применения табличного отображения

- `table-footer-group` — окончание таблицы (`<tfoot>`).
- `table-row` — строка (`<tr>`).
- `table-column-group` — группа колонок (`<colgroup>`).
- `table-column` — колонка (`<col>`).
- `table-cell` — ячейка (`<td>`).
- `table-caption` — подпись к таблице (`<caption>`).

Создадим таблицу для примера из листинга 2.1, где в качестве таблицы будем использовать элемент `people`, строки — элемент `person`, значений ячеек — значения атрибутов `born` и `place-born` и элемента `name`. CSS код представлен в листинге 2.8, а результат в таблице 1.

Листинг 2.8: Использование табличного отображения в CSS коде

```
people { display: table; }
person { display: table-row; }
name, person[born], person[place-born] { display: table-cell; }
```

2.5. Определение стиля элемента

Таблицы стилей CSS позволяют не только выбирать и размещать элементы, но и задавать их стиль (размеры, шрифты, цвета и т.д.). Рассмотрим эти возможности CSS более подробнее.

Определение размеров

- `border-width` — задает ширину рамки элемента. Также позволяет определять ширину для разных сторон. Если указано одно значение, то оно применяется для всех четырех сторон элемента. Если указано два значения, то первое значение применяется

к верхней и нижней сторонам, а второе значение — к правой и левой сторонам. Если указаны три значения, то они применяются в следующем порядке: верхняя сторона, правая и левая сторона, нижняя сторона, если четыре значения — верняя сторона, правая сторона, нижняя сторона, левая сторона. Помимо числовых значений ширину можно задавать в описательном стиле: thin (тонкая), medium (средняя), thick (толстая). В листинге 2.9 для элемента `people` будет использоваться рамка шириной 10px по всем сторонам, а для элемента `person` — рамка средней ширина по верхней стороне, рамка размером в 3 пикселя по правой стороне, тонкая рамка для нижней стороны и рамка в 5 пикселей по левой стороне.

Листинг 2.9: Задание ширины рамок

```
people { border-width: 10px; }
person { border-width: medium 3px thin 5px; }
```

- `font-size` — задает размер шрифта блока. Позволяет это делать описательно (small (маленький), large (большой), medium (средний), x-small (супер маленький) и т.д.), описательно относительно родительского элемента (smaller (меньше) и larger (больше)), в процентах от размера шрифта родительского элемента (например, 80%) и числовое значение в заданных единицах (пикселях, сантиметрах, дюймах, точках и т.д.). В листинге 2.10 для элемента `people` используется маленький шрифт, а для элемента `person` — шрифт размеров в 1 сантиметр.

Листинг 2.10: Задание размера шрифта

```
people { font-size: small; }
person { font-size: 1cm; }
```

- `line-height` — задает высоту строки блока. Позволяет задавать либо с помощью числовых значений в заданных единицах, либо в процентах от размера шрифта блока, либо в множителе, на который будет умножен размер шрифта. В листинге 2.11 высота строки для элемента `person` будет результатом уножения значения 1.5 на размер шрифта элемента `person`.

Листинг 2.11: Задание высоты блока

```
people { line-height: 10px; }
person { line-height: 1.5; }
```

- `margin-left, margin-right, margin-top` и `margin-bottom` — отступ от краев родительского блока, соответственно слева, справа, сверху и снизу. Задается либо в числовых значениях заданных единиц, либо в процентах от ширины блока. Также отступ можно задать с помощью свойства `margin`, количество значений которого соответствует описанию приведенному для свойства `border-width`.
- `padding-left, padding-right, padding-top` и `padding-bottom` — поля блока, соответственно слева, справа, сверху и снизу. Задается либо в числовых значениях заданных единиц, либо в процентах от ширины блока. Также поля можно задать с помощью свойства `padding`, количество значений которого соответствует описанию приведенному для свойства `border-width`.

Задание свойств шрифтов.

- `font-family` — названия шрифтов. Позволяет перечислить через запятую названия шрифтов в порядке приоритетов (если система отображения не поддерживает первый шрифт, то она пробует применить второй, если не поддерживает второй, то третий и т.д.), которые необходимо применить к блоку. Названия шрифтов делят на две группы: специальные (`family-name`) и универсальные (`generic-family`). Чтобы избежать ситуации, когда система отображения не поддерживает все перечисленные шрифты, рекомендуется в качестве последнего шрифта использовать универсальный, который может принимать следующие значения: `serif`, `sans-serif`, `cursive`, `fantasy`, `monospace`.
- `font-style` — стиль шрифта. Может принимать следующие значения: `italic` — курсивный начертание, `normal` — нормальное начертание, `oblique` — наклонное начертание (в отличие от курсивного начертания, которое имитирует рукописный шрифт, наклонное начертание получается путем наклона обычных букв).

- `font-size` — размер шрифта (описание см. выше).

- `font-variant` — регистр шрифта. Может принимать два значения: `normal` — оставляет регистр без изменения, `small-caps` — отображение в виде капители (все строчные символы преобразует в заглавные уменьшенного размера).

- `font-weight` — насыщенность шрифта. Может принимать описательные значения (`bold` — насыщенное начертание или `normal` — обычная насыщенность), описательные значения относительно родителя (`bolder` — насыщеннее, чем шрифт родителя, или `lighter` — светлее, чем шрифт родителя) или числовые значения от 100 до 900 кратные 100 (значение 400 соответствует значению `normal`, а значение 700 – значению `bold`.

- `font-stretch` — задает ширину букв шрифта, что в итоге приводит к уплотнению или расширению текста. Может принимать следующие значения, которые расположены в порядке возрастания ширины: `ultra-condensed, extra-condensed, condensed, semi-condensed, normal, semi-expanded, expanded, extra-expanded, ultra-expanded`.

В листинге 2.12 приведен пример задания шрифта для блока `person`.

Листинг 2.12: Задание шрифта блока

```
person {
    font-family: Helvetica, Arial, sans-serif;
    font-size: x-large;
    font-style: italic;
    font-variant: small caps;
    font-weight: 900;
    font-stretch: semi-expanded
}
```

Задание свойств блока текста.

