Министерство образования и науки
Донецкой Народной Республики
ГОУ ВПО «Донецкий национальный университет»
Кафедра теории упругости и вычислительной математики

Авдюшина Е.В.
Пачева М.Н.

WEB/XML технологии

УЧЕБНОЕ ПОСОБИЕ

Издание второе

для студентов направления подготовки
01.04.02 Прикладная математика и информатика

ДОНЕЦК 2019

УДК 004.9(07)

ББК 3973я73

А 18

Рекомендовано к изданию Ученым советом

ГОУ ВПО «Донецкий национальный университет»

(протокол № 1 25.01.2019 г.)

Авдюшина Е.В., Пачева М.Н. WEB/XML технологии: учеб. пособие / Е.В. Авдюшина, М.Н. Пачева. – Изд. 2-е.- Донецк: ДонНУ, 2019. – 119 с.

Рецензенты:

Щепин Н.Н., кандидат физико-математических наук, доцент, ГОУ ВПО «Донецкий национальный университет»;

Машаров П.А., кандидат физико-математических наук, доцент, ГОУ ВПО «Донецкий национальный университет»

В учебном пособии изложена методика планирования обучения по курсу «WEB/XML технологии»: разбиение учебного материала по темам, учебный материал, вопросы для самоконтроля и список литературы. Учебный материал содержит основные сведения о xml-документах, их стандартах и преобразованиях, поиске данных, веб-сервисах.

Учебное пособие предназначено для студентов специальности «Прикладная математика и информатика» и может быть использовано студентами других направления прикладной математики, информатики и информационных технологий.

УДК 004.9

ББК 3973.2-0.18.1я73

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Учебная дисциплина «Web/XML технологии» относится к циклу вариативной части профессионального блока и состоит из одного модуля и четырех тем.

Содержание дисциплины является логическим продолжением дисциплин бакалаврского цикла по направлению подготовки 01.03.02 Прикладная математика и информатика и формирует основу освоения дисциплин: «Современные технологии разработки приложений под мобильные платформы», «Современные компьютерные технологии», «Распределенные информационные системы».

Целью дисциплины является формирование понимания студентами ключевых понятий XML-технологии и ее применения в Web, изучение возможностей создания и преобразования XML-документов, рассмотрение области применимости XML технологии. Основными задачами определены рассмотрение вопросов совместного использования XML и Web технологий.

Процесс изучения дисциплины направлен на формирование элементов следующих компетенций в соответствии с ГОС ВПО по данному направлению подготовки (профилю): готовности к саморазвитию, самореализации, использованию творческого потенциала; способности самостоятельно приобретать с помощью информационных технологий и использовать в практической деятельности новые знания и умения, в том числе в новых областях знаний, непосредственно не связанных со сферой деятельности, расширять и углублять свое научное мировоззрение; способности использовать и применять углубленные знания в области прикладной математики и информатики; способности разрабатывать и анализировать концептуальные и теоретические модели решаемых научных проблем и задач; способности разрабатывать и применять математические методы, системное и прикладное программное обеспечение для решения задач научной и проектно-технологической деятельности; способности управлять проектами,

планировать научно-исследовательскую деятельность, анализировать риски, управлять командой проекта; способности к преподаванию математических дисциплин и информатики в общеобразовательных организациях, профессиональных образовательных организациях и образовательных организациях высшего профессионального образования.

В результате изучения учебной дисциплины студент должен знать:

- принципы построения XML-документов;
- область применения XML-фалов;
- понятие достоверности XML-документа;
- схеме XML-файлов;
- типы анализаторов XML-документов и правила их использования;
- принципы программного построения и изменения XML-документов;
- основные возможности использования XML-документов с web-технологиях.

Уметь:

- проверять достоверность XML-документов;
- определять тип и анализировать схему XML-файлов;
- использовать различные анализаторы для XML-документов;
- создавать и преобразовывать XML-файлы;
- проводить парсинг сайта;
- применять XML-файлы в различных web-технологиях.

Владеть:

- приемами создания и анализа на достоверность XML-документов;
- программными средствами синтаксического анализа XML-документов;
- навыками использования XML-файлов в web-технологиях.

В рамках изучения дисциплины предусмотрены следующие формы организации учебного процесса: лекции, лабораторные занятия, самостоятельную работу студента.

Лекционные занятия предполагают овладение теоретическими основами дисциплины, лабораторные – для овладения методами решения примеров и задач.

Самостоятельная работа студентов предусматривает выполнение домашних заданий, подготовку к лабораторным занятиям, изучение учебно-методической литературы, составление конспектов, подготовку презентаций и докладов.

Текущий контроль осуществляется путем написания самостоятельных и контрольных работ для проверки текущих знаний теории и практики, модульной контрольной работы по проверке знаний теоретических и практических положений.

Язык XML (Extensible Markup Language) был разработан рабочей группой XML Working Group консорциума World Wide Web Consortium (W3C). XML - язык разметки, разработанный специально для размещения информации в World Wide Web, аналогично языку гипертекстовой разметки HTML (Hypertext Markup Language), который изначально стал стандартным языком создания Web-страниц. Поскольку язык HTML полностью удовлетворяет всем нашим потребностям, возникает вопрос: для чего понадобился совершенно новый язык для Web?

HTML нужен для представления данных в браузере. То есть имеется HTML-код и соответствующий этому HTML-коду определённый вид. Однако современные тенденции требуют не просто отображения данных, но ещё и их грамотной внутренней структуры. Вот именно для создания структуры и существует язык XML.

XML - язык разметки текста, крайне напоминающий HTML. Но его чаще используют как формат хранения данных. Проще говоря – база данных. Он очень удобен, почти во всех библиотеках нашего времени можно найти классы/функции для работы с ним.

Главная особенность XML - это его универсальность. То есть XML понимает любой современный язык. А так как XML - это текстовый файл, то с

ним можно работать и в обычном блокноте. Теперь конкретно к практике, где используется XML:

- Файл-настроек. Настройки в XML-файле очень легко считывать и записывать. По этой причине на Вашем компьютере находятся сотни XML-файлов.
- Мост для обмена данными между программами, написанными на разных языках. Очень важная особенность, следующая из универсальности языка, и это регулярно используется в сложных системах.
- Хранение данных. Фактически, это некий аналог базы данных, но не требующий СУБД (например, MySQL). А благодаря языку запросов XPath становится возможным легко общаться с этой "базой данных".

Таким образом, сочетание простого формального синтаксиса, удобства для человека, расширяемости, а также базирование на кодировках Unicode для представления содержания документов привело к широкому использованию как собственно XML, так и множества производных специализированных языков на базе XML в самых разнообразных программных средствах [9].

XML - это расширяемый язык разметки (Extensible Markup Language), разработанный специально для размещения информации в World Wide Web, наряду с HTML, который давно стал стандартным языком создания Web-страниц.

XML решает ряд проблем, которые не решает HTML, например:

- представление документов любого (не только текстового) типа, например, музыки, математических уравнений и т.д.
- сортировка, фильтрация и поиск информации.
- представление информации в структурированном (иерархическом) виде.

Главная особенность XML - это его универсальность. То есть XML понимает любой современный язык. А так как XML - это текстовый файл, то с ним можно работать и в обычном блокноте .

Назначение XML

- хранение структурированных данных;
- обмен информацией между программами;
- создание на его основе более специализированных языков разметки (например, XHTML).

Преимущества XML:

- XML(человеко-ориентированный);
- XML поддерживает Юникод;
- в формате XML могут быть описаны основные структуры данных - такие как записи, списки и деревья;
- XML - это самодокументируемый формат, который описывает структуру и имена полей также как и значения полей;
- XML имеет строго определённый синтаксис и требования к анализу, что позволяет ему оставаться простым, эффективным и непротиворечивым;
- XML также широко используется для хранения и обработки документов;
- XML - формат, основанный на международных стандартах;
- иерархическая структура XML подходит для описания практически любых типов документов;
- XML представляет собой простой текст, свободный от лицензирования и каких-либо ограничений;
- XML не зависит от платформы;
- XML является подмножеством SGML (который используется с 1986 года). Уже накоплен большой опыт работы с языком и созданы специализированные приложения;
- XML не накладывает требований на расположение символов на строке.

Недостатки XML:

- синтаксис XML избыточен. Размер XML документа существенно больше бинарного представления тех же данных. В грубых оценках величину

этого фактора принимают за 1 порядок (в 10 раз). Размер XML документа существенно больше, чем документа в альтернативных текстовых форматах передачи данных (например JSON, YAML) и особенно в форматах данных оптимизированных для конкретного случая использования;

- избыточность XML может повлиять на эффективность приложения. Для большого количества задач не нужна вся мощь синтаксиса XML и можно использовать значительно более простые и производительные решения. XML не содержит встроенной в язык поддержки типов данных. В нём нет понятий «целых чисел», «строк», «дат», «булевых значений» и т. д. Иерархическая модель данных, предлагаемая XML, ограничена по сравнению с реляционной моделью и объектно-ориентированными графами;
- пространства имён XML сложно использовать и их сложно реализовывать в XML парсерах;
- существуют другие, обладающие сходными с XML возможностями, текстовые форматы данных, которые обладают более высоким удобством чтения человеком.

Тема 1. XML-документы и их схемы

1.1. Стандарт языка XML

Стандарт языка XML (Extensible Markup Language, расширяемый язык разметки) был принят организацией World Wide Web Consortium (W3C, www.w3c.org), которая является контролирующей организацией в области разработки и утверждения веб-стандартов. Благодаря этому возможна разработка настраиваемого языка разметки, обладающего собственным набором свойств. Подобная возможность отсутствует в других языках разметки, таких как HTML (Hypertext Markup Language, гипертекстовый язык разметки) [6]. Как известно, в HTML все используемые элементы являются предопределенными, в результате чего ограничивается область применения этого языка. Фактически XML является языком метаразметки, поскольку (в отличие от обычных языков разметки) позволяет разрабатывать пользовательские языки разметки [4].

В XML имеется большая степень свободы, но вместе с тем возрастает персональная ответственность разработчика. В XML пользователи могут определять свои собственные элементы, но им же приходится принимать решение относительно дальнейшего использования этих элементов. Несмотря на очевидную гибкость, XML-документы подчиняются нескольким правилам, которые позволяют успешно и плодотворно работать с ними. По сути, требования, предъявляемые к XML-документам, значительно строже, нежели в случае с HTML-документами.

На XML-документы накладываются **два особых ограничения: формальная корректность (well-formedness) и действительность (validity).** Согласно рекомендациям консорциума W3C (World Wide Web Consortium), наиболее важным ограничением является формальная корректность.

Согласно спецификации языка XML 1.0, объект данных именуется XML-документом, если является формально корректным. Если это условие выполняется, а также удовлетворяются некоторые дополнительные ограничения, XML-документ считается действительным. Требование действительности состоит в том, что документы должны соответствовать условиям, заданным в определении DTD (Document Type Definition, определение типа документа).

Консорциум W3C определяет понятие формальной корректности следующим образом. Текстовый объект является формально корректным, если

- выступая в качестве единого целого, соответствует сценарию «документ»;

- отвечает всем условиям формальной корректности, описанным в спецификации, помещенной на веб-узле www.w3.org/TR/REC-xml;

- каждая из разбираемых сущностей, на которую имеется прямая или косвенная ссылка в документе, является формально корректной.

Отдельные спецификации, входящие в состав рабочего проекта или рекомендации, называются **сценариями**. Поэтому условие формальной корректности требует, чтобы **документ соответствовал сценарию документа**, в соответствии с которым документ **состоит из трех частей**: пролога (может быть пустым), корневого элемента и необязательной общей части.

Пролог находится в начале XML-документа. В действительности XML-документы не нуждаются в прологе, если они являются формально корректными. Однако консорциум W3C рекомендует включать в пролог, как минимум, XML-объявление, в котором указывается применяемая версия языка XML. Как правило, пролог включает XML-объявления, комментарии, инструкции по обработке, пропуски и объявления типа документа.

Любой XML-документ может (и должен, согласно рекомендациям консорциума W3C) начинаться с XML-объявления. Место для включения XML-объявления —первая строка. Например:

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

При указании XML-объявления применяется элемент <?xml?>. Ранее в этом случае использовался формат записи <?XML?>, но в соответствии с последней рекомендацией повсеместно используются строчные буквы.

В XML-объявлении допускается указание следующих атрибутов:

- version: версия XML-объявления. В настоящее время существует только одна версия XML —1.0. Если в составе документа используется XML-объявление, этот атрибут обязателен для применения;

- encoding: кодировка символов документа. По умолчанию применяется кодировка UTF-8. Можно вместе с тем использовать Unicode, UCS-2 или UCS-4, а также множество других символьных кодировок, например ISO. Этот атрибут необязателен для применения;

- standalone: присваивается значение "yes", если этот документ не ссылается на внешние объекты; в противном случае присваивается значение "по". Этот атрибут необязателен для применения, по умолчанию его значение равно "yes".

Как и в HTML, **комментарий** начинается символами <! - - и заканчивается символами -->.

Если в XML-документ добавляются комментарии, следует придерживаться указанных ниже правил:

- комментарии не должны помещаться перед XML-объявлением;

- нельзя помещать комментарий внутри символов разметки;

- недопустимо использовать символ - - внутри комментария, поскольку при синтаксическом разборе двойной дефис играет роль признака окончания комментария;

- комментарии также позволяют исключить из процесса синтаксического разбора части документов.

Инструкции по обработке управляют функционированием XML-процессора. Они начинаются последовательностью символов <?, а завершаются символами ?>. При этом запрещается использовать тег <?xml?>. Инструкции по

обработке должны быть понятны XML-процессору, то есть они зависят от используемого процессора, а не формируются на основе рекомендаций XML.

Простым примером инструкции по обработке (которая, впрочем, не указана в рекомендациях XML 1.0) является <?xml -stylesheet?>, включающая в документ таблицу стилей. Например:

```
<?xml version =”1.0" standalone="yes"  encoding="UTF-8"?>
<?xml  -stylesheet type="text/css" href="file.css"?>
```

Каждый формально корректный XML-документ включает элемент, который содержит все остальные элементы. В данном случае идет речь о **корневом элементе**. Этот объект — очень важная часть XML-документа, особенно с точки зрения программирования, поскольку синтаксический разбор XML-документа начинается с корневого элемента. **Корневой элемент** документа может включать подэлементы. Вообще говоря, трудно представить XML-документы, корневой элемент которых не содержит других элементов. Формально корректный XML-документ должен включать только один корневой элемент, а все остальные элементы должны входить в его состав.

Необязательная общая часть может состоять из XML-комментариев, инструкций по обработке и пропусков (пробелы, табуляции и т. п.).

Фрагменты XML-документа называются **сущностями**. Сущность является частью документа, и может содержать текстовые или двоичные данные (но не те и другие одновременно). Сущность может ссылаться на другие сущности и таким образом включать их в документ, может получать другие разобранные (символьные данные) или неразобранные (символьные данные, которые могут включать текст, не являющийся XML-кодом, или двоичные данные, не разбираемые XML-процессором) сущности. Другими словами, сущность позволяет определить некий универсальный способ ссылок на единицу хранимых данных в XML, —например, файл с несколькими XML-элементами является сущностью, однако он не будет XML-документом, пока не станет формально корректным.

Использование понятия разобранной сущности в данном контексте означает то, что, если выполняется ссылка на сущности, благодаря чему производится их включение (сущности могут содержать данные из внешних источников) в документ, содержащиеся в них данные будут соответствовать условию формальной корректности.

Затем добавляется корневой элемент <ИмяКорневогоЭлемента>:

```
<?xml version =”1.0" standalone="yes"?>
<ИмяКорневогоЭлемента>
</ИмяКорневогоЭлемента>
```

Естественно, что корневой элемент может содержать другие элементы. Например, в следующий документ добавляются сведения:

```
<?xml version =”1.0" standalone="yes"?>
<ИмяКорневогоЭлемента>
<НазваниеТега1>
</ НазваниеТега1>
<НазваниеТега2>
</ НазваниеТега2>
</ИмяКорневогоЭлемента>
```

Документы XML включают разметку и текстовые данные. **Разметка образует структуру документа.** Она состоит из начальных и конечных тегов, тегов пустых элементов, объектных и текстовых ссылок, комментариев, секции CDATA, определений DTD и инструкций по обработке. Текст в XML-документе, который не относится к категории разметки, представляет собой **текстовые данные**.

Начальный тег (также называемый открывающим тегом) начинается символом < и завершается символом >. Конечные теги (также называемые закрывающими тегами) начинаются символами </ и заканчиваются символом >.

Спецификация XML ограничивает возможности по **выбору названий тегов**: в качестве начального символа можно использовать букву, символ подчеркивания или двоеточие (но не пропуск). В качестве вторых, третьих и т.

д. символов в названиях тегов могут применяться буквы, цифры, символы подчеркивания, дефисы, точки и двоеточия (но не пропуски).

Хотя в рекомендациях XML1.0 явно это не указывается, следует избегать использования двоеточий в именах тегов, поскольку двоеточие используется в XML при указании пространств имен.

Заметьте, что XML-процессор является чувствительным к регистру символов.

Пустые элементы включают лишь один тег (исключая начальный или конечный теги). Как вы помните, пустые элементы в HTML, например <IMG>, <LI>, <HR> и
, ничего не включают (ни данных, ни разметки). Пустые элементы эквивалентны единственному тегу (в HTML отсутствуют закрывающие теги, например, </IMG>, </LI>, </HR> и </BR>). Объявление пустых элементов в XML осуществляется в определении DTD. В XML пустому элементу соответствует завершающий тег />.

Однако разметка не обязательно начинается и заканчивается символами < и >.

Начало разметки может обозначаться символом &, а конец — знаком ; . При наличии общих объектных ссылок (объектная ссылка заменяется объектом, на который устанавливается ссылка в процессе синтаксического разбора) разметка может начинаться с символа % и заканчиваться символом ; (при наличии параметра объектной ссылки, который используется в определении DTD). В случае применения объектных ссылок некоторые части разметки, содержащиеся в документе, могут превращаться в символьные элементы. Например, элемент разметки > который является общей объектной ссылкой, в процессе синтаксического разбора превращается в символ <. Соответственно, элемент разметки &lt в этом случае превращается в символ >.

<First>
Этот текст содержит < First > элемент.
</ First>

Поскольку некоторые элементы разметки при синтаксическом разборе могут превращаться в символы, результаты разбора всегда анализируются. При этом вместо элементов разметки подставляются соответствующие символы. В этом случае элементы разметки называются разобранными символьными данными.

Следует отметить, что такие символы, как пробел, возврат каретки, перевод строки и табуляция, воспринимаются в XML как **пропуск.**

Атрибуты XML напоминают соответствующие атрибуты HTML — пары «имя/значение», позволяющие определить дополнительные данные для начального и пустого тегов. Присваивание атрибуту значения осуществляется с помощью знака равенства.

В соответствии со спецификацией XML 1.0, при назначении имен атрибутам следует придерживаться правил, действительных в случае определения имен дескрипторов. В качестве начального символа в имени атрибута может применяться буква, знак подчеркивания или двоеточие. В качестве следующих символов могут применяться буквы, цифры, символы подчеркивания, дефисы, точки и двоеточия (не допускается использование пропусков, поскольку с их помощью разделяются пары «имя атрибута/значение»). В XML значения атрибутов заключаются в кавычки. Как правило, применяются двойные кавычки, хотя возможны проблемы в случае, если значение атрибута само содержит двойные кавычки.

Атрибуты, как и другие элементы разметки, представлены символами. Даже если атрибуту присваивается числовое значение, оно трактуется в виде текстовой строки (заключается в кавычки).

