МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ УКРАИНЫ

НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
"ХАРЬКОВСКИЙ ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ"

Н.Ф. Хайрова, С.В. Петрасова

ИНФОРМАЦИОННЫЕ ИНТЕЛЛЕКТУАЛЬНЫЕ СИСТЕМЫ И СЕМАНТИЧЕСКИЙ ВЕБ

Учебное пособие
по курсу «Современные интеллектуальные системы
автоматической обработки информации»

Утверждено
редакционно-издательским
советом университета,
протокол № 2 от 24.12.14.

Харьков
НТУ "ХПИ"
2016

УДК 004.89(075.8)
ББК 32.973.202я73
Х12

Рецензенты:
Е. В. Бодянский, д-р.техн.наук, проф. каф. искусств. интеллекта Харьковского национального университета радиоэлектроники;
С. Л. Кривый, д-р. физ-мат. наук, проф. каф. информ. систем Киевского национального университета им. Т. Г. Шевченка

У навчальному посібнику розглядаються основні складові технології семантичного веба: XML, простір імен, універсальний ідентифікатор ресурсів URI, XML Schema, XSL, RDF, RDF Schema та OWL. Особлива увага приділяється використанню DTD та XML Schema, а також моделі DOM XML. Матеріал проілюстровано наочними практичними прикладами, розділи включають лабораторні роботи.

Призначено для студентів спеціальностей «Прикладна лінгвістика», «Прикладна інформатика» та інших інформаційних та комп'ютерних напрямків.

Хайрова Н.Ф.

Х12 Информационные интеллектуальные системы и семантический веб : учебное пособие / Н. Ф. Хайрова, С. В. Петрасова. – Х. : НТУ «ХПИ», 2016. – 170 с. – Рус. яз.

ISBN 978-617-05-0204-9

В учебном пособии рассматриваются основные составляющие технологии семантического веба: XML, пространство имен, универсальный идентификатор ресурсов URI, XML Schema, XSL, RDF, RDF Schema и OWL. Особое внимание уделяется использованию DTD и XML Schema, а также модели DOM XML. Материал проиллюстрирован наглядными практическими примерами, разделы включают лабораторные работы.

Предназначено для студентов специальностей «Прикладная лингвистика», «Прикладная информатика» и других информационных и компьютерных направлений.

Ил. 11. Табл. 11. Библиогр. : 12 наим.

УДК 004.89(075.8)
ББК 32.973.202я73

ISBN 978-617-05-0204-9

ВВЕДЕНИЕ

Курс «Современные интеллектуальные системы автоматической обработки информации» ориентирован на формирование у студентов знаний по теории семантического веба и OWL-онтологий. Программа дисциплины направлена на практическую подготовку студентов, изучающих интеллектуальные сети, распределенные информационные системы, мультиагентные системы и знаниеориентированные технологии.

В пособии рассматриваются основные составляющие технологии семантического веба: технологическая платформа XML, пространство имен, универсальный идентификатор ресурсов URI, XML Schema, XSL, RDF, RDF Schema и OWL. Особое внимание при изложении рекомендаций W3C к расширяемому языку разметки XML уделяется технологиям разработки валидных XML документов, включающих схемы DTD и XML Schema, а также модели DOM-документа XML. В учебном пособии подробно рассматриваются технологии средств выражения семантики данных и их связей языками RDF и OWL, использование которых позволяет разрабатывать онтологии, базы знаний и системы управления знаниями.

С целью закрепления умений работать с языком разметки XML и языком описания семантики web-ресурсов RDF приведены наглядные примеры и лабораторные работы, позволяющие выработать навыки самостоятельного решения задач.

Предназначено для студентов старших курсов технических и гуманитарных специальностей, изучающих интеллектуальные системы обработки информации и семантический веб.

ТЕМА 1

ИНТЕЛЛЕКТУАЛЬНЫЕ ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ

1.1. Базовые функции интеллектуальной информационной системы.

1.2. Интеллектуализация современного Интернета.

1.3. Назначение XML.

1.1. Базовые функции интеллектуальной информационной системы

Под ***интеллектуальными системами (ИС)*** понимают любые биологические, искусственные или формальные системы, проявляющие способность к целенаправленному поведению, предусматривающему:

1) проявление свойств общения;

2) накопление знаний;

3) принятие решений;

4) обучение;

5) адаптацию и т.д.

Под *информационной системой* понимают совокупность технического, программного, организационного обеспечения, а также персонала, предназначенную для того, чтобы своевременно обеспечивать людей надлежащей информацией.

Функции информационной системы:

- воспринимать вводимые пользователем информационные запросы и исходные данные;
- обрабатывать введенные и хранимые в системе данные в соответствии с известным алгоритмом и формировать требуемую выходную информацию.

Если в ходе эксплуатации ИС выяснится потребность в ее модификации, то возникнет необходимость ее переписывания. Это объясняется тем, что полным знанием проблемной области обладает только разработчик ИС, а программа служит “недумающим исполнителем” знания разработчика. Этот недостаток устраняется в интеллектуальных информационных системах. По одному из определений ***интеллект*** представляет собой универсальный алгоритм, способный разра-

батывать алгоритмы решения конкретных задач. А ***интеллектуальными*** считаются задачи, связанные с разработкой алгоритмов решения ранее нерешенных задач определенного типа.

Интеллектуальная информационная система (ИИС) – это информационная система, которая основана на концепции использования базы знаний для генерации алгоритмов решения задач различных классов в зависимости от конкретных информационных потребностей пользователей.

Интеллектуальные информационные технологии – одна из наиболее перспективно развивающихся областей информатики:

- призванная решать задачи в слабоструктурированных предметных областях;
- повышающая уровень интеллектуальной информационной поддержки современных специалистов.

Интеллектуализация компьютеров осуществляется путем разработки как специальной аппаратуры (например, нейрокомпьютеры), так и программного обеспечения (экспертные системы, базы знаний, решатели задач и т.д.).

Система считается *интеллектуальной*, если в ней реализованы следующие три базовые функции:

1. *Функция представления и обработки знаний.* ИС должна быть способна накапливать знания об окружающем мире, классифицировать и оценивать их с точки зрения прагматики и непротиворечивости, инициировать процессы получения новых знаний, соотносить новые знания со знаниями, хранящимися в базе знаний.

2. *Функция рассуждения.* ИС должна быть способна формировать новые знания с помощью логического вывода и механизмов выявления закономерностей в накопленных знаниях, получать обобщенные знания на основе частных знаний и логически планировать свою деятельность.

3. *Функция общения.* ИС должна быть способна общаться с человеком на языке, близком к естественному языку (ЕЯ), и получать информацию через каналы, аналогичные тем, которые использует человек при восприятии окружающего мира.

Классификация ИИС:

I класс: системы с интеллектуальным интерфейсом (коммуникативные способности):

1) системы с естественно-языковым интерфейсом;

2) гипертекстовые системы (технология обработки семантической информации, основанная на использовании гипертекста);

3) контекстные системы и др.

II класс: экспертные системы (решение сложных задач):

1) классифицирующие системы;

2) доопределяющие системы;

3) трансформирующие системы.

III класс: самообучающиеся системы (способность к самообучению):

1) индуктивные системы;

2) нейронные сети;

3) системы, основанные на прецедентах;

4) информационные хранилища.

1.2. Интеллектуализация современного Интернета

Веб 1.0 – технологии разработки сайтов, существовавшие в сети Интернет до появления веб 2.0, обычно включали:

- статичные страницы вместо генерируемого пользователями динамического контента;
- бедную гипертекстовую разметку (большая часть контента дефакто являлась простым текстом, зачастую пренебрегавшим правилами HTML);
- использование фреймов;
- использование специфичных тегов HTML – следствие редактирования страниц в WYSIWYG-редакторах, встроенных в конкретный браузер или сторонников конкретного браузера-участника «войны браузеров»;
- гостевые книги, форумы или чаты – как попытку придания интерактивности;
- указание конкретного разрешения монитора, при котором дизайн сайта отображается корректно (не вылезает за пределы страницы, не разъезжается форматирование);

- крайне редкое и непопулярное использование стилей CSS при оформлении страниц сайта и др.

Основные положительные черты web-1:

- открытый характер Интернета – к сети можно подключиться с помощью любого стандартного оборудования и свободно распространяемых программных средств;
- демократическая организация – использование Интернета не требует существенных финансовых затрат и каких-либо административных решений;
- эффективная как для пользователей, так и для разработчиков приложений клиент-серверная архитектура WWW;
- простота языка разметки HTML, возможность представления с помощью него не только гипертекстовых, но и гипермедийных данных, наличие множества HTML- редакторов и др.

Свойства web-1, сдерживающие развитие и эффективное использование WWW:

- язык HTML ориентирован не на логическую, а на форматную разметку контента, отражающую способ его представления на экране;
- язык HTML имеет слабые средства поддержкии метаданных, описывающих структурные и семантические свойства web-страниц;
- язык HTML идентифицирует информационные ресурсы по их адресам в WWW с помощью URL. К HTML-ресурсам возможен только навигационный доступ по гиперссылкам. Доступ по содержанию обеспечивают специальные средства – поисковые машины;
- около 70 % информационных ресурсов Интернета явно не представлены в web-1, т.е. недоступны для автоматической обработки поисковыми машинами. Подобные ресурсы образуют так называемый ***скрытый,*** или ***глубинный web (deep web)*** – это базы данных, интегрированные в веб-сайты, архивы, мультимедийные файлы, а также многочисленные документы в форматах PDF, Doc, RTF, PostScript и др;
- неразвитость HTML в части поддержки метаданных не позволяет верифицировать информационные ресурсы, ограничивает поисковые возможности web-1 навигационным и контентным методами поиска. Поэтому поисковые машины web-1 характеризуются высоким уровнем поискового шума.

На сегодняшний день необходим переход от документов, читаемых компьютером (machine readable), к документам, понимаемым компьютером (machine understandable).

Недостатки и ограничения технологии Интернета первого поколения (web-1) привели к разработке консорциумом W3C концепции *«семантического веба» (semantic web,* или *web-2)*. Цель реализации Semantic Web состоит в преодолении ограничений технологий web-1 и сохранении их достоинств.

Термин «семантическая паутина» был впервые введен сэром Тимом Бернерсом-Ли в 2001 году. В настоящее время концепция семантической веб-сети была принята и продвигается W3C (World Wide Web Consortium).

Технология семантического веба направлена на интеллектуализацию WWW и базируется на следующих основных компонентах:

- активном использовании метаданных, т.е. дополнительно к видимой человеком информации добавляется служебная информация, позволяющая эффективно использовать данные программным агентам;
- метаязыке XML;
- онтологическом подходе, позволяющем описывать термины и отношения между ними;
- модели RDF, устанавливающей способ представления значений, определенных в онтологии.

Для глобальной реорганизации всемирной сети, в направлении использования стандартов представления данных, понимаемых программными агентами, а не только предназначенных для восприятия человеком, требуется набор взаимосвязанных технологий, представленных на рис.1.1.

URI (Uniform Resource Identifier) – унифицированный индикатор любого ресурса. Представляет собой уникальную символьную строку. Самым известным URI на сегодня является URL (Uniform Resource Locator), являющийся идентификатором ресурса в Internet и дополнительно содержащий информацию о местонахождении адресуемого ресурса.

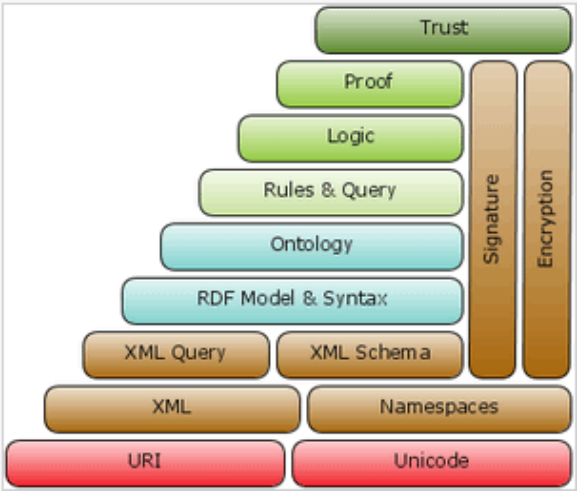


Рисунок 1.1 – Структура Semantic Web

Онтологии – некоторая концептуальная схема, описывающая (формализующая) значения элементов некоторой предметной области. Онтология состоит из набора терминов и правил, описывающих их связи, отношения.

RDF (Resource Description Framework) – современный язык описания онтологии. Атомарным объектом в RDF является триплет: субъект – предикат – объект.

OWL (Web Ontology Language) – язык Веб-онтологий, созданный для представления значения терминов и отношения между этими терминами в словарях. В отличие от RDF, данный язык использует более высокий уровень абстракции, что позволяет языку наряду с формальной семантикой использовать дополнительный терминологический словарь.

Важным преимуществом OWL является то, что в его основу положена четкая математическая модель дескрипционных логик.

Место OWL в общей структуре Semantic WEB с точки зрения консорциума W3C:

- XML предоставляет возможности создания структурированных документов, но не предъявляет к ним никаких семантических требований;

- XML Schema определяет структуру XML документов и дополнительно позволяет использовать конкретные типы данных;
- RDF предоставляет возможность описывать абстрактные модели данных некоторых объектов и отношения между ними. Использует простую семантику на основе XML синтаксиса;
- RDF Schema позволяет описывать свойства и классы RDF-ресурсов, а также семантику отношений между ними;
- OWL расширяет описательные возможности предыдущих технологий. Позволяет описывать отношения между классами (к примеру, непересекаемость), кардинальность (например, «точно один»), симметрия, равенство, перечисляемые типы классов.

В Semantic Web также применяются:

1) универсальные идентификаторы ресурсов;

2) системы обработки правил логического вывода;

3) стандартные протоколы Интернета.

Техническую часть семантической паутины составляет семейство стандартов на языки описания, включающее XML, XML Schema, RDF, RDF Schema, OWL и др.

В основе Semantic Web лежат следующие концепции:

1) расширяемый язык разметки XML;

2) RDF-формат описания ресурсов;

3) онтологии, определяющие термины и отношения между ними.

Основой web-2 служит расширяемый язык разметки XML. Единицей доступа к информационному ресурсу web-2 является XML-документ. Возникла новая технологическая платформа web-2 – ***платформа XML***. Под ней понимается совокупность взаимосвязанных и согласованных стандартов и спецификаций, имеющих общее функциональное назначение.

Ядро платформы XML включает в себя:

- стандарты XML;
- понятие XML-документа;
- способы представления метаданных с помощью XML;
- механизм идентификации ресурсов URI, более общий по сравнению с URL;

- протоколы передачи XML-данных: XMLP (XML Protocol – протокол передачи XML-данных), SOAP (Simple Object Application Protocol – прикладной протокол передачи простых объектов).

Важно отметить, что платформа XML обеспечивает совместимость web-2 с используемыми в настоящее время технологиями web-1. Язык HTML также может быть определен как приложение XML, поэтому переход к платформе XML не грозит потерей информационных ресурсов, накопленных в WWW.

1.3. Назначение XML

Спецификация языка 1.0 XML (Extensible Markup Language – расширяемый язык разметки) была разработана в 1998 году и появилась для перехода Интернета на новый уровень связности. XML является открытым стандартом, которым управляет W3C, придав ему статус формальной рекомендации.

XML представляет собой, с одной стороны, протокол хранения и управления информацией, а с другой, – это семейство технологий, с помощью которых можно осуществлять все: от оформления документов до фильтрации данных.

XML, несмотря на свое название не является языком разметки, это набор правил для создания языков разметки.

Разметка – это информация, добавляемая в документ и в некоторых отношениях усиливающая его смысл. Понятие гипертекста было введено В. Бушем в 1945 году. А начиная с 60-х годов, стали появляться первые приложения, использующие гипертекстовые данные. Основное развитие данная технология получила, когда возникла реальная необходимость объединения множества информационных ресурсов.

В 1986 году ISO был утвержден универсальный стандартизированный язык разметки ***SGML (Standard Generalized Markup Language)***. Этот язык предназначен для создания других языков разметки, он определяет допустимый набор тэгов, их атрибуты и внутреннюю структуру документа. Программное обеспечение для данного языка является сложным и дорогим, и его полезность ограничивается крупными организациями.

В 1991 году Тимом Бернерсом-Ли был разработан язык HTML, который первоначально был всего лишь одним из SGML-приложений. Несмотря на то, что единственное свойство HTML – классифицировать части документа и обеспечивать его правильное отображение в браузере, он является самым популярным языком разметки. Это связано с тем, что HTML достаточно легок для изучения.

Язык HTML предоставляет фиксированный набор элементов, которые вы можете использовать для размещения компонентов на типовой web-странице. Примерами таких элементов являются заголовки, абзацы, списки, таблицы, изображения и связи. Например, HTML отлично подходит для создания личной домашней страницы.

Каждый элемент начинается с начального тега: текста, заключенного в угловые скобки (< >), который содержит имя элемента и дополнительную информацию. Большинство элементов заканчиваются конечным тегом, который повторяет соответствующий начальный тег, за исключением того, что имеет символ косой черты (/) перед именем элемента. Элемент *содержание* представляет собой текст, расположенный между начальным и конечным тегами.