- `text-indent` — отступ первой строки блока. Устанавливается в заданных единицах, либо в процентах от ширины блока.

- `text-align` — выравнивание текста в блоке. Может принимать следующие значения: `center` — выравнивание по центру горизонтали блока, `justify` — выравнивание по ширине блока (чтобы добиться этого системы отображения обычно добавляют дополнительные пробелы к тем пробелам, что уже имеются в тексте), `left` — выравнивание по левому краю блока, `right` — выравнивание по правому краю блока.

- `text-decoration` — эффекты шрифта. Может принимать следующие значения: `blink` — устанавливает мигание текста, `line-through` — перечеркивает текст по середине горизонтали по всей ширине, `overline` — рисует линию над текстом, `underline` — рисует линию под текстом, `none` — отменяет все эффекты.

- `text-transform` — перевод в верхний/нижний регистры. Может принимать следующие значения: `capitalize` — первый символ предложения изменяет на заглавный, остальные на строчные (предложения разделяются знаком точки), `lowercase` — все символы блока заменяет на строчные (нижний регистр), `uppercase` — все символы блока изменяет на прописные (верхний регистр), `none` — отменяет изменения регистра.

- `white-space` — выполнение переноса строк. Может принимать следующие значения: `normal` — текст переносится системой отображения автоматически, `nowrap` — текст не переносится, `pre` — переносит так, как это было указано разработчиком документа, `pre-line` — переноси так, как это указано, но если текст не помещается в блок, то автоматически переносит его на следующую строку.

Задание свойств цвета. С помощью CSS можно задать следующие цвета:

- `color` — цвет самого текста.

- `background-color` — цвет фона за текстом.

- `border-color` — цвет видимого прямоугольника, окружающего текст.

Способы задания цвета следующие:

- именованные константы (aqua, red, green, blue и т.д.).
- компоненты цвета RGB (#FF0000).
- вызов функции `rgb()` с тремя параметрами в десятичной системе счисления (значения от 0 до 255), где параметры отвечают соответственно за красный, зеленый и синий цвета: (rgb(43,43,43)).
- вызов функции `rgb()` с тремя параметрами в процентах: (rgb(16.9%,16.9%,16.9%)).

§ 3. Форматирующие объекты (XSL-FO)

Технология XSL-FO (Extensible Stylesheet Language Formatting Objects) предназначена для более гибкого представления документов XML в различных форматах. В отличии от XSLT результат работы XSL-FO может быть представлен в текстовом формате (например, text, html, xml) и в бинарном формате (например, PDF, RTF, DOC). В то же время, XSL-FO представляет собой полноценное XML приложение, что позволяет использовать технологии XML для создания документа XSL-FO и его использования.

Документ XSL-FO представляет собой объединение данных и объектов форматирования, в том числе, шаблон страницы и форматирование содержимого. В результате, XSL-FO документ имеет достаточно большой объем и сложную структуру. Одним из способов облегчения работы является использование XSLT преобразования XML документа в результате которого мы получим готовый XSL-FO документ.

В документе XSL-FO данные представлены в виде потока блоков, которые во время обработки процессором XSL-FO заполняют макет страницы. Если очередной блок не помещается на страницу, то создается новая страница. Таким образом, размещение на странице блока данных будет определено после обработки XSL-FO документа.

3.1. Создание документа XSL-FO

Рассмотрим общую структуру документа XSL-FO на примере листинга 3.1. Документ должен содержать один корневой элемент `root` в котором описывается пространство имен XSL-FO. Рекомендуемый префикс `fo`.

Листинг 3.1: Простой XSL-FO документ

```
<?xml version="1.0"?>
<fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
  <fo:layout-master-set>
    <fo:simple-page-master master-name="only">
      <fo:region-body/>
    </fo:simple-page-master>
  </fo:layout-master-set>

  <fo:page-sequence master-reference="only">
    <fo:flow flow-name="xsl-region-body">
```

```
      <fo:block>Hello World!</fo:block>
    </fo:flow>
  </fo:page-sequence>
</fo:root>
```

Элемент `root` должен содержать обязательный дочерний элемент `layout-master-set` для описания шаблонов страниц (см. 3.2.) и как минимум один дочерний элемент `page-sequence` содержащий поток данных с форматированием содержимого (см. 3.4.). Объекты форматирования документа XSL-FO встраиваются в содержимое в виде соответствующих элементов и атрибутов.

С точки зрения XSL-FO, документ представляет собой последовательность страниц. Каждая страница представлена в виде набора областей. Область содержит часть потока данных в виде набора блоков. Таблица и список также представляет собой блок, помещаемый в область. Каждый блок может содержать либо набор вложенных блоков либо текстовое содержимое в виде набора строк. При этом, строка состоит из набора внутри строчных элементов (символы, сноски, математические выражения и др.).

Создание конечного документа выполняется с помощью специального компилятора (например, Apache FOP [6]). Пример команды компиляции PDF версии документа представлен в листинге 3.2.

Листинг 3.2: Компиляция PDF версии XSL-FO документа

```
fop simple.fo simple.fop
```

В связи со сложной структурой XSL-FO существует возможность разделения данных и форматирования: данные оформляются в виде XML документа, а форматирование в виде XSL преобразований (см. § 1.). В листинге 3.3 показан фрагмент преобразований XSL для создания XSL-FO документа.

Листинг 3.3: Фрагмент XSLT для создания XSL-FO документа

```
<xsl:template match="dish">
  <fo:block font-size="20pt"
    font-weight="bold" text-align="center">
    <xsl:apply-templates/>
  </fo:block>
</xsl:template>
```

В случае разделения данных и форматирования команда компиляции PDF документа будет выглядеть например как в листинге 3.4.