Полезный атрибут xml:lang. Заслуживает внимания один из атрибутов общего назначения —xml:lang. Этот атрибут применяется в целях указания языка, применяемого при определении содержимого документа и значений атрибутов. Также он применяется вспомогательными программами, например поисковыми механизмами в Веб. Используемый в XML-тегах язык может быть указан с помощью атрибута xml : lang. (В действительных документах этот

атрибут, наравне со многими другими, должен быть объявлен перед применением.)

Атрибуту xml: lang могут быть присвоены перечисленные ниже значения:

- двухбуквенный код языка в том виде, в котором он определен (ISO 639);

- идентификатор языка, зарегистрированный Internet Assigned Numbers Authority (IANA). Эти идентификаторы обозначаются префиксом «i-» (или «I-»);

- идентификатор языка, присвоенный пользователем в частном порядке. Подобные идентификаторы начинаются символами «x-» или «X-».

Ниже приводится соответствующий пример. В данном случае используется атрибут xml: lang, с помощью которого определяется, что применяется английский язык:

```
<p xml:lang="en">The color should be brown.</p>
```

Можно также воспользоваться субкодами языков, которые указываются после кода языка через дефис. Субкод применяется для указания региональных различий или диалектов языка. Например, в следующих примерах одному элементу соответствует британский диалект английского языка, а второму — американский вариант английского языка:

```
<p xml:lang="en-GB">The colour should be brown.</p>
<p xml:lang="en-US">The color should be brown.</p>
```

Построение формально корректной структуры документа

Первое ограничение формально корректной структуры заключается в том, что документ должен начинаться XML-объявлением. Конечно, выполнение этого условия вовсе не обязательно во всех случаях, но требование формальной корректности требует включение XML-объявления, размещенного в заголовке документа (очень важно, чтобы XML-объявление было первой синтаксической конструкцией документа).

При работе с XML-кодом пустые элементы всегда следует завершать символом /> (выполнение этого условия необходимо для выполнения условия формальной корректности).

В формально корректных документах один элемент (корневой) включает все остальные элементы.

Благодаря этому облегчается обработка XML-документов, организованных в виде деревьев.

Поскольку процесс создания формально корректного документа требует корректного вложения элементов, следует проверять, чтобы каждый некорневой элемент содержал только один включающий его родительский элемент.

Обратите внимание на то, что корневой элемент может включать произвольное число дочерних элементов (количество дочерних элементов может быть равным нулю).

Имя атрибута должно быть уникальным в пределах одного и того же начального тега или тега непустого элемента (как указывается в спецификации XML 1.0).

В XML существует пять стандартных объектных ссылок. В процессе обработки XML-документа происходит замена объектной ссылки соответствующим объектом:

& —символ &;

< — символ <;

> —символ >;

' —символ ';

" — символ ".

В процессе разбора текста XML-процессорами предполагается, что символ < всегда обозначает начало тега, а символ & — начало объектной ссылки.

В **секциях** CDATA находятся символьные данные, которые не должны разбираться XML-процессором. Этот ресурс XML весьма полезен для пользователей. Обычно весь текст, включенный в XML-документ, подвергается синтаксическому разбору, причем осуществляется поиск символов < и &.

Секции CDATA указывают XML-процессору на то, что не следует изменять текст, а нужно просто передать его в исходном виде базовому приложению.

Начало секции CDATA обозначается разметкой <! [CDATA[, а завершение — разметкой]]>. Обратите внимание на то, что фактически в секциях CDATA осуществляется поиск, но объектом этого поиска является исключительно текст]]>. Наряду с некоторыми другими последствиями, это означает то, что текст]]> не может включаться в состав секций CDATA. Также не допускается вложение секций CDATA.

Если синтаксис XML-документов успешно выдержал проверку, они называются действительными; в частности, XML-документ считается действительным, если в его состав включено определение **DTD** (document type definition, определение типа документа) или **XML-схема**, а сам документ не противоречит схеме (определению DTD). Это все, что необходимо для создания действительного документа.

1.2. DTD - определение типа документа

Элемент <! DOCTYPE> является объявлением типа документа, которое применяется для указания используемого в документе определения DTD.

Синтаксис и структура элементов определяются посредством определения типа документа (DTD), а определения DTD объявляются посредством объявления типа документа. Как показано ранее, для создания объявления типа документа используется элемент <!DOCTYPE>. Этот элемент может принимать много различных форм; ниже они перечислены (здесь URL —это URL определения DTD, а rootname —имякорневого элемента):

<!DOCTYPE rootname [DTD]>;

<!DOCTYPE rootname SYSTEM URL>;

<!DOCTYPE rootname SYSTEM URL [DTD]>;

<!DOCTYPE rootname PUBLIC identifier URL>;

<!DOCTYPE rootname PUBLIC identifier URL [DTD]>

Определения DTD могут быть внутренними или внешними.

Однако можно также создать **внешние** (external) определения DTD, в которых DTD фактически хранится во внешнем файле (обычно с расширением .dtd). Использование внешних определений DTD облегчает создание XML-приложения, совместно используемого многими пользователями. На практике этот способ поддерживается многими XML-приложениями. Существуют два способа определить внешние DTD: как частные DTD или как публичные DTD. Вначале рассмотрим частные определения DTD.

Частные определения DTD предназначены для использования людьми или группами конфиденциально, а не для всеобщего распространения. Внешнее частное определение DTD задается посредством ключевого слова SYSTEM в элементе <!DOCTYPE> следующим образом:

<!DOCTYPE КорневойЭлемент SYSTEM "имяфайла.dtd">

<!DOCTYPE КорневойЭлемент SYSTEM "URL_file.dtd">

Когда имеется определение DTD, **предназначенное для всеобщего использования,** вместо ключевого слова SYSTEM в объявлении типа документа <!DOCTYPE> используется ключевое слово PUBLIC. Для использования этого ключевого слова следует также создать формальный публичный идентификатор (formal public identifier, FPI), для которого установлены некоторые правила:

- первое поле в FPI определяет подключение DTD к официальному стандарту. Для самостоятельно определенных DTD этому полю должно быть присвоено значение -. Если в данном случае стандарт не применяется, но определение DTD одобрено, используется значение +. Для официальных стандартов это поле является ссылкой на сам стандарт (например, ISO/IEC 13449:2000);

- второе поле описывает имя группы или лица, которое собирается поддерживать или ответственно за это определение DTD. В этом случае

следует воспользоваться уникальным именем, идентифицирующим вашу группу (например, консорциум W3C просто использует W3C);

- третье поле указывает тип описываемого документа, обычно сопровождается уникальным идентификатором некоторого вида (например, версия 1.0). Эта часть должна включать номер обновляемой версии;

- четвертое поле задает используемый определением DTD язык. Например, для английского языка используется EN.Note. Причем двухбуквенный спецификатор языка позволяет задавать максимум 24 х 24 =576 возможных языков; ожидается, что в ближайшее время появятся спецификаторы языка с тремя символами;

- поля в FPI должны разделяться двойной косой чертой (//).

Покажем, как можно изменить предыдущий пример, чтобы включить публичный DTD, содержащий его собственный FPI:

<?xml version =”1.0" standalone="no"?>

<!DOCTYPE DOCUMENT PUBLIC "-//starpowder//Custom XML Version 1.0//EN" "URL_file.dtd">

Фактически можно одновременно использовать внутренние и внешние определения DTD посредством элемента <!DOCTYPE> следующего вида:

<!DOCTYPE rootname SYSTEM URL [DTD]>-для частных внешних DTD и

<!DOCTYPE rootname PUBLIC FPI URL [DTD]> —для публичных внешних DTD. В этом случае внешнее определение DTD задается указателем URL, а внутреннее — определением DTD.

Чтобы использовать DTD, необходимо объявление типа документа. Это означает, что нужно использовать элемент <!DOCTYPE>, являющийся частью пролога документа.

Для объявления синтаксиса элемента в определении DTD используется тег <!ELEMENT> вида

<! ELEMENT NAME CONTENT_MODEL>.

Здесь NAME -это имя объявляемого элемента; CONTENT_MODEL может быть присвоено значение EMPTY или ANY, или смешанное содержимое (другие элементы, а также разобранные символьные данные), или дочерние элементы. Например,

<!ELEMENT DOCUMENT ANY>

Когда элемент объявляется с типом содержимого ANY, он может включать любой тип содержимого: любой элемент в документе, а также разобранные символьные данные. Фактически это означает, что содержимое элементов, объявленное с типом содержимого ANY, не проверяется модулем проверки корректности XML-кода.

Кроме использования типа содержимого ANY путем указания имени этого элемента в круглых скобках, можно определить, что объявляемый элемент содержит другой элемент, как показано ниже

<!ELEMENT DOCUMENT (CUSTOMER)>

В качестве содержимого элемента используются разобранные символьные данные #PCDATA либо другие созданные разработчиком элементы. Для объявления элемента, который может содержать множественные дочерние элементы, есть несколько вариантов. Для обработки множественных дочерних элементов определения DTD используют синтаксис, весьма похожий на обработку регулярных выражений.

Рассмотрим допустимый синтаксис (здесь a и b —это дочерние элементы объявляемого элемента):

- a+ — одно или большее количество появлений элемента a;
- a* — нуль или большее количество появлений элемента a;
- a? — a или ничего;
- a, b — после a следует b;
- a | b —a или b, но не оба сразу;
- (expression) —выражение, заключенное в круглые скобки, означает, что оно обрабатывается как модуль и может включать суффиксные операторы ?, * или +.

Последовательность (sequence) позволяет определить, какие именно дочерние элементы (и в каком порядке) может содержать конкретный элемент. Последовательность — это разделенный запятыми список имен элементов, сообщающий XML-процессору, какие элементы и в каком порядке должны появляться.

DTD является внутренним:

```
<?xml version=”1.0"?>
<!DOCTYPE THESIS [
<!ELEMENT THESIS (P*)>
<!ELEMENT P (#PCDATA)>
]>
<THESIS>
<P>
Это моя статья.
</P>
<P>
Она о научной работе
</P>
<P>
Я занимаюсь научными исследованиями
</P>
</THESIS>
```

Значит, синтаксис каждого элемента задается посредством тега <!ELEMENT>. Используя этот элемент, можно указать, что в качестве содержимого элемента используются разобранные символьные данные #PCDATA либо другие созданные разработчиком элементы. В этом примере указано, что элемент <THESIS> должен содержать только элементы <P>, причем нуль и большее количество элементов <P> (на это указывает символ * после P в элементе <!ELEMENT THESIS (P*)>).

В определении элемента <THESIS> слово #PCDATA указывает, что элемент <P> может содержать только текст, то есть разобранные символьные данные (чистый текст без какой-либо разметки).

```
<!DOCTYPE DOCUMENT [
<!ELEMENT DOCUMENT (CUSTOMER)>
<!ELEMENT CUSTOMER (NAME,DATE, ORDERS)>
<!ELEMENT NAME (LAST_NAME,FIRST_NAME)>
<!ELEMENT LASTJAME (#PCDATA)>
<!ELEMENT FIRST'JAME (#PCDATA)>
<!ELEMENT DATE (#PCDATA)>
<!ELEMENT ORDERS (ITEM)*>
<! ELEMENT ITEM (PRODUCT, NUMBER, PR I CE)>
<!ELEMENT PRODUCT (#PCDATA)>
<!ELEMENT NUMBER (#PCDATA)>
<!ELEMENT PRICE (#PCDATA)>
]>
```

Можно усовершенствовать методику применения операторов +, * и ? в последовательностях, поскольку с помощью круглых скобок возможно создание подпоследовательности (subsequences), то есть последовательности в последовательностях.

```
<!ELEMENT direction (left, right. top?)>
```

Помимо последовательностей, в определениях DTD можно использовать **альтернативы** (choices). Альтернатива позволяет указать, что в конкретном месте появится один из множества элементов. Выбор одного из элементов — <a>, < b > или <c> — выглядит следующим образом: (a | b | c). При использовании этого выражения XML-процессор знает, что в документе может присутствовать только один из элементов — <a>, < b > или <c>. Например,

```
<!ELEMENT p (#PCDATA | I)* >
```

Фактически можно определить, что элемент может содержать как PCDATA, так и другие элементы. Такой тип содержимого называется

смешанным (mixed). Для указания смешанного типа содержимого достаточно перечислить #PCDATA наряду с допустимыми дочерними элементами.

<!ELEMENT PRODUCT (#PCDATA | PRODUCTID)*>

Однако использование смешанного типа содержимого связано с крупным недостатком: можно задавать только имена дочерних элементов, которые могут появиться. Нельзя установить порядок дочерних элементов или количество их появлений. При смешанном типе содержимого нельзя использовать операторы +, * или ?.

Последний оставшийся тип содержимого определения DTD —это пустой тип содержимого (empty-content model). В этом случае объявляемые элементы могут не иметь какого-либо содержимого (PCDATA или другие элементы). Объявить элемент пустым просто; достаточно следующим образом использовать ключевое слово EMPTY:

<!ELEMENT CREDIT_WARNING EMPTY>

<!ELEMENT %title; %content; >

Рассмотрим несколько примеров элемента. Обратите внимание на выражения, начинающиеся с % и заканчивающиеся ;. Эти выражения — параметры объектных ссылок, весьма похожие на общие объектные ссылки, за исключением того, что они используются в определениях DTD, а не в теле документа:

<!ELEMENT CHAPTER (INTRODUCTION, (P | QUOTE | NOTE)*, DIV*)>

Сущности являются составляющими элементами XML-документа. Сущности делятся на два типа: **общие и параметрические.** Общие сущности используются большинством авторов XML-документов, разрабатывающих содержимое XML-документов. Параметрические сущности обладают разносторонними возможностями, а сфера их применения связана с определениями DTD.

На самом деле сущность определяет способ, с помощью которого XML-процессор обращается к элементам данных. Как правило, в качестве сущности выступает обычный текст, хотя также могут применяться данные,

закодированные в двоичном виде. Объявление сущности производится в определении DTD. Обращение к сущности в тексте документа производится с помощью ссылок. Ссылки, имеющие отношение к общим сущностям, начинаются символом &, а завершаются символом ;. Ссылки, обеспечивающие доступ к параметрической сущности, начинаются символом %, а завершаются символом ;. В дальнейшем ссылка, обеспечивающая доступ к сущности, будет называться объектной ссылкой. В процессе синтаксического разбора XML-процессор заменяет объектную ссылку самой сущностью.

Иными словами, сущность объявляется в определении DTD, а дальнейшее обращение к ней производится с помощью объектных ссылок в содержимом документа (для общих сущностей) или в определении DTD — для параметрических сущностей.

Сущности могут быть внутренние и внешние. Внутренняя сущность полностью определяется в теле XML-документа, который ссылается на нее (и фактически сам документ в терминах XML рассматривается в качестве сущности). Содержимое внешних сущностей порождается внешними ресурсами, например, файлами, и ссылка на них обычно включает унифицированный индикатор ресурса, с помощью которого может быть найден этот вид сущности.

Сущности также делятся на разбираемые и неразбираемые. Содержимым разбираемой сущности является формально корректный XML-документ. Неразбираемые сущности содержат информацию, которая не должна подвергаться синтаксическому разбору, — обычный текст или данные в двоичном формате.

Фактически вы уже знакомы с пятью стандартными общими объектными ссылками: <, >, &, " и '. Поскольку эти сущности предопределены в XML, не требуется их объявление в DTD.

Можно также определить собственные сущности путем их объявления в DTD. Для определения сущности следует использовать элемент <!ENTITY> (а

при объявлении элемента — <!ELEMENT>). Объявление обычной сущности выглядит следующим образом:

<! ENTITY NAME DEFINITION>

Здесь NAME - имя сущности, а DEFINITION - ее определение. Имя применяется при обращении к сущности.

Простейшее среди возможных определений сущности — это текст, заменяющий объектную ссылку. Ниже приводится соответствующий пример. В этом случае в DTD определяется общая сущность TODAY, определяющая дату October 15, 2018.

<!ENTITY TODAY "October 15, 2018">

Область применения параметрических сущностей ограничена определениями DTD. Объявление параметрической сущности осуществляется так, как показано в следующем фрагменте кода (обратите внимание на символ %):

<! ENTITY % NAME DEFINITION>

Во-первых, можно вкладывать общие ссылки следующим образом:

<!ENTITY NAME "Alfred Hitchcock">

<!ENTITY SIGNATURE "&NAME; 14 Mystery Drive">

Во-вторых, не допускается зацикливание объектных ссылок, поскольку это может поставить в тупик XML-процессор. Рассмотрим пример:

<!ENTITY NAME "Alfred Hitchcock «&SIGNATURE;">

<!ENTITY SIGNATURE "&NAME; 14 Mystery Drive">

В этом случае XML-процессор пытается разрешить ссылку &NAME;. При этом обнаруживается, что следует подставить текст для сущности SIGNATURE в текст сущности NAME. Однако и для сущности NAME требуется текст из сущности SIGNATURE и т. д., в результате чего образуется бесконечный цикл. Отсюда следует вывод, что в действительных документах циклические объектные ссылки недопустимы.

Более того, нельзя применять общие объектные ссылки для включения текста, который предположительно будет использоваться только в определении

DTD, а не в самом содержимом документа. В качестве примера приведем фрагмент ошибочного кода:

```
<!ENTITY TAGS "(NAME,DATE.ORDERS)">
<!ELEMENT CUSTOMER &TAGS;>
```

Исправить ситуацию в этом случае можно путем применения параметрических сущностей. Общие сущности могут использоваться в определениях DTD для включения текста, который превратится в составную часть документа.

Сущности также могут быть внешними, причем в этом случае придется указать XML-процессору URI-путь к сущности. Можно использовать ссылки на внешние сущности с целью их включения в ваш документ. Также можно определить внешнюю сущность таким образом, что она не будет подвергаться синтаксическому разбору, а это значит, что в документ можно включать данные в двоичном формате (по аналогии с присоединяемой графикой в HTML-документе).

В роли внешних сущностей могут выступать текстовые строки, целые документы или части документов. Самое главное, чтобы после их включения в содержимое документа XML-процессор подтвердил его формальную корректность и действительность.

В определениях DTD внешние сущности объявляются с помощью ключевых слов SYSTEM или PUBLIC. Если использовалось ключевое слово SYSTEM, сущность предназначается для внутреннего применения организацией или частным лицом. Сущности, объявленные с применением ключевого слова PUBLIC, являются глобальными.

В этом случае используется формальный публичный идентификатор (formal public identifier, FPI). Далее приводятся примеры употребления ключевых слов SYSTEM и PUBLIC для объявления внешней сущности.

```
<!ENTITY NAME SYSTEM URI>
<!ENTITY NAME PUBLIC FPI URI>
```

Предположим, что сведения о дате сохранены в текстовом формате: October 15, 2018. Этот текст был включен в файл NameFile.xml. Ниже определяется сущность TODAY, связанная с файлом Primer.xml:

```
<!ENTITY TODAY SYSTEM “Primer.xml”>
```

В следующем примере показано, как можно использовать ссылку на сущность для включения данных в содержимое документа (обратите внимание, что значение атрибута standalone изменено с "yes" на "по", поскольку обрабатывается внешняя сущность.

```
<?xml version =”1.0" standalone=="no"?>
<!DOCTYPE DOCUMENT [
…
<!ENTITY TODAY SYSTEM “Primer.xml”>
]>
<DOCUMENT>
<DATE>&TODAY;</DATE>
…
```

Описанная методика позволяет создавать документы, сформированные на основе других документов.