С помощью языка HTML не могут быть описаны:

- документ, который не содержит типовых компонентов (заголовков, абзацев, списков, таблиц и т.д.). Например, в языке HTML отсутствуют элементы, необходимые для отображения музыкальных символов или математических уравнений;
- база данных наподобии каталога книг. Вы можете использовать HTML-страницу, чтобы хранить и отображать информацию из статической базы данных (например, перечень книг и их описание). Однако, если понадобится осуществить сортировку, фильтрацию, поиск и обработку информации, придется снабдить каждую из составных частей информации соответствующей меткой (как в программе, работающей с базами данных, как, например, Microsoft Access). В языке HTML не предусмотрено соответствующих элементов;
- документ, который нужно представить в виде иерархической структуры. Допустим, вы пишете книгу и хотите разбить ее на части, главы, разделы A, B, C и т.д. В дальнейшем программа может использовать данную структуру документа для создания оглавления, оформ-

ления различных уровней в структуре с помощью различных стилей, извлечения определенных разделов, а также обработки информации иными способами. Однако элемент типа заголовок в HTML содержит лишь описание собственно текста.

Недостатки HTML:

1) HTML имеет фиксированный набор тэгов, и данный набор нельзя расширить или изменить;

2) теги языка HTML показывают только, как должны быть представлены данные, то есть внешний вид документа. HTML не несет информации о значении содержания, заключенного в тэгах, структуре документа.

Язык XML позволяет преодолеть эти ограничения. Основное назначение языка XML – облегчить работу с документами в web.

В 1996 году общественной организацией W3C началась разработка XML, который стал золотой срединой между языками SGML и HTML. Язык XML позволяет разработчику создавать свои собственные теги, но в отличие от SGML он достаточно прост.

При создании документа XML используется не ограниченный набор элементов, как в HTML, а создаются собственные элементы и присваиваются им любые имена по вашему выбору – именно поэтому язык XML является расширяемым (extensible). Поэтому можно использовать XML для описания практически любого документа: от музыкальной партитуры до базы данных.

Три основных способа сообщить браузеру (в частности, Microsoft Internet Explorer 5), как обрабатывать и отображать каждый из созданных XML-элементов:

1. *Таблица стилей.* С помощью данного метода вы связываете таблицу стилей с XML-документом. Таблица стилей представляет собой отдельный файл, содержащий инструкции для форматирования индивидуальных XML-элементов. Вы можете использовать либо каскадную таблицу стилей *(Cascading Style Sheet – CSS)*, которая также применяется для HTML-страниц, либо расширяемую таблицу в формате языка стилевых таблиц *(Extensible Stylesheet Language – XSL)*, обладающую значительно более широкими возможностями, нежели CSS, и разработанную специально для XML-документов. XSL больше напо-

минает язык программирования – с рекурсией, шаблонами и функциями и достаточно сложен.

2. *Связывание данных.* Этот метод требует создания HTML-страницы, связывания с ней XML-документа и установления взаимодействий стандартных HTML-элементов на странице, таких как SPAN или TABLE, с элементами XML. В дальнейшем HTML-элементы автоматически отображают информацию из связанных с ними XML-элементов.

3. *Написание сценария.* В этом методе вы создаете HTML-страницу, связываете ее с XML-документом и имеете доступ к индивидуальным XML-элементам с помощью специально написанного кода сценария (JavaScript или Microsoft Visual Basic Scripting Edition (VBScript)). Браузер воспринимает XML-документ как объектную модель документа (Document Object Model – DOM), состоящую из большого набора объектов, свойств и команд. Написанный код позволяет осуществлять доступ, отображение и манипулирование XML-элементами.

XML не является языком – это спецификация для создания языков разметки. Существуют два способа создания языка, основанного на XML:

1. *Свободный формат (freeformat) XML*. В этом режиме действуют минимальные правила образования и применения тегов. Если документ удовлетворяет минимуму правил XML, он называется ***корректным*** XML документом. Но полезность свободного формата XML ограничена.

Правила XML жестко трактуют все, что касается структуры. Документ должен иметь такую разметку, чтобы не существовало двух способов интерпретации имен, порядка и иерархии элементов. Документ является корректным, если он удовлетворяет ряду минимальных синтаксических требований. Язык XML имеет строго определенный синтаксис:

а) например, в отличие от HTML, каждый элемент XML должен содержать начальный и конечный тег (либо специальный пустой тег);

б) имя *Типа элемента* в начальном теге должно в точности соответствовать имени в соответствующем конечном теге, имена типов элементов чувствительны к регистру;

в) существует только один корневой элемент, который содержит все остальные элементы документа. Все элементы образуют простое иерархическое дерево, с отношением «родитель – потомок».

2. *Моделирование документов (document modeling)* требует создания спецификации, излагающей правила, которым должен отвечать документ. Проверка документа на соответствие своей спецификации называется проверка семантической корректности, или валидация (validation). Если документ признан действительным (valid), это говорит о том, что в нем нет таких ошибок, как некорректная запись тегов, неправильный порядок их следования или отсутствие данных.

Моделирование документов. Чаще всего документы моделируются с помощью определения типа документа (*Document Type Definition DTD).* Используется стандарт моделирования документов XML-схемы (XML Scheme). Схемы используют фрагменты XML, называемые ***шаблонами (templates)***, для того чтобы показать, как должен выглядеть документ.

Язык разметки, созданный с помощью правил XML, называется ***приложением XML***, или иногда ***типом документа***. Существуют сотни общедоступных приложений XML для кодирования чего угодно.

Всякая программа, которая читает и обрабатывает документы XML, называется ***процессором XML***. Первый этап процессора XML представляет собой ***синтаксический анализатор (parser)***, являющийся достаточно строгим, т.е. любая ошибка прекращает дальнейшую обработку. Лишь после исправления всех синтаксических ошибок документ признается *корректным*.

Internet Explorer, Opera, Mozilla производит анализ документов XML и выводит их с помощью каскадных таблиц стилей.

Просмотр XML документа. В XML нет неявных определений стилей. Это значит, что одного документа XML обычно недостаточно для получения форматированного результата. Internet Explorer выводит любой документ XML в виде схемы структуры, если не указана таблица стилей. ***XHTML*** (версия HTML, которая удовлетворяет правилам XML) является языком разметки, в котором неявно заданы стили для элементов. Часто документы XML преобразуют в XHTML, чтобы посмотреть их в браузере как форматированные документы.

ТЕМА 2

ОСНОВНЫЕ ПОНЯТИЯ XML

2.1. Документ XML.

2.2. Отображение XML-документов с использованием каскадной таблицы стилей.

2.3. Пространство имен XML.

2.4. Отображение XML-документов с использованием связывания данных.

2.5. Использование табличного сцепления данных.

2.6. Использование постраничного отображения.

2.1. Документ XML

В XML ***документ*** – это основная единица информации XML, составленная из элементов и разметки в упорядоченном пакете, файл с расширением xml. Документ может содержать текст, базу данных, уравнение или некоторую абстрактную структуру. Документ может располагаться в одном или нескольких файлах, возможно расположенных на разных машинах.

Любой XML-документ состоит из следующих частей:

1) необязательный пролог;

2) тело документа;

3) необязательный эпилог, следующий за деревом элементов.

Документ XML начинается с *пролога*. В прологе содержатся *инструкции*, предназначенные для анализатора XML и приложений. Общий вид инструкции:

<? кому инструкция?>

Кому указывает имя приложения, получателя инструкции. Параметр *инструкция* задает содержание инструкции, передаваемой приложению.

XML-процессор представляет программный модуль, который читает и хранит содержимое XML-документа. ***Приложение*** – это отдельный программный модуль, который получает содержимое документа от XML-процессора, а затем обрабатывает и отображает его содержимое. Если документ отображается в Internet Explorer, то браузер содержит как XML-процессор, так и часть приложения. По наимено-

ванию конкретный XML-процессор определяет, предназначены данные ему или другому процессору.

Инструкции могут помещаться не только в прологе, но и в любом месте XML-документа вне описания тегов.

В простейшем случае пролог просто сообщает (представляет объявление документа XML), что это документ XML и добавляет используемую версию: <?xml version=”1.0” encoding="windows-1251"?> – указывает процессору XML на необходимость использования синтаксического анализатора XML для интерпретации данного документа. Свойство *encoding* указывает кодировку символов, используемых в документе. Строки в кавычках называются ***литералами***.

Например, инструкция *<?xml-stylesheet type="text/css" href="file_2.css"?>* предписывает Internet Explorer использовать CSS-таблицу из файла file_2.css.

Порядок инструкций в прологе нельзя изменить, и первым должно быть объявление XML. После пролога должен идти корневой элемент.

Так как пролог не является обязательным (хотя в спецификации XML заявлено, что вам «следует» включать XML-объявление), то минимальный корректно сформулированный в соответствии с XML-стандартом документ содержит только элемент:

<min>A minimalist text.</min>

Документ состоит из частей, называемых ***элементами***, которые вкладываются друг в друга. На верхнем уровне находится элемент, называемый ***элементом документа,*** или ***корневым элементом***. Элемент, который содержит один или более вложенных элементов, называется ***родительским элементом***. Элемент, содержащийся внутри родительского элемента, называется ***дочерним элементом***, ***субэлементом***, или ***вложенным элементом.***

Элемент обычно состоит из начального тега, содержимого и конечного тега, как показано на рис. 2.1.

Рисунок 2.1 – Элемент XML-документа

Тип, или ***родовой идентификатор элемента Generic Identifier (GI)*** – это имя элемента, которое содержится в стартовой позиции начального тега и в конечном теге (TITLE в нашем примере). Документ может содержать более одного элемента с одинаковыми именами типа. Имя элемента идентифицирует особый тип или класс элемента.

Содержимое элемента – это текст, расположенный между начальным и конечным тегами. Содержимым элемента могут быть:

- вложенные элементы;
- символьные данные или текст;
- инструкции по обработке;
- комментарии (начинается с символов <!-- и заканчивается символами -->. Комментарии можно вставлять в любое место XML-документа, кроме описания тега.);
- ссылки на примитивы и ссылки на символы;
- раздел ***CDATA (character data)*** – это текстовый блок, в котором можно размещать любые символы, за исключением]]>.

Внутри символьных данных в содержимом элемента нельзя помещать некоторые специальные символы (<, &, …), так как это может привести к путанице при обработке документа. Одним из возможных путей преодоления этих ограничений является использование разделов символьных данных. Такой раздел начинается с символов <![CDATA[и заканчивается]]>. Все символы внутри раздела CDATA рассматриваются как литеральная часть символьных данных элемента, а не как XML-разметка. Раздел CDATA, представленный на рис. 2.2, может располагаться в любом месте документа, занимаемом символьными данными. Разделы CDATA не могут быть вложенными.

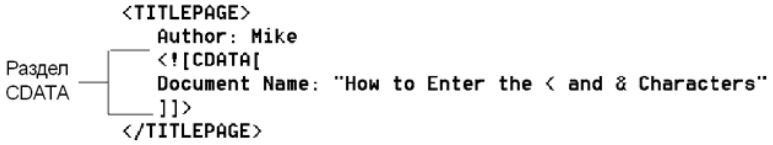


Рисунок 2.2 – Раздел символьных данных

Кроме обычных элементов XML можно использовать ***пустые элементы*** (элементы, не имеющие содержания):

- для указания XML-приложению выполнить действие или отобразить объект, присутствие элемента с определенным именем – без какого-либо содержимого – может послужить важной информацией для приложения. Аналогом в HTML является пустой элемент BR, который является указанием браузеру вставить разрыв строки, а также пустой элемент HR, указывающий на вставку горизонтальной разделительной линии;
- пустой элемент может нести информацию посредством атрибутов, о которых вы узнаете далее.

Аналогом в HTML является пустой элемент IMG (изображение), содержащий атрибуты, которые сообщают процессору, где искать графический файл и как его отобразить.

Возможны два синтаксиса задания пустого элемента:

1) размещения конечного тега сразу же после начального тега. Например, <HR></HR>;

2) можно использовать специальный тег пустого элемента, представленный на рис. 2.3.

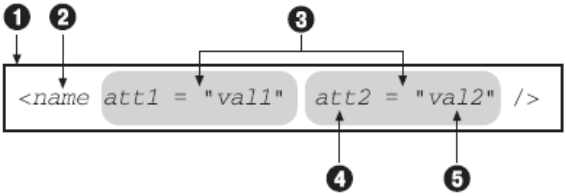


Рисунок 2.3 – Синтаксис задания пустого элемента:
1 – тег, начинающийся с открывающейся угловой скобки, за которым следует имя элемента; 2 – имя элемента; 3 – атрибуты, каждый из которых состоит из имени и значения в кавычках; 4 – имя атрибута; 5 – значение атрибута; заканчивается элемент косой чертой (/) и закрывающейся угловой скобкой

Задание атрибутов элемента. В начальный тег элемента или в тег пустого элемента можно включить одно или несколько описаний атрибутов. Описание атрибута состоит из имени атрибута, знака равенства и в кавычках – значения атрибута. Имена аналогичны требованиям, предъявляемым к именам элементов.

Правила использования имен элементов (типов элемента):

- имя должно начинаться с буквы или символа подчеркивания (_);
- следующие после первого символы могут быть буквами, цифрами, точкой, тире или подчеркиванием;
- не следует использовать имена, начинающиеся с префикса “xml” (в любом сочетании строчных или прописных букв);
- имя, записанное в начальном теге, должно в точности соответствовать имени в конечном теге;
- соблюдение регистра существенно для имен элементов, как и для всего текста в описании разметки.

Каждое имя атрибута может только один раз присутствовать в одном и том же начальном теге или в теге пустого элемента.

Задание атрибутов обеспечивает альтернативный способ (обычно данные, которые необходимо отобразить, включаются в содержимое элемента) помещения информации в элемент. В спецификации XML не установлено строгих разграничений относительно типа информации, которую можно описывать с помощью атрибутов или внутри содержимого элемента.

Когда XML-документ отображается с использованием CSS-таблицы, браузер не выводит атрибуты или их значения.

Если XML-документ отображается с использованием связывания данных, сценария для HTML-страницы либо XSL-таблицы стилей, то возможно отображать значения атрибутов или выполнять соответствующие действия.

Правила задания значений атрибутов:

- строка (литерал) может быть заключена как в одинарные ('), так и в двойные кавычки (");
- строка не может содержать внутри себя тот же символ кавычек, которыми она ограничена;
- строка может содержать ссылку на символ или ссылку на внутренние примитивы общего назначения;
- строка не может содержать символ < (синтаксический анализатор может воспринять этот символ как начало описания XML-разметки);

- строка не может содержать символ &, если это не ссылка на символ или примитив.

Если создается корректно сформированный документ, не имеющий объявления типа документа, вы можете присвоить атрибуту любое значение, соответствующее приведенным выше правилам.

Если создается описание типа документа (т.е. есть модель документа) и внутри него определяются атрибуты, то можно ограничить типы значений, которые могут быть присвоены конкретному атрибуту. Например, можно определить атрибут, которому могут быть присвоены только значения "yes" или "no".

2.2. Отображение XML-документов с использованием каскадной таблицы стилей

Отображение XML-документов с использованием каскадной таблицы стилей осуществляется в два этапа:

1) создание файла таблицы стилей;
2) связывание таблицы стилей с XML-документом.

Таблица стилей состоит из одного или нескольких правил (иногда их называют ***набором правил***). Правило содержит информацию по отображению определенного типа элемента в XML-документе.

Селектор представляет собой имя типа элемента, к которому относится информация по отображению.

Каскадные таблицы стилей CSS не чувствительны к регистру.

Набор наследуемых свойств, которые присвоены определенному элементу, действует на все дочерние элементы, прямо или косвенно вложенные в него, если только они не переустанавливаются впоследствии для определенного дочернего элемента.

Для ненаследуемых свойств, если не задано значение свойства для конкретного элемента, браузер использует значение свойства по умолчанию.

Использование контекстуальных селекторов. ***Контекстуальным (contextual) селектором*** называется селектор, в котором имя элемента может предваряться именами одного или нескольких элементов-предков (родительский, родительский плюс родительский родителя и т.д.). В этом случае правило будет применимо только к элементам с этим именем, которые являются вложенными подобным образом.

Между именами элементов в контекстном селекторе ставят только пробелы.

Родовым (generic) селектором называется селектор, который не включает имен элементов-предков.

Если определенное свойство для одного и того же элемента имеет одну установку в правиле с контекстуальным селектором и другую установку в правиле с родовым селектором, установка в правиле с контекстуальным селектором доминирует, поскольку является более конкретизированной.

Например, приведем фрагмент программного кода XML-документа, где <MAPS> – корневой элемент:

```
<MAPS>
    <CITY>
        <NAME>Santa Fe</NAME>
        <STATE>New Mexico</STATE>
     </CITY>
     <STATE>California</STATE>
</MAPS>
```

Правила присоединенной таблицы стилей:

```
CITY STATE
    {font-style:normal}
STATE
    {font-style:italic}
```

В результате браузер отформатирует "New Mexico" обычным шрифтом, а "California" – курсивом.

Использование атрибута STYLE. Можно в XML-документе использовать специальный атрибут *STYLE*, вместо того чтобы устанавливать одно или несколько определенных свойств отдельного элемента в таблице стилей.

Если значение свойства, установленного с помощью атрибута STYLE, конфликтует со значением свойства, установленного в таблице стилей, установка с помощью атрибута STYLE имеет приоритет. Таким образом, атрибут STYLE является удобным средством, чтобы переустановить – для определенного элемента – значение свойства, присвоенное для типа элемента в присоединенной таблице стилей.

