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 I. Организация данных

Учебное пособие для студентов математического факультета

Петрозаводск
Издательство ПетрГУ
2014

ББК 32.973
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 г.г.

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. — Петрозаводск : Изд-во ПетрГУ, 2014.

ISBN 978-5-8021-2139-9
Ч. 1. Организация данных. — 2014. — 68 с.
ISBN 978-5-8021-2138-2

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

ББК 32.973
УДК 004

ISBN 978-5-8021-2138-2 (Ч. 1.)
ISBN 978-5-8021-2139-9

Содержание

Введение

В современном информационном поле взаимодействие между приложениями или сервисами является одной из актуальных проблем разработки программного обеспечения. Одним из способов решения проблемы обмена данными является организация этих данных с помощью языка разметки XML (Extensible Markup Language) [1].

Основной целью разработчиков XML было создание простого и легко используемого формата для представления данных. В XML данные организованы в формате, понятном как для человека, так и для обработки автоматизированными средствами. И, хотя основной упор был сделан на обработке документов, XML широко используется для представления различных структур данных. Непосредственным конкурентом языка разметки XML является более компактный текстовый формат JSON [2].

XML широко используется на практике. Множество форматов документов используют XML синтаксис, в том числе RSS, Atom, SOAP, XHTML и др. Офисные пакеты, включая Microsoft Office (формат Office Open XML), OpenOffice.org и LibreOffice (формат OpenDocument), построены на основе XML. Протокол обмена мгновенными сообщениями XMPP основывается на XML. Следует также добавить, что XML включен в перечень многоцелевых расширений интернет-почты (MIME): *application/xml* или *text/xml*.

Данное учебное пособие представляет вводный курс для использования языка XML на практике и освещает следующие темы: правила построения XML документов, модель документа (схемы DTD и XSD), организация связей между документами (XLink), языки запросов по XML документам (XPath, XPointer и XQuery).

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. Extensible Markup Language (XML)

1.1. Структура XML документа

XML документ должен иметь строго определенную структуру, задаваемую правилами синтаксиса языка. *Согласованным* или *корректным (well-formed)* документом называют документ, который соответствует спецификации XML. Если документ не соответствует спецификации языка, то он не может считаться XML документом.

Кроме согласованности для ряда задач от XML документа требуется также учитывать дополнительные ограничения предметной области, описываемые с помощью *модели* документа (см. § 2.). *Действительным (valid)* XML документом называют согласованный документ, соответствующий всем ограничениям, определенным в модели XML документа.

Для разбора структуры XML документа будем использовать пример, представленный в листинге 1.1. XML документ состоит из *пролога* и *тела*.

Листинг 1.1: Пример документа XML

```
1  <?xml version="1.0" encoding="KOI8-R" standalone="yes"?>
2  <library>
3      <book>
4          <title lang="ru" type="full">Java</title>
5          <author>Брюс Эккель</author>
6          <publisher>Питер</publisher>
7      </book>
8      <book>
9          <title>Читаемый код</title>
10         <author>Дастин Босуэлл</author>
11         <publisher>Питер</publisher>
12     </book>
13 </library>
```

Первая строка листинга 1.1 представляет собой пролог, определяющий параметры документа: версию стандарта языка XML (`version`), кодировку документа (`encoding`) и автономное использование (`standalone`). Также пролог включает объявления типа документа (DTD, см. § 2.) и инструкции по обработке. Пролог является необязательной

частью документа, но требуется стандартом.

Версия стандарта языка XML является обязательным элементом пролога. В примере 1.1 указана версия 1.0. Также существует версия 1.1 в которой добавлена, в том числе, поддержка Unicode, однако в большинстве случаев для работы с XML документом достаточно использования версии 1.0.

В стандарте языка XML отмечено, что по умолчанию используются кодировки Unicode UTF-8 и UTF-16. Если используемая кодировка отличается от кодировки по умолчанию, то она указывается в прологе. В листинге 1.1 в качестве кодировки указана кодировка "KOI8-R".

Указание автономности в прологе необходимо в том случае, если документ содержит данные из внешних источников, например, объявление сущностей в другом файле (см. п. 2.5.). Значение автономности может принимать два значения "yes" и "no", если пролог не содержит определение автономности, то по умолчанию считается значение "no".

Начиная со второй строки листинга 1.1 идет тело XML документа. Тело представляет собой набор иерархически организованных *элементов*. Элемент состоит из начального тега, содержимого и конечного тега. Начальный и конечный теги заключаются в угловые скобки с тем отличием, что для конечного тега после открывающей угловой скобки требуется добавить символ прямого слеша "/". Между скобками указывается имя тега, которое также является типом элемента. Предъявляются следующие требования к имени тега:

- может начинаться с буквы или символа подчеркивания ("_");
- может содержать буквы, цифры и символы подчеркивания ("_"), дефиса ("-") и точки (".").

Примеры тегов из листинга 1.1: начальных `<library>`, `<title>` и конечных `</library>`, `</title>`.

Помимо имени начальный тег может содержать атрибуты и их значения. В листинге 1.1 приведен пример тега с именем `title` и двумя атрибутами: `lang` и `type`. Значением атрибута `lang` является строка "`ru`", а атрибута `type` — "`full`".

Атрибут формируется в виде пары имя атрибута и значение атрибута. К имени атрибута предъявляются те же требования, что и к имени тега, а значение атрибута должно выполнять следующие правила:

- представляет собой строку, заключенную в одинарные или двойные кавычки;
- внутри строки не может быть той же кавычки, т.е. если значение атрибута заключено в двойные кавычки, то помещение ее в строку не допускается;
- может содержать ссылку на символ или примитив (см. п. 2.5.);
- не может содержать символ “<”;
- не может содержать символ “&”, кроме случая использования сущности (см. п. 2.5.).

Спецификация XML определяет смысл некоторых атрибутов, также называемых зарезервированными, использование которых предполагается с данным указанным смыслом. Перечислим их:

- `xml:lang` — определение языка, на котором представлены данные элемента.
- `xml:space` — определяет обработку пробельных символов и может принимать два значения: `preserve` — сохраняет все пробельные символы в том виде, в каком они представлены в элементе, `default` — пробельные символы обрабатываются согласно правилам принятым для данного конкретного анализатора XML документа. В том случае, если указано какое-либо другое значение, то анализатор может либо указать на ошибку, либо проигнорировать данный атрибут.
- `xml:link` — элемент-ссылка (см. п. 5.1.).
- `xml:attribute` — переназначение атрибута для XLink.

Стоит заметить, что атрибуты не предоставляют каких-либо дополнительных возможностей, так как всю информацию, заключенную в атрибутах, можно представить во вложенных элементах. Не существует общепризнанных правил для использования атрибутов или элементов, однако, можно придерживаться следующего: использовать атрибуты только в том случае, если информация, которую они заключают, не является частью смысловых данных документа. Например, атрибуты `lang` и `type` из листинга 1.1 не несут смысловой нагрузки,

но могут быть полезны при обработке XML документа средствами синтаксического анализа.

Каждый элемент может содержать как данные, так и вложенные элементы. Например, элемент `library` содержит вложенные элементы `book`, а содержимое элемента `title` представлено в виде строки из символов.

Данные представляют собой любые символы за исключением символов "<", "&" и последовательности символов "]]>". В случае, когда в значение элемента или атрибута требуется вставить запрещенные символы можно воспользоваться сущностями (см. п. 2.5.). Например, символ ">" можно заменить на предопределенную сущность ">".

Также можно с помощью специального элемента *CDATA* явно указать данные. Этот элемент говорит средствам синтаксического анализа о том, что далее следуют данные, которые не требуется обрабатывать до завершающей последовательности символов "]]>". Например, в листинге 1.2 элемент *CDATA* используется для размещения примеров кода на языке HTML.

Листинг 1.2: Данные в формате CDATA

```
1 <example><![CDATA[<b>Жирный</b> текст]]></example>
```

В листинге 1.3 представлен вариант решения проблемы, когда внутри элемента CDATA требуется использовать последовательность символов "]]>".

Листинг 1.3: Последовательность "]]>" в формате CDATA

```
1 <![CDATA[ ]] ]]><![CDATA[>]]>
```

Также допускается элемент без данных и вложенных элементов, который называется *пустым элементом*. В листинге 1.4 на первой строке приведен пример такого элемента, а на второй — его сокращенная запись.

Листинг 1.4: Пустой тег

```
1 <hr></hr>
2 <hr/>
```

1.2. Правила построения XML документа

При создании XML документа необходимо соблюдать базовые правила его построения.

- В XML документе допускается использование элементов с одинаковым именем вне зависимости от их взаимного расположения. Смысловое значение каждого из элементов определяется на этапе обработки документа.
- Элементом верхнего уровня (корневым элементом) может быть только один элемент. В листинге 1.5 представлен некорректный документ, т.к. содержит два элемента верхнего уровня *<book>*.

Листинг 1.5: Некорректный документ

```
1 <?xml version="1.0"?>
2 <book></book>
3 <book></book>
```

- Не допускается частичное пересечение содержимого элементов. Например, в листинге 1.6 представлен некорректный документ, в котором выполнено частичное пересечение типов элементов `book` и `magazine`.

Листинг 1.6: Вложение упорядоченным образом

```
1 <?xml version="1.0"?>
2 <library>
3     <book>
4     <magazine>
5     </book>
6     </magazine>
7 </library>
```

- В отличии от HTML в XML для каждого элемента обязательным является наличие начального и конечного тегов. Исключением является сокращенная версия пустого тега.
- Имя типа элемента в начальном теге должно в точности соответствовать имени типа элемента в конечном теге. Также следует учитывать, что имена тегов чувствительны к регистру. В

листинге 1.7 представлен некорректный документ, в котором начальный тег `Book` и конечный тег `book` представляют собой различные имена.

Листинг 1.7: Регистр тегов

```
1 <?xml version="1.0"?>
2 <Book></book>
```

- Имена атрибутов должны быть уникальны в пределах одного элемента. Не допускается использование в одном элементе нескольких атрибутов с одинаковым именем.
- Имена элементов, содержимое элементов, имена атрибутов и содержимое атрибутов должны соответствовать описанным выше требованиям.

1.3. Пространство имен

Одним из основополагающих понятий XML является понятие пространства имен, которое определено в отдельной спецификации, существующей на данный момент в третьей редакции от 8 декабря 2009 года [3]. Одной из причин для введения пространства имен стала необходимость отличать смысл элементов с одним и тем же типом.

Предположим, что в документе с описанием книг из библиотеки тип элемента `date` описывает дату подписания в печать книги, а в документе с описанием книг в книжном магазине — дату поступления книги на склад магазина. В листинге 1.8 представлен один из этих документов.

Листинг 1.8: Описание книги

```
1 <?xml version="1.0"?>
2 <library>
3     <book>
4         <title>Философия Java</title>
5         <author>Брюс Эккель</author>
6         <date>2013-02-28 10:53:20</date>
7     </book>
8 </library>
```

Как можно заметить, без дополнительной информации невозможно определить смысловое значение типа элемента `date` и принадлежность XML документа. Используя пространство имен, можно сопоставить этому элементу смысловое значение, как это сделано в листинге 1.9.