Если требуется использовать глобальную сущность вместо локальной, примените ключевое слово SYSTEM с формальным публичным идентификатором (FPI), как показано в следующем примере:

```
<!ENTITY TODAY SYSTEM
"-//starpowder//Custom Entity Version 1.O//EN" “Primer.xml”>
```

Благодаря определению внешних сущностей обеспечивается их доступность для многих документов, что полезно в том случае, если вы хотите использовать, например, одну и ту же подпись для всех документов или работаете с текстом (например, приветствием), который будет часто изменяться, причем редактирование должно осуществляться централизованным образом.

Обратите внимание на то, что XML-процессоры, не проверяющие действительность документа, считывают определения DTD с целью выборки объявлений пользовательских сущностей, даже если DTD не используется для проверки действительности документа. Разработчики XML-кода иногда добавляют неполные определения DTD к документам, которые не рассматриваются как действительные, в результате чего обеспечивается возможность применения объектных ссылок. Ниже приводится соответствующий пример (обратите внимание на то, что это определение DTD является незавершенным и включает объявление сущности TODAY):

```
<?xml version =”1.0" standalone=="no"?>
<!DOCTYPE DOCUMENT [
<!ENTITY TODAY SYSTEM “Primer.xml”>
]>
```

Один из способов использования внешних общих сущностей — это компоновка документа из частей, при этом каждая часть рассматривается как общая сущность.

В рассматриваемом примере в документ включается сущность, которая ссылается на документ в файле “Part1.xml”.

```
<?xml version =”1.0" standalone=="no"?>
<!DOCTYPE DOCUMENT [
…
<!ENTITY data SYSTEM “Part1.xml”>
]>
<DOCUMENT>
&data;
</DOCUMENT>
```

Файл “Part1.xml” содержит данные для этого документа.

```
<CUSTOMER>
<NAME>…
</NAME> …
```

В XML можно создавать объектные ссылки для символов. Для этого требуется назначить для символа правильный код в применяемой кодировке. Например, в системе кодирования UTF-8 код символа @ —#64(где # указывает на то, что код представлен в шестнадцатеричной системе счисления), так что можно определить сущность view таким образом, чтобы ссылки на view при синтаксическом разборе были заменены символом @:

<! ENTITY view "@">

Фактически можно даже самостоятельно переопределить стандартные объектные ссылки в том случае, когда XML-процессор их не воспринимает.

Обычно для объявления элементов и атрибутов используются параметрические сущности. Ссылки на параметрические сущности могут включаться только в DTD.

Фактически при этом устанавливается дополнительное ограничение: любая ссылка на параметрическую сущность, применяемая в любом определении DTD, может обращаться только к внешнему подмножеству DTD (внешние подмножества представляют внешние компоненты DTD). Можно использовать параметрические сущности во внутренних подмножествах, но только с некоторыми ограничениями, как это будет показано дальше.

В отличие от общих объектных ссылок, параметрические объектные ссылки начинаются с символа %, а не с символа &. Создание параметрической сущности напоминает формирование обычной сущности, но в этом случае, как показано в следующем примере, в элемент <!ENTITY> включается символ %:

<!ENTITY % NAME DEFINITION>

При объявлении внешней параметрической сущности можно также использовать ключевые слова SYSTEM и PUBLIC (где FPI обозначает формальный публичный идентификатор):

<!ENTITY % NAME SYSTEM URI>

<!ENTITY % NAME PUBLIC FPI URI>

Приведем пример использования внутренней параметрической сущности. В данном случае объявляется параметрическая сущность BR, которая вводится в определение DTD в виде конструкции <! ELEMENT BR EMPTY>:

```
<!DOCTYPE DOCUMENT [
<!ENTITY % BR "<!ELEMENT BR EMPTY>">
…
%BR;   ]>
```

Если параметрическая сущность находится во внешнем подмножестве DTD, на нее можно ссылаться в определениях DTD, в том числе при объявлении элементов.

Ниже рассматривается соответствующий пример. Для данного документа используется внешнее подмножество DTD

```
<?xml version =”1.0" standalone=="no"?>
<!DOCTYPE DOCUMENT SYSTEM "Part2.dtd">
<DOCUMENT>…
< /DOCUMENT >
```

Во внешнем подмножестве DTD, Part2.dtd, будут выполнены структурные изменения, в результате чего элемент <DOCUMENT> сможет включать не только элементы <CUSTOMER>, но также и элементы <BUYER> и <DISCOUNTER>. Каждый из рассматриваемых новых элементов, <BUYER> и <DISCOUNTER>, снабжен моделью содержимого, идентичной модели для элемента <CUSTOMER> (то есть эти элементы включают элементы <NAME>, <DATE> и <ORDERS>). В целях небольшой экономии времени эта модель содержимого (NAME,DATE,ORDERS) будет назначена параметрической сущности record:

```
<!ENTITY % record "(NAME,DATE,ORDERS)">
<!ELEMENT DOCUMENT (CUSTOMER | BUYER | DISCOUNTER)>
```

Теперь можно обращаться к параметрической сущности record в произвольном месте. Это означает, что ее можно использовать для объявления элементов <CUSTOMER>, <BUYER> и <DISCOUNTER>/

Этот пример демонстрирует, вероятно, основную причину, в силу которой большинство разработчиков используют параметрические сущности: стремление компактным образом обработать текст, который часто повторяется в элементах, объявленных в DTD. В рассматриваемом примере представлена модель содержимого трех элементов, использующих одну и ту же параметрическую сущность. Также можно легко определить параметрическую сущность, которая позволит указать список атрибутов, одинаковых для требуемого количества элементов. Таким образом, можно управлять объявлениями многих элементов и атрибутов даже в огромном определении DTD. И, если требуется изменить объявление, потребуется лишь модифицировать параметрическую сущность, а не каждое объявление в отдельности.

При объявлении некоторого нового элемента можно захотеть придать ему только атрибуты для обработки изображения и для работы с универсальным идентификатором ресурса (URI), которые уместны в настоящем случае (фактически в этом случае формируются определения DTD в языке разметки XHTML):

<!ATTLIST NEW_ELEMENT %image_attributes; %URI_attributes;>

Совместно с параметрическими сущностями часто используются две важные директивы: INCLUDE и IGNORE. Эти директивы предназначены для включения или удаления разделов определения DTD; директивы имеют следующий формат:

<! [INCLUDE [DTDSection]]>

и

<![IGNORE [DTD Section]]>.

Используя эти директивы, можно изменять определения DTD.

Приведем пример, демонстрирующий использование этих директив:

```
<![ INCLUDE [
<!ELEMENT PRODUCTID (#PCDATA)>
<!ELEMENT SHIP_DATE (#PCDATA)>
<!ELEMENT SKU (#PCDATA)>
]]>
<![ IGNORE [
<!ELEMENT PRODUCTID (#PCDATA)>
<!ELEMENT SHIP_DATE (#PCDATA)>
<!ELEMENT SKU (#PCDATA)>
]]>
```

Возникает вопрос зачем так все усложнять, если можно воспользоваться комментарием для скрытия разделов DTD. Полезность INCLUDE и IGNORE в целях включения или игнорирования разделов DTD станет более наглядной, если вы используете их вместе с параметрическими сущностями с целью параметризации DTD. В процессе параметризации DTD можно включать или игнорировать многие разделы, просто изменяя значение параметрической сущности с IGNORE на INCLUDE и обратно.

В приведенном примере обеспечивается включение (или игнорирование) раздела DTD путем изменения значения параметрической сущности includer. Для использования параметрической сущности в разделах INCLUDE и IGNORE следует использовать внешнее подмножество DTD, для этого будет установлено внешнее подмножество DTD, Part3.dtd:

```
<?xml version =”1.0" standalone=="no"?>
<!DOCTYPE DOCUMENT SYSTEM "Part1.dtd">
<DOCUMENT>…
```

Листинг файла Part1.dtd

```
<!ENTITY % includer "INCLUDE”>
<!ELEMENT DOCUMENT (CUSTOMER)*>
…
```

```
<![ %includer; [
<!ELEMENT PRODUCTID (#PCDATA)>
<!ELEMENT SHIP_DATE (#PCDATA)>
<!ELEMENT SKU (#PCDATA)>
]]>
```

В этом месте программного кода можно включать (или игнорировать) указанный раздел DTD путем изменения значения сущности includer. Подобная методика программирования позволяет легко сгруппировать сущности, которые нужны для одновременной настройки всего DTD.

Атрибуты - это пары «имя/значение», которые можно использовать в открывающих и пустых тегах в целях ввода дополнительной информации.

Можно присваивать им значения, но, пока атрибуты не будут объявлены, документ не будет действительным. Перечень атрибутов в определении DTD объявляется с помощью элемента **<!ATTLIST>**. Ниже приводится общий синтаксис элемента <!ATTLIST>:

```
<!ATTLIST ELEMENT_NAME
ATTRIBUTE_NAME  TYPE  DEFAULT_VALUE
ATTRIBUTE_NAME  TYPE  DEFAULT_VALUE
ATTRIBUTE_NAME  TYPE  DEFAULT_VALUE
ATTRIBUTE_NAME  TYPE  DEFAULT_VALUE>
```

В рассматриваемом примере ELEMENT_NAME определяет имя элемента, для которого объявлены атрибуты, ATTRIBUTE_NAME - имя объявленного атрибута, TYPE - тип атрибута, DEFAULT_VALUE - значение атрибута, заданное по умолчанию.

Ниже приводятся допустимые значения атрибута **TYPE**:

- CDATA - данные в символьном виде (то есть текст, который не включает разметку);

- ENTITIES - имена нескольких сущностей (которые следует объявить в DTD), разделяемые пропусками;

- ENTITY - имя сущности (которое следует объявить в DTD);

- перечисление - список значений, элементы которого представляют допустимые значения атрибутов;

- ID - корректное XML-имя, которое должно быть уникальным (то есть не совпадать ни с каким ID другого атрибута);

- IDREF - содержит значение атрибута ID некоторого элемента; обычно этот другой элемент связан с текущим элементом;

- IDREFS - список, состоящий из ID нескольких элементов, разделенных пропусками;

- NMTOKEN - имя признака, состоящее из одной или более букв, цифр, дефисов, символов подчеркивания, двоеточий и точек;

- NMTOKENS - список, состоящий из нескольких NMTOKEN, разделенных пропусками;

- NOTATION - формат записи файла (который должен быть объявлен в DTD).

Дальше представлены возможные установки **DEFAULT_VALUE**:

- значение - простое символьное значение, заключенное в кавычки;

- #IMPLIED - указывает на то, что атрибут не имеет значения, заданного по умолчанию, и что этот атрибут можно не использовать;

- #REQUIRED - указывает на то, что атрибут не имеет значения, заданного по умолчанию, но ему должно быть присвоено значение. Если атрибут с признаком #REQUIRED пропущен, документ не действителен;

- #FIXED VALUE - в этом случае VALUE - это значение атрибута, которое является обязательным.

Приведем пример, в котором объявляется атрибут TYPE.

<!ATTLIST CUSTOMER TYPE CDATA #IMPLIED>

Здесь применяется простейший способ объявления путем назначения атрибуту типа CDATA —простые символьные данные. Значение, заданное по умолчанию, определяется с помощью атрибута #IMPLIED.

Если вместо значения по умолчанию для атрибута указано ключевое слово #REQUIRED, это означает, что исходное значение не указывается в

объявлении. Оно должно быть представлено пользователем, применяющим подобное определение DTD.

Объявлять атрибуты с ключевым словом #IMPLIED — обычное дело, поскольку в этом случае определяется, что они могут отображаться или не отображаться в элементах, в зависимости от предпочтений автора документа.

Можно установить значение атрибута таким образом, что ему всегда будет присваиваться данная величина. Для этого следует воспользоваться ключевым словом #FIXED, которое определяет постоянное значение атрибута. После этого указывается значение, которое будет присвоено атрибуту. XML-процессор проводит синтаксический разбор атрибута и его значения в процессе выполнения приложений, поэтому он объявляется как #FIXED.

Если явно использовать атрибут, следует присвоить ему то значение, какое было назначено ему по умолчанию в DTD, иначе XML-процессор выведет сообщение об ошибке.

<!ATTLIST CUSTOMER LANGUAGE CDATA #FIXED "EN">

Простейший тип CDATA представляет собой обычные символьные данные. Это означает, что атрибуту может быть присвоено значение в формате текстовой строки, если эта строка не включает символы разметки. Запрет явного использования символов разметки исключает применение любой строки, в которой есть символы <, > или &. Если требуются именно эти символы, воспользуйтесь стандартными объектными ссылками (<, " и &). Эти объектные ссылки будут разбираться и заменяться соответствующими символами. (Поскольку такие значения атрибутов подвергаются синтаксическому разбору XML-процессором, следует очень тщательно подходить к выбору символов, которые могут выглядеть как символы разметки.)

При использовании перечисляемого типа атрибута не применяется ключевое слово, как это делается в других случаях. Вместо этого он снабжается списком (перечислением) возможных значений. Каждое возможное значение должно быть действительным XML-именем (согласно общепринятым

правилам, первым символом имени должна быть буква, символ подчеркивания и т. д.). Ниже приводится соответствующий пример: здесь объявляется атрибут CREDIT_OK, который может принимать только два возможных значения — "TRUE" или "FALSE" — и который по умолчанию имеет значение "TRUE".

<!ATTLIST CUSTOMER CREDIT_OK (TRUE | FALSE) "TRUE">

Разработчики XML-документов обычно используют еще один тип атрибута —NMTOKEN. Этот атрибут может принимать только те значения, которые заданы в соответствии с правилами присвоения имен в языке разметки XML (то есть состоять из одной или больше букв, цифр, символов дефиса и подчеркивания, двоеточий и точек). В частности, значения NMTOKEN не могут включать пропуски. От значений NMTOKEN также требуется, чтобы они состояли из единственного слова, пропуски любого рода недопустимы (весьма полезное ограничение). Обратимся к примеру. В данном случае объявляется атрибут SHIP_STATE, которому назначается код, включающий две буквы. Также объявляется, что атрибут с присвоенным типом значения NMTOKEM не может принимать значения, которые включают более одного элемента

<!ATTLIST CUSTOMER SHIP_STATE NMTOKEN #REQUIRED>

…

<CUSTOMER SHIP_STATE ="CA">

Тип атрибута NMTOKENS означает, что значение атрибута состоит из нескольких лексем NMTOKEN, разделенных пропусками. Предположим, что атрибут CONTACT_NAME типа NMTOKENS содержит имя и фамилию, разделенные пропуском

<!ATTLIST CUSTOMER CONTACT_NAME NMTOKENS #IMPLIED>

…

<CUSTOMER CONTACT_NAME ="George Starr ">

Тип атрибута ID обеспечивает однозначную идентификацию элементов. В силу этих причин перед XML-процессорами выдвигается обязательное условие, согласно которому в документе не может быть двух элементов с

одинаковыми значениями атрибута типа ID (элементу можно назначить только один атрибут такого типа). Значение, присваиваемое атрибуту этого типа, должно соответствовать правилам именования в XML.

<!ATTLIST CUSTOMER CUSTOMER_ID ID #REQUIRED>

…

<CUSTOMER CUSTOMER_ID ="C1232231 ">

Отметим, что тип ID не может использоваться с атрибутами #FIXED (поскольку все такие атрибуты имеют одинаковое значение).

Тип атрибута IDREF предназначен для передачи информации о структуре документа — в частности, о взаимоотношениях между элементами. Атрибуты IDREF включают значение ID другого элемента этого документа.

Например, пусть требуется установить взаимосвязь между элементами отношения «родитель-потомок», которые не отражены в обычной вложенной структуре документа. В этом случае можно установить значение атрибута IDREF элемента-потомка, которое равно значению ID родительского элемента. Тогда приложение могло бы проверять атрибут потомка типа IDREF для определения его предка.

Ниже приводится соответствующий пример. В нем объявляются два атрибута:

CUSTOMER_ID типа ID и EMPLOYER_ID типа IDREF, последний содержит ID работодателя для данного заказчика.

<!ATTLIST CUSTOMER CUSTOMER_ID #REQUIRED EMPLOYER_ID IDREF #IMPLIED>…

<CUSTOMER CUSTOMER_ID ="C1232231 ">…

<CUSTOMER CUSTOMER_ID ="C1232232" EMPLOYER_ID="C1232231">…

Атрибуты могут также объявляться с типом ENTITY, то есть атрибуту присваивается имя объявленной сущности. В следующем примере объявляется сущность SNAPSHOT1, которая ссылается на внешний файл, содержащий

изображение. Затем можно создать новый именованный атрибут, скажем, IMAGE, через который можно обращаться к сущности SNAPSHOT1.

```
<!ATTLIST CUSTOMER IMAGE ENTITY #IMPLIED>
<!ENTITY SNAPSHOT1 SYSTEM "image.gif">
…
<CUSTOMER IMAGE-"SNAPSHOT1">
```

Как и в случае типа атрибута NMTOKEN, которому соответствует множественный тип NMTOKENS, тип атрибута ENTITY имеет свою множественную разновидность - ENTITIES.

Атрибуты этого типа могут включать список имен сущностей, разделяемых пропусками.

```
<!ATTLIST CUSTOMER IMAGES ENTITIES #IMPLIED>
<!ENTITY SNAPSH0T1 SYSTEM "image.gif">
<!ENTITY SNAPSH0T2 SYSTEM "image2.gif">
…
<CUSTOMER IMAGES="SNAPSHOT1  SNAPSH0T2">
```

Последний рассматриваемый здесь тип атрибута NOTATION. Атрибуту, объявленному с этим типом, присваиваются значения, соответствующие формату записей (notations).

Запись определяет формат представления данных, отличных от XML-кода, и применяется для описания внешних сущностей. К числу распространенных типов записей относятся типы MIME, такие как image/gif, application/xml, text/html.

В следующем примере объявляется два формата записи, GIF и JPG, ассоциированные с типами MIME, image/gif и image/jpeg. Затем определяется атрибут, которому может присваиваться то или иное значение.

Для объявления формата записи файла используется элемент определения DTD

```
<! NOTATION NAME SYSTEM "EXTERNAL_ID">
```

Здесь NAME - название формата записи, а EXTERNAL_ID - внешний идентификатор, который используется в качестве формата записи, причем часто используется тип MIME.

Можно также использовать ключевое слово PUBLIC для обозначения глобальной записи, если применяется FPI:

```
<!NOTATION NAME PUBLIC FPI "EXTERNAL_ID">
```

Теперь определяются форматы записи GIF и JPG

```
<!NOTATION GIF SYSTEM "image/gif">
<!NOTATION JPG SYSTEM "image/jpeg">
```

Теперь можно определить именованный атрибут IMAGE_TYPE типа NOTATION, которому назначается формат записи GIF или JPG.

```
<!ATTLIST CUSTOMER IMAGE NMTOKEN #IMPLIED
IMAGE_TYPE NOTATION (GIF | JPG) #IMPLIED>
…
<CUSTOMER IMAGE="image.gif" IMAGE_TYPE=”"GIF”>
```

1.3. Схемы XML документа

Схемы XML – это альтернативный по сравнению с DTD способ определения типа документа. Наиболее часто используемые языки схем XML – это XSD от W3C и XDR от Microsoft.

В схеме XML определяются элементы и их атрибуты. Кроме того, в схеме XML задаются правила порождения элементов (порядок вложенности элементов, последовательность их появления), и, конечно, типизация элементов и атрибутов с возможностью установления ограничений на значения элементов и их атрибутов.

1.3.1. XDR схема

XML-Data – полное имя языка описания схем, предложенного Майкрософт, а *XML-Data Reduced* – это “часть” полной рекомендации. Схема XDR - это экземпляр XML, т.е. соответствует всем синтаксическим правилам и стандартам XML.