Использование атрибута STYLE нарушает принцип CSS относительно хранения информации о форматировании отдельно от определения содержимого документа и структуры XML-файла.

Чтобы установить одно или несколько значений свойств, объявления в значение атрибута STYLE включаются в виде строки, заключенной в кавычки, отделяя индивидуальные объявления точкой с запятой.

Например:

```
<TITLE STYLE='font-style: normal; font-size:14pt'>
    The Adventures of Huckleberry Finn
</TITLE>
```

Импорт других таблиц стилей. Можно с помощью директивы *@import* в одну каскадную таблицу стилей встроить одну или несколько других таблиц стилей. Возможность импорта отдельных таблиц стилей позволяет хранить правила для связанных стилей в отдельных файлах, а затем объединять их при создании документов определенного типа. Директива @import должна располагаться в начале таблицы стилей перед правилами. В начале таблицы стилей можно разместить несколько директив @import:

```
@import url (URL ИмпортТаблСтил);
```

Например:

```
/* Имя файла: Inventory01.css*/
@import url (Book.css);
BOOK
    {display:block;
    margin-top:12pt;
    font-size:10pt}
/* продолжение таблицы стилей... */
```

В случае возникновения конфликта правил основная таблица стилей (из файла, в который осуществляется импорт) имеет приоритет над импортируемыми таблицами стилей. Если импортируется несколько таблиц стилей, правила из таблицы стилей, импортированной последней, имеют приоритет над правилами из ранее импортированных таблиц стилей.

2.3. Пространство имен XML

Создавая XML-документ, размечая его конкретным набором тегов, созданных по правилам XML, мы получаем одну из реализаций языка XML.

Уже создано много общепринятых реализаций XML. Наиболее известные:

- ***XHTML –*** язык гипертекстовой разметки HTML, приведенный в соответствии с правилами XML;
- ***MathML (Mathematical Markup Language) –*** язык записи математических формул;
- ***CML (Chemical Markup Language) –*** язык записи химических формул.

Для того чтобы можно было различать имена тегов и атрибутов, которые имеют разный смысл в разных языках разметок, и можно было включить элементы или атрибуты одного типа документа в другой тип документа – в реализациях XML используют пространства имен.

Пространства имен представляют собой префикс, который отделяется от имен тегов или атрибутов двоеточием. Все имена тегов и атрибутов, префиксы которых связаны одним и тем же идентификатором, образуют одно пространство имен, в котором имена должны быть уникальны. Пространство имен помогает выделить процессору XML различные группы имен для различных способов обработки.

Пространство имен нужно объявить в документе прежде, чем оно будет использовано. Пространство имен действует только в пределах того элемента, атрибутом которого является его декларация (и его дочерних элементов). Объявление имеет вид атрибута внутри элемента. Оно начинается с ключевого слова ***xmlns:*** (сообщающего анализатору, что объявляется пространство имен), затем двоеточие, префикс пространства имен, знак равенства и URI, заключенный в кавычки. Префикс может иметь любое имя, удовлетворяющее правилам именования элементов.

Префикс и идентификатор пространства имен определяются атрибутом xmlns как в корневом, так и в любом другом элементе:

<ntb: notebook xmlns: ntb=”http://some.firm.com/2003/html”>

В этом теге префикс *ntb* определяется и одновременно используется в имени *ntb: notebook*, в дальнейшем имена вложенных тегов, которые относятся к пространству имен http://some.firm.com/2003/html, снабжаются префиксом ntb. В одном элементе можно определить несколько пространств имен.

Идентификатор пространства имен имеет форму *URI (Uniform Resource Identifier)* – уникальная строка символов, идентифицирующая пространство имен. Такой адрес не имеет никакого значения и может не соответствовать никакому действительному адресу Интернета. Программы, использующие документ, не будут обращаться по этому адресу, поскольку там нет никакой web-страницы. Просто идентификатор пространства имен должен быть уникальным во всем Интернете, и разработчики рекомендаций по применению пространств имен W3C рекомендовали использовать для него DNS-имя сайта, на котором размещено определение пространства имен.

Имя вместе с префиксом (например, ntb:notebook) – называется ***расширенным, полным,*** или ***уточненным именем***. Часть имени, записанная после двоеточия, называется ***локальной частью.***

префикс:имя_элемента(атрибута)

Атрибуты элемента не наследуют его пространство имен – если префикс пространства имен для атрибута не указан, то его имя не относится ни к какому пространству имен.

Пространство имен, используемое по умолчанию (default namespage), объявляется без двоеточия в имени атрибута xmlns:

1) элементы из пространства имен по умолчанию не требуют префикса:

<?xml version=”1.0”>

<journal xmlns=”http://www.psycholab.org/mynamespace/”>

2) атрибуты никогда не входят в пространство имен по умолчанию;

3) если имя атрибута записано без префикса, то это означает, что атрибут не относится ни к одному пространству имен. Программы, анализирующие документ, не будут искать такое имя ни в одном пространстве имен.

Пример документа, в котором определены два пространства имен myns и eq:

<? xml version=“1.0”?>

<myns journal xmlns:myns=“http://www.psycholabs.org/ mynamespace /”>

<myns:experiment>

<myns:date> September, 15, 2013</myns:date>

<myns:subject>XML Document </myns:subject>

<eq:formula xmlns:eq=“http://www.mathstuff.org/”>

<eq:variable>P</eq:variable>

</eq:formula>

</myns journa>

Префиксы, начинающиеся с символов *xml* с любым регистром букв, зарезервированы за самим языком XML. Префикс xml не нужно определять в XML-документе, он связан по умолчанию с идентификатором http://www.w3.org/XML/1998/namespace.

Вставка элементов HTML в XML-документы с использованием пространства имен. Когда XML-документ отображается с использованием CSS, можно добавить в него любой стандартный элемент HTML (гиперссылку, изображение, форму).

Для этого лучше в теге корневого элемента объявить пространство имен html, которое можно использовать внутри элемента, в котором его объявили, и внутри любого из его дочерних элементов. Пространство имен html является специальным, зарезервированным пространством имен, которое объявляется:

xmlns:html=’http://www.w3c.org/TR/REC=html40/’

После объявления, если имя XML-элемента совпадает с именем стандартного HTML-элемента (например, IMG, A, HR) и если он принадлежит пространству имен html, то Internet Explorer интерпретирует его как HTML-элемент и поместит на страницу.

Например, вставка IMG-элемента:

<html:IMG xmlns:html=’http://www.w3c.org/TR/REC-html40/’ src=’picture.jpj’ ALIGN=’LEFT’/>

Это корректно сформированный пустой XML-элемент, имеющий три атрибута: первый атрибут объявляет пространство имен, а второй и третий – стандартные HTML-атрибуты.

Например, вставка гиперссылки с помощью следующего XML-элемента:

<html:A xmlns:html='http://www.w3c.org/TR/REC-html40/'
HREF='http://www.edgar.com'>
Edgar Allan Poe
</html:A>

Или, например, вставка двух горизонтальных разделительных линий с использованием следующего XML-элемента:

<html:HR xmlns:html='http://www.w3c.org/TR/REC-html40/' />

Этот элемент вставляет стандартный элемент HTML HR (горизонтальная линия).

2.4. Отображение XML-документов с использованием связывания данных

При связывании данных соединяют XML-документ с HTML-страницей, а затем встраивают стандартные HTML-элементы, такие как SPAN или TABLE, в отдельные XML-элементы. HTML-элементы затем автоматически отображают содержимое XML-элементов, в которые они встроены.

При этом Internet Explorer создает программный объект, называемый ***"Объект исходных данных"*** (Data Source Object, DSO). Объект DSO хранит данные XML и обеспечивает доступ к ним. DSO позволяет осуществлять доступ и манипулирование XML-данными посредством ряда методов, свойств и событий. Internet Explorer берет на себя большую часть работы; не нужно писать сценарии или вызывать методы (функции). Можно использовать одну таблицу HTML для отображения XML-документа, структурированного как набор записей, или можно использовать вложенные HTML-таблицы для отображения XML-документа, содержащего иерархический набор записей (более сложную структуру записей).

Два основные этапа связывания данных:

1. Установить связь XML-документа с HTML-страницей, на которой необходимо отобразить данные XML:

<XML ID="dsoStudent" SRC="file_l .xml"></XML>,

где атрибут ID – это уникальный идентификатор, который используется для доступа к XML-документу с HTML-страницы.

2. Сцепить HTML-элементы с XML-элементами. Когда HTML-элемент сцепляется с XML-элементом, HTML-элемент автоматически отображает содержимое XML-элемента.

Для установки связи XML-документа с HTML-страницей можно использовать две формы записи на HTML-страницу:

- Поместить весь текст XML-документа между начальным и конечным тегами XML документа HTML:

```
<HTML>
<HEAD>
<TITLE> Лабораторная работа </TITLE>
</HEAD>
<BODY>
<XML ID="daoStudent">
<?xml version="1.0" encoding="windows- 1251" ?>
<STUDENT>
   <FAMILY>Иванов</FAMILY>
   <NAME>Сергей </NAME>
   <YEAR>1993</YEAR>
   <GROUP>ИФ 87 </GROUP>
</STUDENT >
</XML>
<! --Другие элементы HTML -->
</BODY>
</HTML>
```

- На странице HTML создать пустой HTML-элемент с именем XML. URL адрес XML-документа представлен значением атрибута XML тега. В качестве значения атрибута SRC может выступать как полный, так и относительный URL адрес XML-документа.

Пример фрагмента данных на HTML-странице:

```
<HTML>
<HEAD>
<TITLE> Лабораторная работа </TITLE>
</HEAD>
<BODY>
<XML ID="daoStudent" SRC="file_1.xml">
</XML>
```

```
<!--Другие элементы HTML -->
</BODY>
</HTML>
```

Текст XML-документа должен содержаться в отдельном файле file_1.xml:

```
<?xml version="l.0" encoding="windows-1251"?>
<STUDENT>
 <FAMILY>Иванов</FAMILY>
 <NAME>Сергей </NAME>
        <YEAR>1993</YEAR>
 <GROUP>ИФ 87 </GROUP>
</STUDENT>
```

В подавляющем большинстве случаев используется вторая форма, так как она больше соответствует основам философии XML, согласно которой собственно данные (XML-документ) хранятся отдельно от информации по их форматированию и обработке (таблицы стилей или в данном случае – HTML-страницы).

Вторая форма облегчает работу с XML-документом, особенно если один документ отображается на нескольких различных HTML-страницах.

Сцепление HTML-элементов с XML-элементами можно осуществить двумя способами:

1) табличное сцепление – сцепление HTML-элемента TABLE с данными XML (в таблице автоматически отображается весь набор записей, принадлежащих XML);

2) сцепление по отдельным записям – сцепление табличных элементов HTML с XML-элементами, так что за один раз отображается только одна запись.

2.5. Использование табличного сцепления данных

Самый простой способ отобразить XML-документ, который состоит из группы записей, – это сцепить HTML-элемент TABLE с данными XML таким образом, чтобы в таблице автоматически отображались сразу все записи (или одна страница записей за один раз, если установить режим постраничного отображения).

Можно использовать один HTML-элемент TABLE для отображения XML-документа, составленного по следующим правилам:

- корневой элемент содержит множество элементов типа запись;
- каждый элемент типа запись содержит одинаковый набор элементов типа полей;
- каждый элемент типа поле содержит только символьные данные.

Например, корневой элемент *FILE_1* содержит набор из четырех элементов-записей (элементы STUDENT), и каждый из элементов-записей имеет одинаковый набор элементов-полей, которые содержат только символьные данные (FAMILY, NAME, YEAR, GROUP).

Сцепление элемента TABLE с данными XML требует следующего синтаксиса:

<TABLE datasrc="#[ссылка]">
<TR>
<TD> <SPAN datafld = "имя элемента "/></SPAN</TD>
... инструкции для каждого поля ...
</TR>
</TABLE>

Элемент TABLE страницы сцеплен со всем XML-документом путем присвоения атрибуту *DATASRC* элемента идентификатора (ID) фрагмента данных, предваренного символом #:

<TABLE DATASRC="#dsoStudent" BORBER="6" CELLPADDING="5" width="100%">

Таблица определена с одной строкой (элемент TR). Каждая ячейка в этой строке, т.е. каждый элемент TD, состоит из элемента SPAN, который сцеплен с одним из полей XML-документа таким образом, что этот элемент отображает содержимое поля.

Так как элемент ID не является связываемым HTML-элементом, то он не может быть непосредственно сцеплен с полями XML, поэтому необходимо использовать элемент SPAN в качестве контейнера для данных XML. Атрибут datafld этого элемента задает конкретное поле, которое должно связываться. Например, для связывания элемента SPAN с элементом *FAMILY* из документа *file_l.xml* можно использовать следующую строку:

<TD> < SPAN DATAFLD ='"FAMILY"> </SPAN> <TD>

Несмотря на то, что в элементе TABLE определена только одна строка, когда браузер отображает таблицу, он повторяет строковый элемент для каждой записи в XML-документе.

Для обеспечения заголовков всех столбцов таблицы можно использовать раздел стандартного заголовка:

```
<THEAD Style="background-color:aqua">
<TH> Фамилия </TH>
... инструкции для каждого заголовка...
</THEAD>
```

2.6. Использование постраничного отображения

Если XML-документ содержит много записей, можно использовать постраничный вывод группы записей за один раз вместо отображения всех записей одновременно в огромной таблице. Для активизации постраничного отображения в обычной связанной таблице выполняются следующие действия:

- для сцепленного элемента TABLE добавляется атрибут DATAPAGESIZE, значение которого равно максимальному числу записей, отображаемых за один раз. Каждая страница записей будет содержать заданное число записей;
- для сцепленного элемента TABLE добавляется атрибут ID и ему присваивается уникальный идентификатор;
- для перемещения между записями таблицы вызываются методы элемента TABLE с использованием значения его уникального идентификатора в качестве объекта, вызывающего метод (представлено в табл. 2.1).

Таблица 2.1 – Использование методов элемента TABLE

Отображает первую страницу записей	ID_таблицы.firstPage ()
Отображает предыдущую страницу записей	ID_таблицы.previous Page()
Отображает следующую страницу записей	ID_таблицы. next Page ()
Отображает последнюю страницу записей	ID_таблицы.lastPage ()

В простейшем случае обращение к методам элемента TABLE заключается в присвоении метода атрибуту ONCLICK HTML-элемента BUTTON.

Задания к лабораторной работе 1
"Создание документа XML"

1. Проработайте предложенный ниже пример создания XML-документа и его отображения с помощью каскадных таблиц стилей:

а) создайте в текстовом редакторе Notepad новый файл и введите текст XML-документа, сохранив с расширением .xml:

```
<?xml version=”1.0” encoding="windows-1251”?>
<!--имя файла:file_1.xml-->
<FILE_1>  <STUDENT>
        <FAMILY>Иванов</FAMILY>
        <NAME>Сергей </NAME>
        <YEAR>1993</YEAR>
        <GROUP>ИФ 87</GROUP>
    </STUDENT>
    <STUDENT>
        <FAMILY>Петрова</FAMILY>
        <NAME>Галина </NAME>
        <YEAR>1992</YEAR>
        <GROUP>ИФ 87</GROUP>
    </STUDENT>
    <STUDENT>
        <FAMILY>Семенов </FAMILY>
        <NAME>Валерий </NAME>
        <YEAR>1993</YEAR>
        <GROUP>ИФ 88</GROUP>
  </STUDENT>
  <STUDENT>
        <FAMILY>Павлова</FAMILY>
        <NAME>Ирина </NAME>
        <YEAR >1994 </YEAR>
        <GROUP>ИФ 89</GROUP>
</STUDENT> </FILE_1>
```

Данный документ состоит из двух основных частей: пролога и корневого документа (называемого также элементом документа). Элемент документа называется здесь FILE_1, его начальный тег – <FILE_1>, а конечный – </FILE_l>, а содержимое – 4 вложенных эле-

мента STUDENT. В свою очередь, каждый элемент STUDENT содержит ряд вложенных элементов;

б) откройте документ с помощью браузера Internet Explorer (IE). После проверки синтаксиса документ отобразится на экране. При наличии ошибок вместо документа на экран будет выдано сообщение о невозможности отобразить страницу;

в) попробуйте изменить степень детализации представления элементов документа. Щелкните на символе знака минус (-) слева от начального тега, чтобы свернуть элемент, либо на знаке плюс (+) рядом со свернутым элементом, чтобы развернуть его. Например, щелкнув на знаке минус (-) рядом с элементом FILE_1, вы получите:

<?xml version="1.0" encoding="windows-1251" ?>

<!--Имя файла:file_1.xml -->

+ <FILE_1>

г) создайте в файле file_2.css каскадную таблицу стилей:

STUDENT

{disfplay:block;

margin-top:12pt;

font-size:10 pt}

FAMILY {font-style:italic}

NAME {font-weight:bold}

д) откройте в текстовом редакторе файл, созданный в первом пункте задания, и второй строкой документа создайте следующую инструкцию по обработке:

<?xml version="l.0" encoding="windows-12 51"?>

<?xml-stylesheet type="text/css" href="fi1e_2.css"?>

<!--имя файла:fale_10.xml-->

<FILE_1>

<STUDENT>

2. Создайте XML-документ, представляющий информацию по определенной вариантом предметной области. Созданный документ должен соответствовать следующим требованиям:

а) документы должны иметь глубину вложенности не менее четырех элементов;

б) число элементов документа, не имеющих вложенных, должно быть не менее пяти;

в) элементы документа должны содержать комментарии о своем содержании;

г) документ должен включать элементы, содержащие символьные данные и дочерний элементы;

3. Создайте таблицу каскадных стилей, которая отформатирует созданный XML-документ. Созданная CSS-таблица должна соответствовать следующим правилам:

а) CSS-таблица должна включать как контекстуальные, так и родовые селекторы;

б) дочерние элементы должны наследовать CSS-формат родительского элемента;

в) созданная CSS-таблица должна импортировать другую таблицу стилей;

г) таблица стилей должна включать использование атрибута STYLE;

д) созданная таблица стилей должна включать:

- свойства шрифта;
- свойство фона;
- свойства разбивки и выравнивания текста;
- свойства текстовых областей;
- свойства полей и границ;
- псевдоэлементы.