Листинг 3.4: Создание PDF из XML и XSLT

```
fop -xml data.xml -xsl transform.xsl -pdf report.pdf
```

3.2. Макет страницы

В стандарте технологии XSL-FO определен единственный вид страницы, описываемый элементом `simple-page-master`. Страница представляет собой прямоугольную форму с фиксированной шириной, высотой и полями со всех четырех сторон. Атрибут шаблона `master-name` позволяет задать связанное с макетом имя. Документ может содержать несколько макетов страниц, связанных с потоками, однако, один поток должен использовать один макет.

Общая структура страницы представлена на рис. 2. Страница имеет 4 регулируемых отступа:

- `margin-top` – верхний отступ страницы;
- `margin-bottom` – нижний отступ страницы;
- `margin-left` – левый отступ страницы;
- `margin-right` – правый отступ страницы.

Размер страницы (ширина `page-width` и высота `page-height`), поля (`padding*`) и отступы определяются с помощью соответствующих атрибутов.

Полезная площадь страницы поделена на 5 областей:

- `region-body` – основная область, обязательный при описании макета элемент;
- `region-before` – пред-область;
- `region-after` – после-область;
- `region-start` – начальная область;
- `region-end` – конечная область.

Местоположение областей определяется направлением текста (см. рис. 3). Основная область занимает всю полезную площадь страницы, а остальные области накладываются на основную. Для избежания

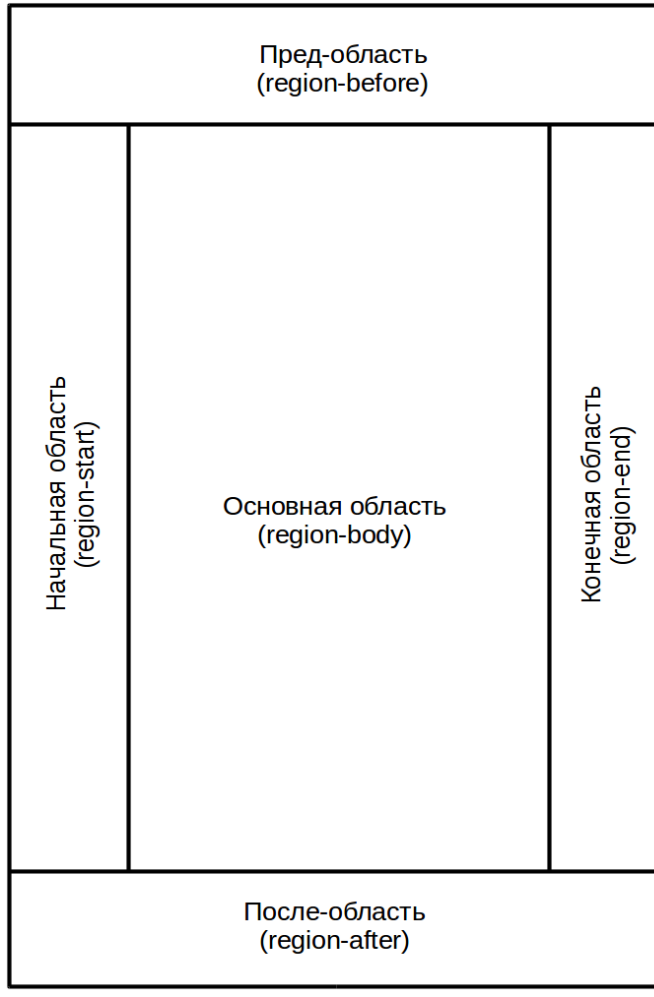

Рис. 2: Структура страницы XSL-FO

наложения необходимо задать размер граничной области с помощью атрибута `extent` и отступы основной области `margin-*`.

В листинге 3.5 показан пример создания страницы с запол-

Рис. 3: Местоположение областей в зависимости от направления текста

нением пред-области и основной области. С помощью атрибута `master-reference` для последовательности страниц выбран используемый шаблон.

Листинг 3.5: Шаблон страницы с несколькими областями

```
<?xml version="1.0"?>
<fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format"
  font-family="Arial">
  <fo:layout-master-set>
    <fo:simple-page-master master-name="only"
    page-height="12cm"
    page-width="12cm"
    margin-left="1cm">
      <fo:region-body margin-top="1cm"/>
      <fo:region-before extent="1cm"/>
    </fo:simple-page-master>
  </fo:layout-master-set>

  <fo:page-sequence master-reference="only">
    <fo:static-content flow-name="xsl-region-before">
      <fo:block>Пред область со сдвигом 1 см.</fo:block>
    </fo:static-content>
    <fo:flow flow-name="xsl-region-body">
```

```
      <fo:block>Основная область с отступом 1 см.</fo:block>
    </fo:flow>
  </fo:page-sequence>
</fo:root>
```

В последовательности страниц `page-sequence` используемый шаблон определяется с помощью атрибута `master-reference`. Последовательность страниц содержит необязательный элемент title титула последовательности, необязательные элементы статического содержимого `static-content` и один обязательный элемент потока данных `flow`. Для создания документов со сложным оформлением используется шаблон с набором мастер страниц. Шаблон оформляется как последовательность дочерних элементов внутри `page-sequence-master` и может состоять из следующих объектов:

- `single-page-master-reference` шаблон одной страницы;
- `repeatable-page-master-reference` повторное использование шаблона для нескольких страниц;
- `repeatable-page-master-alternatives` использование шаблона в зависимости от условий.

Шаблон одной страницы используется для вставки в документ страниц со специальным оформлением и ограниченным содержимым, например, титульной страницы. Пример шаблона буклета с именем `booklet` состоящего из 2 страниц представлен в листинге 3.6. Так как лицевая и оборотная сторона буклета оформляются одинаково, то оформление страницы вынесено в отдельный шаблон с именем `A4`, а при формировании шаблона одной страницы указана ссылка на оформление через атрибут `master-reference`. В примере использован атрибут у блока `page-break-before` для завершения формирования первой страницы и перехода к формированию второй.