Реализуя проверки данных на уровне документа с помощью схемы, приложения, генерирующие и принимающие транзакции, можно оптимизировать для обеспечения максимального быстродействия. Соответствие полей и правильность записей проверяются на уровне экземпляров XML.

Корневым элементом в схеме XDR всегда является элемент ***Schema*:**

```
<Schema name="имя_схемы"
   xmlns="urn:schemas-microsoft-com:xml-data"
xmlns:dt="urn:schemas-microsoft-com:datatypes">
<-- Объявления элементов схемы -->
</Schema>
```

Элемент ***ElementType*** имеет синтаксис:

```
<ElementType
  content="{empty | texOnly | eltOnly | mixed}">
  dt:type "datatype"
  model="{open | closed}"
  name = "idref"
  order="{one | seq | many}"
>
```

Элемент *ElementType* может иметь следующие атрибуты:

1) name - имя элемента;

2) content - содержание элемента. Допустимые значения: empty (пустой элемент), eltOnly (может быть только контейнером для других элементов), textOnly (только текстовые данные), mixed (смешанные данные);

3) dt:type - тип данных элемента;

4) model - может принимать значения open (разрешено использовать элементы, не определенные в схеме) и closed (запрещено использовать элементы, не определенные в схеме);

5) order - порядок следования дочерних элементов в экземпляре XML. Допустимые значения one (предполагается наличие одного документа), many (любое количество элементов в любом порядке) и seq (элементы указываются в строго заданном порядке).

В качестве дочерних элементов для *ElementType* можно использовать следующие*:*

1) element - объявляет дочерний элемент;
2) description - обеспечивает описание элемента ElementType;
3) datatype - обеспечивает тип данных элемента ElementType;
4) group - определяет порядок следования элементов;
5) AttributeType - определяет атрибут;
6) attribute - определяет сведения о дочернем элементе AttributeType.

Для объявления атрибутов используется синтаксис:

```
<AttributeType
   default="default-value"
   dt:type="primitive-type"
   dt:values="enumerated-values"
   name="idref"
   required="{yes|no}"
>
```

В свою очередь элемент *AttributeType* может иметь атрибуты:

1) default - значение по умолчанию;
2) name - имя атрибута;
3) dt:type - один из следующих типов: entity, entities, enumeration, id, idref, nmtoken, nmtokens, notation, string;

4) dt:values - допустимые значения;

5) required - указывает на обязательное наличие атрибута в описании.

Синтаксис для описания элемента *attribute* выглядит следующим образом:

```
<attribute
    default=“default-value”
    type=“attribute-type”
    [required=“{yes|no}”
]>
```

а его возможные значения могут быть такими:

1) default - значение по умолчанию;

2) type - имя элемента AttributeType, определенного в данной схеме. Должно соответствовать атрибуту name элемента AttributeType;

3) required - указывает на обязательное наличие атрибута в описании.

В отличие от DTD схем XDR поддерживает типы данных. Элемент *Schema* имеет следующий атрибут:

xmlns:dt=“urn=schemas-microsoft-com:datatypes”

Тогда описание схемы имеет вид

```
<Schema name="ИмяСхемы"
    xmlns="urn:schemas-microsoft-com:xml-data"
    xmlns:dt="urn:schemas-microsoft-com:datatypes">
  <!-- ... -->
</Schema>
```

Со списком простых типов данных можно ознакомится на странице по адресу:

https://docs.microsoft.com/en-us/previous-versions//ms256065%28v%3dvs.85%29,

с полным списком данных на странице:

https://docs.microsoft.com/en-us/previous-versions//ms256121(v=vs.85)

Индикаторы вхождения в схемах XDR имеют синтаксис:

```
<element
```

type=“element-type”
[minOccur=“{0|1}”]
[maxOccur=“{1|*}”]
>

XDR схема позволяет определять группы содержания. Для этого в элементе *ElementType* может содержаться элемент *group*, имеющий синтаксис:

<group order=“(one|seq|many)” minOccur=“(0|1)” maxOccur=“(1|*)”>
<element type=“ElementType/”>
<element type=“ElementType/”>
<element type=“ElementType/”>
<element type=“ElementType/”>
</group>

1.3.2. XSD схемы

XSD - XML-схемы Schema Definition Language – стандарт, поддерживаемый W3C.

В XSD схеме могут использоваться следующие элементы:

1) All - разрешает отображение элементов группы в произвольном порядке внутри вмещающего элемента;

2) Annotation - создает аннотацию;

3) Any - разрешает любому элементу из данного пространства (пространств) имен присутствовать во вмещающем элементе последовательности или альтернативы;

4) AnyAttribute - разрешает любому атрибуту из данного пространства (пространств) имен присутствовать во вмещающем элементе compiexType или attributeGroup;

5) Appinfo - определяет информацию, используемую приложениями в элементе annotation;

6) Attribute - создает атрибут;

7) AttributeGroup - группирует объявления атрибутов так, что они могут использоваться как группа в определениях составных типов;

8) Choice - разрешает одному и только одному содержащемуся в группе элементу присутствовать во вмещающем элементе;

9) ComplexContent - содержит расширения или ограничения на составной тип, который содержит смешанное содержимое или только элементы;

10) ComplexType - определяет составной тип, поддерживающий атрибуты и содержимое элемента;

11) Documentation - содержит текст, передаваемый пользователю в элементе аннотации;

12) Element - создает элемент;

13) Extension - расширяет простой тип или составной тип, имеющий простое содержимое;

14) Field – указывает Xpath-выражение (XML Path Language), определяющее ограничение (элементы unique, key и keyref);

15) Group - группирует набор объявлений элементов так, чтобы они могли включаться как единое целое в определения составных типов

16) Import - импортирует пространство имен, причем вмещающая схема содержит ссылки на компоненты из схемы этого пространства

17) Include - включает заданный документ схемы в целевое пространство имен вмещающей схемы;

18) Key - указывает, что атрибут или значение элемента должны быть ключевыми;

19) Keyref - указывает, что атрибут или значение элемента соответствуют значению данного ключа или уникального элемента;

20) List - определяет элемент simpleType в качестве списка значений заданного типа данных;

21) Notation - содержит определение записи для описания в XML-документе формата данных, отличных от XML;

22) Redefine - разрешает переопределять в текущей схеме простые и составные типы, а также группы (и группы атрибутов из внешних файлов схемы);

23) Restriction - определяет такие ограничения, как типы данных;

24) Schema - содержит определение схемы;

25) Selector - указывает Xpath-выражение, выбирающее набор элементов для ограничения тождественности элементов unique, key и keyref;

26) Sequence - требует, чтобы элементы группы отображались во вмещающем элементе в заданной последовательности;

27) SimpleContent - содержит расширения или ограничения на элемент complexType с символьными данными или элементом simpl eType в качестве содержимого. Не содержит каких-либо элементов;

28) SimpleType - определяет простой тип;

29) Union - определяет элемент SimpleType в качестве совокупности значений текущего простого типа данных;

30) Unique - указывает на то, что атрибут или значение элемента должны быть уникальными.

В терминах XML-схем элементы, которые включают или имеют атрибуты, называются составными типами ComplexType. Элементы, которые включают только такие простые данные, как числа, строки или даты, но не имеют каких-либо подэлементов, называются простыми типами SimpleType. Кроме того, атрибуты — это всегда простые типы, поскольку значения атрибута не могут содержать какой-либо структуры. Если рассматривать документ как дерево, простые типы не имеют каких-либо подузлов, в то время как составные типы могут включать подузлы.

Различие между простыми и составными типами существенно, поскольку эти типы объявляются различным образом. Составные типы объявляются

самостоятельно; как будет показано далее, спецификация XML-схемы поставляется с большим числом уже объявленных простых типов.

Содержимое смешанного типа – это возможность использовать внутри элемента дочерние элементы и текст (смешанное содержимое).

Простые определяются с помощью вложенного элемента <xs:simpleType>, сложные — <xs:complexType>.

В схемах XML поддерживаются все наиболее распространенные в языках программирования типы данных. Во некоторые из них:

xs:string - строковый тип;

xs:decimal - десятичное число;

xs:boolean true/false;

xs:float - число с плавающей точки;

xs:date – дата в формате год-месяц-число (например, 2016-09-01);

xs:time – время в формате часы:минуты:секунды (например, 14:29:05)

Кроме того, атрибут элемента minOccurs=”0” определяет минимальное число вхождений элемента, а maxOccurs=”число” — максимальное число вхождений элементов.

Ограничения на значение элемента можно устанавливать с помощью конструкции <xs:restriction>. Например:

```
<xs:element name=”возраст”>
<xs:simpleType>
<xs:restriction base=”xs:integer”> <!— ограничения:—>
<xs:minInclusive value=”25”> <!—мин. значение—>
<xs:maxInclusive=”41” ><!—макс. значение>
</xs:restriction>
</xs:simpleType>
</xs:element>
```

Ограничения на значения элемента (используются множества допустимых значений) устанавливаются с помощью <xs:restriction> и <xs:enumeration>

Например:

```
<xs:element name=”Processor”>
<xs:simpleType>
<xs:restriction base=”xs:string”>
<xs:enumeration value=”P4”/>
<xs:enumeration value=”AMD”/>
<xs:enumeration value=”CoreDuo”/>
</xs:restriction>
</xs:simpleType>
</xs:element>
```

Для того, чтобы можно было *использовать ограничение вне определения элемента* Processor (например, еще в элементе «Platform»), нужно следующим образом модифицировать определение элемента Processor:

```
<xs:element name=”Processor” type=”processorType”>
<xs:simpleType name=”processorType”>
<xs:restriction base=”xs:string”>
<xs:enumeration value=”P4”/>
<xs:enumeration value=”AMD”/>
<xs:enumeration value=”CoreDuo”/>
</xs:restriction>
</xs:simpleType>
</xs:element>
```

В отличие от простых элементов сложные элементы могут иметь атрибуты и содержимое смешанного типа.

Последовательность появления дочерних элементов определяется с помощью конструкции <xs:sequence>.

Например, сложный элемент Computer с установленной последовательностью появления элементов (сначала HDD, потом Motherboard) можно определить так:

```
<xs:complexType name=”Computer”>
```

```
<xs:sequence>
<xs:element name=”HDD” type=”xs:string”/>
<xs:element name=”Motherboard” type=”xs:string”/>
</xs:sequence>
<xs:attribute name=”model” type=”xs:string” fixed=”NewModel”/>
<xsd:complexType>
```

Подключить XMLSchema к XML документу можно так:

```
<ProductList xmlns:xs=»http://www.w3.org/2001/XMLSchema-instance» xs:noNamespaceSchemaLocation=»products.xsd»>
…
</ProductList>
```

Вместо «products.xsd» может быть любой допустимый URL, а вместо ProductList любой ваш корневой элемент XML документа.

В заключение приведем пример XSD схемы, описывающей структуру XML документа, содержащего письма электронной почты:

```
<?xml version = "1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2000/10/XMLSchema">
<xsd:element name="m_box">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="message" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
<xsd:element name="message">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="head" minOccurs="1" maxOccurs="1"/>
```

```
                    <xsd:element ref="body" minOccurs="1" maxOccurs="1"/>
                  </xsd:sequence>
                  <xsd:attribute name="uid" use="required" type="xsd:string"/>
                </xsd:complexType>
              </xsd:element>
              <xsd:element name="head">
                <xsd:complexType>
                  <xsd:sequence>
                    <xsd:element ref="to" minOccurs="1" maxOccurs="unbounded"/>
                    <xsd:element ref="from" minOccurs="1" maxOccurs="1"/>
                    <xsd:element ref="date" minOccurs="1" maxOccurs="1"/>
                    <xsd:element ref="subject" minOccurs="1" maxOccurs="1"/>
                     <xsd:element ref="cc" minOccurs="0" maxOccurs="unbounded"/>
                  </xsd:sequence>
                </xsd:complexType>
              </xsd:element>
              <xsd:element name="to" type="xsd:string"/>
              <xsd:element name="from" type="xsd:string"/>
              <xsd:element name="date" type="xsd:string"/>
              <xsd:element name="subject" type="xsd:string"/>
              <xsd:elememt name="cc" type="xsd:string"/>
              </xsd:schema>
```

Для проверки действительности XML документа можно использовать специальные валидаторы, например *W3C валидатор* (http://validator.w3.org/).

Для проверки схем также существуют специальные валидаторы, например *XML Schema валидатор* (http://www.w3.org/2001/03/webdata/xsv).

Согласно спецификации W3C XML программа должна прекратить обработку XML документа, как только будет обнаружена ошибка в этом документе.

1.4. Применение языка XML в специализированных языках разметки

1.4.1. Язык разметки математических текстов MathML

В настоящее время для отображения математических текстов в Internet создан язык MathML. Корпорация W3C в 2003 году утвердила спецификацию языка MathML версии 2.0 и рекомендовала его к использованию.

Язык MathML использует два способа кодирования математических выражений. Один из них непосредственно передает синтаксис формулы (presentation), другой отражает семантику выражения (content), то-есть ее математическое содержание. При этом поскольку язык MathML, являяется подмножеством расширенного языка разметки XML, то и в основу этого языка положена нотация, свойственная XML. Но эта разметка с точки зрения прозрачности восприятия человеком далека от общепринятой системы кодирования математичеих выражений.

Поскольку, как мы указали в начале статьи,система кодирования играет доминирующую роль в формировании математического языка, то можно попытаться использовать эту систему кодирования и для электроной обработки математических текстов. В качестве противовеса MathML мы предлагаем язык разметки MathTextView, нотация которого максимально приближена к нотации, используемой в языках програмирования для отображения математичских выражений с помощью клавиатуры. Такой подход обладает рядом хороших свойств:

- лаконичность нотации;

- нотация сохраняет как синтаксис формулы так и ее семантику;
- принципиальную возможность преобразования нотации к безскобочной польской записи, которая удобна для машинной обработки математических выражений;
- возможность семантического контроля нотации;
- возможность динамического ввода информации (форумы, чаты и др.).

Элементом верхнего уровня в MathML является тэг <math>. Каждый допустимый экземпляр MathML должен быть внутри этого контейнера. Он не допускает вложений, но внутри может быть произвольное число других дочерних элементов.

Например,

```
<?xml version="1.0" encoding="windows-1251"?>
<?xml-stylesheet type="text/xsl"
   href="http://www.w3.org/Math/XSL/pmathml.xsl"?>
<html xmlns="http://www.w3.org/1999/xhtml">
<math xmlns="http://www.w3.org/1998/Math/MathML">
<msup>
<mo class="MathClass-op">sin</mo>
<mn>2</mn>
</msup>
<mrow> <mi>α</mi> </mrow>
<mo class="MathClass-bin">+</mo>
<msup>
<mo class="MathClass-op"> cos</mo>
<mn>2</mn>
</msup>
<mrow><mi>α</mi></mrow>
<mo class="MathClass-rel">=</mo>
```

```
<mn>1</mn></math>
</html>
```

Более подробную информацию о всех элементах специализированного языка формул можно прочитать здесь

https://developer.mozilla.org/ru/docs/Web/MathML/Element

1.4.2. Язык разметки по созданию графических объектов

Формат масштабируемой векторной графики (Scalable Vector Graphics, SVG) является частью семейства стандартов векторной графики. Векторная графика отличается от растровой, в которой определение цвета каждого пиксела хранится в некотором массиве данных. Наиболее распространенными растровыми форматами, используемыми в Интернете в настоящее время, являются JPEG, GIF и PNG, каждый из которых имеет свои достоинства и недостатки.

При создании графического изображения в формате SVG используется совершенно иной процесс, нежели при создании файлов в форматах JPEG, GIF или PNG. Файлы JPEG, GIF и PNG обычно создаются с помощью какой-либо программы редактирования изображений, например, Adobe Photoshop. Изображения в формате SVG, как правило, создаются с использованием какого-либо языка на базе XML. Существуют графические пользовательские интерфейсы редактирования графики в формате SVG, которые генерируют код XML, лежащий в основе изображения.

Элементы XML для создания графики в формате SVG представлены в табл. 4.1.

Таблица 4.1 - Элементы XML для создания графики в формате SVG

Элемент	Описание
line	Создает простую линию
polyline	Формирует фигуры, построенные с использованием нескольких определений линий
rect	Создает прямоугольник
circle	Создает круг
ellipse	Создает эллипс
polygon	Создает многоугольник
path	Позволяет определять произвольные траектории

Корневой тег SVG имеет атрибуты ширины и высоты, которые определяют рабочую область, доступную для рисования. Эти атрибуты действуют так же, как атрибуты высоты и ширины других элементов HTML. В данном случае установлено, что рабочая область занимает все доступное пространство.

Кроме того, в этом примере используется тег style. SVG-графика поддерживает применение стилей к содержимому с помощью самых разнообразных методов. Определение линии можно создать путем задания начальных и конечных координат по осям X и Y относительно рабочей области. Атрибуты x1 и y1 представляют собой координаты начала, а атрибуты x2 и y2 — координаты конца линии. Чтобы изменить направление вычерчивания линии, необходимо просто изменить координаты.

1.5. Вопросы для самоконтроля

1. Области применения XML-файлов в веб-технологиях.
2. Принципы построения XML-файлов.

3. Формально корретный XML-документ.
4. Схема XML.
5. Встроенные простые типы XSD. Пространства имен языка XSD.
6. Связь документа XML со своей схемой.
7. Построение последовательностей в различных схемах.

Тема 2. Синтаксические анализаторы XML-документов и поиск информации

2.1. Основы синтаксического анализа XML-документов

Существует два популярных типа XML-анализаторов, поддерживающих два стиля обработки XML-документов.

– Document Object model (DOM). DOM-анализаторы преобразуют XML-документ в иерархическую древовидную структуру. После этого при помощи API DOM программа может перемещаться по дереву вверх и вниз, следуя иерархии документа. Интерфейс API DOM облегчает программистам доступ к структуре документа и его элементам.

– Simple XPI for XML (SAX)**.** SAX-анализаторы преобразуют XML-документ в последовательность обратных вызовов программы, которые информируют программу о каждой встреченной анализатором части документа. В ответ программа может выполнять определенные действия, например, реагировать на начало каждого раздела документа или на конкретный атрибут. Интерфейс API SAX предлагает использующим его программам более последовательный стиль обработки документа, лучше соответствующий программной структуре приложений, управляемых событиями.

Рассмотрим работу с XML DOM и Java [8], использующего дерево W3C DOM.

Синтаксический разбор XML-доку мента можно выполнить с помощью объекта DocumentBuilderFactory, который применяется для создания объекта класса DocumentBuilder. Название document builder factory (фабрика построителей документа) связано с тем, что он может применяться для создания анализаторов, использующих классы Java, от различных поставщиков анализаторов.

Для работы необходимо импортировать библиотеки

```
import javax.xml.parsers.*;
import org.w3c.dom.*;
```

Для реального разбора XML-документа применяется метод parse объекта DocumentBuilder [5]. Заметим, что методу parse не обязательно передавать имя локального файла — можно передать URL-ссылку на документ в Интернете, с помощью которой будет осуществлена выборка документа.