4. В отчете представить код xml-документа, код таблиц CSS и скриншот табличного представления документа.

5. Варианты предметных областей создаваемых XML-документов:

Вариант 1:	библиографическое описание списка литературы
Вариант 2:	описание фильмов видеотеки
Вариант 3:	список сотрудников организации
Вариант 4:	список моделей мобильных телефонов
Вариант 5:	список студентов факультета
Вариант 6:	список изучаемых дисциплин

ТЕМА 3

ВАЛИДНЫЙ XML-ДОКУМЕНТ

3.1. Преимущество использования валидных документов.

3.2. Объявление и форма записи типа элемента.

3.3. Объявление атрибутов.

3.4. Объявление внешней и полной схем DTD.

3.5. Сцепление по отдельным записям.

3.6. Использование связывания данных по одной записи.

3.7. Связывание валидных документов XML с помощью вложенных таблиц.

3.1. Преимущество использования валидных документов

Мы говорили о корректно сформированных (well-formed) документах. Если документ не является корректным (т.е. не отвечает минимальным требованиям по составлению XML-документа), то он не может считаться XML-документом.

Валидным (valid) называется корректно сформированный документ, отвечающий двум дополнительным требованиям:

1) пролог документа должен содержать *специальное объявление типа документа (DTD – Document Type Definition)*, задающее структуру документа;

2) остальная часть документа должна отвечать структуре, заданной в DTD. Нельзя включить в документ атрибуты или элементы, которые не объвлены в объявлении типа документа. Каждый включаемый в документ элемент или атрибут должен соответствовать специфи-кации, выраженной в соответствующем объявлении.

DTD обеспечивает стандартный шаблон для процессора. При проверке валидности процессор проверяет соответствие документа установленным в шаблоне стандартам.

Преимущества использования валидных документов:

- стандарт XML определяет DTD как "грамматику для определенного класса документов", поэтому использование валидных документов особенно полезно для проверки однородности среди группы схожих документов;

- если документ будет обрабатываться программой пользователя, ориентированной на определенную структуру документа, то пользователи могут быть уверены, что документы, прошедшие проверку на валидность, будут распознаны программой-обработчиком.

Описание или объявление типа документа DTD можно занести в отдельный файл (внешняя схема) или включить его непосредственно во вторую часть пролога XML документа (внутренняя схема).

Внутренняя схема. Объявление типа документа DTD представляет собой блок XML-разметки, который нужно добавить в пролог валидного XML-документа, в любое место после XML-объявления:

<! DOCTYPE Имя

[объявления разметки, описывающие логическую структуру документа, т.е. задающие элементы документа, атрибуты и др.

]

>

Имя указывает на имя элемента документа (или имя корневого элемента). Например, используя пример лабораторной работы 1, следует использовать имя *FILE_1*.

Пробелы при объявлениях можно добавлять в любом количестве. Ключевое слово DOCTYPE всегда указывается только символами верхнего регистра.

3.2. Объявление и форма записи типа элемента

Объявление типа элемента определяет типы элементов, которые может содержать документ, их содержимое (часто описывающее порядок размещения дочерних элементов).

Обобщенная форма записи типа элемента:

<! ELEMENT Имя описание_содержимого>

Объявить определенный тип элемента в данном документе можно только один раз. Ключевое слово *ELEMENT* должно быть указано с помощью символов верхнего регистра.

Описать содержимое элемента можно различными способами:

1) ключевое слово ***EMPTY*** – указывает, что элемент должен быть пустым (ключевое слово EMPTY указывается только с помощью символов верхнего регистра):

<!ELEMENT PICTURE EMPTY>

Валидные элементы, которые можно поместить в документ:

<PICTURE> </PICTURE>

<PICTURE/>

2) ключевое слово ***ANY*** – указывает, что элемент может иметь любое допустимое для этого типа содержимое. То есть элемент этого типа может содержать или не содержать дочерние элементы в любом порядке и с любым количеством вхождений, иметь или не иметь чередующиеся символьные данные. Это наиболее неопределенный тип описания содержимого, дающий возможность создавать типы элементов без ограничений на их содержимое. Ключевое слово ANY всегда указывается только в верхнем регистре;

3) <! ELEMENT Имя (***#PCDATA***)> – элемент с обычным текстовым содержимым и без вложенных элементов (***Parsed Character DATA*** – строка символов, просматриваемая программой-анализатором документа XML), может быть пустым. Ключевое слово #PCDATA всегда указывается только большими буквами:

<!ELEMENT SUBTITLE (#PCDATA)>

Следующие два элемента в соответствии с данной декларацией являются корректными:

<SUBTITLE>A New Task</SUBTITLE>

<SUBTITLE></SUBTITLE>

4) дочернее содержимое элемента. Элемент может содержать дочерние элементы, но не может содержать символьные данные;

5) смешанное содержимое. Элемент может содержать любое количество смешанных данных, чередующихся с дочерними элементами определенных типов.

Задание дочернего содержимого элемента. В тексте документа можно разделять дочерние элементы символами пробела, табуляции, перевода строки – процессор будет их игнорировать и не передаст приложению.

Задание дочерней модели содержимого может иметь следующие формы:

1. Если объявляемый элемент содержит строгую определенную последовательность вложенных дочерних элементов (каждый из которых встречается только один раз), то описание содержимого элемента должно содержать в скобках список их имен дочерних элементов, пе-

речисленных через запятую. Объявляется только один уровень вложенности, следующий уровень будет объявлен только при объявлении вложенного элемента:

<!ELEMENT STUDENT (FAMILY, NAME, YEAR, GROUP)>

<!ELEMENT FAMILY (#PCDATA)>

<!ELEMENT NAME (#PCDATA)>

<!ELEMENT YEAR (#PCDATA)>

<!ELEMENT GROUP (#PCDATA)>

2. Выборочная форма модели содержимого показывает, что элемент может иметь любой из перечисленных дочерних элементов, в любом порядке – дочерние элементы разделяются вертикальной чертой |:

<!ELEMENT STUDENT (FAMILY | NAME | YEAR | GROUP)>

Элемент STUDENT может состоять из одного дочернего элемента NAME или одного дочернего элемента YEAR или одного дочернего элемента GROUP:

<!ELEMENT FAMILY (#PCDATA)>

<!ELEMENT NAME (#PCDATA)>

<!ELEMENT YEAR (#PCDATA)>

<!ELEMENT GROUP (#PCDATA)>

3. Можно изменить две приведенные модели дочернего содержимого, используя дополнительные символы:

Символ	Значение
?	ни одного или один из предшествующих элементов
+	один или более одного из предшествующих элементов
*	ни одного или более одного из предшествующих элементов

<!ELEMENT MOUNTAIN (NAME+, HEIGHT?, STATE)>

Элемент MOUNTAIN может содержать один или более элементов NAME, один или ни одного элемента HEIGHT и один элемент STATE, все в строго заданном порядке.

<! DOCTYPE FILE_1

[

<!ELEMENT FILE_1(STUDENT+)>

<!ELEMENT STUDENT (FAMILY, NAME, YEAR, GROUP)>

<!ELEMENT FAMILY (#PCDATA)>
<!ELEMENT NAME (#PCDATA)>
<!ELEMENT YEAR (#PCDATA)>
<!ELEMENT GROUP (#PCDATA)>
]
>

4. Можно использовать символы ?, + или * для модификации всей модели содержимого, помещая символы сразу после закрывающих скобок:

<!ELEMENT FILM (STAR | NARRATOR | INSTRUCTOR)+>

Объявление дает возможность включить один или несколько дочерних элементов любого из этих трех типов в любом порядке. Такое объявление делает корректными следующие элементы:

<FILM>
<NARRATOR>Bertram Wooster</NARRATOR>
<STAR>Sean Connery</STAR>
<NARRATOR>Plug Basham</NARRATOR>
</FILM>
<FILM>
<STAR>Sean Connery</STAR>
<STAR>Meg Ryan</STAR>
</FILM>
<FILM>
<INSTRUCTOR>Stinker Pike</INSTRUCTOR>
</FILM>

5. Можно формировать сложные модели содержимого элемента путем вложения выборочной модели содержимого внутрь последовательной модели либо последовательной модели в выборочную модель.

Например, DTD задает, что каждый элемент FILM должен иметь один дочерний элемент TITLE; за ним должен следовать один дочерний элемент CLASS; после него должен идти один дочерний эле-мент STAR, NARRATOR или INSTRUCTOR:

<!DOCTYPE FILM
[
<!ELEMENT FILM (TITLE, CLASS, (STAR | NARRATOR | | INSTRUCTOR))>

<!ELEMENT TITLE (#PCDATA)>

<!ELEMENT CLASS (#PCDATA)>

<!ELEMENT STAR (#PCDATA)>

<!ELEMENT NARRATOR (#PCDATA)>

<!ELEMENT INSTRUCTOR (#PCDATA)>

]

>

В соответствии с этим DTD следующий элемент *Документ* будет корректным:

<FILM>

<TITLE>The Net</TITLE>

<CLASS>fantastic</CLASS>

<STAR>Sandra Bullok</STAR>

</FILM>

так же, как:

<FILM>

<TITLE>How to Use XML</TITLE>

<CLASS>instructional</CLASS>

<INSTRUCTOR>Penny Donaldson</INSTRUCTOR>

</FILM>

Задание смешанного содержимого элемента позволяет задавать символьные данные с необязательными дочерними элементами (любая комбинация текстовых данных и перечисленных дочерних элементов).

Для этого каждый тип дочернего элемента перечисляется в модели содержимого элемента, разделяя их символами | и помещая * в конце всей модели содержимого. Каждое имя элемента может использоваться в модели содержимого один раз.

Например:

*<! ELEMENT имя (#PCDATA | дочерний_элемент)**>*

или

*<!ELEMENT TITLE (#PCDATA | SUBTITLE)**>*

В соответствии с этим объявлением следующие элементы TITLE являются допустимыми:

<TITLE>

Moby-Dick

<SUBTITLE>Or, the Whale</SUBTITLE>

</TITLE>
<TITLE>
<SUBTITLE>Or, the Whale</SUBTITLE>
Moby-Dick
</TITLE>
<TITLE>
Moby-Dick
</TITLE>
<TITLE>
<SUBTITLE>Or, the Whale</SUBTITLE>
<SUBTITLE>Another Subtitle</SUBTITLE>
</TITLE>
<TITLE></TITLE>

3.3. Объявление атрибутов

В валидном XML-документе нужно также исчерпывающе объявить все атрибуты, которые предполагается использовать для элементов документа. Обычно атрибуты каждого элемента объявляются в одном месте с помощью *списков объявлений атрибутов (attribute declaration lists)*. Объявление атрибута выполняет следующие три функции:

1) определяет имена атрибутов, которые определены для данного элемента, т.е. их можно включать в начальный тег элемента;

2) устанавливает тип данных каждого элемента или список разрешенных значений;

3) задает описание режима атрибута (задает обязательность для каждого атрибута). Если атрибут необязателен, в объявлении списка атрибутов указывается, что должен делать процессор, если атрибут опущен (например, может содержаться значение атрибута по умолчанию, которое будет использовать процессор).

Синтаксис списка объявлений атрибутов представлен на рис. 3.1.

Тип атрибутов можно задать тремя различными способами:

1) строковый тип задается с использованием ключевого слова CDATA. Строковый тип атрибута позволяет присваивать атрибуту любое символьное значение (кроме символов <, >, &, “), задаваемое в кавычках;

2) маркерный тип;

3) нумерованный (перечисляемый) тип.

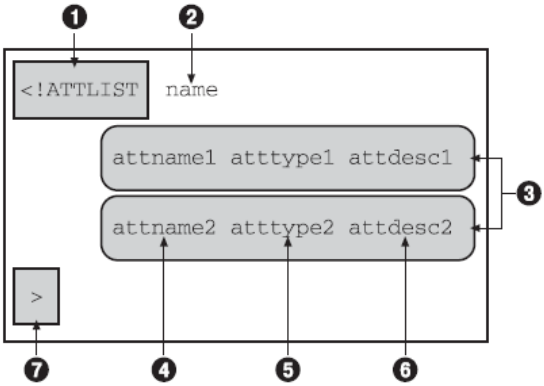


Рисунок 3.1 – Синтаксис списка объявлений атрибутов:
объявление начинается со строки <!ATTLIST (1); name – имя элемента (2), которому принадлежат атрибуты, далее следуют определения одного или нескольких атрибутов (3): attname1, attname2 – имя атрибута (4); atttype1, atttype2 – тип данных атрибута или перечисление значений (5); attdesc1,attdesc2 – описание режима атрибута (6); > – завершающий разделитель (7)

Маркерный тип. Значения, которые можно присвоить атрибуту маркерного типа, имеют ряд ограничений. Вид ограничения указывается с помощью специальных ключевых слов. Некоторые маркерные типы атрибутов:

- ***ID*** обозначает, что атрибуту каждого элемента должно быть присвоено уникальное значение; начинаться оно должно с буквы или символа подчеркивания (_), за которыми могут идти или не идти другие буквы, цифры, символы точки (.), тире (–) или символы подчеркивания. Данный тип элемента может иметь только один атрибут типа ID, а в объявлении значения атрибута по умолчанию должно фигурировать #REQUIRED (обязательный) или #IMPLIED;
- ***IDREF*** – значение атрибута должно совпадать со значением атрибута элемента типа ID внутри документа. Другими словами, этот тип атрибута является ссылкой на уникальный идентификатор другого

атрибута (указывает на элементы, содержащие атрибуты ID другого атрибута);

- ***NMTOKEN (name token)*** – строка символов, начинающаяся с буквы, далее может содержать буквы, цифры, точки (.), тире (–) или символы подчеркивания (_). Элементарное имя может также содержать двоеточие (:), но не на первом месте (не может содержать пробелов);
- ***NMTOKENS (name token list)*** – тип атрибута похож на тип NMTOKEN, но значение может содержать несколько элементарных имен, разделенных пробелами, внутри строки в кавычках;
- ***IDREF***.

Перечисляемый тип. Значение, которые можно присвоить атрибуту перечисляемого типа, должно совпадать с одним из имен, приведенных в списке типов атрибутов. Имена могут иметь одну из следующих форм:

1) в круглых скобках перечисляется список элементарных имен (перечисляемых значений), разделенных вертикальной чертой |.

Например, если значение атрибута Class ограничивается словами “fictional”, “instructional” или “documentary”, то атрибут определяется следующим образом:

```
<?xml version=”1.0” encoding = “windows-1251” ?>
<!DOCTYPE FILM
[
   <!ELEMENT FILM (TITLE, (STAR | NARRATOR |
| INSTRUCTOR))>
<!ATTLIST FILM
   Class (fictional | instructional | documentary) “fictional”>
<!ELEMENT TITLE (#PCDATA)>
<!ELEMENT STAR (#PCDATA)>
<!ELEMENT NARRATOR (#PCDATA)>
<!ELEMENT INSTRUCTOR (#PCDATA)>
]
>
<FILM Class=”instructional”>
   <TITLE> Использование XML </TITLE>
```

<NARRATOR> Иванов Геннадий </NARRATOR>

</FILM>

2) ключевое слово NOTATION, за которым идет пробел, затем открывающая скобка, затем список имен нотаций, разделяемых символами |, после чего следует закрывающая скобка. Каждое из этих имен должно точно соответствовать имени нотации, объявленному в DTD.

Режимы атрибутов. Если пользователь не указывает значение атрибута, процессор XML предполагает, что в DTD указано значение по умолчанию, его помещают в кавычках после указания режима. Значение по умолчанию должно соответствовать заданному типу атрибута:

- ***#IMPLIED*** – атрибут является необязательным, ему не присваивается значение по умолчанию;
- ***#REQUIRED*** – обязательный атрибут, обычно используется, когда нет значения по умолчанию и атрибут нельзя оставить пустым;
- ***#FIXED AttValue*** (*AttValue* – значение атрибута по умолчанию) – значение уже присвоено, пользователю нельзя его изменять. То есть атрибут всегда должен иметь одно и то же значение, даже если он опущен, атрибуту будет присвоено значение по умолчанию:

<!ATTLIST FILM Class CDATA #FIXED “documentary”>

В соответствии с этим объявлением два элемента будут корректными:

<FILM> Изучаем XML </FILM>

<FILM Class="documentary"> Изучаем XML </FILM>

3.4. Объявление внешней и полной схем DTD

Внешняя схема DTD имеет смысл в том случае, когда DTD являются общими для целой группы документов. Можно поместить все или часть DTD документа в отдельный файл, а затем ссылаться на этот файл из объявления типа документа.