Листинг 3.6: Использование шаблона одной страницы

```
<?xml version="1.0"?>
<fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format"
  font-family="Arial">
  <fo:layout-master-set>
    <fo:simple-page-master master-name="A4"
      page-width="297mm" page-height="210mm">
      <fo:region-body/>
```

```
    </fo:simple-page-master>
    <fo:page-sequence-master master-name="booklet">
      <fo:single-page-master-reference master-reference="A4"/>
      <fo:single-page-master-reference master-reference="A4"/>
    </fo:page-sequence-master>
  </fo:layout-master-set>

  <fo:page-sequence master-reference="booklet">
    <fo:flow flow-name="xsl-region-body">
      <fo:block>Лицевая сторона</fo:block>
      <fo:block page-break-before="always"/>
      <fo:block>Оборотная сторона</fo:block>
    </fo:flow>
  </fo:page-sequence>
</fo:root>
```

Шаблон в `repeatable-master-reference` используется до окончания потока. При необходимости можно ограничить число использований с помощью атрибута `maximum-repeats`. Условие использования шаблона в `repeatable-page-master-alternatives` определяется дочерним элементом `conditional-page-master-reference`. Последний имеет следующие атрибуты:

- `master-reference` ссылка на используемый шаблон;
- `page-position` страница для которой применяется шаблон: первая (значение `first`), последняя (`last`), внутренние (`rest`) или все (`any`);
- `odd-or-even` использование шаблона на четных (`even`), нечетных (`odd`) или на всех страницах (`any`);
- `blank-or-not-blank` использование шаблона на пустых (`blank`), не пустых (`not-blank`) иил на всех страницах (`any`).

В листинге 3.7 показан пример создания сложного оформления документа, состоящего из титульной страницы `title`, не более 10 повторов использования обычного шаблона `A4` и использование условных шаблонов.

Листинг 3.7: Документ со сложным оформлением

```
<?xml version="1.0"?>
<fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format"
  font-family="Arial">
```

```
  <fo:layout-master-set>
    <fo:simple-page-master master-name="title"
      page-width="210mm" page-height="297mm">
      <fo:region-body/>
    </fo:simple-page-master>
    <fo:simple-page-master master-name="A4"
      page-width="297mm" page-height="210mm">
      <fo:region-body/>
    </fo:simple-page-master>
    <fo:simple-page-master master-name="blank"
      page-width="210mm" page-height="210mm">
      <fo:region-body/>
    </fo:simple-page-master>
    <fo:page-sequence-master master-name="report">
      <fo:single-page-master-reference
        master-reference="title"/>
      <fo:repeatable-page-master-reference
        master-reference="A4" maximum-repeats="10"/>
      <fo:repeatable-page-master-alternatives>
       <fo:conditional-page-master-reference
        blank-or-not-blank="blank" master-reference="blank"/>
       <fo:conditional-page-master-reference
        blank-or-not-blank="not-blank" master-reference="A4"/>
      </fo:repeatable-page-master-alternatives>
    </fo:page-sequence-master>
  </fo:layout-master-set>

  <fo:page-sequence master-reference="report">
    <fo:flow flow-name="xsl-region-body">
      <fo:block>Это документ!</fo:block>
    </fo:flow>
  </fo:page-sequence>
</fo:root>
```

3.3. Структура блока

Блок `block` представляет собой прямоугольную область и состоит из 4 элементов (см. рис. 4):

- поле блока, определяется атрибутами `padding-*` (см. примеры в листинге 3.8);

Листинг 3.8: Примеры оформления полей блока

```
<fo:block padding-before="18pt"
```

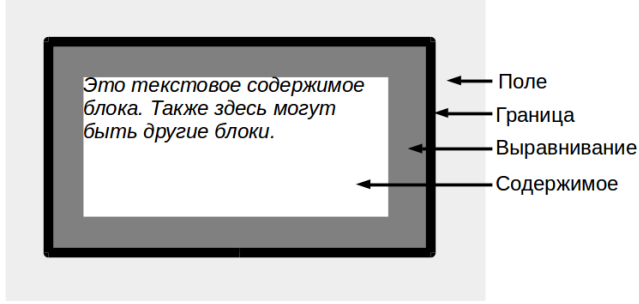

Рис. 4: Структура блока XSL-FO

```
            padding-bottom="18pt">
  Поля 18pt сверху и снизу.
</fo:block>

<fo:block padding="2cm">
  Поля 2 см. со всех сторон блока
</fo:block>

<fo:block padding="1cm 3cm">
 Поля 1 см. сверху и снизу, 3 см. по сторонам
</fo:block>

<fo:block padding="1cm 3cm 2cm">
 Поля 1 см. сверху, 3 см. по сторонам и 2 см. снизу
</fo:block>

<fo:block padding="1cm 3cm 2cm 4cm">
 Поля 1 см. сверху, 3 см. слева, 2 см. справа, 4 см. снизу
</fo:block>
```

- граница блока, определяется атрибутами `border-*` (см. примеры в листинге 3.9);

Листинг 3.9: Примеры оформления границ блока

```
<fo:block border-color="gray"
          border-style="groove"
          border-width="medium"
```

```
            text-align="center">
    Серая граница по всему блоку
  </fo:block>

  <fo:block border-top-color="black"
            border-top-style="solid"
            border-top-width="thick">
    Толстая рамка сверху
  </fo:block>

  <fo:block border-top="dashed 1pt #C00000"
            border-bottom="1pt dashed #C00000">
    Пунктирная граница сверху и снизу
  </fo:block>
```

- выравнивание внутри блока, определяется атрибутами `margin-*`;
- содержимое блока.

Размер блока определяется внешними границами его полей. Свойства блока определяется атрибутами. При формировании страницы блок обрабатывается аналогично элементу `DIV` из HTML или `display:block` из CSS. Большинство названий свойств блока идентично названиям из каскадных таблиц стилей CSS (см. § 2.). В листинге 3.10 показано сравнение оформлений блоковых элементов.