Листинг 1.9: Использование пространства имен

```
1 <?xml version="1.0"?>
2 <library xmlns:lib="http://example.com/2013/Library">
3     <lib:book>
4         <lib:title>Философия Java</lib:title>
5         <lib:author>Брюс Эккель</lib:author>
6         <lib:date>2013-02-28 10:53:20</lib:date>
7     </lib:book>
8 </library>
```

Использование пространства имен состоит из двух частей: объявления пространства и указание принадлежности элемента пространству. Пространство имен объявляется с помощью специального атрибута *xmlns*, содержащего обязательное значение унифицированного идентификатора ресурса (Uniform Resource Identifier, URI), формат которого определен в RFC3986 [4]. Также можно через двоеточие указать в имени атрибута необязательное значение префикса пространства. В листинге 1.9 пространство имен объявлено для типа элемента `library`, где URI является "*http://example.com/2013/Library*", а префиксом пространства – "*lib*".

Стоит особо отметить, что URI не должен обязательно быть адресом существующего в сети Интернет ресурса. Синтаксические анализаторы XML не обращаются по этому URI за какой-либо дополнительной информацией, предназначенной для последующей обработки документа. Разработчики спецификации XML предполагали, что информация по данному URI должна предназначаться для человека и разъяснять ему все особенности и назначение данного пространства имен. При таком подходе становится понятно, что все URI пространств имен должны быть уникальны.

Принадлежность элемента пространству указывается через двоеточие в типе элемента с помощью объявленного префикса, например, `<lib:date></lib:date>`. При этом стоит учитывать, что префикс должен присутствовать как в начальном теге, так и в конечном.

Объявленное пространство можно использовать только для элемента, в котором оно объявлено, и для всех вложенных в него элементов. Это свойство используется при создании пространства имен по умолчанию, для которого в объявлении не указывается префикс. В этом случае элемент, в котором объявлено пространство по умолчанию, и все вложенные в него элементы будут принадлежать этому пространству имен (листинг 1.10).

Листинг 1.10: Пространство имен по умолчанию

```
1  <?xml version="1.0"?>
2  <library xmlns="http://example.com/2013/Library"
3    xmlns:shop="http://example.com/2013/BookShop">
4      <book>
5          <title>Философия Java</title>
6          <author>Брюс Эккель</author>
7          <date>2013-02-28 10:53:20</date>
8          <shop:date>2013-03-19 15:10:00</shop:date>
9      </book>
10 </library>
```

Ничего не запрещает пользователю использовать несколько пространств имен для одного элемента. Пример такого документа приведен в листинге 1.11, в котором для описания книги используется одинаковый тег `date`, принадлежащий разным пространствам имен, что делает документ более понятным.

Листинг 1.11: Несколько пространств имен

```
1  <?xml version="1.0"?>
2  <lib:library
3      xmlns:lib="http://example.com/2013/Library"
4      xmlns:shop="http://example.com/2013/BookShop">
5      <lib:book>
6          <lib:title>Философия Java</lib:title>
7          <lib:author>Брюс Эккель</lib:author>
8          <lib:date>2013-02-28 10:53:20</lib:date>
9          <shop:date>2013-03-19 15:10:00</shop:date>
10     </lib:book>
11 </lib:library>
```

§ 2. Определение типа документа (DTD)

Автоматизированная обработка XML документа (например, извлечение данных из элементов структуры документа или значений атрибутов) с помощью программной системы требует от создателя документа следовать жестким ограничениям в использовании структуры элементов, имен тегов, атрибутов и прочего. Некорректное описание структуры XML документа может значительно искажать результаты обработки. При большом объеме данных поиск ошибки в таком случае превращается в трудоемкую задачу.

Для избежания появления в документе некорректной структуры было введено понятие модели XML документа.

2.1. Модель XML документа

Процесс формального определения структуры документа называется моделированием документа. Модель XML документа определяет следующие положения:

- имена тегов, разрешенные к использованию в XML документе;
- структура документа (иерархия элементов);
- атрибуты и возможность их применения для элементов;
- дополнительные механизмы для управления моделью.

При наличии модели XML документа специальный анализатор может проверить действительность документа и в случае несоответствия документу модели предпринять соответствующие шаги.

Однако, создание и использование модели XML документа влечет за собой ряд сложностей:

- дополнительные затраты на сопровождение модели (в том случае если меняется структура XML документа, то эти изменения помимо самих документов должны быть также отражены в модели XML документа);
- замедление работы (необходимо дополнительное время для проверки документа на соответствие модели);

- ограниченный набор допустимых элементов и атрибутов (это может быть проблемой в динамически развивающихся предметных областях);
- возможные проблемы с пространством имен.

Таким образом, необходимость создания и использования модели XML документа зависит от решаемых задач, но в большинстве случаев, когда к работе с XML документами подключается человек, использование модели XML документа является оправданным.

2.2. Элементы DTD

Одной из реализаций модели является модель DTD (Document Type Definition) [5], или схема DTD, которая позволяет реализовывать все положения, описанные в п. 2.1.

Схема DTD может быть внешней или внутренней. Внешняя схема находится в отдельном файле и подключается с помощью объявления “DOCTYPE”. Пример подключения внешней схемы DTD представлен в листинге 2.1.

Листинг 2.1: Подключение внешней схемы DTD

```
1 <!DOCTYPE book SYSTEM
2   “http://www.library.org/dtd/book.”dtd>
3 <book>
4 </book>
```

Имя схемы DTD, указанное после ключевого слова “DOCTYPE”, должно соответствовать имени корневого элемента. Ключевые слова SYSTEM и PUBLIC определяют способ поиска файла со схемой DTD:

- при указании SYSTEM анализатор открывает файл со схемой по указанному URI;
- при указании PUBLIC анализатор выполняет поиск схемы по внутренней базе данных.

При указании ключевого слова PUBLIC так же можно указать резервный URI для случая, когда используется специализированный анализатор или когда схема не найдена во внутренней базе данных.

Внутренняя схема DTD описывается непосредственно в XML документе внутри объявления “DOCTYPE”. При этом допускается совмещение внешней и внутренней схемы DTD. Пример подключения с использованием резервного URI и совмещения внутренней и внешней схем представлен в листинге 2.2. В этом примере анализатор будет выполнять поиск схемы DTD по идентификатору “world/book.dtd”, а в случае отсутствия схемы в базе данных откроет схему по URI “http://www.library.org/dtd/book.dtd”.

Листинг 2.2: Совмещение внутренней и внешней схем DTD

```
1 <!-- DTD -->
2 <!DOCTYPE book PUBLIC "world/book.dtd" "http://www.
      library.org/dtd/book.dtd"
3 [
4 <!-- содержание внутренней схемы DTD -->
5 ]>
6 <!-- XML документ -->
7 <book>
8 </book>
```

Синтаксис схемы DTD представляет собой последовательный набор объявлений, порядок которых имеет значение в следующих случаях:

- при указании объявления более одного раза — выбирается первое;
- объявление параметрической сущности должно быть указано до первого использования (см. п. 2.5.).

Объявление может представлять собой элемент, атрибут, нотацию или сущность.

Объявление элемента. Шаблон объявления нового элемента в пространстве имен представлен в листинге 2.3.

Листинг 2.3: Объявление элемента

```
1 <!ELEMENT name content-model>
```

Поле *name* является именем элемента, а поле *content-model* — описанием содержания этого элемента, которое может быть следующим:

- *EMPTY* — пустой элемент. Данная директива означает, что элемент не может содержать каких-либо данных или вложенных элементов. В качестве примера такого элемента можно привести элемент *
*, который используется в языке разметки HTML для обозначения перевода строки.
- *#PCDATA* (Parsed Character DATA) — символьные, или текстовые, данные. Пример использования данного описания представлен в листинге 2.4, в котором элемент *author* объявлен как текстовый, а в XML документе имеет значение "Стивенс Уильям".

Листинг 2.4: Текстовый элемент

```
1  <?xml version="1.0" encoding="utf-8" ?>
2  <!DOCTYPE author
3    [
4      <!ELEMENT author (#PCDATA)>
5    ]>
6    <!-- XML документ -->
7    <author>Стивенс Уильям</author>
```

- Набор других элементов. В листинге 2.5 элемент *book* объявлен как набор из элементов *title, author*.

Листинг 2.5: Сложный элемент

```
1  <?xml version="1.0" encoding="utf-8" ?>
2  <!DOCTYPE book
3    [
4      <!ELEMENT book (title, author)>
5      <!ELEMENT title (#PCDATA)>
6      <!ELEMENT author (#PCDATA)>
7    ]>
8    <!-- XML документ -->
9    <book>
10     <title>programming on c++</title>
11     <author>Стивенс Уильям</author>
12   </book>
```

Более подробно синтаксис задания вложенных элементов будет рассмотрен в п. 2.2.

- *ANY* — элемент без ограничений на содержимое, позволяющий включать любое содержание в элемент. Данное описание не рекомендуется использовать.
- смешанное содержимое может позволять как текстовые данные, так и вложенную структуру элементов. На использование смешанного содержимого накладываются следующие ограничения:
  - нельзя определить порядок следования элементов, входящих в смешанное содержимое;
  - нельзя определить количество повторов элементов, входящих в смешанное содержимое;
  - одно и то же имя не может быть представлено в объявлении смешанного содержимого более одного раза.

  В листинге 2.6 приведено определение и пример такого элемента. Элемент *book* может быть как текстовым полем, так и элементом, содержащим элемент *author*.

Листинг 2.6: Элемент со смешанным содержимым

```
1  <?xml version="1.0" encoding="utf-8" ?>
2  <!DOCTYPE library
3    [
4      <!ELEMENT library (book+)>
5      <!ELEMENT book (#PCDATA| title)*>
6      <!ELEMENT title (#PCDATA)>
7    ]>
8    <!-- XML документ -->
9  <library>
10     <book>Стивенс Уильям
11         <title>Programming on C++</title>
12     </book>
13 </library>
```

Правила описания сложных элементов. Для описания сложной структуры элемента существуют специальные правила, которые позволяют задавать последовательность, обязательность и частоту присутствия элементов, из которых должен или может состоять данный элемент. Разберем более подробно данные правила.

- Необходимая последовательность, или логическое “И”, означает, что перечисленные элементы должны следовать строго в заданной последовательность. В схеме DTD необходимая последовательность задается с помощью знака “,” (запятая). В листинге 2.5 приведен пример элемента *book*, который должен в обязательном порядке состоять из элементов *title* и *author*.

  При этом не допускается, чтобы элемент *book* состоял из какого-либо одного из элементов *title* или *author*, был в обратной последовательности *author* и *title* или пустым.

- Альтернатива, или логическое “ИЛИ”, позволяет перечислять возможную последовательность элементов. Альтернатива задается с помощью знака “|” (вертикальная черта). В листинге 2.7 представлен пример использования альтернативы, в котором элемент *book* может содержать либо элемент *author*, либо элемент *editor*.