Таблица 2.1 - Методы класса javax.xml.parsers.DocumentBuilderFactory

Метод	Описание
protected DocumentBuilderFactory()	Заданный по умолчанию конструктор
Abstract Object getAttribute (String name)	Возвращает определенные значения атрибута
boolean isCoalescing()	Принимает истинное значение, если фабрика сконфигурирована для создания анализаторов, преобразующих узлы CDATA в текстовые узлы
boolean isExpandEntityReferences()	Принимает истинное значение, если фабрика сконфигурирована для формирования анализаторов, расширяющих узлы ссылок для XML-сущностей
boolean isIgnoringComments()	Принимает истинное значение, если фабрика формирует анализаторы, игнорирующие комментарии
boolean isIgnoringElementContentWhitespace()	Принимает истинное значение, если фабрика продуцирует анализаторы, которые не замечают игнорируемые пропуски в содержимом элемента (например, те, которые используются в качестве отступа при указании

	элементов)
boolean isNamespaceAware()	Принимает истинное значение, если фабрика генерирует анализаторы, использующие пространства имен XML
boolean isValidating()	Принимает истинное значение, если фабрика формирует анализаторы, проверяющие действительность XML-содержимого в процессе выполнения операций синтаксического разбора
Abstract DocumentBuilder newDocumentBuilder ()	Создает новый объект DocumentBuilder
Static DocumentBuilder Factorynewlnstance() DocumentBui1derFactory	Возвращает новый объект DocumentBui1derFactory
Abstract void setAttribute (String name. Object value)	Устанавливает определенные атрибуты
void setCoalescing (boolean coalescing)	Требует наличия анализатора для преобразования узлов CDATA в текстовые узлы
void setExpandEntityReferences (boolean expandEntityRef)	Требует наличия анализатора для расширения узлов ссылок на XML-сущности
void setIgnoringComments (boolean ignoreComments)	Требует наличия анализатора для игнорирования комментариев
void setlgnoringElementContent Whitespace(boolean whitespace)	Требует наличия анализаторов, исключающих игнорируемые пропуски
void setNamespaceAware (boolean awareness)	Требует наличия анализатора, обеспечивающего поддержку пространства имен XML
void setValidating (boolean validating)	Требует наличия анализатора,

	проверяющего действительность документов в процессе синтаксического разбора

Таблица 2.2 - Методы из класса javax.xml.parsers.DocumentBuilder

Метод	Описание
Protected DocumentBuilder()	Заданный по умолчанию конструктор
Abstract DOMImplementation getDOMImplementation()	Возвращает объект DOMImplementation
Abstract boolean isNamespaceAware()	Принимает истинное значение, если анализатор сконфигурирован таким образом, что «понимает» пространство имен
Abstract boolean isValidating()	Принимает истинное значение, если данный анализатор сконфигурирован для проверки действительности XML-документов
Abstract Document newDocument() Document tree	Возвращает новый экземпляр объекта DOM Document в целях формирования модели DOM
Document parse(File f)	Разбирает содержимое файла в виде XML-документа и возвращает новый объект DOM Document
Abstract Document parse (InputSource is)	Разбирает содержимое определенного исходного файла в виде XML-документа и возвращает новый объект DOM Document
Document parse(InputStream is)	Разбирает содержимое определенного входного потока InputStream в виде XML-документа, возвращая при этом новый

	объект DOM Document
Document parse(InputStream is, String systemId)	Разбирает содержимое определенного входного потока InputStream в виде XML-документа, возвращая новый объект DOM Document
Document parse(String uri)	Разбирает содержимое, заданное определенным URI, в виде XML-документа, возвращая новый объект DOM Document
Abstract void setEntityResolver (EntityResolver er)	Указывает объект EntityResolver, используемый для разрешения сущностей
Abstract void setErrorHandler (ErrorHandler eh)	Указывает обработчик ошибок, применяемый для отображения отчета об ошибках

Для реального разбора XML-документа применяется метод parse объекта DocumentBuilder. Заметим, что методу parse не обязательно передавать имя локального файла — можно передать URL-ссылку на документ в Интернете, с помощью которой будет осуществлена выборка документа.

Пример документа для разбора

```
<?xml version="1.0" encoding="utf-8"?>
<flowers>
 <rose id="1" id_pr="34">
  <sort>красная</sort>
  <height>0.5</height>
 </rose>
 <rose id="5" id_pr="98">
  <sort>желтая</sort>
  <height>0.5</height>
 </rose>
```

```
</flowers>
```

Фрагмент программы для создания объекта Document

```
public class PR_DOM {
        public static void main(String[] args)
{
try
{
DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
DocumentBuilder builder = factory.newDocumentBuilder();
File fl=new File("pr.xml");
Document doc = builder.parse(fl);
}
catch(Exception e)
{e.printStackTrace();}
        }
        }
```

Таблица 2.3 - Методы интерфейса org.w3c.dom.Document

Метод	Определение
Attr createAttribute(String name)	Создает объект Attr с указанным именем
Attr createAttributeNS(String namespaceURI, String qualifiedName)	Создает атрибут с указанным именем и пространством имен
CDATASection createCDATASection (String data)	Создает узел CDATASection, значением которого является определенная строка
Comment createComment(String data)	С помощью заданной строки создает узел Comment
DocumentFragment createDocumentFragment()	Создает пустой объект DocumentFragment
Element createElement (String tagName)	Создает элемент указанного типа

Element createElementNS(String namespaceURI, String qualifiedName)	Создает элемент с определенным уточняющим именем, а также с унифицированным указателем ресурса (URL)
EntityReference createEntityReference(String name)	Создает объект EntityReference
ProcessingInstruction createProcessingInstruction (String target, String data)	Создает узел ProcessingInstruction
Text createTextNode(String data)	Создает текстовый узел на основе указанной строки
DocumentType getDoctype()	Возвращает для документа определение типа документа (DTD)
Element getDocumentElement()	Поддерживает непосредственный доступ к элементу Document
Element getElementById(String elementId)	Возвращает элемент с указанным ID
NodeList getElementsByTagName (String tagname)	Возвращает все элементы с определенным наименованием тега
NodeList getElementsByTagNameNS(String namespaceURI, String local Name)	Возвращает все элементы с определенным именем и пространством имен
DOMImplementation getImplementation()	Получает объект DOMImplementation, который обрабатывает данный документ
Node importNode(Node importedNode, boolean deep)	Импортирует узел в данный документ из другого документа

2.2. DOM-модель документа

Интерфейс Document основан на интерфейсе Node, который поддерживает W3C-объект Node. Узлы представлены в виде единственного узла на дереве документа (напомним, все, что находится на дереве документа, включая текст и комментарии, воспринимается как узлы). Интерфейс Node располагает большим числом методов, которые применяются при обработке узлов [2]. Например, для получения информации об узле можно воспользоваться такими методами, как getNodeName и getNodeValue. Этот интерфейс также располагает элементами данных, которые называются полями. Здесь хранятся постоянные значения, соответствующие узлам различных типов.

Объект Document является внутренним представлением древовидной структуры XML-документа. На рис.2.1 представлена иерархия интерфейсов.

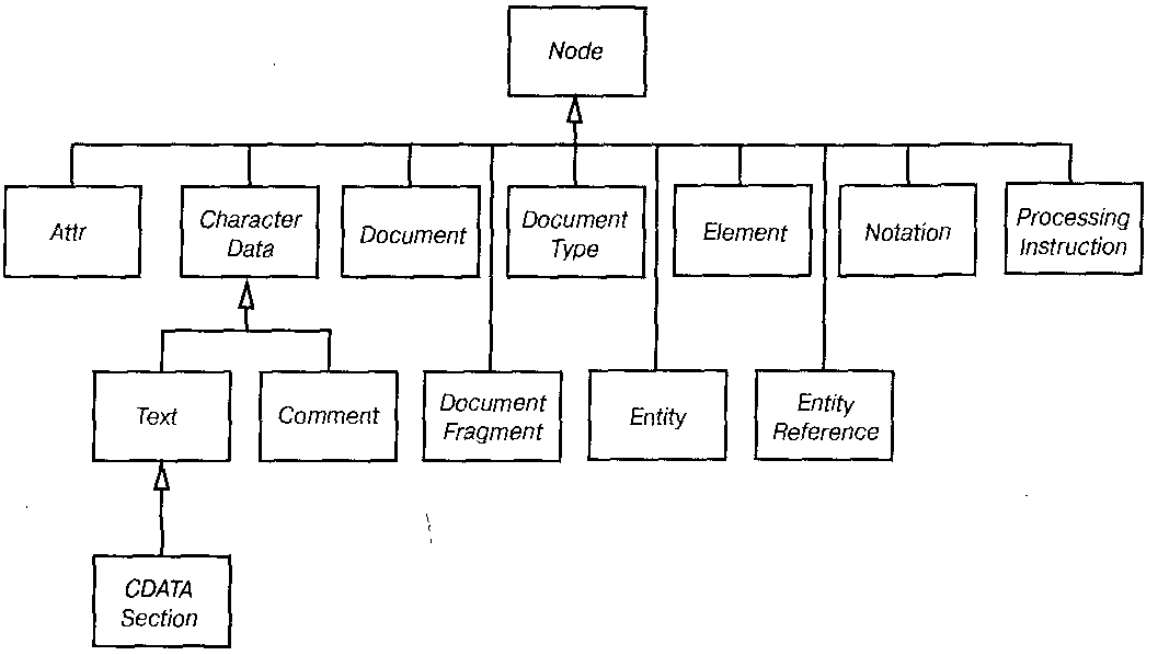

Рис.2.1 – Структура DOM-модели документов

Поля интерфейса Node имеют названия:

static short ATTRIBUTE_NODE

static short CDATA_SECTION_NODE

static short COMMENT_NODE

static short DOCUMENT_FRAGMENTJIODE

static short DOCUMENT_NODE

static short DOCUMENT_TYPE_NODE

static short ELEMENT_NODE

static short ENTITY JIODE

static short ENTITY_REFERENCE_NODE

static short NOTATION_NODE

static short PROCESSINGJNSTRUCTIONJIODE

static short TEXT_NODE

Таблица 2.4 - Методы интерфейса org.w3c.dom.Node

Метод	Описание
Node appendChild(Node newChild)	Добавляет указанный узел в конец списка дочерних элементов текущего узла
Node cloneNode(boolean deep)	Возвращает дубликат данного узла
NamedNodeMap getAttributes()	Возвращает атрибуты данного узла, если последний является элементом
NodeList getChildNodes ()	Возвращает все дочерние элементы для данного узла
Node getFirstChild ()	Возвращает первый дочерний элемент для данного узла
Node getLastChild()	Возвращает последний дочерний элемент для данного узла
String getLocalName()	Возвращает локальную часть полного наименования данного узла
String getNamespaceURI()	Возвращает URI пространства имен для данного узла
Node getNextSibling()	Возвращает узел, следующий за данным узлом
String getNodeName()	Возвращает название данного узла

short getNodeType()	Возвращает тип объекта узла
String getNodeValue ()	Возвращает значение этого узла
Document getOwnerDocument()	Возвращает объект Document для данного узла
Node getParentNode()	Возвращает родительский элемент для данного узла
String getPrefix()	Возвращает префикс пространства имен для данного узла
Node getPreviousSibling()	Возвращает узел, предшествующий данному узлу
boolean hasAttributes()	Принимает истинное значение, если данный узел имеет атрибуты
boolean hasChildNodes()	Принимает истинное значение, если данный узел имеет дочерние элементы
Node insertBefore(Node newChild, Node refChild)	Включает новый узел перед существующей ссылкой на дочерний узел
boolean isSupported(String feature, String version)	Принимает истинное значение, если реализуется указанное свойство
void normalize()	Помещает все текстовые узлы в «нормальную» XML-форму
Node removeChild(Node oldChild)	Удаляет дочерний узел, возвращая его при этом
Node replaceChild(Node newChiId, Node oldChild)	Заменяет дочерний узел в списке дочерних узлов и возвращает «старый» дочерний узел
Void setNodeValue(String nodeValue)	Устанавливает значение узла
void setPrefix(String prefix)	Устанавливает префикс пространства имен для данного узла

Для дальнейшей обработки xml-документа необходимо работать с корневым элементом, использую функцию getDocumentElement. Для работы с внутренними тегами корневого элемента можно воспользоваться функцией getChildNodes, которая возвращает объект типа NodeList.

Пример работы с корневым элементом:

```
Element root=doc.getDocumentElement();
System.out.println("корень="+root.getNodeName());
String sroot=root.getTagName();
NodeList list = root.getChildNodes();
System.out.println("list="+list.getLength());
```

В результате работы функции getDocumentElement будет получен элемент flowers.

Интерфейс NodeList поддерживает упорядоченный набор узлов. Функциональность интерфейса определена функциями интерфейса List:

int getLength() - определяет количество узлов в списке;

Node item(int index) - обеспечивает доступ к элементу по индексу index в данном наборе.

В дальнейшем можно перебирать все элементы списка типа NodeList, используя один из вариантов цикла [7]:

```
for (int i = 0; i < list.getLength(); i++)
Node nd=list.item(i);
```

или

```
for (Node nn : list)
Node nd=nn;
```

Пример разбора документа.

```
import java.io.File;
import java.io.IOException;
import javax.xml.parsers.DocumentBuilder;
```

```
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;
import javax.xml.soap.Text;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;
import org.xml.sax.SAXException;
public class PR_DOM {
        public static void main(String[] args) throws Exception{
                DocumentBuilderFactory factory =
                        DocumentBuilderFactory.newInstance();
                        DocumentBuilder builder = factory.newDocumentBuilder();
                        File fl=new File("pr.xml");
                        Document doc = builder.parse(fl);
                        Element root=doc.getDocumentElement();
                        System.out.println("корень="+root.getNodeName()); //
                String sroot=root.getTagName();
                        NodeList list = root.getChildNodes();
                        System.out.println("list="+list.getLength());
                        for (int i = 0; i < list.getLength(); i++)
                        {
                                Node nd=list.item(i);
                                if(nd.getNodeType()==Node.ELEMENT_NODE)
                                {Element fl1=(Element)nd;
                                System.out.println("цветок="+fl1.getNodeName());
                                 if(fl1.getTagName()=="rose")
                                    {System.out.println("a1="+fl1.getAttribute("id"));
                           System.out.println("a2="+fl1.getAttribute("id_pr"));
                           NodeList H=fl1.getElementsByTagName("height");
```

```
                String Hflow="";
                if(H.getLength()>0) Hflow=H.item(0).getTextContent();
                        System.out.println("высота="+Hflow);
                    }
                NodeList ns=fl1.getElementsByTagName("sort");
                for (int j = 0; j < ns.getLength(); j++)
                { String sflow=ns.item(j).getTextContent();
                        System.out.println("сорт="+sflow);
                }
                }
                        }

        }
        }
```

При работе с полученным узлом необходимо учитывать тип обрабатываемого узла. Для уточнения типа узла можно воспользоваться методом узла getNodeType. Для обработки различных типов узлов удобно применить оператор множественного выбора switch:

```
int type = nd.getNodeType();
switch (type) {
case Node.DOCUMENT_NODE: {
//
break;
}
case Node. ELEMENT_NODE: {
int length = (nd.getAttributes() != null) ?
        nd.getAttributes().getLength() : 0;
Attr []attributes= new Attr [length];
for (int k = 0; k < length; k++) {
attributes[k] = (Attr)nd.getAttributes().item(k);
```

}

Таблица 2.5 - Методы интерфейса Attr

Метод	Описание
String getName()	Получение имени данного атрибута
Element getOwnerElement ()	Получение узла Element, к которому присоединен данный атрибут
boolean getSpecified()	Принимает истинное значение, если данный атрибут наверняка получает значение в исходном документе
String getValue()	Получает значение атрибута в виде строки
void setValue(String value)	Устанавливает значение атрибута в виде строки

Поскольку интерфейс Attr построен на основе интерфейса Node, для получения названия атрибута и значения можно воспользоваться либо методами getNodeName и getNodeValue, либо методами Attr, getName и getValue. В данном случае применяются методы getNodeName и getNodeValue.

В модели W3C DOM указывается, что текст в элементах следует сохранять в текстовых узлах, причем эти узлы имеют тип Node.TEXT_NODE. Для этих узлов в отображаемую строку добавляется строка текущих отступов, а затем с помощью метода trim Java-объекта String убираются начальные и завершающие пропуски:

```
String newText = nd.getNodeValue().trim();
```

Проверка наличия только текста может быть записана следующим образом:

```
if(newText.indexOf("\n") < 0 && newText.length() > 0) {
// необходимые действия
}
```

Методы интерфейса NamedNodeMap приведены в табл. 2.6.

Таблица 2.6 - Методы интерфейса NamedNodeMap

Метод	Описание
Int getLength()	Возврат количества узлов для данной карты
Node getNamedItem(String name)	Получение узла с указанным именем
Node getNamedltemNS(String namespaceURI, String localName)	Получение узла, для которого указаны атрибуты namespaceURL и localName
Node item (int index)	Получение элемента карты по заданному индексу
Node removeNamedItem (String name)	Удаление узла с указанным именем
Node removeNamedltemNS(String namespaceURI, String local Name)	Удаление узла, для которого указаны атрибуты namespaceURL и 1 ocal Name
Node setNamedltem(Node arg)	Добавление узла, определенного атрибутом nodeName
Node setNamedltemNS(Node arg)	Добавление узла, определенного атрибутами namespaceURL и localName

Интерфейс Node предоставляет в распоряжение пользователя множество методов по обновлению документов путем добавления или исключения узлов. В их число входят appendChild, insertBefore, removeChild, replaceChild и т. д. Эти методы обеспечивают изменение XML-документов «на лету».

Этапы создания нового XML-документа следующие:

1) создаем объекты типа DocumentBuilderFactory, DocumentBuilder, Document. Эти действия можно проводить отдельно как было показано ранее при разборе существующего элемента, а можно объединить

```
Document doc=
DocumentBuilderFactory.newInstance().newDocumentBuilder().newDocument();
```

2) создаем корневой элемент и добавляем его к объекту Document

```
// Корневой элемент
Element fls = doc.createElement("flowers");
```

```
doc.appendChild(fls);
```

3) создаем следующий элемент и добавляем его к созданному ранее элементу

```
// Элемент типа rose
Element rs = document.createElement("rose ");
fls.appendChild(rs);
```

4) для атрибута создаем объект типа Attr, задаем его значение и добавляем его к созданному ранее элементу

```
// Определяем идентификатор сотрудника
Attr id = document.createAttribute("id");
id.setTextContent("1");
rs.setAttributeNode(id);
```

или

```
rs.setAttribute("id", "1");
```

5) продолжаем процесс для всех данных;

6) для сохранения текстового представления в файл

```
Transformer transformer = TransformerFactory.newInstance().newTransformer();
DOMSource source = new DOMSource(doc);
StreamResult result =
    new StreamResult(new File(“Имяфайла.xml”));
transformer.transform(source, result);
```

7) для вывода на консоль необходимо задать

```
StreamResult result = new StreamResult(System.out);
```

8) возможность добавления дополнительных пробелов при выводе определяется таким образом

```
transformer.setOutputProperty(OutputKeys.INDENT, "yes");
```

2.3. SAX-анализатор

По существу, обработка XML-документов осуществлялась в стиле, представленном в простом интерфейсе прикладного программирования для XML (Simple API for XML, SAX). Вместо того чтобы выполнять навигацию по всему документу, пользователь получает его в исходном виде. Затем вызываются на выполнение различные конструкции case, обрабатывающие разные узлы документа. Именно эта задача выполняется с помощью SAX [2]. Этот интерфейс основан на событии. Если анализатор SAX встречает элемент или инструкцию по обработке, то воспринимает их в качестве событий, вызывая соответствующий код. При этом не требуется выполнять навигацию по всему документу —документ сам «приходит» к вам. Простота и скорость работы интерфейса SAX объясняются тем, что выполняемые им операции ограничены чтением, а методики DOM реализуют операции чтения/записи.