Листинг 3.10: Сравнение оформления CSS и XSL-FO

```
// оформление блока в CSS
dish {
  display: block;
  font-family: Helvetica, Arial, sans-serif;
  font-size: 20pt;
  font-weight: bold;
  text-align: center
}

<!-- оформление блока в XSL-FO -->
<fo:block font-family="Helvetica, Arial, sans-serif"
  font-size="20pt" font-weight="bold" text-align="center">
  Текст блока.
</fo:block>
```

3.4. Наполнение документа

Размещение содержимого. Существует два способа размещения содержимого в элементе `page-sequence` XSL-FO документа: потоковые данные (элемент `flow`) и статичное содержимое (элемент `static-content`).

Потоковые данные могут содержать неограниченное количество блоков данных, которые в процессе компиляции распределяются по страницам документа. Размещение элементов потоковых данных определяется с помощью атрибута `flow-name` и имеет следующие значения:

- `xsl-region-body` основная область;
- `xsl-region-before` пред-область;
- `xsl-region-after` после-область;
- `xsl-region-start` начальная область;
- `xsl-region-end` конечная область.

Существует ограничение на размещение элементов: не может быть более одного элемента в одной области одной последовательности. Пример потокового содержимого представлен в листинге 3.11.

Листинг 3.11: Потоковое содержимое

```
<fo:page-sequence master-reference="A4">
 <fo:flow flow-name="xsl-region-body">
  <fo:block font-size="20pt" font-family="serif"
    line-height="30pt">
   Hydrogen
  </fo:block>
  <fo:block font-size="20pt" font-family="serif"
    line-height="30pt" >
   Helium
  </fo:block>
 </fo:flow>
</fo:page-sequence>
```

Статичное содержимое ограничено в объеме и располагается на каждой странице документа. Статичное содержимое оформляется аналогично потоковым данным. В документе XSL-FO объявление статичного содержимого должно быть до объявления элемента `flow`.

Пример использования статичного содержимого представлен в листинге 3.12.

Листинг 3.12: Статичное содержимое

```
<fo:page-sequence master-name="A4">
 <fo:static-content flow-name="xsl-region-before">
  <fo:block>The Periodic Table</fo:block>
 </fo:static-content>
 <fo:flow flow-name="xsl-region-body">
  <fo:block font-size="20pt" font-family="serif"
    line-height="30pt">
   Hydrogen
  </fo:block>
 </fo:flow>
</fo:page-sequence>
```

Нумерация страниц. Нумерация страниц определяется с помощью следующих атрибутов элемента `page-sequence`.

- `initial-page-number` – номер первой страницы в последовательности страниц. Значением атрибута может быть любое число или одно из следующих ключевых слов: `auto` – следующий порядковый номер страницы или 1 если это первая страница; `auto-odd` – аналогично `auto`, только первая страница последовательности будет с нечетным номером; `auto-even` – аналогично `auto`, только первая страница последовательности будет с четным номером.

- `force-page-count` – ограничение на номер последней страницы. При необходимости вставляется пустая страница в конец последовательности. Значением атрибута может выступать одно из следующих ключевых слов: `auto` – номер последней страницы определяется ограничениями атрибута `initial-page-number`; `even` – общее число страниц четно; `odd` – общее число страниц нечетно; `end-on-even` – последняя страница с четным номером; `end-on-odd` – последня страница с нечетным номером; `no-force` отсутствие требований к номеру последней страницы.

- `format` – формат номера страницы (см. в § 1.).

- `letter-value` – интерпретация букв (см. в § 1.).

- `country` – код страны по стандарту RFC 1766.
- `language` – код языка по стандарту RFC 1766.
- `grouping-separator` – символ разделения групп (см. в § 1.).
- `grouping-size` – количество цифр в группе (см. в § 1.).

Размещение номера на странице выполняется с помощью элемента `page-number`.

Таблицы. Таблица представляет собой самостоятельный контейнер блоков и не требует упаковки в блоковый элемент. Таблица описывается с помощью следующих элементов:

- `table-and-caption` – контейнер для таблицы и подписи;
- `table-caption` – подпись к таблице;
- `table` – контейнер таблицы;
- `table-header` – заголовок;
- `table-footer` – окончание;
- `table-body` – тело;
- `table-column` – колонка;
- `table-row` – строка;
- `table-cell` – ячейка.

Корневым элементом может быть либо `table`, либо `table-and-caption` (содержит элементы `table` и `caption`). Элемент `table` содержит элементы `table-header`, `table-body` и `table-footer`. Тело таблицы (элемент `table-body`) содержит набор строк `table-row` с ячейками `table-cell`.

Пример таблицы представлен в листинге 3.13.

Листинг 3.13: Таблица в документе XSL-FO

```
<?xml version="1.0"?>
<fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
 <fo:layout-master-set>
  <fo:simple-page-master master-name="empty">
   <fo:region-body/>
```

```
  </fo:simple-page-master>
 </fo:layout-master-set>

 <fo:page-sequence master-reference="empty">
  <fo:flow flow-name="xsl-region-body">
   <fo:table border="solid" border-collapse="collapse">
    <fo:table-header>
     <fo:table-row>
      <fo:table-cell>
       <fo:block>Заголовок 1</fo:block>
      </fo:table-cell>
      <fo:table-cell>
       <fo:block>Заголовок 2</fo:block>
      </fo:table-cell>
     </fo:table-row>
    </fo:table-header>
    <fo:table-body>
     <fo:table-row>
      <fo:table-cell>
       <fo:block>Ячейка 1</fo:block>
      </fo:table-cell>
      <fo:table-cell>
       <fo:block>Ячейка 2</fo:block>
      </fo:table-cell>
     </fo:table-row>
    </fo:table-body>
   </fo:table>
  </fo:flow>
 </fo:page-sequence>
</fo:root>
```

Списки. Список представляет собой контейнер блоков и не требует упаковки в блочный элемент. Список описывается с помощью следующих элементов:

- `list-block` – контейнер списка;
- `list-item` – контейнер элемента списка;
- `list-item-label` – маркер списка;
- `list-item-body` – содержимое элемента списка.