Листинг 2.7: Альтернатива

```
1  <?xml version="1.0" encoding="utf-8" ?>
2  <!DOCTYPE library
3    [
4      <!ELEMENT library (book*)>
5      <!ELEMENT book (author|editor)>
6      <!ELEMENT author (#PCDATA)>
7      <!ELEMENT editor (#PCDATA)>
8    ]>
9  <!-- XML документ -->
10 <library>
11   <book><author>Стивенс Уильям</author></book>
12   <book><editor>A. Ахо</editor></book>
13 </library>
```

- Объединение содержимого задается с помощью круглых скобок. Объединение используется для обозначения последовательности элементов. Например, в листинге 2.8 объединение используется для задания альтернативы из элементов *author* и *editor*.

Листинг 2.8: Объединение содержимого

```
1  <?xml version="1.0" encoding="utf-8" ?>
2  <!DOCTYPE book
3    [
4      <!ELEMENT book (title, (author|editor))>
5      <!ELEMENT author (#PCDATA)>
6      <!ELEMENT editor (#PCDATA)>
7      <!ELEMENT title (#PCDATA)>
8    ]>
9  <!-- XML документ -->
10 <book>
11   <title>Programming on C++</title>
12   <author>Стивенс Уильям</author>
13 </book>
```

Частота присутствия элементов. Для любого элемента схемы или объединения содержимого можно задать количественные характеристики.

- Необязательность присутствия. Задается с помощью знака “?” и означает, что элемент или последовательность элементов может не присутствовать в данном контексте.
- Одно или более присутствии. Задается с помощью знака “+” и означает, что элемент или последовательность элементов должны присутствовать по крайней мере один раз.
- Любое количество присутствие. Задается с помощью знака “*” и означает, что элемент или последовательность элементов может присутствовать любое количество раз, в том числе 0 раз.

В листинге 2.9 показаны примеры использования частоты присутствия элементов: элемент *isOriginal* может быть не определен в элементе *book*, элемент *author* должен обязательно присутствовать в элементе *book* хотя бы один раз, а элемент *editor* может быть использован произвольное число раз.

Листинг 2.9: Частота присутствия элементов

```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE book
  [
    <!ELEMENT book (title, author+, editor*, isOriginal?)
    >
    <!ELEMENT author (#PCDATA)>
    <!ELEMENT title (#PCDATA)>
    <!ELEMENT editor (#PCDATA)>
    <!ELEMENT isOriginal EMPTY>
  ]>
<!-- XML документ -->
<book>
  <title>Programming on C++</title>
  <author>Стивенс Уильям</author>
</book>
```

2.3. Атрибуты в DTD

В DTD описание атрибутов элементов выполняется в виде отдельных объявлений. При этом одно объявление может содержать несколько атрибутов, относящихся к одному элементу. Для атрибута определяется имя, тип или значение и режим. Общая структура объявления атрибутов представлена в листинге 2.10.

Листинг 2.10: Структура объявления атрибута

```
<!ATTLIST name
    attname atttype attdesc attdefault
>
```

Поле *name* определяет имя элемента, к которому данный список атрибутов относится. Поля *attname, atttype, attdesc* определяют соответственно имя атрибута, тип или значение и режим использования. *attdefault* является необязательным полем и определяет значение по умолчанию. Правила использования умолчания зависят от значения *attdesc*. В случае, когда требуется описать больше одного атрибута поля *attname, atttype, attdesc, attdefault* повторяются.

Типы атрибутов. Атрибут может иметь следующий тип:

- CDATA — любые символьные данные.
- NMTOKEN — метка имени. Означает, что значение атрибута должно следовать правилам задания имени в XML документе (начинаться с буквы, отсутствовать пробелы и др.).
- NMTOKENS — множество меток имени, разделенных пробелами.
- ID — идентификатор элемента. Также должен соответствовать правилам задания имени в XML документе, но при этом добавляется условие уникальности на уровне элементов в документе. Стоит также заметить, что уникальность идентификатора глобальна, не зависит от вложенности или типа элемента, т.е. невозможно указывать одинаковые идентификаторы для элементов на разных уровнях вложенности или для элементов разных типов.
- IDREF — ссылка на другой элемент. В качестве значения ссылки должно использоваться значение идентификатора этого другого элемента. В случае если указан несуществующий идентификатор, то документ считается некорректным. Также стоит отметить, что не запрещено в качестве ссылки указывать идентификатор этого же элемента.
- IDREFS — список ссылок, разделенных пробелами, на другие элементы. В качестве ссылки указывается идентификатор.
- ENTITY — имя сущности (см. п. 2.5.). В случае, если не существует сущности с указанным именем, документ считается некорректным.
- ENTITIES — список имен сущностей, перечисленных через запятую.
- список значений представляет собой строго определенный набор значений, которые может принимать атрибут. Значения перечисляются через символ “|”.

В листинге 2.11 представлены примеры определений атрибутов различных типов и соответствующие им значения в документе XML.

Листинг 2.11: Примеры определения атрибутов

```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE library
  [
    <!ELEMENT library (book|author)*>
    <!ELEMENT book EMPTY>
    <!ELEMENT author EMPTY>
    <!ATTLIST book
        book_id ID #REQUIRED
        title CDATA #IMPLIED
        rel_book IDREF #IMPLIED
        authors IDREFS #IMPLIED
        cover ENTITY #IMPLIED
        published (true|false) #REQUIRED>
    <!ATTLIST author
        author_id ID #REQUIRED
        name CDATA #IMPLIED>
    <!NOTATION jpeg PUBLIC "JPG">
    <!ENTITY c SYSTEM "cover.jpg" NDATA jpeg>
  ]>
<!-- XML документ -->
<library>
    <book book_id="ID1" title="Интересная книга" authors="
    A1 A3" cover="c" published="true"/>
    <book book_id="ID2" rel_book="ID1" title="еще одна
    интересная книга" authors="A1 A2" published="false"/>
    <author author_id="A1" name="Иванов" />
    <author author_id="A2" name="Петров" />
    <author author_id="A3" name="Сидоров" />
</library>
```

Атрибут *book_id* содержит уникальный идентификатор книги. Связь между книгами “один к одному” описывается с помощью атрибута *rel_book*. Связь с авторами описывается с помощью атрибута *authors*. Изображение обложки книги находится во внешнем файле *cover.jpg* и связано с книгой с помощью сущности *c* через атрибут *cover*. Атрибут *published* может принимать значения только *true* или *false*. При указании отличного от этих двух значений документ считается некорректным.

Режимы атрибутов. Для каждого атрибута можно указать его режим. Существуют следующие режимы атрибутов:

- #IMPLIED — указание атрибута не является обязательным.
- #REQUIRED — указание атрибута обязательно. Если атрибут не указан в данном режиме, то документ считается некорректным.
- #FIXED — значение атрибута строго фиксировано и определяется в схеме DTD. В листинге 2.12 представлен пример такого атрибута. Указание какого-либо другого значения этого атрибута делает документ некорректным.

Листинг 2.12: Фиксированное значение атрибута

```
1  <?xml version="1.0" encoding="utf-8" ?>
2  <!DOCTYPE library
3    [
4      <!ELEMENT library (book*)>
5      <!ELEMENT book EMPTY>
6      <!ATTLIST book published CDATA #FIXED "true">
7    ]>
8  <!-- XML документ -->
9  <library>
10     <book published="true"/>
11     <book />
12 </library>
```

- Значение по умолчанию. В данном режиме указывается значение атрибута, которое будет использоваться в том случае, если атрибут не указан. В листинге 2.13 таким значением для атрибута *published* является значение *"true"*. Значение по умолчанию может быть полезно при обработке XML документа средствами XSL. Применительно к листингу 2.13 книга без указания атрибута *published* с идентификатором *"ID1"* считается уже опубликованной, что позволяет предпринять соответствующие действия. Также стоит отметить, что значение по умолчанию может быть использовано с любым типом атрибута.

Листинг 2.13: Значение атрибута по умолчанию

```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE library
  [
    <!ELEMENT library (book*)>
    <!ELEMENT book EMPTY>
    <!ATTLIST book published (true|false) "true"
                  id ID #REQUIRED>
  ]>
<!-- XML документ -->
<library>
    <book published="false" id="ID1"/>
    <book id="ID2"/>
</library>
```

2.4. Нотация

Нотация была введена для определения формата внешних (не XML) данных, например, файлов с изображениями или звуковых файлов, на которые, возможно, необходимо ссылаться в XML документе. Благодаря нотации, обработчик XML документа может корректно определить тип файла, на который ссылаются в документе, и предпринять соответствующие действия.

Общая структура объявления нотации представлена в листинге 2.14.

Листинг 2.14: Структура объявления нотации

```
<!NOTATION name (PUBLIC|SYSTEM) ID>
```

name определяет имя нотации. В случае если за именем указана директива *PUBLIC*, то *ID* должен быть именем соответствующего стандарта. Пример нотации для файлов с изображениями в формате PNG приведен в листинге 2.15.

Листинг 2.15: Нотация для файлов в формате PNG

```
<!NOTATION png PUBLIC "PNG 1.2">
```

Если после имени указана директива *SYSTEM*, то *ID* должен указывать обработчику XML документа на способ взаимодействия с файлами данного формата. Это может быть путь до приложения, с помощью которого может быть открыт данный файл, или просто идентификатор приложения, по которому обработчик самостоятельно определит приложения для открытия файла. Если файлы предназначены для передачи с помощью Интернет протоколов, то в качестве *ID* может быть указан MIME тип файла. Пример нотации с указанием MIME типа приведен в листинге 2.16.

Листинг 2.16: Нотация с указанием MIME типа

```
1 <!NOTATION png SYSTEM "image/png">
```

Также существует возможность указания способа взаимодействия в директиве *PUBLIC* после указания стандарта. Пример см. в листинге 2.17.

Листинг 2.17: Нотация для файлов в формате PNG

```
1 <!NOTATION png PUBLIC "PNG 1.2" "image/png">
```

Пример использования нотации см. в листинге 2.11.

2.5. Сущности

В языке XML сущности (entity) позволяют однократно определить фрагмент кода или данных для дальнейшего многократного использования в XML документе. С помощью сущностей можно использовать в XML документе нестандартные символы и сделать документ более наглядным и удобным для использования.

Сущности можно классифицировать на два вида: внешние и внутренние. Внешние сущности объявляются за пределами данного документа, а внутренние включаются непосредственно в сам документ. Объявление секции с сущностями в общем случае выглядит следующим образом: <!DOCTYPE имя объявление>. Имя должно совпадать с именем корневого тега XML документа. А объявление представляет собой собственно объявление сущностей. В случае если объявление внутреннее необходимо заключить его в квадратные скобки.

Для того чтобы использовать объявленную сущность, необходимо сослаться на нее в требуемом месте документа. В зависимости от типа

сущности (общая или параметрическая) ссылка имеет формат *&имя;* или *%имя;* соответственно.

В листинге 2.18 приведен пример определения сущности, которое выполняется с помощью специальной директивы ENTITY. В данном примере определена сущность с именем *publisher* и значением “Питер” и показан пример ее использования.