В этом случае DTD или часть DTD, содержащаяся в отдельном файле, называется ***внешней схемой*** DTD.

Многие стандартные XML-приложения основаны на общем DTD, включаемом во все XML-документы, отвечающие этому приложению.

В объявлении типа документа, использующем внешнюю схему, все, что указывается в квадратных скобках, – ***внутренняя схема*** DTD:

<!DOCTYPE корневой_элемент [

<ELEMENT корневой_элемент (модель содержания)…>

]>

Внешняя схема DTD обычно содержится в отдельных файлах с расширением .DTD. Полная схема DTD для экземпляра XML состоит из комбинации внутреннего и внешнего подмножества DTD, если они существуют.

Внешняя схема DTD. В объявление типа документа включается ключевое слово SYSTEM, после которого указывается адрес URI файла, содержащего схему DTD. В настоящее время URI практически аналогичен стандартному (URL) Internet-адресу.

Может быть использовано ключевое слово PUBLIC, если схема DTD общедоступна и хранится в специальной библиотеке или репозитарии:

<!DOCTYPE корневой_элемент SYSTEM адрес файла>

Адрес файла указывается в виде системного литерала, заключенного в одинарные или двойные кавычки, и содержать любые символы, кроме кавычек. Системный литерал задает унифицированный идентификатор ресурса (URI) файла, содержащего внешнее подмножество DTD. URI представляет собой чрезвычайно гибкую систему нотации для адресации ресурсов. В настоящее время URI практически аналогичен стандартному Internet-адресу, известному как унифицированный указатель ресурса (URL).

Например:

<?xml version="1.0" encoding="windows-1251" ?>

<!DOCTYPE SIMPLE SYSTEM "Simple.dtd">

<SIMPLE> This is an extremely simplistic XML document. </SIMPLE>

Файл Simple.dtd должен иметь следующее содержимое:

<!ELEMENT SIMPLE ANY>

Файл, содержащий внешнюю схему DTD, может включать любые ранее описанные объявления разметки, которые включены во внутреннее подмножество DTD.

Полная схема DTD. Чтобы использовать одновременно внешнюю и внутреннюю схемы, следует в объявлении DTD после ключевого слова SYSTEM вместе с системным литералом, задающим местонахождение файла с внешним подмножеством DTD, внутри квадратных скобок ([ ]) указать объявление разметки внутреннего подмножества DTD.

Если используется полная схема DTD, XML-процессор объединяет содержимое внешнего и внутреннего подмножеств DTD таким образом:

- в общем случае он осуществляет слияние двух подмножеств, чтобы сформировать полный DTD;
- если атрибут с одним и тем же именем и типом элемента объявляется более одного раза, XML-процессор использует первое объявление и игнорирует все последующие;
- внутреннее подмножество DTD имеет приоритет перед внешним подмножеством DTD (даже если ссылка на внешнее подмножество идет первой в объявлении типа документа). Таким образом, любой атрибут, определенный во внутреннем подмножестве, доминирует над атрибутом с тем же именем и типом элемента, объявленным во внешнем подмножестве;
- XML-процессор игнорирует повторные объявления атрибутов и примитивов, повторное объявление элемента (даже если он объявлен тем же самым образом) является допустимым.

Например:

```
<?xml version="1.0" encoding="windows-1251" ?>
<!DOCTYPE BOOK SYSTEM "Book.dtd"
[
   <!ATTLIST BOOK ISBN CDATA #IMPLIED  Year CDATA "2000">
   <!ELEMENT TITLE (#PCDATA)>
]
>
<BOOK Year="1998">
   <TITLE>The Scarlet Letter</TITLE>
</BOOK>
```

Содержимое файла *Book.dtd*, в котором хранится внешнее подмножество DTD:

<!ELEMENT BOOK ANY>

<!ATTLIST BOOK ISBN NMTOKEN #REQUIRED>

В рассмотренном примере объединенный DTD определяет два элемента: *TITLE* и *BOOK*, а также два атрибута для элемента BOOK: *ISBN* и *Year*. XML-процессор считает, что атрибут ISBN имеет тип CDATA и объявление значения по умолчанию *#IMPLIED*, поэтому приведенный элемент (в котором не указан ISBN) является корректным.

Объединение внутреннего и внешнего подмножеств DTD XML-процессором предоставляет возможность использовать общий DTD (например, такой, который используется для XML-приложений) в качестве внешнего подмножества DTD, а затем адаптировать (или субклассировать) DTD для конкретного документа путем включения внутреннего подмножества.

Можно заставить XML-процессор игнорировать часть внешнего подмножества DTD с помощью раздела IGNORE.

Например, можно использовать раздел IGNORE при разработке документа с целью временного отключения необязательного блока, подобно "комментированию" фрагмента кода, который необходимо временно игнорировать.

Раздел IGNORE начинается с символов <![IGNORE[и заканчивается символами]]>.

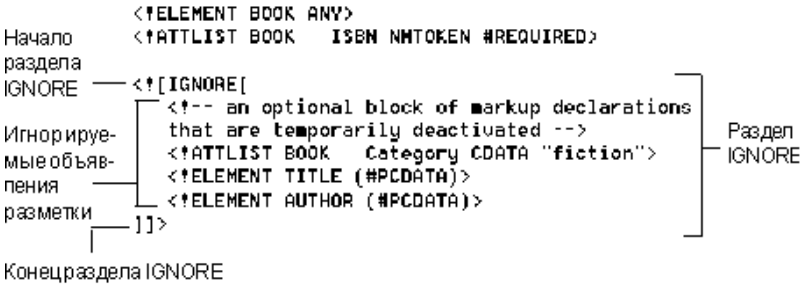


Рисунок 3.2 – Раздел IGNORE

Например, на рис. 3.2 представлено описание внешнего подмножества DTD, включающего раздел IGNORE.

Если необходимо временно восстановить блок объявлений разметки в разделе IGNORE, достаточно заменить ключевое слово IGNORE на INCLUDE, не удаляя при этом символы-ограничители (<![, [и]]>), как в следующем примере:

```
<![INCLUDE[
    <!-- необязательный блок объявлений разметки, который временно восстановлен -->
    <!ATTLIST BOOK  Category CDATA "fiction">
    <!ELEMENT TITLE (#PCDATA)>
    <!ELEMENT AUTHOR (#PCDATA)
]]>
```

Впоследствии вы можете снова быстро отключить раздел, вернув заголовок IGNORE. Раздел INCLUDE, вложенный в раздел IGNORE, также игнорируется.

Можно использовать разделы IGNORE и INCLUDE только во внешнем подмножестве DTD либо во внешнем параметрическом примитиве.

3.5. Сцепление по отдельным записям

Связывание данных работает только с XML-документом, который симметрично структурирован, например, таким как база данных, – элементы документа могут быть инерпретированы как набор записей и полей. В простейшем случае такой документ состоит из корневого элемента, содержащего множество элементов одинакового типа (записи), каждый из которых имеет одинаковый набор дочерних элементов, все из которых содержат символьные данные (поля).

Если структура документа такова, что не допускает связывания данных, то можно использовать метод создания сценариев.

Когда Internet Explorer открывает HTML-страницу, со встроенным XML-документом он создает программный объект – объект исходных данных (DSO), который хранит данные XML и обеспечивает доступ к этим данным. DSO хранит данные XML как набор записей, т.е. множество записей и их полей.

Когда сцепляется HTML-элемент с XML-элементом, DSO автоматически предоставляет значение XML-элемента и управляет всеми его свойствами. DSO также позволяет напрямую осуществлять доступ и манипулирование имеющимся набором записей посредством ряда методов, свойств и событий.

Методы представляют собой функции, которые можно вызывать со страницы для доступа или модификации набора записей. Например, можно использовать методы для перемещения между записями. Свойства представляют собой установленные на данный момент параметры, которые можно считывать и в ряде случаев изменять на странице. Например, можно считать свойство, которое сообщает, достигнута ли последняя запись. События представляют собой определенные смены состояний (например, изменение значений записи), которыми можно управлять посредством функции сценария, который создается для страницы.

3.6. Использование связывания данных по одной записи

Связывание данных по одной записи используется для HTML-элементов, которые не являются таблицами и не включены в связанную таблицу.

HTML-элемент – например, SPAN, BUTTON или LABEL – связывается с отдельным полем XML. После этого HTML-элемент автоматически отображает содержимое поля XML, с которым он связан (сцеплен).

Например, следующий HTML-элемент SPAN сцеплен с полем TITLE XML-документа, доступ к которому осуществляется через фрагмент данных dsoBook:

<SPAN DATASRC="#dsoBook" DATAFLD="TITLE"></SPAN>

Поскольку данный HTML-элемент не имеет множественных частей подобно таблице, он способен отобразить значение поля только для одной записи за раз. Чтобы использовать связывание данных по одной записи, XML-документ должен быть организован как простой набор записей. Наипростейшим случаем связывания данных по одной записи является случай, когда XML-документ состоит только из одной записи:

<?xml version="1.0"?>

```
<STUDENT>
   <GROUP> &quote; ИФ-80 &quote;</GROUP>
   <SURNAME> Иванов </SURNAME>
   <NAME>Геннадий</NAME>
   <BIRTHDAY> 29 ноября 1992</BIRTHDAY>
</STUDENT>
```

HTML-страница, которая связывает отдельный элемент SPAN с каждым из полей рассматриваемого документа (GROUP, SURNAME, NAME, BIRTHDAY):

```
<!--имя файла: lab4.html -->
<HTML>
<HEAD>
  <TITLE>Лабораторная работа 4</TITLE>
</HEAD>
<BODY>
  <XML ID="dsoLab4" SRC="lab4.xml"></XML>
  <H1> Список студентов </H1>
  <SPAN STYLE="font-style:italic"> Группа </SPAN>
  <SPAN STYLE="font-weight:bold" DATASRC="#dsoLab4"
DATA-FLD="GROUP"></SPAN>
<BR>
  <SPAN STYLE="font-style:italic"> Фамилия </SPAN>
  <SPAN STYLE="font-weight:bold" DATASRC="#dsoLab4"
DATA-FLD="SURNAME"> </SPAN>
<BR>
  <SPAN STYLE="font-style:italic"> Имя</SPAN>
  <SPAN STYLE="font-weight:bold" DATASRC="#dsoLab4"
DATA-FLD="NAME"> </SPAN>
<BR>
  <SPAN STYLE="font-style:italic"> Дата рождения</SPAN>
  <SPAN STYLE="font-weight:bold" DATASRC="#dsoLab4"
DATA-FLD="BIRTHDAY"> </SPAN>
<BR>
 </BODY>
 </HTML>
```

Если XML-документ содержит более одной записи, связывание по записям становится более сложным, так как HTML-элемент может отобразить только одну запись. Отображаемая в данный момент запись называется ***текущей записью***. Изначально текущей является первая запись в документе.

Связывание данных по одной записи иногда называют еще ***связыванием по текущей записи***.

DSO (объект исходных данных), ассоциированный с XML-документом, предоставляет ряд методов (функций), которыми можно воспользоваться при перемещении между записями, приведенными в табл. 3.1. Эти методы принадлежат объекту recordset DSO.

Таблица 3.1 – Методы объекта recordsetDSO

Метод объекта recordsetDSO	Переход от текущей записи к	Пример вызова
moveFirst	первой записи в документе	dsoInventory.recordset. moveFirst()
movePrevious	предыдущей записи	dsoInventory.recordset. movePrevious()
moveNext	следующей записи	dsoInventory.recordset. moveNext()
moveLast	последней записи в документе	dsoInventory.recordset. moveLast()
move	записи с указанным номером	dsoInventory.recordset. move(5) (Переход к пятой записи. Записи нумеруются, начиная с нуля)

В примерах вызовов, приведенных в последней графе, предполагается, что HTML-страница содержит фрагмент данных XML с идентификатором (ID) dsoInventory.

Составной объект recordset DSO соответствует стандарту технологии доступа к данным, которую Microsoft назвала ActiveX Data Objects (ADO). Можно использовать объект общего назначения ADO recordset совместно с множеством различных источников данных, а не только с XML DSO.

Можно обращаться к этим методам из написанного кода сценария. Однако самый простой способ их вызова – это присвоить имя метода атрибуту ONCLICK элемента BUTTON, как в примере ниже:

```
<BUTTON ONCLICK="dsoLab4.recordset.moveFirst()">
  Первая запись
</BUTTON>
```

Этот элемент отображает кнопку. Когда пользователь щелкает мышью по кнопке, вызывается метод, присвоенный атрибуту ONCLICK, dsoLab4.recordset.moveFirst().

Методы JavaScript, которые встречались раньше, были защищены от достижения начала или конца файла благодаря тому, что игнорировали вызовы записей, выходящих за рамки действия документа. Методы DSO не предоставляют подобной защиты. Если текущей является первая запись, вызов метода movePrevious приводит к перемещению в зону начала файла (BOF), где нет записей, поэтому сцепленный элемент будет пуст. Аналогично, вызов метода moveNext, если текущей является последняя запись, приводит к перемещению в зону конца файла (EOF), поэтому сцепленный элемент также будет пуст.

Для избегания этого используется поддерживаемое объектом recordset свойство BOF (Beginning Of File – метка начала файла), которое принимает значение true (истина), если достигнуто начало файла, а также свойство EOF (End Of File – метка конца файла), которое принимает значение true (истина), если достигнут конец файла. Можно использовать указанные свойства для определения этих состояний и внесения необходимых корректировок.

Можно использовать свойства BOF и EOF, применив инструкцию if и сценарий кнопки. Например, приведенный ниже код предписывает при щелчке на кнопке в случае, если достигнуто начало файла, быстро отобразить первую запись:

```
<BUTTON ONCLICK="dsoLab4.recordset.movePrevious();
   if (dsoLab4.recordset.BOF)
   dsoLab4.recordset.moveNext()">
Назад
</BUTTON>
```

Следующий код проверяет достижение конца файла:

```
<BUTTON ONCLICK="dsoLab4.recordset.moveNext();
```

if (dsoLab4.recordset.EOF)

dsoLab4.recordset.movePrevious()">

Вперед

</BUTTON>

Атрибуту ONCLICK (либо другим атрибутам, относящимся к событиям, например, ONMOUSEOVER) можно присвоить целый блок кода сценария.

Существуют другие способы связывания нетабличных HTML-элементов. В самой простой форме, применимой к XML, связывание данных подразумевает извлечение текстовых данных из элементов XML с помощью программы для их сопоставления с элементами HTML.

Возможность связывания поддерживается следующими элементами HTML:

- A;
- APLET;
- BUTTON;
- DIV;
- FRAME;
- IMG;
- INPUT;
- LABEL;
- MARQUE;
- SELECT;
- SPAN;
- TEXTAREA.

Например, элементы DIV и SPAN используются как контейнеры для данных XML, которые размещаются среди других элементов HTML для контроля над отображением данных в окне браузера. Когда связывается элемент HTML с элементом XML, браузер отображает текстовые данные элемента XML.

Синтаксис элементов DIV и SPAN:

<DIV datasrc=”#[ссылка]” datafld=”[имя элемента XML] “> </DIV>

<SPAN datasrc=”#[ссылка]” datafld=”[имя элемента XML] “> </SPAN>

Атрибут *datasrc* указывает на источник данных, которые должны связываться с элементом HTML. Атрибут datafld задает конкретное поле, которое должно связываться. Элементы DIV и SPAN – это элементы контейнеры, которые могут содержать данные, стили или другие элементы HTML, предназначенные для использования на HTML-странице.

Элемент A (“анкер”, используется для создания гиперссылок):

```
<A DATASRC="dsoInventory" DATAFLD="REVIEWS">
  Click here for reviews
</A>
```

Это свойство, как и атрибут HREF элемента, устанавливает URL для гиперссылки. Следовательно, из поля XML извлекается URL гиперссылки для сцепленного элемента A, а не его текстовое содержимое. XML-поле REVIEWS в рассматриваемом примере должно содержать корректный URL.

Свойство src – это свойство, как и атрибут SRC элемента, задает URL файла, содержащего графические данные:

```
<?xml version="1.0" encoding="windows-1251" ?>
<!--Имя файла: lab4.xml-->
<STUDENT>
   <GROUP>  ИФ-80 </GROUP>
   <SURNAME> Иванов </SURNAME>
   <NAME>Геннадий</NAME>
   <BIRTHDAY> 29 ноября 1992</BIRTHDAY>
   <FOTO>64.jpg</FOTO>
</STUDENT>
<!-- Имя файла: lab4.htm -->
<HTML>
 <HEAD>
   <TITLE>Лабораторная работа 4</TITLE>
 </HEAD>
 <BODY>
   <XML ID="ds" SRC="lab4.xml"></XML>
   <H1> Список студентов </H1>
```

```
<SPAN STYLE="font-style:italic"> Группа </SPAN>
<SPAN STYLE="font-weight:bold" DATASRC="#ds" DATA-
FLD="GROUP"></SPAN>
<BR>
<SPAN STYLE="font-style:italic"> Фамилия </SPAN>
<SPAN STYLE="font-weight:bold" DATASRC="#ds" DATA-
FLD="SURNAME"> </SPAN>
<BR>
<SPAN STYLE="font-style:italic"> Имя</SPAN>
<SPAN STYLE="font-weight:bold" DATASRC="#ds" DATA-
FLD="NAME"> </SPAN>
<BR>
<SPAN STYLE="font-style:italic"> Дата рождения</SPAN>
<SPAN STYLE="font-weight:bold" DATASRC="#ds" DATA-
FLD="BIRTHDAY"> </SPAN>
<BR>
<IMG DATASRC="#ds" DATAFLD="FOTO" >
</BODY>
</HTML>
```

В данном примере элемент *IMG* сцеплен с полем *FOTO* XML-документа и поэтому отобразит файл *64.jpg*, содержащий фотографию сотрудника.