Контейнер списка содержит набор контейнеров элемента списка. Каждый контейнер элемента списка содержит пару: маркер и содержимое.

По умолчанию, маркер и содержимое начинаются с левой стороны используемой области и накладываются друг на друга. Для устранения наложения используются следующие атрибуты (см. рис. 5):

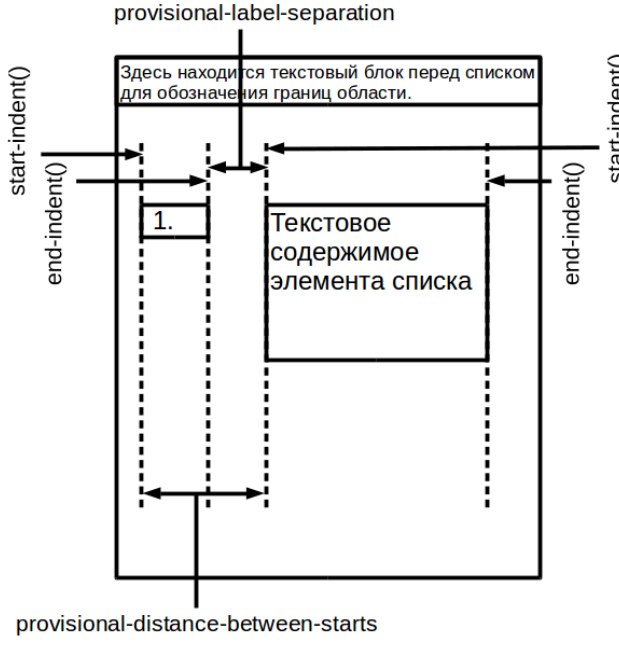

Рис. 5: Отступы в элементе списка XSL-FO

- `provisional-distance-between-starts` – расстояние между началом маркера и левой границей содержимого;
- `provisional-label-separation` – расстояние между окончанием маркера и левой границей содержимого;
- `start-indent` – в элементе `list-item-label` атрибут обозначает начало маркера, а в элементе `list-item-body` — левую границу содержимого;

- `end-indent` – в элементе `list-item-label` атрибут обозначает конец маркера, а в элементе `list-item-body` — правую границу содержимого;

Для вычисления местоположения начала и окончания маркера и границ содержимого существуют соответствующие функции `label-start()`, `label-end()`, `body-start()`, `body-end()`.

Пример оформления списка с вычислением местоположений маркеров и содержимого представлен в листинге 3.14.

Листинг 3.14: Список в документе XSL-FO

```
<?xml version="1.0"?>
<fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
 <fo:layout-master-set>
  <fo:simple-page-master master-name="only">
   <fo:region-body/>
  </fo:simple-page-master>
 </fo:layout-master-set>

 <fo:page-sequence master-reference="only">
  <fo:flow flow-name="xsl-region-body">
   <fo:list-block provisional-distance-between-starts="0.3cm"
provisional-label-separation="0.15cm">
    <fo:list-item>
     <fo:list-item-label end-indent="label-end()">
      <fo:block>*</fo:block>
     </fo:list-item-label>
     <fo:list-item-body start-indent="body-start()">
      <fo:block>Первый элемент списка</fo:block>
     </fo:list-item-body>
    </fo:list-item>
    <fo:list-item>
     <fo:list-item-label>
      <fo:block>2)</fo:block>
     </fo:list-item-label end-indent="label-end()">
     <fo:list-item-body start-indent="body-start()">
      <fo:block>Actinium</fo:block>
     </fo:list-item-body>
    </fo:list-item>
   </fo:list-block>
  </fo:flow>
 </fo:page-sequence>
</fo:root>
```

Вне-строчные форматирующие объекты. Существуют следующие вне-строчные форматирующие объекты:

- `float` – плавающий блок;
- `footnote` – сноска;
- `footnote-body` – текст сноски.

Вне-строчные форматирующие объекты вставляются в потоковое содержимое страницы в зависимости от наличия свободного места и условий. Таким образом, вне-строчный форматирующий объект может не находиться в результирующем файле рядом с определенными по соседству объектами. Например, плавающий блок отображается при наличии места на текущей странице или на следующей, а блоки перед и после плавающего блока отображаются последовательно.

Пример оформления сноски представлен в листинге 3.15. Важно отметить, что документ XSL-FO не отслеживает правильность нумерации и цитирования сносок. Это можно сделать с помощью использования элемента `number` в шаблоне документа XSLT.

Листинг 3.15: Сноска в документе XSL-FO

```
<?xml version="1.0"?>
<fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
 <fo:layout-master-set>
  <fo:simple-page-master master-name="empty">
   <fo:region-body/>
  </fo:simple-page-master>
 </fo:layout-master-set>

 <fo:page-sequence master-reference="empty">
  <fo:flow flow-name="xsl-region-body">
   <fo:block>Привет мир!
    <fo:footnote>
     <fo:inline font-size="smaller" vertical-align="super"> *
     </fo:inline>

     <fo:footnote-body font-size="smaller">
      <fo:block>
       <fo:inline font-size="smaller" vertical-align="super">
          *)
       </fo:inline>
       Здесь идет текст сноски. Сноска размещается внизу страницы.
      </fo:block>
```

```
      </fo:footnote-body>
     </fo:footnote>
  </fo:block>
  </fo:flow>
 </fo:page-sequence>
</fo:root>
```

Линейки и пунктиры. В XSL-FO под линейкой понимается блок, содержащий горизонтальную линию, а под пунктиром – внутристрочный элемент "раздвигающий" по краям страницы блоки. Аналогом линейки является элемент `hr` HTML разметки. Пунктиры часто используются в оглавлениях раздвигая текст и номера страница.

Линейки и пунктиры создаются элементом `leader`. Оформление линейки и пунктира задается с помощью следующих атрибутов.