Листинг 2.18: Сущности в XML документе

```
<?xml version="1.0"?>
<!DOCTYPE library
[
    <!ENTITY publisher "Питер">
]>
<library>
    <book>
        <title>Философия Java</title>
        <author>Брюс Эккель</author>
        <publisher>&publisher;</publisher>
    </book>
</library>
```

Если объявление внешнее, то необходимо использовать следующий синтаксис: <!DOCTYPE имя [SYSTEM|PUBLIC] “URI”>. После имени указывается одна из директив SYSTEM или PUBLIC, за которой следует уникальный идентификатор ресурса (URI), по которому XML анализатор должен непосредственно обратиться за определениями сущностей. В случае если указана директива *SYSTEM*, то XML анализор должен при каждой обработке извлекать сущности по указанному URI, если используется *PUBLIC*, то считается, что документ по указанному URI не может быть изменен, а следовательно XML анализатор может кэшировать данные, находящиеся в нем, и использовать их, не обращаясь к документу по URI. В листинге 2.19 приведены два файла *entities.dtd*, в котором определены сущности, и *library.xml*, который эти сущности использует. Из указанного URI понятно, что файл *entities.dtd* должен находиться в той же самой файловой системе, что и файл *library.xml*, в структуре каталогов *xmlstuff* и *dtds*.

Листинг 2.19: Внешние сущности

```
entities.dtd
    <!ENTITY publisher "Питер">
    <!ENTITY author "Брюс Эккель">

library.xml
<?xml version="1.0"?>
<!DOCTYPE library SYSTEM
            "/xmlstuff/dtds/entities.dtd">
<library>
    <book>
        <title>Философия Java</title>
        <author>&author;</author>
        <publisher>&publisher;</publisher>
        <translator>E. Матвеев</translator>
    </book>
    <book>
        <title>Читаемый код</title>
        <author>Дастин Босуэлл</author>
        <publisher>&publisher;</publisher>
    </book>
</library>
```

Ничего не мешает использовать внешнее и внутреннее объявление сущностей одновременно. В листинге 2.20 представлен пример такого XML документа.

Следует отметить, что после подстановки значения сущности вместо ссылки разбор продолжается с точки перед появлением сущности. Таким образом, можно в значении сущности использовать ссылки на другие сущности.

Общие сущности делятся на символьные, не анализируемые и со смешанным содержанием. Символьные сущности используются для представления различных символов. При этом существуют предопределенные сущности, нумерованные и именованные сущности. Предопределенные сущности поддерживаются любыми анализаторами XML документов. К категории предопределенных сущностей относятся символы амперсанта (&), больше (>), меньше (<), апостроф (') и двойные кавычки ("). Они имеют как нумерованные, так и именованные представления. Например, символ больше (>) в нумерован-

ном представлении имеет вид &*#62;*, а в именованном представлении – &*gt;*.

Листинг 2.20: Внешние и внутренние сущности

```
entities.dtd
    <!ENTITY publisher "Питер">
    <!ENTITY author "Брюс Эккель">

library.xml
<?xml version="1.0"?>
<!DOCTYPE library SYSTEM "/xmlstuff/dtds/entities.dtd"
[
    <!ENTITY translator "Е. Матвеев">
]>
<library>
    <book>
        <title>Философия Java</title>
        <author>&author;</author>
        <publisher>&publisher;</publisher>
        <translator>&translator;</translator>
    </book>
    <book>
        <title>Читаемый код</title>
        <author>Дастин Босуэлл</author>
        <publisher>&publisher;</publisher>
    </book>
</library>
```

Не анализируемые сущности представляют собой ссылку на объект, например, изображение, не требующее обработки анализатором XML документов. Пример определения и использования не анализируемой сущности *c* представлен в листинге 2.11 Сущности со смешанным содержанием могут содержать как текстовые данные, так и фрагменты XML. Пример определения и использования сущности представлен в листинге 2.21.

Параметрические сущности предназначены для использования только при описании схем DTD и не должны содержать XML содержимое или встречаться внутри XML документа. Пример объявления и использования параметрической сущности представлен в листинге 2.22.

Листинг 2.21: Сущность со смешанным содержанием

```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE longdoc
[
<!ELEMENT longdoc ANY>
<!ENTITY part1 "p1.xml">
]>
<longdoc>&part1;</longdoc>
```

Листинг 2.22: Параметрические сущности

```
<!ENTITY % content "para | note | warning">
<!ELEMENT chapter (title, epigraph, %content;+)>
```

§ 3. Схема документа на языке XSD

В отличие от описания модели с помощью DTD формат схемы XML Schema Definition (XSD) [6] основан на XML. Таким образом, мы получаем древовидное описание модели документа XML.

3.1. Определение и использование схемы

Общая структура схемы XSD представлена в листинге 3.1.

Листинг 3.1: Структура XSD

```
1 <?xml version="1.0"?>
2 <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
3   <!-- описание схемы документа -->
4 </xsd:schema>
```

В начале документа присутствует XML пролог. В XSD корневым элементом должен быть элемент `schema`. Все элементы, относящиеся к описанию схемы XSD, должны принадлежать пространству имен `http://www.w3.org/2001/XMLSchema`. При описании схемы документа рекомендуется использовать префикс `xsd`.

Как правило, XSD схема используется в приложениях для проверки правильности входного XML документа. Например, проверка XML документа в языке PHP с помощью XSD схемы представлена в листинге 3.2.

Листинг 3.2: Использование XSD в PHP

```
1 $xml = new DOMDocument();
2 $xml->load('example.xml');
3
4 if (!$xml->schemaValidate('example.xsd')) {
5     print '<b>XML document not valid!</b>';
6 }
```

Также можно указать используемую XSD схему непосредственно в XML файле с помощью атрибутов корневого элемента (см. листинг 3.3). Это позволяет проверять правильность XML документа без использования сторонних средств, например, в Web-обозревателях.

Листинг 3.3: Подключение XSD в XML документе

```
<root
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="example.xsd">
<!-- тело документа -->
</root>
```

3.2. Определение элементов и атрибутов

С точки зрения XSD схемы элементы XML делятся на простые и сложные. Простые элементы не содержат атрибутов и вложенных элементов, только данные. Остальные элементы являются сложными.

Определение простого элемента должно заключаться в тег `element` и содержать имя определяемого элемента (атрибут `name`) и его тип (атрибут `type`). Пример определений простого элемента представлен в листинге 3.4.

Листинг 3.4: Простой элемент

```
<xsd:element name='book' type='xsd:string'>
```

Значение по умолчанию указывается с помощью атрибута `default`. Обязательное значение указывается с помощью атрибута `fixed`.

Добавление атрибута к элементу представлено в листинге 3.5.

Листинг 3.5: Определение атрибута для элемента

```
<?xml version="1.0" encoding="utf-8" ?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name='book'>
    <xsd:complexType>
      <xsd:simpleContent>
        <xsd:extension base="xsd:string">
          <xsd:attribute name="lang" type="xsd:string" />
        </xsd:extension>
      </xsd:simpleContent>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

Тег `complexType` определяет, что далее следует описание сложного типа элемента (см. 3.5.). Тег `simpleContent` определяет, что элемент не будет содержать вложенных элементов, а будет основываться на простом типе, возможно, с добавлением атрибута. В теге `extension` определяются атрибуты и ограничения (см. 3.4.). Наконец, с помощью тега `attribute` определяется атрибут элемента с именем, указанным в `name`, и типом, указанным в `type`. В примере из листинга 3.5 это соответственно `lang` и `string`.

Атрибуты определяются как необязательные по умолчанию. Для указания обязательности необходимо определить атрибут `use` со значением `required`.

В листинге 3.6 приведен пример определения вложенных элементов, последовательность которых задается с помощью тега `sequence`. Разработчику XML документа необходимо строго придерживаться данной последовательности: при измененном порядке следования элементов документ будет считаться некорректным.

Листинг 3.6: Определение вложенных элементов

```
1  <?xml version="1.0" encoding="utf-8" ?>
2  <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
3    <xsd:element name='book' type='BookType'/>
4    <xsd:complexType name='BookType'>
5      <xsd:sequence>
6        <xsd:element name='title' type='xsd:string'/>
7        <xsd:element name='author' type='xsd:string'/>
8      </xsd:sequence>
9    </xsd:complexType>
10 </xsd:schema>
```

В XSD схеме также существует тег `any`, соответствующий произвольному элементу XML. Пример использования тега `any` представлен в листинге 3.7. Атрибут `minOccurs` определяет минимальное число использования тега `any`, что соответствует произвольной последовательности элементов.

Аналогично для использования произвольных атрибутов в XSD существует тег `anyAttribute`.

Листинг 3.7: Произвольное содержимое

```
<xsd:element name="person">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="firstname" type="xsd:string"/>
      <xsd:any minOccurs="0"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

3.3. Простые типы данных

XSD схема обладает более богатым набором простых типов данных по сравнению с DTD. Можно выделить следующие категории простых типов.

Численные типы. Часто используют следующие типы:

- “decimal” – простое вещественное число с плавающей запятой;
- “integer” – целое число;
- “positiveInteger”, “negativeInteger” – положительное и отрицательное целое число;
- “nonPositiveInteger”, “nonNegativeInteger” – не положительное и неотрицательное целое число.

Кроме этого, также существуют стандартные типы “float” (32-битное вещественное), “double” (64-битное вещественное), “byte” (8-битное целое), “unsignedInt”, “long”, “unsignedLong” и т.д.

Строковые типы. Основным типом данной категории является “string” – простая строка, в т.ч. пустая. Также используют следующие типы:

- “token” – строка без пробельных символов;
- “language” – код языка, например, “ru-RU”;
- “ID” – идентификатор без пробельных символов;

- “name” – имя элемента XML (см. ограничения в п.1.1.);
- “anyURI” – URI адрес.

Типы даты и времени. К этой категории относятся следующие типы:

- “time” – время в формате “hh:mm:ss zone”;
- “date” – дата в формате “yyyy-mm-dd zone”;
- “dateTime” – дата и время в формате “yyyy-mm-ddThh:mm:ss zone”;
- “duration” – длительность в формате “P*Y*M*DT*H*M*S”.

В форматах даты и времени используются стандартные обозначения: “hh” – часы, “mm” – минуты для времени или номер месяца для даты, “ss” – секунды, “yyyy” – год, “dd” – день, “zone” – зона. Управляющие и пробельные символы в формате значимы.

Формат длительности следует читать следующим образом: управляющий символ “P”, количество лет, управляющий символ “Y”, количество месяцев и т.д. Например, значение “P1Y2M3DT10H30M45S” расшифровывается как интервал в 1 год 2 месяца 3 дня 10 часов 30 минут и 45 секунд.

Двоичные и логические типы. Документ XML не позволяет хранить бинарные данные напрямую, однако предоставляет возможность хранения закодированного представления. К этой категории можно отнести следующие типы:

- “boolean” – логический тип, допускает значения `true` и `false` или 1 и 0 соответственно;
- “binary” – двоичные числа;
- “hexBinary” – 16-ричные числа;
- “base64” – данные, закодированные в формате base64.

Объединение типов. В ряде случаев элемент может содержать значения различных типов. Например, размер шрифта можно указывать как с помощью целого числа, так и с помощью заранее определенных именованных констант. В этом случае элемент XML можно определить с помощью объединения типов. Пример объединения представлен в листинге 3.8.