3.7. Связывание валидных документов XML с помощью вложенных таблиц

Если XML-документ отображается с использованием связанных данных, то включение DTD и превращение его в валидный обеспечит требуемую симметричность организации набора записей и элементов в документе.

Включение DTD особенно полезно для документа, который имеет более сложную иерархическую структуру записей, чем та, которую можно отобразить с помощью вложенных таблиц.

Если документ, отображаемый с помощью связывания данных, содержит ошибки валидности, сообщение об ошибке не появится, но в сцепленных элементах не будут отображены никакие данные. Чтобы

увидеть сообщение об ошибке в связном XML-документе, следует протестировать данный документ на валидность.

Следует помнить, что в объявлении элемента для записи (такой, как STUDENT) необходимо использовать модель содержимого, которая исчерпывающе описывает все поля записи и вложенные записи. При этом нельзя использовать спецификацию содержимого ANY, что приведет к нарушению связей между данными.

Пример использования валидного XML-документа для связывания данных:

```
<?xml version="1.0" encoding="windows-1251" ?>
<!-- Имя файла: lab4_Valid.xml -->
<!DOCTYPE INVENTORY
  [
  <!ELEMENT INVENTORY (CATEGORY*)>
  <!ELEMENT CATEGORY (CATNAME, BOOK*)>
  <!ELEMENT CATNAME (#PCDATA)>
  <!ELEMENT BOOK (TITLE, AUTHOR, BINDING, PAGES,
PRICE)>
  <!ELEMENT TITLE (#PCDATA)>
  <!ELEMENT AUTHOR (#PCDATA)>
  <!ELEMENT BINDING (#PCDATA)>
  <!ELEMENT PAGES (#PCDATA)>
  <!ELEMENT PRICE (#PCDATA)>
  ]
>
<INVENTORY>
  <CATEGORY>
    <CATNAME>Учебная литература</CATNAME>
    <BOOK>
      <TITLE>Освой XML за 21 день</TITLE>
      <AUTHOR>Деван Шеперд</AUTHOR>
      <BINDING>твердый переплет</BINDING>
      <PAGES>414</PAGES>
      <PRICE>300 грв</PRICE>
    </BOOK>
    <BOOK>
```

```
      <TITLE>Изучаем XML</TITLE>
      <AUTHOR>Эрик Рэй</AUTHOR>
      <BINDING>твердый переплет</BINDING>
      <PAGES>385</PAGES>
      <PRICE>290 грв</PRICE>
    </BOOK>
  </CATEGORY>
  <CATEGORY>
    <CATNAME>Исторические романы</CATNAME>
    <BOOK>
      <TITLE>Россия молодая</TITLE>
      <AUTHOR>Ю. Герман</AUTHOR>
      <BINDING>мягкая обложка</BINDING>
      <PAGES>225</PAGES>
      <PRICE>186 грв</PRICE>
    </BOOK>
    <BOOK>
      <TITLE>Алая королева</TITLE>
      <AUTHOR>Ф. Грегори</AUTHOR>
      <BINDING>бумажная обложка</BINDING>
      <PAGES>295</PAGES>
      <PRICE>104 грв</PRICE>
    </BOOK>
</CATEGORY>
 <CATEGORY>
    <CATNAME>Детективы</CATNAME>
    <BOOK>
      <TITLE>Шерлок Холмс и не только</TITLE>
      <AUTHOR>А. Конан Дойл</AUTHOR>
      <BINDING>твердый переплет</BINDING>
      <PAGES>324</PAGES>
      <PRICE>114грв</PRICE>
    </BOOK>
    <BOOK>
      <TITLE>Большой Боб. Рука</TITLE>
      <AUTHOR>Ж. Сименон</AUTHOR>
```

```
      <BINDING>твердый переплет</BINDING>
      <PAGES>438</PAGES>
      <PRICE>435грв</PRICE>
    </BOOK>
  </CATEGORY>
</INVENTORY>
```

Элементы данного XML-документа представляют собой иерархический набор записей, поэтому его нельзя отобразить как простой набор записей, где каждая запись состоит из фиксированного набора полей, каждое из которых содержит только символьные данные.

В иерархическом наборе записей каждая запись может содержать, в дополнение к фиксированному набору полей, переменное число вхождений (ноль или более вложенных записей). В представленном примере валидного документа корневой элемент INVENTORY содержит группу записей CATEGORY. Каждая запись CATEGORY начинается с поля CATNAME, которое содержит только символьные данные, а затем следуют нуль или несколько вложенных записей BOOK. Каждая вложенная запись BOOK имеет пять полей: TITLE, AUTHOR, BINDING, PAGES, PRICE.

HTML страница, использующая вложенную таблицу для отображения иерархической структуры записей валидного XML-документа:

```
<HTML>
<HEAD>
  <TITLE> Классификация литературы </TITLE>
</HEAD>
<BODY>
  <XML ID="dsoLabVal" SRC="lab4_valid.xml">
  </XML>
  <TABLE DATASRC="#dsoLabVal" BORDER="1">
    <THEAD>
      <TH>Классификация литературы</TH>
    </THEAD>
    <TR>
      <TD><SPAN DATAFLD="CATNAME"></SPAN></TD>
    </TR>
    <TR>
```

```
<TD>
  <TABLE DATASRC="#dsoLabVal" DATAFLD="BOOK"
    BORDER=0 CELLSPACING=10>
    <THEAD>
      <TH>Название</TH>
      <TH>Автор</TH>
      <TH>Обложка</TH>
      <TH>Количество страниц</TH>
      <TH>Цена</TH>
    </THEAD>
    <TR ALIGN="CENTER">
      <TD><SPAN DATAFLD="TITLE"
        STYLE="font-style:italic"></SPAN></TD>
      <TD><SPAN DATAFLD="AUTHOR"></SPAN></TD>
      <TD><SPAN DATAFLD="BINDING"></SPAN></TD>
      <TD><SPAN DATAFLD="PAGES"></SPAN></TD>
      <TD><SPAN DATAFLD="PRICE"></SPAN></TD>
    </TR>
  </TABLE>
</TD>
</TR>
</TABLE>
</BODY>
</HTML>
```

Внешняя таблица сцеплена с XML-документом:

```
<TABLE DATASRC="#dsoLabVal" BORDER="1">
```

и включает заголовок:

```
<THEAD>
    <TH>Классификация литературы</TH>
</THEAD>
```

и две строки таблицы (два элемента TR). Браузер повторяет эти две строки для каждой записи верхнего уровня (т.е. для каждой записи CATEGORY). В первой строке отображается поле CATNAME, подобно простому набору записей:

```
<TR>
  <TD><SPAN DATAFLD="CATNAME"></SPAN></TD>
```

</TR>

Вторая строка не отображает поле, а содержит вложенную таблицу, которая отображает содержимое каждой вложенной записи BOOK внутри текущей категории (вложенная таблица выделена цветом в листинге программы).

При этом вложенную таблицу необходимо сцепить не только с XML-документом (DATASRC="#dsoLabVal"), но и с вложенными записями BOOK (DATAFLD="BOOK"), чтобы в таблице отображалось содержимое каждой записи BOOK, вложенной в текущую запись CATEGORY. Другими словами, строковый элемент (TR) в этой таблице будет повторен для каждого из этих элементов BOOK. Тогда как внешняя таблица по умолчанию сцеплена с записями верхнего уровня – в данном случае, с записями CATEGORY – поэтому каждая из этих записей отображается при переходе к новой категории.

Задания к лабораторной работе 2 "Отображение XML-документов с использованием связывания данных"

1. Сцепите HTML-элемент ***TABLE*** с данными документа XML ***file_1.xml***, созданного в лабораторной работе 1:

а) в текстовом редакторе Notepad создайте новый файл ***lab2.html***:

```
<!--имя файла: lab2.html -->
<HTML>
<HEAD>
   <T1TLE>Лабораторная работа 2<TITLE>
</HEAD>
<BODY>
        <XML ID="dsoStudent" SRC="file_l. xml"></XML>
        <H2> Список студентов </H2>
        <TABLE DATASRC="#dsoStudent“ BORDER="6"
CELLPADDING="5" width="100%">
                <THEAD style="background-color: maroon">
                <TH> Фамилия </TH>
                <TH> Имя </TH>
                <TH> Год рождения </TH>
                <TH> Группа </TH>
```

```
                    </THEAD>
                    <TR ALIGN='"center">
                <TD><SPAN      DATAFLD="family"      STYLE="font-
style:italic"></SPAN></TD>
                        <TD><SPAN
DATAFLD="NAME"></SPAN></TD>
                        <TD><SPAN
DATAFLD="YEAR"></SPAN></TD>
                        <TD><SPAN
DATAFLD="GROUP"></SPAN></TD>
                    </TR>
                </TABLE>
    </BODY>
    </HTML>
```

б) сохраните файл в той папке, где находится file_1.xml и откройте в браузере Internet Explorer;

в) объясните значение каждого элемента и атрибута.

2. Создайте новый файл ***lab2_str.html***, в котором реализовать постраничный вывод записей документа file_l.xml:

а) добавьте элементу TABLE атрибут ID, присвоив ему уникальное значение, и атрибут DATAPAGESIZE, присвоив ему значение 2:

```
<TABLE ID="IrvertT" DATAPAGESIZE="2" DATASRC="#dsoStudent"
BORDER="6" CELLPADDING="5"' width="100%">
```

б) в верхнюю часть страницы добавьте четыре элемента BUTTON, с атрибутом ON-CLICK, значение которого для каждой кнопки определяется одним из методов элемента TABLE:

```
<BUTTON ONCLICK="InvertT.nextPage()"> Следующая страница
</BUTTON>
```

в) созданный документ должен выглядеть в Internet Explorer 5 следующим образом:

Список студентов

Щелчок мышью на кнопке «Следующая страница» должен приводить к отображению следующих двух записей. Аналогично должно соответствовать подписи действие при щелчке мыши по трем другим кнопкам.

3. Используя варианты предметных областей, создайте XML-документ, содержащий не менее 15 записей. Используя сцепление HTML-элемента TABLE с данными документа XML, реализуйте постраничный вывод записей в браузере (по пять записей на одной странице) и кнопки постраничного перехода.

4. Варианты предметных областей создаваемых XML-документов:

Вариант 1:	библиографическое описание списка литературы
Вариант 2:	описание фильмов видеотеки
Вариант 3:	список сотрудников организации
Вариант 4:	список моделей мобильных телефонов
Вариант 5:	список студентов факультета
Вариант 6:	список изучаемых дисциплин

ТЕМА 4

ИСПОЛЬЗОВАНИЕ ПРИМИТИВОВ

4.1. Типы примитивов (сущностей)

Механизм примитивов в XML предназначен для:

- повышения производительности;
- встраивания различных типов данных XML-документа.

Семантические сущности (или примитивы) ничего не добавляют к разметке, это просто удобный способ облегчить написание, сопровождение и чтение XML.

В спецификации XML – термин примитив (entity), в широком смысле, относится к любому из следующих типов единиц хранения информации для XML-документов:

- собственно XML-документ как целое;
- внешнее подмножество DTD;
- внешний файл, определенный как внешний примитив в DTD и допускающий использование посредством ссылки;
- строка в кавычках, определенная как внутренний примитив в DTD и допускающая использование посредством ссылки.

Существует множество разновидностей сущностей, которые классифицируются по трем признакам:

1. *Общие и параметрические.* Общий примитив включает содержимое документа – XML-текст или другие текстовые или нетекстовые данные, которые можно использовать внутри корневого элемента. Параметрический примитив содержит XML-текст, который может быть помещен в DTD.

2. *Внутренние и внешние.* Внутренний примитив содержится внутри строки в кавычках. Внешний примитив содержится в отдельном файле.

3. *Разбираемые и неразбираемые.* Разбираемый примитив содержит XML-текст (символьные данные, разметка или то и другое). Когда используется ссылка на разбираемый примитив, ссылка замещается содержимым примитива (замещающий текст), который становится составной частью документа. Синтаксический анализатор XML разбирает (сканирует) содержимое примитива точно так же, как он сканирует непосредственно введенный в документ текст. Неразбираемый примитив может содержать любой тип данных: XML-данные или, что чаще, не XML-данные. Не XML-данные могут представлять собой либо текстовые данные (например, название) или нетекстовые данные (например, графические данные для изображения). Неразбираемый примитив нельзя непосредственно вставить в документ посредством ссылки на него.

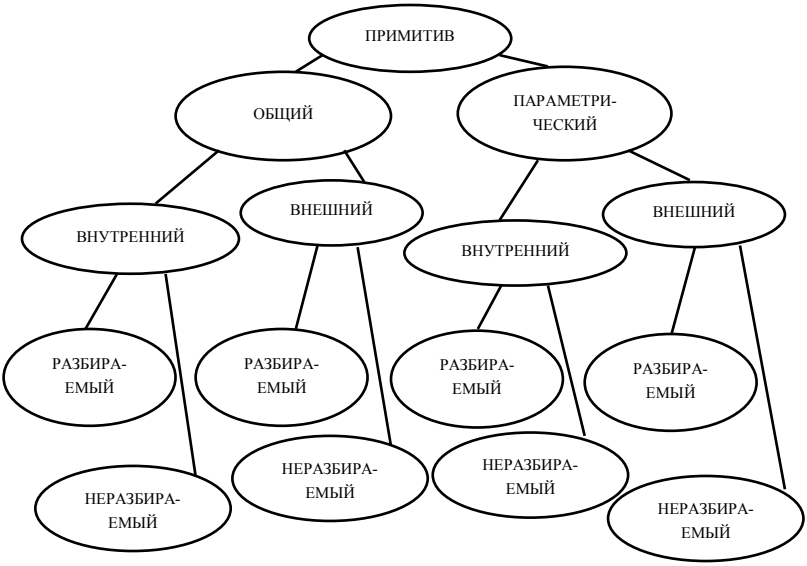


Рисунок 4.1 – Типы сущностей в XML

Реально в XML имеется только пять типов сущностей, представленных на рис. 4.1 (так как три типа сущностей в XML не поддерживаются):

1) общие внутренние разбираемые;

2) общие внешние разбираемые;

3) общие внешние неразбираемые;

4) параметрические внутренние разбираемые;

5) параметрические внешние разбираемые.

Сущность состоит из имени и значения. Объявление сущности, в котором имя связывается со значением, создается в DTD документа. Когда анализатор просматривает XML-документ, он находит ссылки на сущности (entity references), обозначающие &имя_примитива, и заменяет их соответствующим текстом или разметкой.

4.2. Объявление общих примитивов

Общие внутренние разбиваемые сущности. Объявление общей внутренней разбиваемой сущности имеет следующую форму записи:

<!ENTITY Имя_примитива Значение_примитива>

Имя примитива отвечает общим требованиям имен XML, общий примитив может иметь такое же имя, что и параметрический примитив, так как они занимают различные пространства имен.

Значение примитива представляет собой литерал (группу символов, заключенных в кавычки). Содержание литерала должно быть корректным для места, в которое предполагается вставлять примитив (вложенные элементы, символьные данные, значение атрибута):

```
<!DOCTYPE ARTICLE
[
   <!ELEMENT ARTICLE (TITLEPAGE, INTRODUCTION, SECTION*)>
   <!ELEMENT TITLEPAGE (#PCDATA | SUBTITLE)*>
   <!ELEMENT SUBTITLE (#PCDATA)>
   <!ELEMENT INTRODUCTION (#PCDATA)>
   <!ELEMENT SECTION (#PCDATA)>
   <!ENTITY title
      "The Story of XML
      <SUBTITLE> The Future Language of the Internet </SUBTITLE>">
]>
```

Сущность *title* содержит символьные данные и элемент *SUBTITLE*, его можно корректно вставить только в элемент *TITLEPAGE*:

<TITLEPAGE>

Title: &title;

Author: Michael Young

</TITLEPAGE>

XML-процессор заменит ссылку на примитив (&title;) содержимым примитива и обработает содержимое, как если бы вы непосредственно набрали его в документе в позиции ссылки, подобно следующему:

<TITLEPAGE>

Title: The Story of XML

<SUBTITLE>The Future Language of the Internet</SUBTITLE>

Author: Michael Young

</TITLEPAGE>

Общие внешние разбираемые сущности. Объявление для общей внешней разбираемой сущности имеет следующую форму записи:

<!ENTITY ИмяПримитива SYSTEM СистемЛитерал>

СистемЛитерал описывает местонахождение файла, содержащего данные примитива. Системный литерал может быть ограничен одинарными (') или двойными (") кавычками и содержать любые символы, кроме кавычек, используемых как ограничители.

Системный литерал задает унифицированный идентификатор ресурса (URI) файла, содержащего данные примитива. На сегодняшний день URI – практически то же самое, что и стандартный Internet-адрес, известный как унифицированный указатель ресурса (URL). Вы можете использовать либо полный URI, либо относительный.

Файл общего внешнего разбираемого примитива может содержать только те составляющие, которые могут быть корректно вставлены в элемент.