- `leader-alignment` – выравнивание линейки. Допускаются следующие значения атрибута: `reference-area`, `page`, `none` и `inherit`.
- `leader-length` – длина линейки, например, `12mm`.
- `leader-pattern` – шаблон оформления. Атрибут может принимать следующие значения: `space` (пробелы), `rule` (линейка), `dots` (точки), `use-content` или `inherit`. В случае значения `use-content` используются символы из содержимого элемента `leader`.
- `leader-pattern-width` – ширина шаблона оформления.
- `rule-style` – стиль линейки. Атрибут может принимать значения аналогичные значениям элемента CSS `border-style`.
- `rule-thickness` – толщина линейки.

В листинге 3.16 показано создание линейки и пунктира.

Листинг 3.16: Линейки и пунктиры в документе XSL-FO

```
<?xml version="1.0"?>
<fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
 <fo:layout-master-set>
  <fo:simple-page-master master-name="only">
   <fo:region-body/>
  </fo:simple-page-master>
```

```
 </fo:layout-master-set>

 <fo:page-sequence master-reference="only">
  <fo:flow flow-name="xsl-region-body">
   <fo:block>
    <fo:leader leader-length="7.5in" leader-pattern="rule"
       rule-thickness="2pt" color="green" />
   </fo:block>
   <fo:block text-align-last="justify">
     Hello
     <fo:leader leader-pattern="dots" />
     World!
   </fo:block>
  </fo:flow>
 </fo:page-sequence>
</fo:root>
```

Графика. Графические изображения встраиваются в документ в зависимости от их типа:

- XML-графика (например, SVG или MathML) встраивается с помощью элемента `instream-foreign-object`;
- остальные типы изображений (например, JPEG ли PNG) встраиваются с помощью элемента `external-graphic`.

Основное различие заключается в способе обработки изображения: XSL-FO компилятор строит изображение из содержимого элемента `instream-foreign-object` и использует существующее изображение в элементе `external-graphic`.

Элемент `external-graphic` работает аналогично элементу `img` HTML разметки и не содержит тела. Путь к изображению (URI) определяется с помощью атрибута `src`. Элемент `external-graphic` относится к классу внутри-строчных элементов, для оформления изображение в виде блока можно заключить его в элемент `block`.

Элемент `instream-foreign-object` содержит изображение, описанное в формате XML. Способ построения изображения компилятор определяет по указанному пространству имен. Успешность построения изображения зависит от поддерживаемых форматов используемым компилятором.

Тип изображения определяется с помощью атрибута `content-type`. Допускается использование MIME типов с префиксом

`content-type` (например,`content-type="content-type:image/png"`) или префикс пространства имен с префиксом `namespace-prefix` (например, `xmlns:svg="http://www.w3.org/2000/svg" content-type="namespace-prefix:svg"`).

Для указания размера изображения доступны следующие атрибуты.

- `content-width` – ширина изображения.
- `content-height` – высота изображения.
- `width` – ширина зоны для размещения изображения.
- `height` – высота зоны для размещения изображения.
- `scaling` – масштабирование изображения. Если изображение не помещается в прямоугольную зону, то изображение будет масштабироваться. Масштабирование может быть пропорциональным (значение `uniform`) или под заданный размер, т.е. искаженным (значение `non-uniform`).
- `scaling-method` – метод масштабирования. Допускаются следующие значения: `auto` – автоматическое определение метода, `integer-pixels` – использование целых значений соотношений размеров исходного и результирующего изображений или `resample-any-method` – использование сглаживания.

Ссылки. В XSL-FO ссылки определяются с помощью элемента `basic-link`. Ссылка является внутри-строчным форматирующим объектом, предназначена как для навигации внутри документа, так и для открытия внешних ресурсов.

Содержимое элемента определяет отображение ссылки в исходном документе. Элемент поддерживает следующие атрибуты.

- `external-destination` – ссылка на удаленный ресурс. Ресурс определяется с помощью значения атрибута в виде URI.
- `internal-destination` – ссылка на внутренний объект. Объект определяется с помощью значения атрибута в виде идентификатора `ID`.
- `indicate-destination` – выделение целевого ресурса при переходе (значения `true` или `false`).

- `show-destination` – способ отображения целевого ресурса. Допускаются значения `new` (новое окно для отображения целевого ресурса) или `replace` (замена в текущем окне).
- `destination-placement-offset` – смещение целевого ресурса вниз. Длина смещения определяется значением атрибута.
- `target-presentation-context` – отображение части целевого ресурса. Значением может быть или URI определяющий отображаемую часть целевого ресурса (например, с помощью XPointer) или `use-target-processing-context` (часть определяется с помощью `external-destination` вместе с атрибутом `target-processing-context`).
- `target-processing-context` – базовый URI целевого ресурса. В этом случае значение атрибута `external-destination` будет рассматриваться относительно базового URI.
- `target-stylesheet` – используемая для отображения целевого ресурса таблица стилей. Значением атрибута может выступать URI до таблицы стилей.

В листинге 3.17 показано создание ссылок на внутренний и внешний ресурсы.

Листинг 3.17: Ссылки в документе XSL-FO

```
<?xml version="1.0"?>
<fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
 <fo:layout-master-set>
  <fo:simple-page-master master-name="only">
   <fo:region-body/>
  </fo:simple-page-master>
 </fo:layout-master-set>

 <fo:page-sequence master-reference="only">
  <fo:flow flow-name="xsl-region-body">
   <fo:block>
    <fo:basic-link internal-destination="textID">
     Ссылка на внутренний блок</fo:basic-link>
   </fo:block>
   <fo:block>
    <fo:basic-link
     external-destination="http://cs.karelia.ru/~kulakov/
     courses/xml/" show-destination="new">
```

```
      Ссылка на внешний ресурс</fo:basic-link>
    </fo:block>

    <fo:block page-break-before="always"/>
    <fo:block id="textID">Целевой блок для внутренней
     ссылки</fo:block>
  </fo:flow>
 </fo:page-sequence>
</fo:root>
```

Дополнительные атрибуты и свойства. XSL-FO поддерживает большое количество различных свойств для оформления документа. Ниже приведены некоторые из них.