Листинг 3.8: Объединение типов

```
<?xml version="1.0" encoding="utf-8" ?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<xsd:element name="size">
  <xsd:simpleType>
    <xsd:union>
      <xsd:simpleType>
        <xsd:restriction base="xsd:positiveInteger">
          <xsd:minInclusive value="8"/>
        </xsd:restriction>
      </xsd:simpleType>

      <xsd:simpleType>
         <xsd:restriction base="xsd:string">
           <xsd:enumeration value="small"/>
         </xsd:restriction>
      </xsd:simpleType>
    </xsd:union>
  </xsd:simpleType>
</xsd:element>
</xsd:schema>
```

3.4. Грани

Накладываемые на содержимое элементов ограничения называются гранями. Выделяют следующие варианты граней:

- “enumeration” — определяет список конкретных значений, которые может принимать элемент. В листинге 3.9 значением элемента `car` могут быть только две марки автомобилей: `Audi` и `Mercedes`. В любом другом случае элемент считается некорректным.

Листинг 3.9: Перечисление всех возможных значений

```
1  <?xml version="1.0" encoding="utf-8" ?>
2  <xsd:schema xmlns:xsd="http://www.w3.org/2001/
       XMLSchema">
3  <xsd:element name="car">
4    <xsd:simpleType>
5      <xsd:restriction base="xsd:string">
6        <xsd:enumeration value="Audi"/>
7        <xsd:enumeration value="Mercedes"/>
8      </xsd:restriction>
9    </xsd:simpleType>
10 </xsd:element>
11 </xsd:schema>
```

- “length” — жестко задает длину значения элемента. В листинге 3.10 значение элемента `car` не может быть меньше или больше 4 символов. В этом случае документ считается некорректным.

Листинг 3.10: Ограничение длины значения элемента

```
1  <?xml version="1.0" encoding="utf-8" ?>
2  <xsd:schema xmlns:xsd="http://www.w3.org/2001/
       XMLSchema">
3  <xsd:element name="car">
4    <xsd:simpleType>
5      <xsd:restriction base="xsd:string">
6        <xsd:length value="4"/>
7      </xsd:restriction>
8    </xsd:simpleType>
9  </xsd:element>
10 </xsd:schema>
```

- “maxLength” и “minLength” — задает соответственно максимальную и минимальную границы длины значения элемента. В листинге 3.11 значение элемента `car` может быть не менее 5 символов и не более 10.

Листинг 3.11: Ограничение длины значения элемента

```
1  <?xml version="1.0" encoding="utf-8" ?>
2  <xsd:schema xmlns:xsd="http://www.w3.org/2001/
       XMLSchema">
3  <xsd:element name="car">
4    <xsd:simpleType>
5      <xsd:restriction base="xsd:string">
6        <xsd:minLength value="5"/>
7        <xsd:maxLength value="10"/>
8      </xsd:restriction>
9    </xsd:simpleType>
10 </xsd:element>
11 </xsd:schema>
```

- "minExclusive" и "maxExclusive" — задает соответственно минимальное и максимальное значение элемента исключая граничное.
- "minInclusive" и "maxInclusive" — задает соответственно минимальное и максимальное значение элемента включая граничное. В листинге 3.12 элемент `year` может принимать значения от 1991 до 2015.

Листинг 3.12: Задание области допустимых значений

```
1  <?xml version="1.0" encoding="utf-8" ?>
2  <xsd:schema xmlns:xsd="http://www.w3.org/2001/
       XMLSchema">
3  <xsd:element name="year">
4    <xsd:simpleType>
5      <xsd:restriction base="xsd:integer">
6        <xsd:maxInclusive value="2015"/>
7        <xsd:minExclusive value="1990"/>
8      </xsd:restriction>
9    </xsd:simpleType>
10 </xsd:element>
11 </xsd:schema>
```

- "fractionDigits" — задает количество цифр после запятой для дробных чисел.

- “totalDigits” — задает общее количество цифр.
- “pattern” — позволяет задавать регулярное выражение для содержимого элементов. В листинге 3.13 для элемента `file` задано ограничение такое, что его значение может оканчиваться только на выражение *.txt*.

Листинг 3.13: Регулярное выражение

```
1  <?xml version="1.0" encoding="utf-8" ?>
2  <xsd:schema xmlns:xsd="http://www.w3.org/2001/
       XMLSchema">
3  <xsd:element name="file">
4    <xsd:simpleType>
5      <xsd:restriction base="xsd:string">
6        <xsd:pattern value=".*\.txt"/>
7      </xsd:restriction>
8    </xsd:simpleType>
9  </xsd:element>
10 </xsd:schema>
```

- “whiteSpace” — задает значение пробельных символов (перевод строки, завершение строки, табуляция, пробел) в содержимом элементов. Может принимать одно из следующих значений:
  - “preserve” — пробельные символы остаются в том виде, в котором они заданы в XML документе;
  - “replace” — все пробельные символы заменяются на пробелы;
  - “collapse” — все пробельные символы заменяются на пробелы, а также удаляются все пробелы в начале и конце значения и множество пробелов заменяется на один пробел.

3.5. Определение сложных типов

Определение сложных типов выполняется с помощью тега `complexType`. К сложным типам в XSD относятся пустые элементы

и элементы содержащие вложенные элементы и/или текст. В листинге 3.14 определен сложный тип `imageType` с одним атрибутом `href` для описания элемента без содержимого.

Листинг 3.14: Сложный пустой элемент

```
1 <?xml version="1.0" encoding="utf-8" ?>
2 <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
3   <xsd:complexType name='imageType'>
4     <xsd:attribute name="href" type="xsd:anyURI" />
5   </xsd:complexType>
6
7   <xsd:element name='image' type='imageType'/>
8 </xsd:schema>
```

Сложный тип можно описать в виде самостоятельного объекта или внутри описания элемента. В листинге 3.14 показано определение сложного типа с именем `imageType` и его использование для описания элемента `image`.

В случае когда требуется определить расширение простого типа для элемента с простым телом, внутри элемента `complexType` используется дочерний элемент `simpleContent` с указанием типа содержимого.

Для описания вложенных элементов используются следующие модели групп:

- последовательность (sequence) — элементы должны идти в строгом соответствии с их объявлением, является аналогом “,” из DTD;
- выбор (choice) — выбирается один из элементов из списка, является аналогом “|” из DTD;
- произвольный порядок (all) — элементы должны быть однократно использованы в произвольном порядке.

Пример использования модели групп представлен в листинге 3.6.

Для определения элемента со смешанным содержимым необходимо указать в теге `complexType` атрибут `mixed` со значением `true`.

§ 4. Язык запросов XPath

Язык запросов к элементам XML документа XPath [7] позволяет осуществлять выборку вершин или набора вершин из дерева XML документа по заданным пользователем критериям (положению, относительному положению, типу, содержимому и др.). Язык XPath различает различные типы вершин, также называемые узлами: корневой узел (весь XML документ), элемент, текст, атрибут, комментарий, инструкции обработки, пространство имен. XPath выполняет поиск по конечному дереву XML, т.е. с учетом подставленных значений сущностей и значений по умолчанию. Таким образом нельзя отличить генерируемые конструкции от содержимого XML документа.

Описание последовательности переходов между узлами XML дерева называется маршрутом. В XPath существуют простые маршруты, составные маршруты и сложные маршруты. Простой маршрут определяет искомый узел дерева XML без указания переходов. Составной маршрут содержит объединение нескольких маршрутов. Результатом составного маршрута будет объединение результатов исходных маршрутов. Сложный маршрут содержит комбинацию простых маршрутов в виде последовательности переходов между узлами XML дерева.

4.1. Простые и составные маршруты

Выделяют следующие простые маршруты:

- маршрут до корневого узла — обозначается с помощью знака “/” и выбирает все дерево XML;
- маршрут до элемента — обозначается именем элемента и выбирает все элементы с указанным именем относительно текущего контекста;
- маршрут до атрибута — обозначается в виде комбинации символа “@” и имени атрибута и выбирает содержимое атрибута относительно текущего контекста;
- маршрут к текстовому содержимому — обозначается в виде функции “text()” и выбирает все текстовое содержимое элементов относительно текущего контекста;

- маршрут к комментарию — обозначается в виде функции “comment()” и выбирает содержимое XML комментариев относительно текущего контекста.

Для всех маршрутов кроме маршрута до корневого узла важно значение текущего контекста, т.е. местоположение относительно которого осуществляется поиск в дереве XML. Указание текущего контекста выполняется с помощью символов “//”, например, “//@person”.

Приведем несколько примеров маршрутов, используя листинг 4.1, и результаты выборок по этим маршрутам.

Листинг 4.1: Пример XML документа

```
1  <people>
2    <person born="1912" died="1954" id="p342">
3      <name>Алан Тьюринг</name>
4      <!-- Существовало ли понятие «специалист по
5      информатике» во времена Тьюринга? -->
6      <profession>математик</profession>
7      <profession>криптограф</profession>
8      <homepage xlink:href="http://www.turing.org.uk/"/>
9    </person>
10 </people>
```

- “//profession”

  ```
  <profession>математик</profession>
  <profession>криптограф</profession>
  ```

- “//@xlink:href”:

  ```
  http://www.turing.org.uk/
  ```

- “//comment()”:

  ```
  <!-- Существовало ли понятие «специалист по
  информатике» во времена Тьюринга? -->
  ```

- “//text()”:

  ```
  Алан Тьюринг
  математик
  криптограф
  ```

Маршруты объединяются в составной маршрут с помощью символа "|". Пример составного маршрута: `//text() | //comment()`.

Например, для маршрута `//name|//homepage` будет получен следующий результат:

```
<name>Алан Тьюринг</name>-- NODE --
<homepage href="http://www.turing.org.uk/" />
```

Также XPath позволяет использовать выражения, называемые подстановочными. Существуют три таких выражения:

- "*" — соответствует любому узлу элемента. Перед `*` может присутствовать префикс пространства имен: `svg:*`. Например, для маршрута `*` из листинга 4.1 будут получены следующие узлы:

```
<people>
  <person born="1912" died="1954" id="p342">
    <name>Алан Тьюринг</name>
    <!-- Существовало ли понятие «специалист по
    информатике» во времена Тьюринга? -->
    <profession>математик</profession>
    <profession>криптограф</profession>
    <homepage xlink:href="http://www.turing.org.uk/"/>
  </person>
</people>
```

- "node()" — соответствует всем типам узлов: элементов, текста, атрибутов, инструкций обработки, пространств имен и комментариев.

- "@*" соответствует атрибутам. Например, для маршрута "//@*" результатом будет следующее множество узлов:

```
born="1912"
died="1954"
id="p342"
xlink:href="http://www.turing.org.uk/"
```

4.2. Сложные маршруты XPath

Сложный маршрут XPath делится на шаги адресации. Каждый шаг адресации состоит из трех частей:

- ось поиска или направление поиска;
- критерий узла или условие проверки узлов;
- предикат содержащий дополнительные условия отбора.