Можно вставить общий внешний разбираемый примитив только внутрь содержимого элемента:

<!DOCTYPE ARTICLE

[

<!ELEMENT ARTICLE (TITLEPAGE, INTRODUCTION, SECTION)>*

<!ELEMENT TITLEPAGE (#PCDATA)>

<!ELEMENT INTRODUCTION ANY>

<!ELEMENT SECTION (#PCDATA)>

<!ENTITY topics SYSTEM "Topics.xml">

]>

Содержимое файла Topics.xml:

<HEADING>Topics</HEADING>

Standard XML Applications

Этот типичный файл внешнего примитива содержит два пункта, которые можно включить в XML-элемент: вложенный элемент и блок символьных данных. Его содержимое может быть корректно вставлено в элемент *INTRODUCTION*, который может иметь любой тип содержимого:

<INTRODUCTION>

What this article:

&topics;

</INTRODUCTION>

4.3. Нотации и неанализируемые данные

Для объявления общего внешнего неразбираемого примитива используется механизм нотаций.

XML разработан главным образом, чтобы служить контейнером для текстовой информации. Но он предоставляет механизм взаимодействия с данными, не являющимися XML. Этот механизм описывается спецификацией нотаций.

Нотация является особой меткой, сообщающей процессору XML о том, какого типа данные он рассматривает в настоящий момент.

Нотация описывает определенный формат данных. Это делается путем указания адреса описания формата, адреса программы, которая может обрабатывать данные в этом формате, либо просто описание формата.

Можно использовать нотацию, чтобы описать формат общего внешнего неразбираемого примитива (см. в предыдущем разделе), либо присвоить нотацию атрибуту, который имеет нумерованный тип NOTATION.

Нотация имеет следующую форму записи:

<!NOTATION ИмяНотации SYSTEM СистемЛитерал>

Здесь ***ИмяНотации*** есть имя нотации. Вы можете выбрать любое имя при условии, что оно начинается с буквы или символа подчеркивания (_), после чего могут следовать или не следовать другие буквы, цифры, точки (.), тире (–) или символы подчеркивания.

Лучше выбирать информативное имя, позволяющее идентифицировать формат. Например, если вы определяете нотацию, описывающую точечный формат *(bitmap),* то следует использовать имя *BMP*.

СистемЛитерал есть системный литерал, который может быть ограничен одинарными (') или двойными (") кавычками и содержать любые символы, за исключением символа кавычек, используемого в качестве ограничителя. Вы можете включить в системный литерал любое описание формата, которое проинформирует приложение, как отображать или обрабатывать XML-документ:

- URI программы, которая может обрабатывать или отображать формат данных:

<!NOTATION BMP SYSTEM "Pbrush.exe">

<!NOTATION GIF SYSTEM "http://bogus.com/ShowGif.exe">

- URI документа в сети, который описывает формат данных, например:

<!NOTATION STRANGEFORMAT SYSTEM "http://bogus.com/ StrangeFormat.htm">

- простое описание формата, например:

<!NOTATION DOC SYSTEM "Microsoft Word document">

- номер международного стандарта форматов:

SYSTEM “ISO 8601:1988”.

Объявление общего внешнего неразбираемого примитива. Объявление для общего внешнего неразбираемого примитива имеет следующую форму записи:

<!ENTITY ИмяПримитива SYSTEM СистемЛитерал NDATA ИмяНотации>

Ключевое слово ***NDATA*** указывает, что файл примитива содержит неразбираемые данные (они не обрабатываются синтаксическим анализатором).

ИмяНотации, объявленной в DTD, описывает формат данных, содержащихся в файле примитива, или указывает на местонахождение

программы, которая может обрабатывать эти данные. Об объявлении нотации будет говориться дальше.

Файл неразбираемого внешнего примитива может содержать любой тип текста или нетекстовые данные, которые должны соответствовать описанию формата, определяемого соответствующей нотацией.

Практически единственный способ использования этого типа примитива состоит в присвоении его имени атрибуту с типом ENTITY или ENTITIES.

Маркерные типы атрибута ENTITY и ENTITIES.

1. ***ENTITY.*** Значение атрибута должно совпадать с именем примитива, объявленного в DTD. Этот примитив не обрабатывается синтаксическим анализатором и ссылается на внешний файл, обычно содержащий не XML-данные.

2. ***ENTITIES.*** Этот тип атрибута похож на тип ENTITY, за исключением того, что значение может содержать имена нескольких неанализируемых примитивов – разделенных пробелами – внутри строки в кавычках. Например, если вы назначили атрибуту *Source* тип *ENTITIES* следующим образом:

```
<!ELEMENT IMAGE EMPTY>
<!ATTLIST IMAGE Source ENTITIES #REQUIRED>
```

то сможете использовать его для ссылки на несколько не анализируемых примитивов (допустим, примитивов, содержащих графические данные в альтернативных форматах), например, так:

```
<IMAGE Source="LogoGif LogoBmp" />
```

Например:

```
<?xml version="1.0" encoding="windows-1251" ?>
<!DOCTYPE BOOK
[
   <!ELEMENT BOOK (TITLE, AUTHOR, COVERIMAGE)>
   <!ELEMENT TITLE (#PCDATA)>
   <!ELEMENT AUTHOR (#PCDATA)>
   <!ELEMENT COVERIMAGE EMPTY>
   <!ATTLIST COVERIMAGE  Source ENTITY #REQUIRED>
   <!NOTATION GIF SYSTEM "ShowGif.exe">
   <!ENTITY faun SYSTEM "Faun.gif" NDATA GIF>
]>
```

```
<BOOK>
   <TITLE>The Marble Faun</TITLE>
   <AUTHOR>Nathaniel Hawthorne</AUTHOR>
   <COVERIMAGE Source="faun" />
</BOOK>
```

DTD определяет файл *Faun.gif*, который содержит рисунок обложки книги, как общий внешний неразбираемый примитив с именем *faun*. Имя нотации этого примитива – *GIF*. Она указывает на местонахождение программы, которая отображает графические файлы в формате GIF *(ShowGif.exe)*.

DTD также определяет пустой элемент с именем COVERIMAGE и атрибут типа ENTITY для этого элемента с именем *Source*.

В элементе *Документ* атрибуту *Source* элемента *COVERIMAGE* присвоено имя внешнего примитива, который содержит графические данные для отображения рисунка обложки. Поскольку Source имеет тип ENTITY, можно присвоить ему имя общего внешнего неразбираемого примитива:

```
<?xml version="1.0"?>
<!DOCTYPE doc
[
   <!ELEMENT doc ANY>
   <!ELEMENT graphic EMPTY>
   <!ATTLIST graphic Source ENTITY #REQUIRED>
   <!NOTATION jpeg SYSTEM "image/jpeg">
   <!NOTATION png SYSTEM "image/png">
<!ENTITY bob "pictures/bob.jpeg">NDATA jpeg>
<!ENTITY judy "pictures/judy.jpeg " NDATA png>
]>
<doc>
   <graphic source="bob"/>
   <graphic source="judy"/>
</doc>
```

Процессор XML, обнаружив элемент *<graphic>*, находит имя сущности в атрибуте source. Поскольку сущность объявляется как неанализируемая, процессор не обрабатывает ее как данные XML, а передает прямо в ту часть программы, которая умеет ее обрабатывать.

4.4. Объявление параметрических примитивов

Форма объявления разметки параметрического примитива аналогична форме объявления, используемой для общих примитивов.

Параметрическая сущность (parameter entity) содержит текст из DTD и может использоваться во внутреннем и внешнем подмножествах. Она не может содержать текст XML, и ссылка на параметрическую сущность не может находиться внутри документа XML.

Чтобы отличать параметрические сущности от общих, при объявлении параметрических сущностей перед их именем ставится знак процента (%), а в ссылке на них – знак процента вместо амперсанта.

Объявление параметрического внутреннего разбираемого примитива имеет следующую форму записи:

<!ENTITY % ИмяПримитива ЗначениеПримитива>

Параметрический примитив может иметь такое же имя, что и общий примитив в документе. Параметрические и общие примитивы занимают различные пространства имен. Имя примитива также может совпадать с именем элемента или атрибута; имя должно начинаться с буквы или символа подчеркивания (_), вслед за которым могут следовать или не следовать буквы, цифры, точки (.), тире (–) или символы подчеркивания.

Строка ***ЗначениеПримитива*** должна содержать один или несколько типов объявлений разметки, которые разрешено использовать в DTD. В частности, параметрический примитив может содержать:

- объявления типа элемента;
- объявления списка атрибутов;
- объявления общих примитивов;
- объявления нотаций;
- инструкции по обработке;
- или комментарии (объявления параметрических примитивов и ссылки не допускаются):

<!ENTITY % content "para | note | warning ">

<!ENTITY % id.att "id ID #REQUIRED">

<!ELEMENT chapter (title, epigraph, (%cotent;)+)>

<!ATTLIST chapter % id.att;>

<!ELEMENT appendix (title, (%cotent;)+)>

<!ELEMENT appendix % id.att;>

Параметрические сущности упрощают разработку DTD. Параметрические сущности определяются для общих частей DTD.

Например, следующее DTD объявляет параметрический внутренний разбираемый примитив с именем *author*, который содержит три объявления разметки: комментарий, объявление типа элемента и объявление списка атрибутов. Содержимое примитива, т.е. замещающий его текст, помещается в конец DTD посредством ссылки на параметрический примитив (%author;):

```
<!DOCTYPE BOOK
[
   <!ENTITY % author
      "<!-- информация об авторе -->
      <!ELEMENT AUTHOR (#PCDATA)>
      <!ATTLIST AUTHOR  Nationality CDATA 'American'>"
   >
   <!ELEMENT BOOK (TITLE, AUTHOR)>
   <!ELEMENT TITLE (#PCDATA)>
   %author;
]>
```

Объявление параметрического внешнего разбираемого примитива. Внешние параметрические сущности напоминают внешние общие сущности – импортируют текст из другого файла. Сущности этого типа применяются для импортирования частей DTD, находящихся в других файлах, и называются ***разбиением на модули (modularizing)***:

<!ENTITY % ИмяПримитива SYSTEM СистемЛитерал>

СистемЛитерал описывает местонахождение файла, содержащего данные примитива. Системный литерал может быть заключен в одинарные (') или двойные (") кавычки:

<!ENTITY % declarations SYSTEM "Declarations.dtd">

Файл параметрического внешнего примитива должен содержать законченные объявления разметки любых типов, допустимых в DTD. В частности, он может содержать объявления типа элемента, объявления списка атрибутов, объявления примитивов, объявления нотаций, инструкции по обработке или комментарии.

Параметрический внешний разбираемый примитив работает во многом аналогично внешней схеме DTD. Но параметрические внешние примитивы обеспечивают большую гибкость:

- можно включать несколько файлов внешних объявлений, в любом порядке;
- подмножество внутреннего DTD обрабатывается раньше, чем внешнее подмножество DTD.

Например, вы занимаетесь продажей книг, CD-ROM, плакатов и другой продукции. Вы можете поместить объявления для каждого вида продукции в отдельный файл. Это позволит вам объединять эти группы объявлений различными способами. Например, вы хотели бы создать XML-документ, который описывает только имеющиеся у вас в наличии книги и CD-ROM. Для этого вы можете поместить объявления для книг и CD-ROM в DTD документа с помощью параметрических внешних разбираемых примитивов, как показано в следующем примере XML-документа:

```
<?xml version="1.0" encoding="windows-1251" ?>
<!DOCTYPE INVENTORY
[
   <!ELEMENT INVENTORY (BOOK | CD)*>
   <!ENTITY % book_decls SYSTEM "Book.dtd">
   <!ENTITY % cd_decls SYSTEM "CD.dtd">
   %book_decls;
   %cd_decls;
]>
<INVENTORY>
   <BOOK>
            <BOOKTITLE>The Marble Faun</BOOKTITLE>
            <AUTHOR>Nathaniel Hawthorne</AUTHOR>
            <PAGES>473</PAGES>
   </BOOK>
   <CD>
            <CDTITLE>Concerti Grossi Opus 3</CDTITLE>
            <COMPOSER>Handel</COMPOSER>
            <LENGTH>72 minutes</LENGTH>
   </CD>
```

```
    <BOOK>
        <BOOKTITLE>Leaves of Grass</BOOKTITLE>
        <AUTHOR>Walt Whitman</AUTHOR>
        <PAGES>462</PAGES>
    </BOOK>
    <!-- дополнительные виды продукции… -->
</INVENTORY>
```

Содержимое файла примитива Book.dtd:

```
<!ELEMENT BOOK (BOOKTITLE, AUTHOR, PAGES)>
<!ELEMENT BOOKTITLE (#PCDATA)>
<!ELEMENT AUTHOR (#PCDATA)>
<!ELEMENT PAGES (#PCDATA)>
```

Содержимое файла примитива CD.dtd:

```
<!ELEMENT CD (CDTITLE, COMPOSER, LENGTH)>
<!ELEMENT CDTITLE (#PCDATA)>
<!ELEMENT COMPOSER (#PCDATA)>
<!ELEMENT LENGTH (#PCDATA)>
```

4.5. Вставка ссылок на примитив

После объявления примитива возможна вставка содержимого примитива (замещающий текст) в документ с помощью ссылок на примитив:

- ссылка на общий примитив: *&ИмяПримитива*;
- ссылка на параметрический примитив: *%ИмяПримитива*;
- общий внешний неразбираемый примитив нельзя вставить с использованием ссылки. Для того чтобы его использовать, нужно присвоить его имени атрибут, имеющий тип ENTITY или ENTITIES;
- вставка ссылок на символы позволяет вставить любой символ, которого нет на клавиатуре, или вставить символ, который не допускается в данном контексте (например, символы < или & в составе символьных данных элемента). XML проектировался таким образом, чтобы использовать преимущества набора символов UNICODE (16-разрядный набор символов содержит 65536 символов), а не ASCII. Ссылка на символ имеет две различные формы:

1) &# код в десятичном формате;

2) &#x ASCII код в шестнадцатеричном формате:

<TITLE> < Title page > </TITLE> //добавляет <>

<!ENTITY heading1 “% Complete”> //добавляет %, который не может быть непосредственно введен как литерал в значение внутреннего примитива;

- XML содержит пять встроенных ссылок (ссылки на предопределенный примитив), аналогичных ссылке на соответствующий символ:

Ссылка	Описания
&amp;	Знак амперсанда (&)
&apos;	Апостроф (’)
&gt;	Больше (>)
&lt;	Меньше (<)
&quote;	Кавычка (")

<TITLE> <Title page ></TITLE>

<!ENTITY heading "Christopher "Kit" Carson">

Предопределенные примитивы похожи на другие общие внутренние разбираемые примитивы, за исключением того, что ссылки на них вы можете использовать без определения примитивов – и вы можете вставлять их в те же места, что и примитивы данного типа, а именно:

- в содержимое элемента;
- в значение атрибута (как значение по умолчанию в объявлении атрибута или в начальном теге элемента);
- в значение в объявлении внутреннего примитива.

Задания к лабораторной работе 3
"Создание валидного документа XML"

1. Создать внешнее определение типа документа (файл list.dtd), которое определяло бы XML-совместимый формат для хранения данных, аналогичных представленным:

Номер	1
Фамилия	Иванов
Имя	Сергей
Год	1993
Группа	ИФ 87
Телефон	1111111 (гор.) 067-1111111 (моб.)
Номер	2
Фамилия	Петрова
Имя	Г алина
Год	1992
Группа	ИФ 87
Номер	3
Фамилия	Семенов
Имя	Валерий
Группа	ИФ 88
Телефон	3333333 (гор.) 095-3333333 (моб.), 067-3333333 (моб.)
Номер	4
Фамилия	Павлова
Имя	Ирина
Год	1994
Группа	ИФ 89
Телефон	4444444 (гор.) 095-4444444 (моб.)

2. На основе созданного файла DTD создать файл StudList.xml, который должен содержать в себе выше указанную информацию. При этом обеспечить выполнение следующих условий:

а) корневым элементом списка студентов должен быть элемент FILE_1;

б) каждая запись должна быть представлена элементом Student;

в) у каждого элемента Student должен быть обязательный атрибут Num (с информацией о номере записи);

г) в каждый элемент Student должны быть вложены обязательные элементы Family (с информацией о фамилии студента), Name (с

информацией об имени студента), Group (с информацией о группе студента) и может быть необязательный элемент Year (с информацией о годе рождения студента). Каждый элемент Family, Name, Group, Year може быть вложен в элемент Student только один раз;

д) в каждый элеменг Student могут быть вложены неограниченное число раз необязательные элементы Phone;

е) для каждого элемента Phone должен быть предусмотрен обязательный атрибут PhoneType с двумя допустимыми значениями: L – для городских телефонов и M – для мобильных телефонов.

3. Проверить синтаксическую и логическую правильность полученных документов при помощи анализатора XML_Validator.hta.

4. Используя предыдущий XML-документ в качестве примера, преобразовать созданный в лабораторной работе №1, согласно варианту, корректно сформированный документ в валидный. Документ должен включать:

а) полную схему DTD, состоящую из комбинации внутреннего и внешнего подмножества DTD;

б) элементы с текстовым содержимым, элементы со смешанным содержимым и элементы с последовательным и выборочным дочерним содержимым;

в) необязательные и обязательные атрибуты строкового и нумерованного типа.