Для работы необходимо подключить библиотеки

```
import org.xml.sax.helpers.DefaultHandler;
import org.xml.sax.*;
```

Класс DefaultHandler включает набор методов, предварительно определенных и вызываемых анализатором SAX, включая следующие методы обратного вызова:

- startDocument вызывается, если встречается начало документа;
- endDocument вызывается, если встречается конец документа;
- startElement вызывается, если встречается открывающий тег элемента;
- endEl ement вызывается, если встречается закрывающий тег элемента;
- characters вызывается, если XML-анализатор считывает текстовые символы.

Все необходимые методы обратного вызова реализованы в классе DefaultHandler, но они находятся в «спящем состоянии». Это означает, что

требуется реализовать применяемые методы, например startDocument (обнаружение начала документа) или endDocument (обнаружение конца документа). Метод startElement вызывается всякий раз, когда анализатор SAX встречает начало элемента, а метод endEl ement вызывается в том случае, если анализатор SAX «видит» завершение элемента. Обратите внимание на то, что методу startElement передаются два имени элементов: local Name и qual if iedName. Аргумент local Name применяется при обработке пространства имен; этот аргумент содержит имя элемента без какого-либо префикса пространства имен. Аргумент qualifiedName включает полное уточненное имя элемента, включая префикс пространства имен.

Методы, включенные в класс DefaultHandler, приведены в табл.2.7.

Таблица 2.7 - Методы класса DefaultHandler

Метод	Описание
DefaultHandler()	Конструктор класса
void characters(char[] ch, int start, int length)	Обработка символьных данных внутри элемента
Void endDocument ()	Обработка конца документа
void endElement (String uri, String local Name, String qName)	Обработка конца элемента
void endPrefixMapping (String prefix)	Обработка конца отображения пространства имен
void error(SAXParseException e)	Обработка восстанавливаемой ошибки анализатора
void fatal Error (SAXParseException e)	Отчет о фатальной ошибке во время разбора
void ignorableWhitespace(char[] ch, int start, int length)	Обработка игнорируемых пропусков в содержимом элемента (применяемых для формирования отступов в документе)

void notationDecl (String name, String publicld, String systemld)	Обработка объявления нотации
void processinglnstruction(String target, String data)	Выполнение XML-инструкции по обработке (например, JSP-директивы)
InputSource resolveEntity(String publicld, String systemld)	Разрешение внешней сущности
void setDocumentLocator (Locator locator)	Установка для событий документа объекта Locator
void skippedEntity(String name)	Обработка пропущенной XML-сущности
void startDocument()	Обработка начала документа
void startElement(String uri, String local Name, String qName,Attributes attributes)	Обработка начала элемента
void startPrefixMapping(String prefix, String uri)	Обработка начала отображения пространства имен
void unparsedEntityDecl (String name, String publicld. String systemld, String notationName)	Обработка неразобранного объявления сущности
void warning(SAXParseException e)	Обработка предупреждения анализатора

Для создания анализатора SAX используется Java-класс SAXParserFactory (фабрика), с помощью которого создается объект из класса SAXParser. Фактически синтаксический разбор выполняется с помощью метода parse объекта SAXParser. Этот метод передается объекту, чьи методы должны вызываться в случае, когда анализатор находит начало XML-элемента, завершение элемента, а также другие составные компоненты, приведенные в табл.2.8 - 2.10.

Таблица 2.8 - Методы интерфейса java.xml.parsers.SAX ParserFactory

Метод	Определение
protected SAXParserFactory()	Заданный по умолчанию конструктор
abstract boolean getFeature (String name)	Возвращает определенное запрашиваемое свойство
boolean isNamespaceAware()	Принимает истинное значение, если фабрика производит анализаторы, использующие пространства имен XML
boolean isValidating ()	Принимает истинное значение, если фабрика производит анализаторы, проверяющие действительность XML-содержимого
static SAXParserFactory newInstance()	Получение нового объекта SAXParser
abstract SAXParser newSAXParser()	Создание нового объекта SAXParser
abstract void setFeature(String name, boolean value)	Установка определенного запрашиваемого свойства
void setNamespaceAware (boolean awareness)	Запрашивает анализатор, используемый для поддержки пространства имен XML
void setValidating (boolean validating)	Запрашивает анализатор, используемый для проверки действительности XML-документов

Таблица 2.9 - Методы класса SAXParser

Метод	Описание
protected SAXParser ()	Заданный по умолчанию конструктор
abstract Parser getParser()	Возвращает анализатор SAX
abstract Object getProperty (String name)	Возвращает определенное запрашиваемое свойство
abstract XMLReader getXMLReader()	Возвращает применяемый объект

	XMLReader
abstract boolean isNamespaceAware()	Принимает истинное значение, если данный анализатор сконфигурирован для представления пространств имен
abstract boolean isValidating()	Принимает истинное значение, если анализатор сконфигурирован для проверки действительности XM L-документов
void parse (File f, DefaultHandler dh)	Анализирует содержимое файла, указанного с помощью определенного объекта DefaultHandler
void parse(File f, HandlerBase hb)	Анализирует содержимое файла, указанного с помощью определенного объекта HandlerBase
void parse(InputSource is, DefaultHandler dh)	Анализирует содержимое, указанное InputSource с помощью определенного объекта DefaultHandler
void parse(InputSource is, HandlerBase hb)	Анализирует содержимое, указанное InputSource с помощью определенного объекта HandlerBase
void parse(InputStream is, DefaultHandler dh)	Анализирует содержимое указанного экземпляра InputStream с помощью определенного объекта DefaultHandler
void parsednputStream is, DefaultHandler dh, String systemld)	Анализирует содержимое указанного экземпляра InputStream с помощью определенного объекта DefaultHandler
void parse(InputStream is, HandlerBase hb)	Анализирует содержимое указанного экземпляра InputStream с помощью определенного объекта HandlerBase
void parse(InputStream is, HandlerBase	Анализирует содержимое указанного

hb, String systemld)	экземпляра InputStream с помощью определенного объекта HandlerBase и системного ID
void parse(String uri, DefaultHandler dh)	Анализирует содержимое, описанное данным универсальным идентификатором (URI) с помощью определенного объекта DefaultHandler
void parse(String uri, HandlerBase hb)	Анализирует содержимое, описанное указанным URI с помощью некоторого объекта HarfdlerBase
abstract void setProperty(String name, Object value)	Устанавливает определенное свойство объектаXMLReader

Таблица 2.10 - Методы интерфейса Attributes

Метод	Описание
int getlndex (java.lang. String quali fiedName)	Получение индекса атрибута, представленного уточненным именем
int getlndex(String uri, String localPart)	Получение индекса атрибута с помощью пространства имен и локального имени
int getLength()	Получение количества атрибутов в списке
String getLocalName (int index)	Получение локального имени атрибута по индексу
String getQName (int index)	Получение уточненного имени атрибута по индексу
String getType (int index)	Получение типа атрибута по индексу
String getType(String qualifiedName)	Получение типа атрибута по уточненному имени
String getType(String uri, String local	Получение типа атрибута на основе

Name)	пространства имен и локального имени
String getURI (int i ndex)	Получает URI пространства имен атрибута на основе индекса
String getValue (int index)	Получает значения атрибута на основе индекса
String getValue(String qualifiedName)	Получение значения атрибута на основе уточненного имени
String getValue(String uri, String local Name)	Получение значения атрибута на основе названия пространства имен и локального имени

Количество атрибутов определяется с помощью метода getLength интерфейса Attributes. Названия и значения атрибутов определяются с помощью методов getLocalName и getValue, причем индекс применяется для установки ссылок на атрибуты. Обратите внимание на то, что сначала проверяется наличие атрибутов путем тестирования значения аргумента attributes (не должен быть равным нулю).

Интерфейс DefaultHandler определяет несколько методов обратного вызова, применяемых для обработки предупреждений и ошибок. В число этих методов входят: warning (обработка предупреждений анализатора), error (обработка ошибок анализатора), а также fatal Error (обработка серьезных ошибок, нарушающих функционирование анализатора). Каждый из этих методов передает объект класса SAXParseException.

2.4. JDOM-анализатор

JDOM является уникальным Java-инструментом для работы с XML, он создан для обеспечения быстрой разработки XML-приложений. В его проекте использованы синтаксис и семантика языка Java.

JDOM может использоваться как альтернатива пакету org.w3c.dom для программного манипулирования XML-документами. Анализаторы JDOM и DOM могут использоваться одновременно. JDOM не занимается разбором исходного текста XML, хотя он обеспечивает классы-оболочки, которые берут на себя большую часть работы по конфигурированию и выполнению реализации парсера. JDOM использует сильные стороны существующих API.

Сначала рассмотрим ограничения W3C DOM:

- независимость от языка. DOM не была разработана только для языка Java. Хотя этот подход сохраняет очень похожий API для разных языков, он также делает этот API более громоздким для программистов, которые используют стиль языка Java. Например, хотя язык Java имеет встроенный в язык класс String, спецификация DOM определяет собственный класс Text;
- строгая иерархия. API DOM следует непосредственно самой спецификации XML. В XML все является узлами, в DOM интерфейс на базе узлов, где почти все строится на методах, которые возвращают Node. Это просто с точки зрения полиморфизма, но трудно и громоздко работать с этим в языке Java, где явное схождение от Node к типам листьев приводит к громоздкому и плохо понимаемому коду;
- управляемый интерфейсом. DOM API состоит только из интерфейсов (единственным, вполне достаточным исключением является класс Exception). В сферу интересов W3C не входит обеспечение реализаций, только определение интерфейсов, что имеет смысл. Но это также означает, что использующий API Java-программист при создании объектов XML в некоторой степени ущемлен, так как стандарты W3C затрудняют использование классов родовых фабрик и подобных гибких, но менее непосредственных шаблонов. Для определенных вариантов использования, когда XML-документы строятся только парсером и никогда - прикладным кодом, это не имеет значения. Но когда использование XML становится более широким, не все проблемы продолжают оставаться управляемыми только парсером, и разработчики

приложений нуждаются в удобном способе конструировать объекты XML программно.

Для программистов эти ограничения означают тяжелый и громоздкий API, который может быть трудно изучить и использовать. Напротив, JDOM сформулирован как легкий API, прежде всего ориентированный на Java.

Преимущества JDOM:

- JDOM специализирован для платформы Java. API использует, где возможно, встроенную в Java поддержку String, так что текстовые значения всегда доступны как String. Он также использует классы коллекций платформы Java 2, такие как List и Iterator, обеспечивая богатую среду для работы программистов, хорошо знакомых с языком Java;
- нет иерархий. В JDOM элемент XML является экземпляром класса Element, атрибут XML является экземпляром класса Attribute, а сам XML-документ является экземпляром класса Document. Поскольку все они представляют разные концепции в XML, они всегда представляются собственными типами, а не как аморфные "узлы";
- управляемый классом. Поскольку объекты JDOM являются непосредственными экземплярами таких классов, как Document, Element и Attribute, создание их настолько легко, насколько легко использование оператора new языка Java. Это также означает, что нет необходимости в интерфейсе фабрики для конфигурирования - JDOM готов к использованию прямо из jar.

JDOM использует стандартные шаблоны кодирования Java: использует оператор Java new вместо сложных шаблонов фабрик, делая манипулирование объектами легким даже для начинающего. Например, необходимо построить простой XML-документ с нуля, используя JDOM.

```
<?xml version="1.0" encoding="UTF-8"?>
<car vin="123fhg5869705iop90">
  <!--Description of a car-->
  <make>Toyota</make>
```

```
  <model>Celica</model>
  <year>1997</year>
  <color>green</color>
  <license state="CA">1ABC234</license>
</car>
```

Сначала создадим корневой элемент и добавим его в документ:

```
Element carElement = new Element("car");
Document myDocument = new Document(carElement);
```

Этот шаг создает новый элемент org.jdom.Element и делает его корневым элементом org.jdom.Document myDocument. Поскольку XML-документ должен всегда иметь единственный корневой элемент, Document принимает в своем конструкторе Element.

Добавляем атрибут vin:

```
carElement.addAttribute(new Attribute("vin", "123fhg5869705iop90"));
```

Добавляем элемент make:

```
Element make = new Element("make");
make.addContent("Toyota");
carElement.addContent(make);
```

Поскольку метод addContent класса Element возвращает Element:

```
carElement.addContent(new
Element("make").addContent("Toyota"));
```

Завершим конструирование документа:

```
carElement.addContent(new
Element("model").addContent("Celica"));
carElement.addContent(new Element("year").addContent("1997"));
carElement.addContent(new Element("color").addContent("green"));
carElement.addContent(new Element("license")
   .addContent("1ABC234").addAttribute("state", "CA"));
```

Методы addContent класса Element всегда возвращают сам Element, а не имеют декларацию void. Добавление комментария или других стандартных типов XML проделывается тем же способом:

```
carElement.addContent(new Comment("Description of a car"));
```

Манипулирование документов выполняется в том же стиле. Например, чтобы получить ссылку на элемент year, используем метод getChild класса Element:

```
Element yearElement = carElement.getChild("year");
```

Этот оператор будет возвращать первый дочерний Element с именем элемента year. Если элемента year нет, вызов вернет null. Можно удалить из документа элемент year:

```
boolean removed = carElement.removeChild("year");
```

Этот вызов удалит только элемент year; остальная часть документа останется неизмененной.

Для вывода окончательного документа на консоль используется класс JDOM XMLOutputter:

```
try {
   XMLOutputter outputter = new XMLOutputter("  ", true);
   outputter.output(myDocument, System.out);
} catch (java.io.IOException e) {
   e.printStackTrace();
}
```

XMLOutputter имеет несколько опций форматирования: задано, что дочерние элементы отступали на два пробела от родительских элементов; что есть переводы строки между элементами. XMLOutputter может выводить либо во Writer, либо в OutputStream. Чтобы выводить в файл, необходимо изменить строку вывода на:

```
FileWriter writer = new FileWriter("/some/directory/myFile.xml");
outputter.output(myDocument, writer);
writer.close();
```

Одним из интересных свойств JDOM является его функциональная совместимость с другими API. Используя JDOM, можно выводить документ не только в Stream или в Reader, но также и в поток событий SAX или в DOM Document. Эта гибкость позволяет JDOM использоваться в гетерогенной среде или добавляться в систему, уже применяющую другие методы для обработки XML.

Другое использование JDOM состоит в возможности читать и манипулировать уже существующими XML-данными. Чтение правильно форматированного XML-файла выполняется при помощи одного из классов в org.jdom.input. В данном примере применен SAXBuilder:

```
try {
  SAXBuilder builder = new SAXBuilder();
  Document anotherDocument =
    builder.build(new File("/some/directory/sample.xml"));
} catch(JDOMException e) {
  e.printStackTrace();
} catch(NullPointerException e) {
  e.printStackTrace();
}
```

2.5. Вопросы для самоконтроля

1. Что такое DOM?

2. Какой объект представляет верхний уровень объектной иерархии и содержит методы для работы с документом?

3. Какой объект предназначен для манипулирования с отдельным узлом дерева документа?

4. Какой объект представляет собой список узлов – поддеревья и содержит методы, при помощи которых можно организовать процедуру обхода дерева?

5. Какой объект позволяет получить всю необходимую информацию об ошибке, произошедшей в ходе разбора документа?

6. Чем задается иерархия узлов в DOM?

7. С помощью какого метода осуществляется ссылка на начальный объект в "дереве" документа?

8. Какой метод используется для создания нового узла элемента с указанным именем?

9. Какой метод добавляет объект Node в конец списка узлов-потомков?

10. Какой метод объекта Document используется для получения коллекции всех элементов, соответствующих какому-либо тегу?

11. Какой метод создает текстовый узел с указанным текстом?

12. Какой метод создает новый узел атрибута с указанным именем?

13. Что из перечисленного формирует, так называемые, узлы DOM?

14. Какие узлы являются базовыми строительными блоками XML?

15. Какие узлы содержат информацию об элементном узле, но не рассматриваются как потомки элемента?

16. Какой узел является общим предком для всех других узлов в документе?

17. Что такое XML DOM?

18. Какой метод вставляет новый дочерний узел перед существующим дочерним узлом?

19. На сколько частей / уровней разделяется DOM?

20. Что выполняет указанный фрагмент кода?

21. Как называется структура дерева?

22. Что выполняет указанный фрагмент кода?

```
var x = xmlDoc.getElementsByTagName("title")[0].childNodes[0];
x.nodeValue = "Easy Cooking";
```

23. Что выполняет указанный фрагмент кода?

x = xmlDoc.getElementsByTagName('title').length;

24. Что выполняет указанный фрагмент кода?

x = xmlDoc.getElementsByTagName('book')[0].attributes;

25. Какой фрагмент кода создает элемент (<edition>) со значением, и добавляет его после последнего ребенка первого <book> элемента?

26. Как в DOM XML можно получить доступ к узлам?

27. В каком фрагменте кода происходит удаление узла из дерева?

28. Каким консорциумом стандартизирован DOM?

29. Какой метод относится к объекту Document?

30. В XML DOM отношения между узлами определены в виде каких свойств узлов?

Тема 3. XSLT-преобразования и поиск информации

3.1. Проведение трансформации xml-документов

XSL (Extensible Stylesheet Language) язык таблиц стилей. Язык XSL фактически состоит из двух частей: языка преобразований и языка форматирования. Язык, предназначенный для выполнения преобразований, позволяет конвертировать структуры документов в различные формы (например, PDF, WML, HTML или иные типы схем), в то время как язык форматирования используется для оформления и определения стилей документов различными способами. Обе части языка XSL могут функционировать совершенно независимо одна от другой, поэтому их можно рассматривать в качестве независимых языков разметки. На практике перед выполнением форматирования производится преобразование документа, поскольку при этом добавляются необходимые в процессе форматирования теги.

Язык преобразований XSL часто называют XSLT, что соответствует рекомендациям W3C, опубликованным 16 ноября 1999 года. Рекомендации консорциума W3C для XSLT (текущая версия 1.0) можно найти на веб-узле по адресу www.w3.org/TR/xslt.

Выполнить преобразования документа возможно тремя способами:

- с помощью сервера — серверная программа, например Java или JavaServer Page (JSP), применяет таблицу стилей для автоматического преобразования документа и представления его клиенту;
- с помощью клиента — клиентская программа, например браузер, выполняет преобразование путем чтения таблицы стилей, указанной с помощью инструкции по обработке <?xml-stylesheet?>. Эти функции может выполнять Internet Explorer (в некотором объеме);

- с помощью отдельной программы — некоторые программы, обычно основанные на Java, предназначены для выполнения XSLT-преобразования.

Существует возможность проводить преобразование с помощью программирования. Например, на языке Java можно воспользоваться классом TransformerFactory для формирования нового объекта из класса Transformer.

Методы класса TransformerFactory показаны в табл. 3.1, а методы класса TransTransformer - в табл. 3.2.