Сложные маршруты могут быть представлены в сокращенном и полном вариантах. Сокращенный вариант маршрута содержит ограниченный набор осей поиска:

- "/" — перемещение по иерархии вниз. Например, для маршрута `/people/person/name` знак "/" разделяет уровни иерархии и позволяет спуститься до элемента с именем `name`. Результатом для данного маршрута будет следующее множество узлов:

```
<name>Алан Тьюринг</name>
```

- "." — текущий узел.
- ".." — родительский узел. Ссылка на родительский узел может быть полезна, например, при необходимости его выбрать, проверив дочерние элементы на соответствие некоторым условиям. Предположим, что нам необходимо выбрать элемент, содержащий атрибут `xlink:href`. Этого мы можем достичь при помощи маршрута `//@xlink:href/..`, который возвратит следующие результат:

```
<homepage xlink:href="http://www.turing.org.uk/"/>
```

- "//" — потомки контекстного узла. Позволяет задавать все дочерние элементы (на любом уровне вложенности) для текущего узла. Например, маршрут `//name` выберет все элементы с именем `name`, начиная от корневого, а маршрут `/people/person//name`, начиная с элемента `person`.

Предикаты определяются на значение элементов или атрибутов и задаются после имени элемента в квадратных скобках. Например, маршрут `//person[profession='физик']` даст в результате только

те элементы, именем которых является `profession`, а значением — `''физик''` и которые были вложены в элемент с именем `person`. Рассмотрим некоторые особенности определения предикатов:

- использование операторов сравнения: <, >, >=, <= и !=. Например, для поиска людей с датой рождения после 1900 года можно построить следующий маршрут `//person[@born >= 1900]`.
- использование логических операторов and и or. Например, для поиска людей с датой рождения после 1900 года, но до 1950 года можно построить следующий маршрут `//person[@born >= 1900 and @born >= 1950]`.
- набор узлов истинен, в том случае если он не пуст. Например, для маршрута `//person[name]` в результат будут включены только те элементы с именем `person`, которые имеют вложенные элементы с именем `name`.
- строка истинна, в том случае если она не пуста. Например, маршрут `//person[name/text()]` вернет все элементы с именем `person`, вложенные элементы `name` которых имеют не пустое значение.
- число истинно, в том случае если оно равно позиции элемента. Например, маршрут `//profession[2]` выберет единственный элемент с именем `profession`, который идет под номером 2. Для листинга 4.1 это будет:

  ```
  <profession>криптограф</profession>
  ```

С точки зрения типов выражения языка XPath можно разделить на числовые, строковые и логические. Под числовое значение отводится восемь байт с плавающей запятой и позволяется совершать над ними пять базовых математических операций: +, -, *, div (деление) и mod (остаток). Например, мы можем задать следующий маршрут `//person[@born + 1 > 1912]`. Строковым значением является последовательность из символов Unicode, заключающаяся в одинарные или двойные кавычки. Для сравнения строк могу применяться операторы = (равно) и != (не равно). Язык XPath поддерживает два логических значения `true()` и `false()`, позволяет их сравнивать с помощью

Название	Полное определение	Сокращенная запись
Дочерние узлы	child	/
Родительские узлы	parent	..
Собственная ось	self	.
Атрибуты	attribute	@
Ось потомков с включением контекстного узла	descendant-or-self	//

Таблица 1: Основные оси XPath

операторов =, !=, <, >, >= и <= и инвертировать с помощью функции `not()`. Например, маршрут `//person[not(@died)]` возвратит все элементы `person` без атрибута `died`.

4.3. Оси XPath

Ось и критерий разделяются двойными двоеточиями "::". Например, сокращенный маршрут `people/person/@id` в полной версии будет выглядеть так: `child::people/child::person/attribute::id`, где `child` означает, что далее необходимо переместиться к дочерним элементам (первый `child` от корневого элемента, второй — от элемента `people`), а `attribute` означает, что необходимо извлечь атрибут с именем `id` у элемента с именем `person`.

В таблице 1 приведены основные оси языка XPath, их полное определение и сокращенная запись.

Помимо основных осей существуют дополнительные оси поиска.

- Ось предков (ancestor): все узлы элементов, содержащие контекстный узел.
- Ось следующих одноуровневых узлов (following-sibling): следующие за контекстным узлом и содержащиеся в том же узле родительского элемента.
- Ось предыдущих одноуровневых узлов (preceding-sibling): предшествующие контекстному узлу и содержащиеся в том же узле родительского элемента.

- Ось следующих узлов (following): следующие после контекстного узла.
- Ось предыдущих узлов (preceding): предшествующие началу контекстного узла.
- Ось пространств имен (namespace): все пространства имен в области действия контекстного узла, объявленные либо в контекстном узле, либо в одном из его предков.
- Ось потомков (descendant): все потомки контекстного узла.
- Ось предков, включая контекстный узел (ancestor-or-self): все предки контекстного узла, включая сам контекстный узел.

4.4. Функции XPath

Язык XPath поддерживает возможность обработки результата с помощью встроенных функций. Функция может возвратить значение одного из четырех типов: логическое, числовое, строковое или набор узлов. Функции могут вызываться для набора узлов, строки или числа. Рассмотрим несколько примеров функций:

- `position()` — положение текущего узла в контекстном списке. Например, маршрут `//profession[position()=2]` выберет только один элемент типа `profession`, стоящий на второй позиции.
- `last()` — количество узлов в контекстном наборе. Например, маршрут `//profession[last()>1]` вернет все элементы типа `profession`, количество которых больше одного. Для листинга 4.2 результат будет следующим:

```
<profession>математик</profession>
<profession>криптограф</profession>
```

Листинг 4.2: Пример XML документа для функции last

```
1 <people>
2   <person>
3     <profession>математик</profession>
```

```
4       <profession>криптограф</profession>
5     </person>
6     <person>
7       <profession>физик</profession>
8     </person>
9   </people>
```

- `count()` — в качестве аргумента принимает набор узлов и подсчитывает их количество. Например, `//people[count(profession)>1]` вернет только те элементы типа `people`, у которых количество дочерних элементов типа `profession` больше одного.
- `starts_with (a, b)` — возвращает истинное значение, в том случае если "a" начинается с "b". Например, маршрут `//profession[starts_with(text(), "мате")]` вернет только те элементы типа `profession`, которые начинаются с "мате".
- `sum` — возвращает сумму для значений набора узлов, переданного в качестве аргумента. Например, маршрут `sum(//value)` для листинга 4.3 вернет значение 10.

Листинг 4.3: Пример XML документа для функции sum

```
1  <values>
2      <value>3.4</value>
3      <value>5.6</value>
4      <value>1</value>
5  </values>
```

§ 5. Описание связей в XML

XML документ предназначен для хранения в первую очередь текстовых данных в виде организованной структуры. Однако, часто требуется в XML документе взаимодействие с различными ресурсами, например, текстовыми файлами, XML документами, двоичными файлами (графическими или звуковыми), сервисами (канал новостей или редактор электронной почты) и т.д. Связать ресурс с данными XML документа можно с помощью адреса URI ресурса или с помощью непосредственного внедрения ресурса с использованием тега `CDATA` и кодированием в base64. Внедрение ресурса зачастую неоправданно, т.к. приводит к значительному увеличению XML документа и к невозможности использования динамических ресурсов. С другой стороны, для автоматизированной обработки XML документа обработчику необходимо знать механизм реализации связи с ресурсом, что приводит нас к необходимости универсального способа описания связей с ресурсами.

В качестве общепринятого средства описания связей с ресурсами выступает расширяемый язык соединений XLink (XML Linking Language) [8], синтаксис которого основан на атрибутах XML документа. Для указания связи с ресурсом в виде части внешнего XML документа используется язык XPointer [9].

5.1. Расширяемый язык соединений XLink

XLink позволяет организовывать как простые ссылки, связывая XML документ с ресурсом, так и многонаправленные, задавая множество различных путей между любым количеством документов.

Простая ссылка Простая ссылка представляет собой однонаправленную связь между двумя ресурсами. Источником, или начальным ресурсом, для такой связи является сам XML элемент содержащий ссылку, а целью, или конечным ресурсом, — ресурс заданный с помощью URI.

Рассмотрим пример, в котором необходимо при описании книги указывать ссылку на ее текст. Ничего не мешает нам указывать ссылку в виде вложенного элемента или простого атрибута, однако использование XLink является более универсальным способом и позволит сторонним обработчикам использовать ссылку по назначению.

В листинге 5.1 приведено описание элемента `novel` (книга) с указанием ссылки на текст книги на стороннем ресурсе.

Листинг 5.1: Простая ссылка

```
1 <?xml version="1.0" encoding="KOI8-R"
      standalone="yes"?>
2 <novel xmlns:xlink= "http://www.w3.org/1999/xlink"
3       xlink:type="simple"
4       xlink:href="http://cs.karelia.ru/xml_book_p1.pdf">
5   <title>Технологии XML. Часть 1</title>
6   <year>2014</year>
7 </novel>
```

Для определения пространства имен XLink рекомендуется использовать префикс `xlink`, а уникальным идентификатором этого пространства является следующий URI: `http://www.w3.org/1999/xlink`.

Атрибут `type` определяет тип ссылки, в данном случае типом является простой тип (`simple`). Атрибут `href` определяет ресурс, на который задается ссылка, при этом могут использоваться как относительные, так и абсолютные ссылки (URL, в примере это `http://cs.karelia.ru/xml_book_p1.pdf`).

Атрибуты ссылок Помимо атрибутов `type` и `href` спецификация XLink определяет ряд других глобальных атрибутов: `show, actuate, role, arcrole, title, label, from, to`. Все атрибуты помимо `type` являются необязательными. Рассмотрим эти атрибуты более подробно.

Атрибут `show` задает для приложения обработчика XML документа действие, которое необходимо выполнять при активизации данной связи. Может принимать следующие значения:

- `new` — открыть содержимое URI ссылки в новом окне;
- `replace` — заменить содержимое в текущим окне содержимым по URI;
- `embed` — вставить содержимое URI в текущее окно на место элемента ссылки;
- `other` — делать нечто отличное от открытия, замены или вставки; точное поведение задается конкретным обработчиком;

- `none` — отсутствие какого-либо поведения.

Атрибут `actuate` задает момент времени выполнения действия. Может принимать следующие значения:

- `onLoad` — выполнять действие сразу же, как только обработчик распознает ссылку;
- `onRequest` — выполнять действие по запросу пользователя;
- `other` — момент времени выполнения действия задается конкретным обработчиком;
- `none` — никаких указаний на момент выполнения действия не следует.

В листинге 5.2 обработчику XML документа указывается, что необходимо заменять текущее содержимое на содержимое по ссылке по запросу пользователя.