Атрибут `id` имеет тип ID, может применяться к любому элементу и позволяет использовать идентификацию элемента внутри документа. При генерации XSL-FO документа с помощью XSLT преобразований полезным будет использование функции `generate-id()`.

Разрыв страницы может быть выставлен вручную с помощью следующих атрибутов элемента `block`.

- `keep-with-next` – мера усилий для сохранения текущего объекта на одной странице вместе со следующим. Значением атрибута может быть число или ключевые слова `auto` (на усмотрение компилятора) или `always` (максимальное усилие).
- `keep-with-previous` – мера усилий для сохранения текущего объекта на одной странице вместе с предыдущим. Варианты значения атрибута аналогичны предыдущим.
- `keep-together` – мера усилий для сохранения объекта на одной странице. Варианты значения атрибута аналогичны предыдущим.
- `break-before` – выполнить разрыв перед объектом. Значением атрибута могут быть следующие ключевые слова: `column` (разрыв колонки), `page` (разрыв страницы), `even-page` (разрыв страницы и перенос на следующую четную страницу), `odd-page` (разрыв страницы и перенос на следующую нечетную страницу) или `auto` (разрыв на усмотрение компилятора).
- `break-after` – выполнить разрыв после объекта. Варианты значения атрибута аналогичны предыдущим.

Переносы строк регулируются следующими атрибутами.

- `hyphenate` – использование автоматических переносов (значение `true`) или запрет автоматических переносов (`false`).
- `hyphenation-character` – символ обозначения переноса.
- `hyphenation-keep` – определение мест, где допускаются переносы. Значением атрибута могут быть `none`, `column`, `page` или `inherit`.
- `hyphenation-ladder-count` – количество идущих подряд строк с переносами.
- `hyphenation-push-character-count` – минимальное количество перенесенных символов слова для расстановки автоматических переносов.
- `hyphenation-remain-character-count` – минимальное количество символов предшествующих переносу для расстановки автоматических переносов.

Абзацные отступы регулируются следующими атрибутами.

- `start-indent` – смещение всех срок абзаца от начального края (т.е. для русского и английского языков – слева).
- `end-indent` – смещение всех строк абзаца от конечного края.
- `text-indent` – смещение первой (красной) строки абзаца.
- `last-line-end-indent` – смещение последней строки.

Приложение. Варианты практических заданий

Задание 1. “XSL”

- Создать XML документ в любом текстовом или специализированном редакторе содержащий вопросы и ответы к ним для проверки знаний по темам XML (5 вопросов) и XSL (5 вопросов).
- Дополнить XML документ из задачи №1 темой XSL с 5 вопросами;
- Используя XSL преобразования сформировать опросник и проверочный документ (с правильными ответами) в форматах HTML и XML (1 случайный вопрос из каждой темы);
- Реализовать опросник с сохранением результатов ответов пользователей в отдельном файле;
- Используя XSL преобразования сформировать отчет о результатах тестирования.

Задание 2. “CSS”

- Дополнить XML документ из задания №1 темой CSS с 5 вопросами.
- С помощью CSS оформить XML документ следующим образом:
  - Вопросы представлены в виде списка.
  - Для каждого вопроса представлены варианты ответов в виде вложенного списка с выравниванием по левому краю.
  - Правильные ответы отмечены курсивным и полужирным начертанием зеленого цвета для текста, размер которого должен быть на 25% больше остального текста.
  - Сформировать таблицу, состоящую из двух колонок: первая колонка — тема, вторая колонка — количество вопросов по данной теме. Таблица должна содержать заголовок

("Тема", "Количество вопросов"), последняя строка должна быть оформлена в виде подвала и первая колонка содержать текст "Итого:", а во второй колонке должно быть количество всех вопросов.

Задание 3. "XSL-FO"

При выполнении данного задания рекомендуется использовать `Apache FOP` [6].

- Дополнить XML документ из задания №2 темой XSL-FO с 5 вопросами.
- Сформировать PDF файл "Вопросы по темам XML содержащий следующую информацию:
  - Титульная страница с шапкой документа (университет, факультет), автором документа, названием документа, логотипом(ами);
  - Содержание/оглавление;
  - Перечень вопросов с разбиением на темы и вариантами ответов;
  - Перечень правильных ответов в отдельной секции в конце документа с разбиением на темы.
- Документ должен содержать хотя бы одно графическое изображение, номер страницы в нижнем колонтитуле (кроме титульной), краткое название документа в верхнем колонтитуле (кроме титульной). Документ должен быть оформлен в книжном формате (правые и левые страницы разворота).

Список литературы

[1] Extensible Markup Language [Электронный ресурс]. 2005. Режим доступа: http://www.w3.org/xml/

[2] XSL Transformations (XSLT) [Электронный ресурс]. 1999. Режим доступа: http://www.w3.org/TR/xslt

[3] XML Path Language (XPath) [Электронный ресурс]. 1999. Режим доступа: http://www.w3.org/TR/xpath/

[4] XML specification DTD [Электронный ресурс]. 2002. Режим доступа: http://www.w3.org/2002/xmlspec/dtd/2.10/xmlspec.dtd

[5] Cascading Style Sheets Level 2 Revision 1 (CSS 2.1) Specification [Электронный ресурс]. 2014. Режим доступа: http://www.w3.org/TR/xslt

[6] The Apache FOP Project [Электронный ресурс]. 2013. Режим доступа: http://xmlgraphics.apache.org/fop/

Учебное издание

Кулаков Кирилл Александрович;
Димитров Вячеслав Михайлович

Технологии XML. Часть II. Преобразование данных

Учебное пособие для студентов математического факультета

Публикуется в авторской редакции
Компьютерная верстка *В. М. Димитрова*

Подписано в печать 09.10.15. Формат 60×84 1/16
Бумага офсетная. 3 уч.-изд. л. Тираж 100 экз. Изд. №134

Отпечатано в типографии Издательства ПетрГУ
185910, г. Петрозаводска, пр. Ленина, 33