Таблица 3.1. Методы класса TransformerFactory

Метод	Описание
protected TransformerFactory ( )	Заданный по умолчанию конструктор
abstract Source getAssociatedStylesheet(Source source, String media, String title. String charset)	Получает спецификации таблицы стилей
abstract Object getAttribute (String name)	Возвращает определенные атрибуты
abstract ErrorLi stener getErrorListener()	Возвращает обработчик ошибочного события для TransformerFactory
abstract boolean getFeature (String name)	Возвращает значение свойства
abstract URIResolver getURIResolver()	Получает объект, используемый по умолчанию в процессе преобразования для разрешения URI-ссылок
static TransformerFactory newInstance()	Получает новый объект TransformerFactory
abstract Templates newTemplates (Source source)	Обрабатывает Source в объекте Templates (компилированное представление для источника)
abstract Transformer newTransformer()	Создает новый объект Transformer, который использует источник для

	преобразований
abstract Transformer newTransformer(Source source)	Создает новый объект Transformer, который применяет Source для выполнения преобразования
abstract void setAttribute(String name, Object value)	Устанавливает определенные атрибуты
abstract void setErrorListener (ErrorListener listener)	Устанавливает слушатель событий, связанных с ошибками в Java
abstract void setURIResolver(URIResolver resolver)	Устанавливает объект, применяемый по умолчанию во время преобразования для разрешения URI-ссылок

Таблица 3.2 - Методы класса javax.xml.transform.Transformer

Метод	Описание
protected Transformer()	Заданный по умолчанию конструктор
abstract void clearParameters()	Очищает все параметры, установленные с помощью setParameter
abstract ErrorListener getErrorListener()	Возвращает обработчик сообщения об ошибке
abstract Properties getOutputProperties()	Возвращает копию выводимых свойств
abstract String getOutputProperty(String name)	Возвращает значение выводимого свойства
abstract Object getParameter (String name)	Возвращает параметр, установленный с помощью setParameter либо setParameters
abstract URIResolver getURIResolver()	Возвращает объект, используемый

	для разрешения URI-ссылки
abstract void setErrorListener(ErrorListener listener)	Устанавливает слушатель событий об ошибке Java
abstract void setOutputProperties(Properties oformat)	Устанавливает выводимые свойства
abstract void setOutputProperty(String name, String value)	Устанавливает выводимое свойство
abstract void setParameter(String name, Object value)	Устанавливает параметр
abstract void setURIResolver(URIResolver resolver)	Устанавливает объект, применяемый для разрешения URI-ссылки
abstract void transform(Source xmlSource. Result outputTarget)	Выполняет преобразование

Например, можно применить преобразование таким образом [3]:

```
try
{
TransformerFactory transformerfactory
  =TransformerFactory.newInstance();
Transformer transformer =transformerfactory.newTransformer
    (new StreamSource(new File("ИмяФайлаПреобразования.xsl")));
transformer.transform(new StreamSource
   (new File("ИмяИсходногоФайла.xml")),
    new StreamResult(new File("ИмяРезультата.html")));
}
catch(Exception e) {}
```

В рассматриваемом случае файлы обрабатываются с помощью методов из классов StreamSource и StreamResult, что соответствует требованиям,

выдвигаемым классом Transformer. Методы класса StreamSource показаны в табл. 3.3, а методы класса StreamResult - в табл. 3.4.

Таблица 3.3 - Методы класса java.xml.transform.stream Stream Source

Метод	Описание
StreamSource()	Заданный по умолчанию конструктор
StreamSource(File f)	Конструирует StreamSource на основе объекта File
StreamSource(InputStream StreamSource inputStream)	Конструирует StreamSource на основе байтового потока
StreamSource(InputStream StreamSource, inputStream, String systemId)	Конструирует StreamSource на основе байтового потока
StreamSource(Reader reader)	Конструирует StreamSource на основе объекта-читателя символов
StreamSource(String systemld)	Конструирует StreamSource на основе URL-ссылки
InputStream getlnputStream()	Получает байтовый поток, установленный с помощью setByteStream
String getPublicId()	Получает общедоступный идентификатор, установленный с помощью setPublicId
Reader getReader ()	Получает поток символов, который установлен с помощью setReader
String getSystemId()	Получает идентификатор системы, который установлен с помощью setSystemID
void set InputStream (InputStream inputStream)	Устанавливает поток байтов, применяемый в качестве вводных

	данных
void setPublicId(String publicId)	Устанавливает общедоступный идентификатор для данного Source
void setReader (Reader reader)	Устанавливает, что вводные данные служат для чтения символов
void setSystemId(File f)	Устанавливает идентификатор системы на основе ссылки File
void setSystemId(String systemld)	Устанавливает идентификатор системы для данного Source

Таблица 3.4 - Методы результирующего класса javax.xml.transform.StreamResult

Метод	Описание
StreamResult()	Заданный по умолчанию конструктор
StreamResult (File f)	Конструирует StreamResult на основе объекта File
StreamResult(OutputStream outputStream)	Конструирует StreamResult на основе байтового потока
StreamResult (String systemld)	Конструирует StreamResult на основе URL-ссылки
StreamResuit(Writer writer)	Конструирует StreamResult на основе потока символов
OutputStream getOutputStream()	Получает поток байтов, устанавливаемый с помощью setOutputStream
String getSystemld()	Получает идентификатор системы, устанавливаемый с помощью setSystemld
Writer getWriter()	Получает поток символов, устанавливаемый с помощью setWriter
void setOutputStream(OutputStream outputStream)	Устанавливает записываемый ByteStream

void setSystemld (File f)	Устанавливает идентификатор системы на основе ссылки File
void setSystemld (String systemID)	Устанавливает SystemID, который может применяться при ассоциации с потоком байтов или символов
void setWriter (Writer writer)	Устанавливает объект писателя, который применяется при получении результата

3.2. Определение правил преобразования

В процессе XSLT-преобразований в качестве вводных данных используется дерево документа, в результате обработки которого формируется дерево результатов. В этом случае документы представляют собой деревья, построенные на основе узлов. Язык XSLT распознает семь типов XSLT-узлов. Эти узлы представлены в следующем списке, где также указан порядок их обработки XSLT-процессорами:

- корень документа — начало документа;
- атрибут — данный узел включает значение атрибута наравне с объектными ссылками, которые могут расширяться, и пропусками, которые могут удаляться;
- комментарий — данный узел включает текст комментария без знаков <!-- и -->;
- элемент — этот узел содержит все символьные данные элемента, включая символьные данные любого дочернего элемента;
- пространство имен — данный узел содержит URI пространства имен;
- инструкция по обработке — этот узел включает текст инструкции по обработке, куда не входят символы <? и ?>;

- текст — данный узел содержит текст узла.

Для определения обрабатываемого узла (узлов) XSLT предлагает различные способы, обеспечивающие установку соответствия с узлами или их выбора. Например, символ /означает корневой узел. Для начала разработаем небольшой пример, заменяющий корневой узел (а следовательно, весь документ) HTML-страницей.

Таблицы стилей XSLT должны представлять собой формально корректные XML-документы, поэтому создание таблицы стилей начнем с XML-объявления. Затем воспользуемся элементом <stylesheet>. Вообще говоря, таблицы стилей XSLT используют пространство имен xsl, которое после завершения стандартизации языка XSLT соответствует пространству имен www.w3.org/1999/XSL/Transform. Необходимо также в элемент <stylesheet> включить атрибут version, присваивая ему номер текущей версии, 1.0:

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

Для обработки определенных узлов XML-документа XSLT использует шаблоны. После установки соответствия с узлом (или узлами) к нему XSLT-процессор применяет шаблон, который преобразует узел, формируя вывод. С помощью элемента <xsl:template> в таблице стилей устанавливается шаблон (правило).

Например, для замены корневого элемента XML-файла на HTML-файл необходимо применить xsl-файл (назовем его *XSLHTML.xsl)*

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

```
<HEAD>
<TITLE>
Элементарные преобразования
</TITLE>
</HEAD>
<BODY>
Это содержимое документа
</BODY>
</HTML>
</xsl:template>
</xsl:stylesheet>
```

Можно заменить шаблон

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

шаблоном с названием корневого элемента

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

Для проверки работы преобразования воспользуемся браузером. Для этого в xml-файл допишем строку

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

Например:

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xml" href="XSLHTML.xsl"?>
<PLANETS>
<PLANET>
<NAME>Mercury</NAME>
<MASS UNITS="(Earth-1)">.0553</MASS>
<DAY UNITS="days">58.65</DAY>
<RADIUS UNITS="miles">1516</RADIUS>
<DENSITY UNITS="(Earth-1)">.983</DENSITY>
<DISTANCE UNITS="million mnes">43.4</DISTANCE>
</PLANET>
```

```
<PLANET>
<NAME>Venus</NAME>
<MASS UNITS="(Earth-1)">.815</MASS>
<DAY UNITS="days">116.75</DAY>
<RADIUS UNITS="miles">3716</RADIUS>
<DENSITY UNITS="(Earth =1)">.943</DENSITY>
<DISTANCE UNITS="m1llion miles">66.8</DISTANCE>
</PLANET>
<PLANET>
<NAME>Earth</NAME>
<MASS UNITS="(Earth-1)">1</MASS>
<DAY UNITS="days">l</DAY>
<RADIUS UNITS="miles">2107</RADIUS>
<DENSITY UNITS="(Earth =1)">1</DENSITY>
<DISTANCE UNITS="million miles">128.4</DISTANCE>
</PLANET>
</PLANETS>
```

Элемент <xsl : apply-templates>. Для применения шаблонов по отношению к дочерним элементам узла, с которым установлено соответствие используется элемент <xsl :apply-templates>.

```
<xsl:template match="/">
<HTML>
<xsl:apply-templates/>
</HTML>
</xsl:template>
<xsl:template match="ИмяКорневогоЭлемента">
<P>
Данные
</P>
</xsl:template>
```

Для получения доступа к этим данным можно воспользоваться атрибутом select из элемента <xsl:value-of >.

```
<xsl:template match=" ИмяКорневогоЭлемента">
<xsl:value-of select="ИмяТега"/>
</xsl:template>
```

Получение значений узлов с помощью <xsl:value-of>. Для получения значений используется синтаксис:

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

Тогда для получения значений планет:

```
<?xml version=1.0"?>
<xsl:stylesheet version=1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="PLANETS">
<HTML>
<xsl:apply-templates/>
</HTML>
</xsl:template>
<xsl:template match="PLANET">
<xsl:value-of select="NAME"/>
</xsl:template>
</xsl:stylesheet>
```

Свойства атрибута select во многом схожи со свойствами атрибута match для элемента <xsl: template>, за исключением того факта, что атрибут select обладает более универсальными свойствами. Благодаря этому можно выделить узел или набор узлов из XML-документа.

Атрибут select является атрибутом для элементов <xsl:apply-templates>, <xsl :value-of>, <xsl:for-each> и <xsl :sort>.

Получение значений всех узлов с помощью <xsl:for-each>. Атрибут select элемента <xsl :value-of> выделяет только первый элемент <NAME>. Для

выполнения цикла по всем возможным соответствиям можно воспользоваться элементом <xsl :for-each>.

Например,

```
<?xml version=1.0"?>
<xsl:stylesheet version=1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="PLANETS">
<HTML>
<xsl:apply-templates/>
</HTML>
</xsl:template>
<xsl:template match="PLANET">
<xsl:for-each select="NAME">
<P>
<xsl:value-of select=”.”/>
</P>
</xsl :for-each>
</xsl:template>
</xsl:stylesheet>
```

Для установки соответствия с потомками элементов <PLANET>, применяется выражение "PLANET/NAME", а выражение "PLANET/*/NAME" использовалось для установки соответствия со всеми элементами <NAME>, являющимися потомками элементов <PLANET>. Но существует более простой способ выполнения подобных операций: воспользуйтесь выражением "PLANET//NAME", которое устанавливает соответствие со всеми элементами <NAME>, находящимися внутри элементов <PLANET>, независимо от того, сколько уровней вовлечено в рассмотрение. (Соответствующие элементы называются потомками элемента <PLANET>.) Другими словами, выражение "PLANET//NAME" соответствует "PLANET/NAME", "PLANET//NAME", "PLANET/*/*/NAME" и т. д.:

<xsl:template match=”PLANETS//NAME”>

<H3><xsl:value-of select="."/></H3>

</xsl:template>

3.3. Вопросы для самоконтроля

1. Из скольких частей состоит язык таблиц стилей XS?

2. Для чего используется язык преобразований XSL?

3. Как называют язык преобразований XSL?

4. Выполнить преобразования документа возможно с помощью:

5. Для обработки определенных узлов XML-документа XSLT использует…?

6. С помощью которого тега в таблице стилей устанавливается шаблон?

7. Какое из расширения является расширением результирующего XSL-преобразования?

8. К файлу какого из перечисленных расширений может применяться XSL-преобразование?

9. Как текущая версия рекомендации консорциума W3C для XSLT?

10. С чего начинается файл таблицы стиля?

11. В файле какого расширения хранится таблица стиля

12. Атрибут select является атрибутом для элементов…?

13. Каких элементов нет в языке преобразований XSL?

14. Для применения шаблонов по отношению к дочерним элементам узла, с которым установлено соответствие используется элемент

15. Что нужно написать в теге <xsl:when test="@id = '1'"> на месте @, чтобы дать понять что “id” является атрибутом?

16. Какие атрибуты являются обязательными для элемента sort?

17. При преобразовании с помощью языка java кодировка указывается в:

18. Укажите правильное написание выбор корневого элемента документ:

19. Элемент <xsl:choose> используется вместе с элементами:

20. Какой элемент используется для извлечения значения отобранного XML элемента и добавления его в выходной поток преобразовываемого документа?

Тема 4. Web-сервисы

4.1. Введение в понятие веб-сервисы

Технология Web-сервисов — это технология создания распределенных систем, составленных из взаимодействующих между собой программных продуктов, созданных и работающих на основе различных платформ.

Web-сервисы призваны согласовывать работу больших, состоящих из множества частей приложений, предоставляя для приложений бизнес-функции обмена данными.

Помимо функции обмена данными между различными приложениями и платформами, Web-сервисы могут выступать как повторно-используемые компоненты приложения, предоставляющие разнообразные сервисы — от прогноза погоды до перевода с одного языка на другой.

Web-сервисы представляют собой программные компоненты, имеющие идентификатор URI, и взаимодействие с которыми осуществляется по Интернету с помощью открытых протоколов [1].

Коммуникация с Web-сервисами может выполняться с помощью различных транспортных протоколов, таких как HTTP, HTTPS, FTP, SMTP, BEEP, при этом Web-сервисы можно подразделить на три вида: SOAP Web-сервисы, ориентированные на модель RPC — вызов удаленных процедур, XML Web-сервисы, ориентированные на сообщения, и RESTful Web-сервисы.

Первая группа Web-сервисов — это Web-сервисы, взаимодействие с которыми производится с использованием XML-сообщений по SOAP-протоколу (Simple Object Access Protocol), и имеющие интерфейсы, описанные в формате WSDL (Web Services Description Language). Такое описание интерфейса сервиса обеспечивает автоматическую генерацию кода на клиентской стороне, необходимого для связи с сервисом. Описание WSDL Web-сервиса может быть доступно клиенту с помощью реестра UDDI

(Universal Description, Discovery, and Integration), в котором Web-сервис предварительно зарегистрирован. SOAP-протокол может использовать различные транспортные протоколы — HTTP, FTP SMTP и др., однако чаще всего SOAP используется поверх HTTP. SOAP-сообщения, участвующие в обмене между клиентом и SOAP RPC Web-сервисом, имеют строго определенную структуру для передачи имени вызываемой удаленной процедуры и ее параметров, а также результата ее вызова.

Вторая группа Web-сервисов — это XML Web-сервисы, ориентированные на сообщения. Эти XML Web-сервисы обеспечивают низкоуровневую обработку XML-сообщений, при этом Web-сервис обрабатывает полученные XML-данные целиком, как они есть, и полностью формирует ответное XML-сообщение. XML Web-сервисы могут передавать и получать сообщения как в формате SOAP, так и в чистом XML-формате.

Третья группа Web-сервисов — это RESTful Web-сервисы, представляющие удаленные ресурсы, доступные с помощью HTTP-запросов. RESTful Web-сервисы обеспечивают взаимодействие с удаленными ресурсами, передавая клиенту их представление. RESTful Web-сервисы идентифицируются URL-адресом и обрабатывают HTTP-методы GET, PUT, POST и DELETE в ответ на запрос клиента. Технология REST Web-сервисов также может использовать WSDL-описание и SOAP-протокол для передачи сообщений, но может обходиться и без них.

Технология Web-сервисов развивается под эгидой организации W3C.

Модель Service Oriented Model (SOM) описывает сервис и действия. Модель SOM определяет сервис как ресурс, представляющий возможность выполнения задач, реализующих функциональность, которая определена поставщиком сервиса. Чтобы сервис мог использоваться, он должен быть реализован хотя бы одним агентом.

Web-сервис, так же как и Web-ресурс, идентифицируется URI-адресом, однако главное отличие Web-сервиса от Web-ресурса в том, что Web-сервис необязательно должен иметь представление — данные состояния ресурса в

общедоступных форматах XML, HTML, CSS, JPEG, PNG, получаемые с помощью метода HTTP GET.

Web-сервис имеет свое описание, которое представляет собой набор документов, подлежащий компьютерной обработке и содержащий описание интерфейса Web-сервиса и его поведения. Описание Web-сервиса может быть реализовано с помощью языка Service Description Language (SDL) в виде набора XML-документов.

Модель Resource Oriented Model (ROM) описывает ресурсы. Модель ROM определяет ресурс как объект, имеющий имя-идентификатор, свое представление и собственника, обладающего правом назначать политику ресурса, которая определяет правила взаимодействия с ресурсом.

Web-сервисы являются разновидностью ресурсов, которые имеют описание, пригодное для компьютерной обработки и содержащее идентификатор ресурса в виде URI-адреса. С помощью описания потребитель узнает о полезности ресурса и устанавливает связь с ним. Описание облегчает поиск и доступ к ресурсу, включая в себя информацию о местонахождении ресурса, способах доступа к ресурсу и политиках ресурса.

Как было сказано ранее, Web-сервисы — это разновидность ресурсов, отличающаяся от обычных Web-ресурсов тем, что Web-сервисы необязательно должны иметь свое представление — данные состояния ресурса, получаемые с помощью запроса HTTP GET [5].

Модель Policy Model описывает политики сервиса, представляющие собой ограничения на допустимые действия или состояния агентов или их собственников.

Политики могут определять ограничения относительно доступа к ресурсам/состояний ресурсов или возможных действий/состояний агентов поставщиков или потребителей сервисов.

Политики могут быть двух типов:

- политики разрешений;

- политики обязательств.

Политика разрешений позволяет агентам выполнять определенные действия, иметь доступ к определенным ресурсам, достигать определенного состояния. За выполнением политики разрешений следит механизм защиты разрешений, гарантирующий соответствие использования сервиса — политики, установленной поставщиком сервиса.

Политика обязательств выдвигает требования для агентов исполнять определенные действия или достигать определенных состояний. За выполнением политики обязательств следит механизм аудита, проверяющий выполнение обязательств, установленных поставщиком сервиса. Политики могут иметь описание в формате, пригодном для компьютерной обработки. Описание политики определяет ее и используется для решения применения политики к конкретной ситуации.

Web-сервисы являются программными компонентами распределенных приложений, имеющих сервис-ориентированную архитектуру.

Распределенные системы могут создаваться с помощью технологии Web-сервисов, реализующей архитектуру SOA, или технологий COM/CORBA. Однако технологии COM/CORBA не обеспечивают в полной мере кроссплатформенность.

Реализация архитектуры SOA с помощью технологии Web-сервисов более приемлема для создания распределенных приложений, компоненты которых взаимодействуют через Интернет, разворачиваются и работают на различных платформах.

В архитектуре SOA (Service-Oriented Architecture) распределенные системы состоят из взаимодействующих агентов поставщиков и потребителей сервисов, выполняющих определенные задачи.

Агенты распределенной системы не работают в одной и той же среде выполнения и связываются друг с другом с помощью протоколов через сеть.

В распределенных системах скорость выполнения задач зависит от скорости удаленного доступа к агентам, а надежность — от ошибок коммуникации между агентами.

Web-сервисы реализуются в виде программных компонентов на основе какой-либо из платформ, например Microsoft .NET или Java, и разворачиваются на сервере приложений, например IIS или GlassFish. При этом кроссплатформенность взаимодействия с Web-сервисами обеспечивается XML-форматом их описания и XML-форматом сообщений, участвующих в обмене с Web-сервисами. Однако такая архитектура налагает и ограничения на применение технологии Web-сервисов.

Взаимодействие клиента с Web-сервисом может быть синхронным или асинхронным. В случае *синхронного взаимодействия*, после отправки клиентом запроса Web-сервису, все его действия блокируются до тех пор, пока не будет получен ответ. При *асинхронном взаимодействии* действия клиента не блокируются, а обработка ответа производится после его получения от Web-сервиса.

4.2. Протокол SOAP

SOAP - протокол обмена структурированными сообщениями в распределённой вычислительной среде. Поддерживается консорциумом W3C http://www.w3.org/TR/soap/). Протокол SOAP создан в 1998 году командой разработчиков под руководством Дейва

Винера (Dave Winer), работавшей в корпорации Microsoft и фирме Userland, но затем передан в консорциум W3C. Последняя версия стандарта на сегодняшний день - SOAP 1.2. В версии 1.1 SOAP расшифровывался как Simple Object Access Protocol — простой протокол доступа к объектам.

Это название отражало его первоначальное назначение — обращаться к методам удаленных объектов. Сейчас назначение SOAP изменилось, поэтому разные разработчики предлагали свои варианты расшифровки. Поэтому в версии 1.2 аббревиатуру решили никак не расшифровывать.

Протокол SOAP не различает вызов процедуры и ответ на него, а просто определяет формат послания (message) в виде документа XML. Послание может содержать вызов процедуры, ответ на него, запрос на выполнение каких-то других действий или просто текст. Спецификацию SOAP не интересует содержимое послания, она задает только его оформление.

SOAP основан на языке XML и расширяет некоторый протокол прикладного уровня - HTTP, FTP, SMTP и т.д. Как правило чаще всего используется HTTP.

Вместо использования HTTP для запроса HTML-страницы, которая будет показана в браузере, SOAP отправляет посредством HTTP-запроса XML-сообщение и получает результат в HTTP-отклике. Для правильной обработки XML-сообщения процесс-слушатель HTTP (напр. Apache или Microsoft IIS) должен предоставить SOAP-процессор, или, другими словами, должен иметь возможность обрабатывать XML.

SOAP является самой главной частью технологии Web-сервисов. Он осуществляет перенос данных по сети из одного места в другое. SOAP обеспечивает доставку данных веб-сервисов. Он позволяет отправителю и получателю XML-документов поддерживать общий протокол передачи данных, что обеспечивает эффективность сетевой связи.

SOAP – это базовая однонаправленная модель соединения, обеспечивающая согласованную передачу сообщения от отправителя к получателю, потенциально допускающая наличие посредников, которые могут обрабатывать часть сообщения или добавлять к нему дополнительные элементы. Спецификация SOAP содержит соглашения по преобразованию однонаправленного обмена сообщениями в соответствии с принципом «запрос/ответ», а также определяет как осуществлять передачу всего XML-документа.

SOAP предназначен для поддержания независимого абстрактного протокола связи, обеспечивающего коммуникацию двух и более приложений,

сайтов, предприятий и т.п., реализованных на разных технологиях и аппаратных средств.

Общая структура SOAP сообщения SOAP-сообщение представляет собой XML-документ; сообщение состоит из трех основных элементов: конверт (SOAP Envelope), заголовок (SOAP Header) и тело (SOAP Body).

Пример SOAP сообщения:

```
<SOAP-ENV:Envelope
    xmlns:SOAP-ENV="http://www.w3.org/2003/05/soap-envelope"
xmlns:t="www.example.com">
<SOAP-ENV:Header>
</SOAP-ENV:Header>
<SOAP-ENV:Body>
<t:CurrentDate>
<Year>2011</Year>
<Month>February</Month>
<Day>12</Day>
<Time>18:02:00</Time>
</t:CurrentDate>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Конверт (SOAP Envelope) является самым «верхним» элементом SOAP сообщения. Содержит корневой элемент XML-документа. Описывается с помощью элемента Envelope с обязательным пространством имен http://www.w3.org/2003/05/soap-envelope для версии 1.2 и http://schemas.xmlsoap.org/soap/ для версии 1.1.

У элемента Envelope могут быть атрибуты xmlns, определяющие пространства имен, и другие атрибуты, снабженные префиксами. Envelope может иметь необязательный дочерний элемент Header с тем же пространством имен — заголовок. Если этот элемент присутствует, то он должен быть первым прямым дочерним элементом конверта.

Следующий дочерний элемент конверта должен иметь имя Body и то же самое пространство имен - тело. Это обязательный элемент и он должен быть вторым прямым дочерним элементом конверта, если есть заголовок, или первым — если заголовка нет.

Версия 1.1 позволяла после тела сообщения записывать произвольные элементы, снабженные префиксами. Версия 1.2 это запрещает.

Элементы Header и Body могут содержать элементы из различных пространств имен. Конверт изменяется от версии к версии.

SOAP-процессоры, совместимые с версией 1.1, при получении сообщения, содержащего конверт с пространством имен версии 1.2, будут генерировать сообщение об ошибке. Аналогично для SOAP-процессоров, совместимых с версией 1.2. Ошибка — VersionMismatch.

Заголовок SOAP (SOAP Header) Первый прямой дочерний элемент конверта. Не обязательный. Заголовок кроме атрибутов xmlns может содержать 0 или более стандартных атрибутов:

• encodingStyle;

• actor(или role для версии 1.2);

• mustUnderstand;

• relay.

В SOAP-сообщениях могут передаваться данные различных типов (числа, даты, массивы, строки и т.п.). Определение этих типов данных выполняется в схемах XML (обычно — XSD). Типы, определенные в схеме, заносятся в пространство имен, идентификатор которого служит значением атрибута encodingStyle. Атрибут encodingStyle может появиться в любом элементе SOAP-сообщения, но версия SOAP 1.2 запрещает его появление в корневом элементе Envelope. Указанное атрибутом encodingStyle пространство имен будет известно в том элементе, в котором записан атрибут, и во всех вложенных в него элементах. Какие-то из вложенных элементов могут изменить пространство имен своим атрибутом encodingStyle. Стандартное пространство имен, в котором расположены имена типов данных SOAP 1.1,

называется http://schemas.xmlsoap.org/soap/encoding/. У версии 1.2 - http://www.w3.org/2003/05/soap-encoding. Идентификатор того или иного пространства имен, в котором определены типы данных, обычно получает префикс enc или SOAP-ENC.

Атрибут actor Тип данных URI. Задает адрес конкретного SOAP-сервера, которому предназначено сообщение.

SOAP-сообщение может пройти через несколько SOAP-серверов или через несколько приложений на одном сервере. Эти приложения выполняют предварительную обработку блоков заголовка послания и передают его друг другу. Все эти серверы и/или приложения называются SOAP-узлами (SOAP nodes). Спецификация SOAP не определяет правила прохождения послания по цепочке серверов. Для этого разрабатываются другие протоколы, например, Microsoft WS-Routing.

Атрибут actor задает целевой SOAP-узел — тот, который расположен в конце цепочки и будет обрабатывать заголовок полностью. Значение http://schemas.xmlsoap.org/soap/actor/next атрибута actor показывает, что обрабатывать заголовок будет первый же сервер, получивший его. Атрибут actor может встречаться в отдельных блоках заголовка, указывая узел-обработчик этого блока. После обработки блок удаляется из SOAP-сообщения. В версии 1.2 атрибут actor заменен атрибутом role, потому что в этой версии SOAP каждый узел играет одну или несколько ролей. Спецификация пока определяет три роли SOAP-узла:

- роль http://www.w3.org/2003/05/soap-envelope/role/ultimateReceiver играет конечный, целевой узел, который будет обрабатывать заголовок;
- роль http://www.w3.org/2003/05/soap-envelope/role/next играет промежуточный или целевой узел. Такой узел может играть и другие, дополнительные роли;
- роль http://www.w3.org/2003/05/soap-envelope/role/none не должен играть ни один SOAP-узел.

Распределенные приложения, исходя из своих нужд, могут добавить к этим ролям другие роли, например, ввести промежуточный сервер, проверяющий цифровую подпись и определить для него эту роль какой-нибудь строкой URI.

Значением атрибута role может быть любая строка URI, показывающая роль узла, которому предназначен данный блок заголовка. Значением по умолчанию для этого атрибута служит пустое значение, то есть, просто пара кавычек, или строка URI http://www.w3.org/2003/05/soap-envelope/role/ultimateReceiver.

Значение атрибута role показывает, что блок должен быть обработан узлом, играющим роль, определенную такой же строкой.

Атрибут mustUnderstand Тип данных — boolean. По умолчанию 0. Если значение равно 1, то SOAP-узел при обработке элемента обязательно должен учитывать его синтаксис, определенный в схеме документа, или совсем не обрабатывать сообщение. Это повышает точность обработки сообщения.

В версии SOAP 1.2 вместо цифр нужно писать true или false.

Атрибут relay имеет тип данных — boolean.

Показывает, что заголовочный блок, адресованный SOAP-посреднику, должен быть передан дальше, если он небыл обработан. Необходимо отметить, что если заголовочный блок обработан, правила обработки SOAP требуют, чтобы он был удален из уходящего сообщения. По умолчанию, необработанный заголовочный блок, предназначенный роли, которую исполняет SOAP-посредником, должен быть удален перед отправкой сообщения.

Все прямые дочерние элементы заголовка называются блоками заголовка (в версии 1.1. - статьями). Блоки заголовка используются для расширения сообщений децентрализованным способом путем добавления таких функций как аутентификация, администрирование транзакций и т.п. Их имена обязательно должны помечаться префиками.

В блоках заголовка могут быть атрибуты role, actor и mustUnderstand. Действие этих атрибутов относится только к данному блоку. Это позволяет обрабатывать отдельные блоки заголовка промежуточными SOAP-узлами, чья роль совпадает с ролью, указанной атрибутом role.

Ниже дан пример такого блока:

```
<env:Header>
<t:Transaction xmlns:t="http://example.com/transaction"
env:role="http://www.w3.org/2003/05/soap-
envelope/role/ultimateReceiver"
env:mustUnderstand="true">
5
</t:Transaction>
</env:Header>
```

Элементы, вложенные в блоки заголовка, уже не называются блоками. Они не могут содержать атрибуты role, actor и mustUnderstand.

Тело SOAP (SOAP Body) Элемент Body обязательно записывается сразу за элементом Header, если он есть в сообщении, или первым в SOAP-сообщении, если заголовок отсутствует. В элемент Body можно вложить произвольные элементы, спецификация никак не определяет их структуру.

Определен только один стандартный элемент, который может быть в теле сообщения -Fault, содержащий сообщение об ошибке.

Если SOAP -сервер, обрабатывая поступившее SOAP-сообщение, обнаружит ошибку,то он прекратит обработку и отправит клиенту SOAP-

сообщение, содержащее один элемент Faultc сообщением об ошибке.

В версии 1.1 элемент Fault имел дочерние элементы:

• код ошибки faultcode — предназначено для программы, обрабатывающей ошибки;

• описание ошибки faultstring – словесное описание типа ошибки, предназначено для человека;

• место обнаружения ошибки faultactor– адрес URI сервера, заметившего ошибку. Промежуточные SOAP-узлы обязательно записывают этот элемент, целевой SOAP-сервер не обязан это делать;

• детали ошибки detail — описывают ошибки, встреченные в теле Body послания, но не в его заголовке. Если при обработке тела ошибки не обнаружены, то этот элемент отсутствует.

Пример сообщения об ошибке:

```
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/">
<env:Body>
<env:Fault>
<faultcode>env:MustUnderstand</faultcode>
<faultstring>SOAP Must Understand Error</faultstring>
</env:Fault>
</env:Body>
</env:Envelope>
```

В версии SOAP 1.2 содержание элемента Fault изменилось . Как описано в пространстве имен http://www.w3.org/2003/05/soap-envelope, в него входят два обязательных элемента и три необязательных элемента.

Обязательные элементы:

• код ошибки code. Он содержит обязательный вложенный элемент value с кодом ошибки и необязательный вложенный элемент subcode, также содержащий элемент value с уточняющим кодом ошибки и элемент subcode, и далее все повторяется рекурсивно;

• причина ошибки Reason. Содержит необязательный атрибут xml:lang, указывающий язык сообщения, и произвольное число вложенных элементов с описанием ошибки.

Необязательные элементы Node — адрес URI промежуточного SOAP-узла, заметившего ошибку. получить ссылку на часть сообщения (SOAPPart), из части - ссылку на конверт, из конверта - ссылки на заголовок и тело и т.д. Поскольку узлы и элементы SAAJ (SOAP with Attachments API for Java)

реализуют интерфейсы DOM Node и Element, работать с содержимым сообщения SOAP можно, используя только API DOM или только API SAAJ, или переключаясь с одного API на другой.

Все сообщения в SAAJ посылаются и принимаются через соединения. Соединения представляются объектом SOAPConnection, получаемого фабричным методом. Сообщения посылаются через объект SOAPConnection при помощи метода call, аргументами которого являются сообщение и адрес. Метод посылает сообщение и возвращает ответ, между посылкой сообщения и получением ответа метод блокируется.

SAAJ обеспечивает соединение клиента и сервиса «точка в точку» и обмен сообщениями «запрос-ответ». До получения ответа клиент блокируется. Однако сервис не всегда может быть доступен, и большое число прикладных задач и не требует ответа от сервиса. Для того, чтобы допускать продолжение работы клиента при отсутствии ответа от сервиса необходим механизм асинхронного обмена сообщениями. Модель обмена «запрос-ответ» также может быть реализована в асинхронном режиме. Эту задачу решает пакет JAXM, который обычно используется в приложениях совместно с SAAJ. Подготовка сообщения производится средствами SAAJ. Посылается же сообщение через посредство провайдера сообщений - некоторого кода, который обеспечивает доставку сообщения адресату. Соединение с провайдером - объект типа ProviderConnection – получается фабричным методом или через службу именования (JNDI - Java Naming and Directory Interface). Сообщение отсылается адресату методом провайдера send.

Получатель сообщения должен реализовывать интерфейс OnewayListener (если он не отправляет ответа) или интерфейс ReqRespListener (если он отправляет ответное асинхронное сообщение). Оба интерфейса описывают обязательный для реализации метод onMessage, который вызывается при получении сообщения. Аргументом метода является объект типа SOAPMessage- полученное сообщение. В первом интерфейсе метод не

возвращает ничего, а во втором он возвращает также объект типа SOAPMessage- ответное сообщение.

4.3. Вопросы для самоконтроля

1) Что такое веб сервисы?
2) В чем разница между SOA и web service?
3) Что такое SOAP?
4) Объясните понятие WSDL.
5) Что такое JAX-WS?
6) Расскажите о JAXB.
7) Можем ли мы посылать soap сообщения с вложением?
8) Объясните элемент SOAP envelope.
9) Как определяется пространство имен SOAP?
10) Что вы знаете о кодировании в SOAP (encoding)?
11) Что определяет атрибут encodingStyle в SOAP?
12) Какие два конечных типа веб сервисов используют JAX-WS? Какие существуют правила для кодирования записи header?
13) Какие вы можете выделить различия между SOAP и другими техниками удаленного доступа?
14) Когда можно использовать GET запрос вместо POST для создания ресурса?
15) Какая разница между GET и POST запросами?
16) Что означает WADL?
17) Какие два способа получения заголовка HTTP запроса в JAX-RS вы знаете?
18) Как скачать файл с помощью JAX-RS?

СПИСОК ЛИТЕРАТУРЫ

Основная

1. Машнин Т. С. Web-сервисы Java / Т. С.Машнин.- СПб.: БХВ-Петербург, 2012. - 560 с.

2. Хантер Д., Рафтер Дж. XML. Работа с XML, 4-е издание / Д.Хантер, Дж.Рафтер.- М.: Диалектика, 2009.- 1344 с.

3. XML и Java2 [Электронный ресурс]. - К.: Мультитрейд, 2005. - 1 электрон. опт. диск (CD-ROM).

Дополнительная

4. Как программировать на XML / Х. М. Дейтел, П. Д. Дейтел, Т. Р. Нието и др. ; Пер. с англ. А. И. Тихонова. - М. : БИНОМ, 2001. - 934 с. Каб7 (1)

5. Маслов В.В. Основы программирования на языке Java: Учеб. Курс. / В.В.Маслов.- М.: Горячая Линия-Телеком, 2000. - 131 с. АНЛ (1), Чз1 (1)

6. Основы Web-технологий : учеб. пособие для студентов вузов, обучающихся по специальности 351400 "Прикладная информатика" / П.Б. Храмцов, С. А. Брик, А. М. Русак, А.И. Сурин. - 2-е изд. - М. : Интернет-Ун-т информ. технологий : БИНОМ. Лаб. знаний, 2007. - 374 с. АУЛ(1), Чз1(1)

7. Шилд Г. Полный справочник по Java. 7 изд.: Пер. с англ. / Г.Шилд.- М.: Издательский дом "Вильямс", 2007.- 1036 с.

8. Эккель Б. Философия Java. Библиотека программиста. 4-е изд. – СПб.: Питер, 2009 . – 640 с.

Информационные ресурсы

1. http://comp-science.narod.ru/mml/mathml.htm
2. http://math.accent.kiev.ua/article/05/png_htm/05_00_png.htm
3. http://fsweb.info/web/mathml/basis.html
4. http://www.w3ii.com/en-US/java_xml/java_sax_parser.html
5. http://spec-zone.ru/RU/Java/Tutorials/jaxp/sax/parsing.html

6. http://computersbooks.net/index.php?id1=4&category=language-programmer&author=flenagan-d&book=2003&page=227
7. https://studfiles.net/preview/3073078/page:5/
8. http://www.quizful.net/post/sax-parser-java
9. https://xsltdev.ru/xslt/xsl-template/
10. http://umihelp.ru/articles/daniil_sirotkin/xslt-shpargalka-xsl-apply-templates/
11. http://citforum.ck.ua/internet/xmlxslt/xmlxslt.shtml
12. http://xmlhack.ru/books/xslt/ch_09_09.html
13. http://crypto.pp.ua/2010/06/rabota-s-xslt-v-java/
14. https://www.javaworld.com/article/2076171/java-se/xml-document-processing-in-java-using-xpath-and-xslt.html
15. http://www.java2s.com/Tutorials/Java/Java_XML/0200__Java_XSLT_Intro.htm
16. https://www.ibm.com/developerworks/ru/library/x-jaxmsoap/index.html
17. https://javarush.ru/groups/posts/1168-veb-servisih-shag-1-chto-takoe-veb-servis-i-kak-s-nim-rabotatjh
18. https://www.w3schools.com/xml/xml_services.asp

УЧЕБНОЕ ИЗДАНИЕ

Рекомендовано Ученым советом
ГОУ ВПО «Донецкий национальный университет»
(протокол № 1 25.01.2019 г.)

Авдюшина Елена Владимировна
Пачева Марина Николаевна

WEB/XML технологии

УЧЕБНОЕ ПОСОБИЕ

Издание второе

для студентов направления подготовки

01.04.02 Прикладная математика и информатика

Авторская верстка
Компьютерный дизайн: Е.В. Авдюшина

Адрес издательства:
ГОУ ВПО «Донецкий национальный университет»,
ул. Университетская, 24. г. Донецк, 283055

Подписано в печать 20.02.2019 г.
Формат 60×84/16. Бумага офисная.
Печать – цифровая. Усл.-печ. л. 4,5.
Тираж 100 экз. Заказ № 26 – март.19.
Донецкий национальный университет
283001, г. Донецк, ул. Университетская, 24.
Свидетельство про внесение субъекта
издательской деятельности в Государственный реестр
серия ДК № 1854 от 24.06.2004г.