Листинг 5.2: Простая ссылка с указанием типа и времени действия

```
1  <?xml version="1.0" standalone="yes"?>
2  <novel xmlns:xlink= "http://www.w3.org/1999/xlink"
3       xlink:type="simple"
4       xlink:href="http://cs.karelia.ru/xml_book_p1.pdf"
5       xlink:actuate="onRequest" xlink:show="replace"
6       xlink:title= "Полный текст учебного пособия"
7       xlink:role="http://cs.karelia.ru/~kulakov/courses/xml"
8       >
9    <title>Технологии XML. Часть 1</title>
10   <year>2014</year>
11 </novel>
```

С точки зрения смысла, или семантики, ссылки могут являться указателем на различного рода отношения. Например, это могут быть отношения наследования (родитель-наследник), последовательности (предыдущий-следующий), взаимоотношений (работодатель-работник, покупатель-поставщик) и др. Указание такой семантики возможно при помощи атрибутов `title`, `role` и `label`. Атрибут `title` обычно содержит небольшой объем обычного текста, описывающий удаленный ресурс. Атрибут `role` содержит URI, указывающий на более полное описание удаленного ресурса. Атрибут `label` содержит

идентификатор ссылки, позволяющий ссылаться на нее при описании путей или взаимоотношений между ссылками.

Атрибуты `arcrole, from, to` будут подробно разобраны при описании ребер расширенных ссылок.

Расширенные ссылки Помимо простых ссылок существуют расширенные ссылки, описывающие набор ресурсов и пути движения между ними. Каждый путь соединяет в точности два ресурса. Любой отдельный ресурс может быть соединен с любым подмножеством других ресурсов набора, в том числе и с пустым подмножеством. Также ресурс может соединяться сам с собой.

Расширенная ссылка задается с помощью указания атрибута `type` в значении `extended`. Большинство расширенных ссылок содержат локальные ресурсы (`resource`), удаленные ресурсы (`locator`) и ребра между этими ресурсами (`arc`).

Элементы локаторы идентифицируются типом `locator` и содержат ссылку на удаленный ресурс. Предположим, что необходимо создать документ с описанием книг и указанием всех изданий для каждой книги. Для решения этой задачи в элементе книги мы можем использовать расширенную ссылку с указанием нескольких ссылок локаторов (по количеству изданий). Пример реализации документа с ссылками локаторами приведен в листинге 5.3.

Листинг 5.3: Ссылки локаторы

```
1  <?xml version="1.0" standalone="yes"?>
2  <novel xmlns:xlink= "http://www.w3.org/1999/xlink"
3       xlink:type="extended">
4    <title>Технологии XML. Часть 1</title>
5    <year>2014</year>
6
7    <edition xlink:type="locator" xlink:href="urn:isbn:0688069444" />
8    <edition xlink:type="locator" xlink:href="urn:isbn:0192839306" />
9    <edition xlink:type="locator" xlink:href="urn:isbn:0700609857" />
10 </novel>
```

Элементы локальных ресурсов идентифицируются типом `resource` и обычно ссылаются на ресурсы не являющиеся частью документа, содержащего расширенную ссылку.

Элементы ребра идентифицируются типом `arc` и позволяют связывать два ресурса. Для описания связи необходимо указать значения атрибутов `from` (источник ссылки) и `to` (цель ссылки). В листинге 5.4

приведен пример описания ребер в серии книг. В данном примере в качестве источников и целей для ребер использованы ссылки локаторы, идентификаторы которых заданы в атрибуте `label`. Схему организации ребер в листинге 5.4 можно увидеть на рисунке 1.

Листинг 5.4: Ссылки ребра

```
<?xml version="1.0" standalone="yes"?>
<series xlink:type="extended"
     xmlns:xlink="http://www.w3.org/1999/xlink">

  <!-- элементы локаторы -->
  <novel xlink:type="locator" xlink:label="x1"
   xlink:href="http://cs.karelia.ru/xml_book_p1.pdf">
    <title>Технологии XML. Часть 1</title>
    <year>2014</year>
  </novel>

  <novel xlink:type="locator" xlink:label="x2"
   xlink:href="http://cs.karelia.ru/xml_book_p2.pdf">
    <title>Технологии XML. Часть 2</title>
    <year>2014</year>
  </novel>

  <novel xlink:type="locator" xlink:label="x3"
   xlink:href="http://cs.karelia.ru/xml_book_p3.pdf">
    <title>Технологии XML. Часть 3</title>
    <year>2015</year>
  </novel>

  <!-- ребра -->
  <next xlink:type="arc" xlink:from="x1" xlink:to="x2" />
  <next xlink:type="arc" xlink:from="x2" xlink:to="x3" />
  <previous xlink:type="arc" xlink:from="x2" xlink:to="x1" />
  <previous xlink:type="arc" xlink:from="x3" xlink:to="x2" />
</series>
```

Также как и в случае с атрибутом `role` для ребра простой ссылки существует атрибут `arcrole`, представляющий собой URI, по которому находится более подробное описание этого ребра.

Ссылки также разделяют на внешние, входящие и исходящие. Ссылки между полностью удаленными ресурсами называются внешними ссылками. Ссылки из удаленного ресурса на локальный ресурс называются входящими ссылками (`to` — метка элемента ресурса). Ссылки из локального ресурса на удаленный называются исходящими

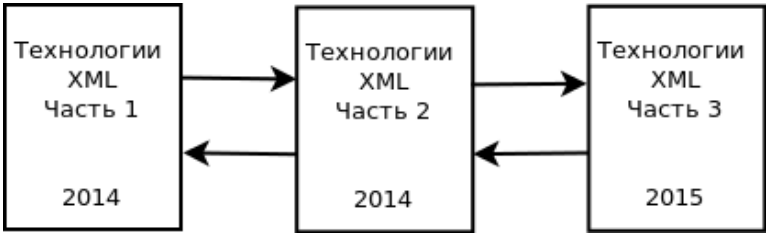

Рис. 1: Схема организации ребер для листинга 5.4

ссылками (`from` — метка элемента ресурса). XML документ, содержащий входящие или внешние ссылки, называется базой ссылок.

5.2. Язык XPointer

Язык XPointer основан на языках XPath и XLink, но добавляет несколько новых возможностей, таких как точка, интервал, а также сокращенный синтаксис для распространенных форм языка XPath. Выражение на языке XPointer заключается в аргументе для функции `xpointer` и возвращает ссылку на корректный XML документ. Например, маршрут `xpointer(//profession)` для листинга 4.1 возвратит два XML документа 5.5 и 5.6.

Листинг 5.5: Первый результат

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <profession>математик</profession>
```

Листинг 5.6: Второй результат

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <profession>криптограф</profession>
```

При дублировании выражений язык XPointer вернет либо указатель на первое, либо на второе в том случае, если первое выражение отсутствует. Например, для маршрута `xpointer(//name)xpointer(//homepage)` возвратится указатель на XML документ с элементом типа `name` (листинг 5.7), а для маршрута `xpointer(//first-name)xpointer(//homepage)` на XML документ с элементом типа `homepage` (листинг 5.8).

Листинг 5.7: Указатель на элемент типа name

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <name>Алан Тьюринг</name>
```

Листинг 5.8: Указатель на элемент типа homepage

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <homepage href="http://www.turing.org.uk/" />
```

Интервалы и точки Понятие точки было введено для возможности указания чего-либо отличного от узла, например, позиции внутри содержимого узла. Для определения точки существуют две функции: `start_point` и `end_point`. `start_point` задает точку непосредственно перед элементом, например, для маршрута `xpointer(start_point(//name))` — перед элементом `name`. `end_point` задает точку за конечным тегом.

В свою очередь интервал представляет собой промежуток анализируемых символьных данных между двумя точками. Существует несколько функций для работы с интервалами:

- `range` — извлекает набор узлов по маршруту, включая начальный и конечный теги. Например, для маршрута `xpointer(range(//name))` результат будет следующим:

  ```
  <name>Алан Тьюринг</name>
  ```

- `range_inside` — набор узлов по маршруту XPath.
- `range_to` — по отношению к контекстному узлу и принимает в качестве аргумента набор узлов.
- `string_range` — интервал внутри текста.

§ 6. Язык запросов XQuery

Одной из областей использования XML является хранение данных. Древовидная структура документа позволяет использовать XML как базу данных. Однако использование реляционной модели для XML документов не оправдано в том числе из-за гибкой структуры и разреженности данных. Тем не менее язык запросов XQuery [10] позволяет выполнять поиск данных в формате реляционной модели.

Язык XQuery основан на использовании XPath. Основными отличиями XQuery от XPath является возможность указания нескольких источников данных, формирование результата и объединение данных.

В листинге 6.1 описан пример XML документа на котором будет показано использование языка XQuery.

Листинг 6.1: документ business.xml

```
<firm>
  <department chief_id="132">
    <name>Отдел разработок</name>
  </department>
  <staff>
    <employee vat="100122567432" id="1">
      <name>Ложкин П. Л.</name>
      <function>Директор</function>
      <salary>16000</salary>
    </employee>
    <employee vat="1001345647" id="132" manager_id="13">
      <name>Иванов И. И.</name>
      <function>Начальник отдела</function>
      <salary>10000</salary>
    </employee>
    <employee vat="1001225674" id="124" manager_id="132">
      <name>Сидоров С. С.</name>
      <function>Программист</function>
      <salary>8000</salary>
    </employee>
    <employee vat="1001225674" id="13" manager_id="1">
      <name>Петров П. П.</name>
      <function>Руководитель проекта</function>
      <salary>12000</salary>
    </employee>
    <employee vat="1001225674" id="146" manager_id="132">
      <name>Павлов Д. П.</name>
      <function>Программист</function>
      <salary>8000</salary>
```

```
30      </employee>
31    </staff>
32  </firm>
```

6.1. Выражение FLOWR

Основной частью XQuery является выражение FLOWR. Это аббревиатура из первых букв операторов, входящих в выражение:

- for - связывание переменной с выражением;
- let - связывание переменной с результатом;
- order by - сортировка потока кортежей;
- where - условие на кортежи;
- return - результат выражения.

Оператор for связывает одну или более переменных с выражениями, создавая поток кортежей. В потоке каждый кортеж связывает данную переменную с одним из значений, получаемых в результате вычисления выражения. Таким образом, оператор for является в некоторой степени оператором цикла. В XQuery переменная обозначается в виде комбинации знака "$" и имени переменной. Как правило, источником выступает XML файл, подключение которого к запросу выполняется с помощью функции "doc()".

Пример использования оператора for показан в листинге 6.2. После оператора for идет объявление переменной `$a`, которая связывается с помощью ключевого слова `in` с потоком объектов по XPath адресу `//employee/name` из XML документа `file:///business.xml`. Затем текстовое содержимое (функция `data()`) каждого значения XPath запроса с помощью оператора `return` добавляется к результату работы XQuery запроса.

Листинг 6.2: Использование оператора for

```
1  for $a in doc('file:///business.xml')//employee/name return
       data($a)
```

Как видно из примера использование всех операторов необязательно.

Оператор `for` можно использовать несколько раз в одном запросе для объединения нескольких документов. Пример многократного использования показан в листинге 6.3. В случае нескольких операторов `for` формируется общий поток из всех комбинаций кортежей. Для организации взаимосвязи между потоками и отсева лишних кортежей можно воспользоваться предикатами, как это показано в примере (`[@vat = $p/vat]`). В результате мы получаем переменные `$p` и `$n` содержащие связанные элементы и эта связь используется для формирования результата в операторе return.

Листинг 6.3: Объединение нескольких документов

```
1 for $p IN document("taxpayers.xml")//person
2 for $n IN document("business.xml")//employee[@vat = $p/vat]
3 return <person><vat>{ $p/vat }</vat>{ $n/name }<tax>{ $p/tax }
      </tax></person>
```

Оператор `let` связывает переменные с полным результатом вычисления выражения, добавляя эти связи к кортежам, полученным оператором `for`, или создавая единственный кортеж (при отсутствии оператора `for`). Оператор `let` также можно использовать несколько раз в одном запросе.

Пример использования оператора `let` показан в листинге 6.4. С помощью оператора `for` переменной `$a` присваивается последовательно значение результата запроса XPath `//employee/name`. С помощью оператора `let` переменной `$b` присваивается значение результата запроса XPath `$a/../function`. Как можно заметить, для использования в операторе `let` доступно все дерево XML с контекстным узлом в переменной `$a`. В результате с помощью оператора `return` выводится найденные пары элементов. При этом стоит учитывать, что результат XQuery запроса может не быть согласованным XML документом.

Листинг 6.4: Использование оператора let

```
1 for $a in doc('file:///business.xml')//employee/name let $b:=
      $a/../function  return ($a, $b)
```

Оператор `order by` сортирует поток кортежей в возрастающем или убывающем порядке. Пример использования оператора `order by` показан в листинге 6.5. Сортировка по убыванию выполняется по числовому значению элемента `salary`. Для получения числа используется встроенный оператор конвертации значения `xs:decimal`.

Листинг 6.5: Использование оператора order by

```
1 for $a in doc('file:///business.xml')//employee order by
      xs:decimal(data($a/salary)) descending return ($a/name,$a/
      salary)
```

Оператор `where` оставляет в потоке только те кортежи, которые удовлетворяют условию, являющемуся параметром данного оператора. Оператор `where` можно использовать только один раз, но в нем определить несколько критериев сортировки. Пример использования оператора `where` показан в листинге 6.6. Оператор `where` оставляет только те кортежи, у которых дочерний элемент `function` имеет значение `Программист`.

Листинг 6.6: Использование оператора where

```
1 for $a in doc('file:///business.xml')//employee where $a/
      function='Программист'  return ($a/name/text(), $a/function
      /text())
```

Также в XQuery запросах доступны следующие операторы:

- выражения и функции XPath;
- список констант, например, `(7, 9, <thirteen/>)`;
- численные интервалы, например, 1 to 100;
- объединение (“,”);
- операторы множества (“|” или union, intersect, except).

Кроме этого в XQuery запросах доступно условное выражение if-then-else. Пример использования условного выражения представлен в листинге 6.7. В случае, если атрибут `@type` выводимого кортежа содержит значение `Journal`, то выводится элемент `editor`, иначе — элемент `author`.

Листинг 6.7: Использование условного выражения

```
1 for $h in document("library.xml")//holding
2 return
3   <holding>
4     { $h/title ,
5       if ($h/@type = "Journal")
6       then $h/editor
```

```
7       else $h/author
8     }
9   </holding>
```

6.2. Примеры запросов XQuery

Далее будет показан ряд примеров использования XQuery запросов для выборки и представления данных.

В листинге 6.8 показан пример соединения данных из различных ветвей одного XML документа.

Листинг 6.8: Соединение данных

```
1 for $a in doc('file:///business.xml')//employee
2 for $b in doc('file:///business.xml')//department
3 where $a/@chief_id = $b/@id
4 return concat($b/name,' - ',$a/name)
```

Следующий пример в листинге 6.9 выбирает всех сотрудников, которые имеют подчиненных. Т.к. один сотрудник может иметь несколько подчиненных, то используется функция `distinct-values()` для получения перечня уникальных значений. В примере используется вложенный запрос для формирования значения переменной `$x` в операторе let.

Листинг 6.9: Поиск сотрудников с подчиненными

```
1  let $x:={
2    for $a in
3       doc('file:///business.xml')//employee
4    for $b in
5       doc('file:///business.xml')//employee
6      where $b/@id = $a/manager_id
7      return $b/name
8    }
9  for $a in $x
10 return distinct-values($a/name)
```

В следующем примере в листинге 6.10 для каждого программиста выбирается разность между его зарплатой и средней зарплатой программистов. В XPath такой запрос реализовать не получится. Для получения средней зарплаты в переменной `$a` используется функция avg.

Листинг 6.10: Разность между значением и средним

```
1 let $a := avg(doc('file:///business.xml')//employee[function='
      Программист']/salary)
2 for $b in  doc('file:///business.xml')//employee[function='
      Программист']
3 return ( $b/name/text(), $b/salary - $a)
```

В языке XQuery существуют два квантора:

- some-in-satisfies — хотя бы одно из проверяемых условий должно быть выполнено;
- every-in-satisfies — должны быть выполнены все проверяемые условия.

Кванторы полезны в случае, когда нам необходимо проверить на выполнение условия набор значений. В листинге 6.11 показан пример использования квантора every-in-satisfies для выборки названий книжек, где в каждом параграфе содержится слово “здесь”.

Листинг 6.11: Использование кванторов

```
1 for $b in document("bib.xml")//book where every $p in $b//
      paragraph satisfies contains($p,"здесь")
2 return $b/title
```

Кванторы можно также использовать для поиска элементов не удовлетворяющих условиям. в листинге 6.12 показан пример выбора сотрудников не имеющих подчиненных. В примере выбираются те элементы, в которых имя не совпадает ни с одним элементом в списке в переменной **$x**.

Листинг 6.12: Использование кванторов

```
1 let $x:=<x>{
2     for $a in doc('file:///business.xml')//employee
3     for $b in doc('file:///business.xml')//employee
4     where $b/@id = $a/manager_id
5     return $b/name} </x>
6 for $c in doc('file:///business.xml')//employee
7   where every $h in $x/name satisfies $h!=$c/name
8 return $c/name/text()
```

В следующем примере (см. листинг 6.13) выбираются имена и зарплаты 10 самых высокооплачиваемых сотрудников фирмы (без учета

комиссионных). Такой запрос представляет некоторую трудность в SQL.

Листинг 6.13: 10 самых высокооплачиваемых сотрудников

```
let $x:=<x>{
 for $a in doc('file:///business.xml')//employee
 order by xs:decimal($a/salary) descending
 return <y>{$a/name, $a/salary}</y> }</x>
for $b in $x/y[position()<=10]
return concat($b/name,' – ',$b/salary)
```

Язык XQuery поддерживает создание собственных функций. Такие функции могут быть полезны для упрощения структуры запроса или для использования в рекурсивных выражениях. В листинге 6.14 показан пример выбора всех прямых начальников сотрудника `Иванова И. И.` Такой запрос удобнее всего реализовать с помощью рекурсивных выражений и собственных функций.

Листинг 6.14: Собственные функции

```
declare namespace mySpace = "http://mySpace"
define function mySpace:getManager($a as element(employee)*)
 as element(employee)* {
  for $b in doc('file:///business.xml')//employee
  where $b/@id = $a/manager_id
  return ($a, $b, mySpace:getManager($b))
 }
for $a in doc('file:///business.xml')//employee[name='Иванов
    И. И.']
return mySpace:getManager($a)/name/text()
```

Приложение. Варианты практических заданий

Задание 1. “XML, DTD”

- Составить вопросы и ответы к ним для проверки знаний по темам лекций XML (5 вопросов) и DTD (5 вопросов) в формате “вопрос - варианты ответа”.
- Создать XML документ в любом текстовом или специализированном редакторе содержащий разбитые по темам вопросы, варианты ответов и ответы.
- Написать DTD определение для XML документа. Связать XML документ и DTD определение.
- Провести валидацию XML документа любым из известных способов.

Форма отчетности: XML файл, DTD файл, определить связь файлов.

Задание 2. “XSD схема”

- Составить вопросы и ответы к ним для проверки знаний по теме лекций XSD (5 вопросов) в формате “вопрос - варианты ответа”.
- Дополнить XML документ из задачи №1 составленными вопросами.
- Написать XSD схему для XML документа из задачи №1. В схеме документа использовать целевое пространство имен.
- Провести валидацию XML документа с использованием XSD схемы любым из известных способов.

Форма отчетности: XML файл, XSD файл, определить связь файлов.

Задание 3. “XLink, XPath, XPointer”

- Написать скрипт(ы) PHP, выполняющий выбор темы из XML файла, просмотр вопросов по выбранной теме

- Дополнить скрипт(ы) PHP возможностью оставлять примечания в отдельном XML файле к теме или к вопросу по теме. Связь между примечаниями, вопросами и темами организовать с использованием XLink и XPath. Примечание должно содержать дату, автора и ссылки на выбранные вопросы и темы.
- Дополнить XML файл с примечаниями ссылками на часть текста вопроса (например, первые 20 символов) с использованием XPointer. Модифицировать скрипт(ы) PHP для сохранения ссылки на часть текста вопроса при создании примечания и для просмотра части текста при отображении примечания.
- Дополнить XML документ из задачи №1 вопросами и вариантами ответов по темам XLink, XPath и XPointer (по 5 вопросов).

Форма отчетности: XML файл, PHP скрипт(ы) в виде web-приложения.

Задание 4. "XQuery"

Все запросы выполняются для XML документа из задачи №1. Рекомендуется использовать средство IPSI-XQuery Interpreter.

- Дополнить XML документ из задачи №1 темой XQuery с 5 вопросами;
- Составить список вопросов (по 2 вопроса из каждой темы);
- Выбрать все вопросы, у которых более одного правильного ответа;
- Выбрать все вопросы, у которых в тексте имеются слова "XML"и (and) "что";
- Подсчитать количество вопросов, у которых есть 4 и более вариантов ответа.

Форма отчетности: XML файл, XQuery запросы.

Список литературы

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

[2] Bray T. The JavaScript Object Notation (JSON) Data Interchange Format. RFC7159 [Электронный ресурс]. 2014. Режим доступа: http://www.rfc-editor.org/rfc/rfc7159.txt

[3] Namespaces in XML 1.0 (Third Edition) [Электронный ресурс]. 2009. Режим доступа: http://www.w3.org/TR/xml-names/

[4] Berners-Lee T., Fielding R., Masinter L. Uniform Resource Identifier (URI): Generic Syntax. RFC3986 [Электронный ресурс]. 2005. Режим доступа: http://www.rfc-editor.org/rfc/rfc3986.txt

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

[6] XML Schema [Электронный ресурс]. 2000. Режим доступа: http://www.w3.org/XML/Schema.html

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

[8] XML Linking Language (XLink) Version 1.0 [Электронный ресурс]. 2001. Режим доступа: http://www.w3.org/TR/xlink/

[9] XML Pointer Language (XPointer) [Электронный ресурс]. 2002. Режим доступа: http://www.w3.org/TR/xptr/

[10] XQuery 1.0: An XML Query Language [Электронный ресурс]. 2010. Режим доступа: http://www.w3.org/TR/xquery/

Учебное издание

Кулаков Кирилл Александрович
Димитров Вячеслав Михайлович

Технологии XML. Часть I. Организация данных

Учебное пособие

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

Подписано в печать 23.06.14. Формат 60×84 1/16
Бумага офсетная. 3 уч.-изд. л. Тираж 55 экз. Изд. №195

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

Для заметок

Для заметок