5. Варианты предметных областей создаваемых XML-документов:

Вариант 1:	библиографическое описание списка литературы
Вариант 2:	описание фильмов видеотеки
Вариант 3:	список сотрудников организации
Вариант 4:	список моделей мобильных телефонов
Вариант 5:	список студентов факультета
Вариант 6:	список изучаемых дисциплин

6. В отчет включить тексты созданных файлов и скриншоты результатов работы анализатора XML.

ТЕМА 5
XML SCHEMA

5.1 Использование схем в качестве альтернативы DTD.

5.2. Основы синтаксиса XML Schema.

5.1. Использование схем в качестве альтернативы DTD

Объявление типа документа с помощью DTD имеет ряд недостатков:

- синтаксис DTD отличается от синтаксиса XML, а это не очень удобно;
- синтаксис DTD тяжело читать, и язык DTD недостаточно гибок;
- DTD не поддерживает пространства имен, что снижает полезность его использования при увеличении степени распределености информации XM;
- DTD практически не поддерживает наследование;
- в DTD практически отсутствуют типы данных (кроме PCDATA, CDATA), что было бы полезно при описании содержимого элементов.

По этим причинам предложен ряд альтернатив DTD, так называемые ***схемы***. Схемы описывают структуру и содержание данных в документе XML:

- схемы подобно DTD позволяют проверить корректность XML;
- схемы используют синтаксис XML;
- схемы XML поддерживают типы данных, которые являются расширяемыми. Кроме предопределенных типов данных, можно создавать пользовательские типы, типы наследники и т.д.;
- использование схем переводит XML на двухдокументальную модель (есть схема документа и экземпляр документа, который должен быть грамматически правильным, чтобы его мог обработать анализатор XML).

Одной из таких альтернатив является XML Schema (XShema). ***XML Schema*** предоставляет больший контроль над типами данных и

шаблонами, образуя более удобный язык для соблюдения строгих требований ввода данных.

5.2. Основы синтаксиса XML Schema

Корневым элементом в схеме XML является элемент ***Schema***, который содержит все остальные элементы в документе схемы:

<?xml version =”1.0”?>

<xsd:schema xmlns:xsd=”http://www.w3.org/2001/XMLSchema”>

...

</xsd:schema>

Определение элемента. Элемент определяется *<xsd:element…/>*. Определение элемента заключается в определении его имени и модели контента, модель контента определяется ее типом. Тип элемента может быть простым, включенным в спецификацию XML-схем и сложным.

В табл. 5.1 представлены элементы, которые не содержат атрибутов или других элементов, относятся к простому типу, предопределенному или определенному пользователем.

Таблица 5.1 – Предопределенные простые типы для значений элементов XML

Простой тип	Описание	Пример
string	строка	«это тестовая строка»
booleann	логический	true, false, 1, 0
float	одинарная точность	32 разряда с плавающей точкой
double	двойная точность	64 разряда с плавающей точкой
decimal	десятичная	-43.21, 0, 123.4, 1500.00
timeInstant	1989-07-17T11: 30:00.000-05:00	17 июля 1989 года 11:30 по европейскому времени
binary		110010100110111
uri-reference		http://www.w3.org
integer		-123456, -1, 0, 1, 123456
non-positive-integer		-123456, -1, 0
negative interger		-123456, -1
long		-1, 12345654321

Продолжение табл. 5.1

Простой тип	Описание	Пример
short		-1, 12345
byte		-1, 126
non-negative-interger		0, 1, 123456
unsigned-long		0, 12345654321
unsigned-int		0, 12345678

Элементы, которые не содержат атрибутов или других элементов, могут быть отнесены к простому типу, предопределенному или определенному пользователем, они объявляются следующим образом:

<xsd:element name=’age’ type=’integer’/>

<xsd:element name=’price’ type= ‘decimal’/>

Атрибуту *name* присваивается имя типа элемента, а атрибуту *type* устанавливается значение типа элемента.

Элементы с атрибутами или элементы, содержащие вложенные элементы, имеют комплексный тип. Необходимо создать специальный элемент *complexType*:

```
<xsd:complexType name=”Address”>
   <xsd:element name=”number” type=”xsd:decimal”/>
   <xsd:element name=”street” type=”xsd:string”/>
   <xsd:element name=”city” type=”xsd:string”/>
   <xsd:element name=”province” type=”xsd:string”/>
</xsd:complexType>
```

Так выглядит в XML фрагмент экземпляра документа:

```
<address>
   <number>510</number>
   <street>Yellowbrick Road</street>
   <city>Munchkinville</city>
   <province>Negbo</province>
</address>
```

Атрибуты объявляются посредством использования элемента *attribute*. Например, USAddress определен как сложный тип, и внутри его определения мы видим пять объявлений элементов и одно объявление атрибута:

<xsd:complexType name="USAddress">

<xsd:sequence>

<xsd:element name="name" type="xsd:string"/>

<xsd:element name="street" type="xsd:string"/>

<xsd:element name="city" type="xsd:string"/>

<xsd:element name="state" type="xsd:string"/>

<xsd:element name="zip" type="xsd:decimal"/>

</xsd:sequence>

<xsd:attribute name="country" type="xsd:NMTOKEN" fixed="US"/>

</xsd:complexType>

Из этого определения можно сделать вывод о том, что любой элемент, представленный в примере, чей тип объявлен как USAddress должен состоять из пяти элементов и одного атрибута. Эти элементы должны называться *name, street, city, state* и *zip*, как задано значениями объявления атрибута name. Элементы обязаны появляться в той же последовательности, в которой они объявлены. Первые четыре элемента будут содержать строковое значение, а пятый – числовое. Элемент, тип которого объявлен как USAddress, может представляться с атрибутом country, который должен содержать строку US.

Ограничение модели содержимого. Модели содержимого элементов, которые могут включать дочерние узлы, могут быть одного из трех видов: <xs:sequence>, <xs:choice> или <xs:all>, называемых ***композиторами.***

Когда мы используем элемент-композитор *«sequence»,* мы сообщаем о том, что элементы, включенные в него, должны всегда располагаться в указанной в схеме последовательности, а также все из них являются обязательными. В XML-схемах есть еще два элемента-композитора: «choice» и «all». Композитор *«choice»* сообщает о том, что должен быть какой-то один из перечисленных в нем элементов, а композитор *«all»* – любая комбинация перечисленных элементов.

XML Schema позволяет ограничивать модели содержимого. Например, можно использовать атрибуты *minOccurs, maxOccurs*, устанавливающие минимальное и максимальное число появлений элемента в документ:

- minOccurs=”0” делает элемент необязательным;

- maxOccurs=”*” – элемент может появиться в документе множество раз.

Например:

```
<messageList>
  <message>
    <phone>71239876543</phone>
    <text>Тестовое сообщение 1</text>
    <date>2013-07-20T12:00:00</date>
    <type>12</type>
  </message>
  <!-- ... -->
  <message>
    <phone>71239876543</phone>
    <text>Тестовое сообщение N</text>
    <date>2013-07-20T12:00:00</date>
    <type>12</type>
  </message>
</messageList>
```

Схема для такого комплексного типа будет выглядеть так:

```
<complexType name="Message">
  <sequence>
    <element name="phone" type="string" minOccurs="1" maxOccurs="1" />
    <element name="text" type="string" minOccurs="1" maxOccurs="1" />
    <element name="date" type="dateTime" minOccurs="1" maxOccurs="1" />
    <element name="type" type="decimal" minOccurs="1" maxOccurs="1" />
  </sequence>
</complexType>
<element name="messageList" type="MessageList" />
<complexType name="MessageList">
  <sequence>
    <element minOccurs="1" maxOccurs="unbounded" name="message" type="Message"/>
```

</sequence>

</complexType>

В первом блоке идет знакомое нам декларирование комплексного типа *«Message»*. В каждом простом типе, входящем в «Message», были добавлены новые уточняющие атрибуты «minOccurs» и «maxOccurs». Первый (minOccurs) сообщает о том, что в данной последовательности должно быть минимум по одному элементу типа *«phone», «text», «date»* и *«type»*, в то время как следующий (maxOccurs) атрибут нам декларирует, что таких элементов в нашей последовательности максимум по одному. В результате, когда мы пишем свои схемы для каких-либо данных, нам предоставляется широчайший выбор по их настройке.

Второй блок схемы декларирует элемент *«messageList»* типа «MessageList». «MessageList» представляет собой комплексный тип, который включает минимум один элемент *«message»*, но максимальное число таких элементов не ограничено.

XML Schema позволяет устанавливать пределы и ограничения на величину данных. Такие ограничения устанавливаются с помощью ***граней или фасетов (facets)***, представляющих собой свойства. В XML Schema 13 таких граней: max-inclusive (максимальное значение), precision (точность), scale (масштаб), pattern (кодировка), enumeration (перечисление), max-length (максимальная длина) и другие.

XML Schema позволяет в качестве шаблонов на содержание элементов использовать регулярные выражения.

Например, сложный тип Address имеет атрибут *postalcode* с типом PCode, который определен посредством объявления <pattern>. Этот шаблон задан регулярным выражением [A-Z]-\d{3}, который нужно читать: «любой символ алфавита, за которым следует дефис и три десятичных цифры:

<xsd:complexType name=”Address”>

<xsd:element name=”number” type=”xsd:decimal”/>

<xsd:element name=”street” type=”xsd:string”/>

<xsd:element name=”city” type=”xsd:string”/>

<xsd:element name=”province” type=”xsd:string”/>

</xsd:complexType>

<xsd:simpleType name=”PCode” base=”xsd:string”>

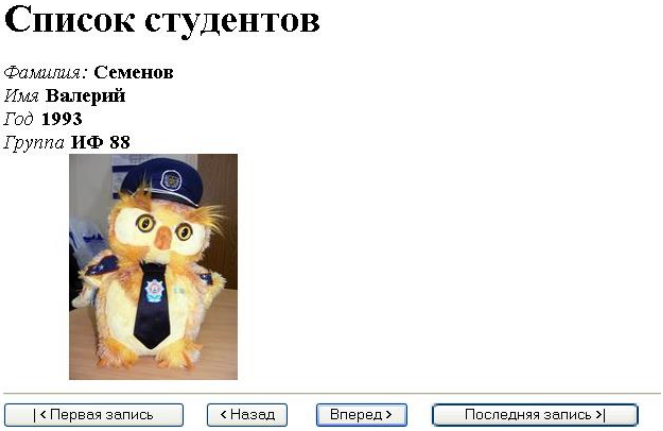
```
    <xsd:pattern value="[A-Z]-\d{3}"/>
</xsd:simpleType>
```

XML Schema предоставляет альтернативу DTD, позволяющую создателям документов проектировать поля, определяя их со значительно большей подробностью. Но DTD по-прежнему имеет ряд преимуществ: компактный размер, изученный уже синтаксис, простоту.

Задания к лабораторной работе 4

"Сцепление XML-документа с HTML по отдельным записям"

1. Создайте html документ, связанный с xml-документом по отдельной записи, с добавлением графических файлов и работающих кнопок перехода между записями, аналогично приведенному рисунку:



2. Используя вложенные таблицы, свяжите созданный в третьей лабораторной работе, согласно своему варианту, валидный XML-документ с HTML-документом.

ТЕМА 6

ОТОБРАЖЕНИЕ XML-ДОКУМЕНТОВ С ПОМОЩЬЮ СЦЕНАРИЕВ ОБЪЕКТНОЙ МОДЕЛИ ДОКУМЕНТА

6.1. Объектная модель документа XML-DOM

XML DOM (Document Object Model) – это объектная модель, которая охватывает содержание всего документа и позволяет писать сценарии, позволяющие манипулировать объектами документа с помощью событий, свойств и методов.

Консорциум W3C использует термин Document Object Model для обозначения объектной модели, которая обеспечивает доступ как к HTML-элементам, так и к XML-документу.

DOM хранит данные в иерархической, древообразной структуре, отражающей структуру XML-документа, которую обычно сравнивают с деревом, содержащим узлы. Эту модель можно использовать для доступа к любым компонентам XML-документа:

- элементам;
- атрибутам;
- инструкциям по обработке;
- комментариям;
- объявлениям нотаций и примитивов.

Каждый ***узел*** в структуре документа может содержать любое количество дочерних узлов, а все узлы, за исключением корневого, также обладают и родительским узлом. Доступ к каждому узлу осуществляется по имени. Каждый узел представляет собой объект, со своими свойствами и методами.

Узел представляет собой наименьшую единицу в иерархической структуре документа XML. Объект ***Node*** – это один узел в дереве документа, а объект ***Document*** – его корневой узел.

В табл. 6.1 приводится список основных узлов, доступных в модели DOM для языка XML.

Таблица 6.1 – Узлы, доступные в модели DOM

Тип узла	Компоненты XML-документа, представляемые узлом	Имя узла (свойство nodeName объекта)
Document	Корневой узел иерархии документа (т. е. он представляет весь XML-документ)	#document
Node	Ссылка на компонент документа, такой как элемент, атрибут, комментарий или текстовая строка	Имя компонента элемента
Element	Элемент документа	Имя типа элемента
Text	Текст, принадлежащий элементу, атрибуту или примитиву, которые представлены родителем этого узла	#text
Attribute	Атрибут элемента, представлен в виде пары имя – значение.	Имя атрибута
Processing-Instruction	Инструкция по обработке (объявление XML или пользовательская инструкция по обработке)	Предназначение инструкции по обработке (например, xml)
Comment	Комментарий	#comment
CDATASection	Раздел CDATA	#cdata-section
DocumentType	Объявление типа документа	Имя корневого элемента, содержащееся в объявлении DOCTYPE
Entity	Объявление примитива в DTD	Имя примитива
Notation	Объявление нотации в DTD	Имя нотации

DOM обеспечивает и использует организованную структуру объектов и сценариев для доступа к ее узлам, а также манипулирования ними.

Сценарий может указывать на узел по его относительному или абсолютному положению, например, первый узел в структуре документа.

Каждое из имен узлов можно получить из свойства узла *nodeName*. Имена узлов бывают двух типов:

1) начинающиеся с символа #, представляют компоненты XML, не поименованные в документе, например, комментарий в XML-документе не обладает именем, в связи с этим DOM использует стандартное имя #comment;

2) имена узлов, получающиеся из имен, присвоенных соответствующим компонентам в XML-документе (например, узел Element, представляющий элемент типа BOOK, также должен носить имя BOOK).

Значение узла можно получить с помощью свойства узла *nodeValue*. Если компонент XML имеет соответствующее значение (например, атрибут или Processing-Instruction будет хранить полное содержимое инструкции по обработке, за исключением предназначения – имени процессора, кому адресована), это значение будет храниться в значении узла. Если компонент XML не имеет значения (например, элемент Document), DOM устанавливает в качестве значения узла *null*.

DOM организует узлы XML-документа в виде древообразной иерархической структуры, которая отражает иерархическую структуру самого документа. При этом создается единственный узел Document, который представляет весь XML-документ и служит корневым элементом в этой иерархии.

Например, на рис. 6.1 показана иерархическая организация узлов для XML-документа:

```
<?xml version="1.0" encoding="windows-1251" ?>
<!-- Имя файла: Inventory Dom.xml -->
<INVENTORY>
  <BOOK Binding="бумажная обложка">
    <TITLE>Приключения Гекльберри Финна</TITLE>
    <AUTHOR Born="1835">Марк Твен</AUTHOR>
    <PAGES>298</PAGES>
    <PRICE>$5.49</PRICE>
  </BOOK>
    <BOOK Binding="твердый переплет">
    <TITLE>Моби Дик</TITLE>
    <AUTHOR Born="1819">Герман Мелвилл</AUTHOR>
```

```
    <PAGES>724</PAGES>
    <PRICE>$9.95</PRICE>
  </BOOK>
</INVENTORY>
```

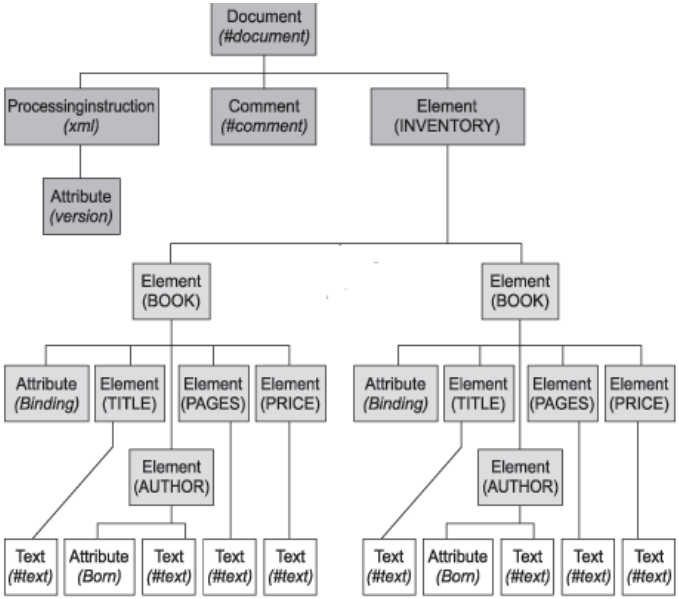


Рисунок 6.1 – Иерархическая организация узлов XML-документа

Все типы узлов используют общий набор свойств и методов, разработанных для работы с узлами вообще. Кроме общих свойств и методов каждому типу узла присущи дополнительные свойства и методы, разработанные для работы с определенным XML-компонентом, который представляет узел. Например, узел *Document* имеет свойство *parseError*, которое содержит информацию о любой ошибке, возникающей в процессе обработки документа. Данное свойство присуще только узлу Document.

6.2. Общие свойства узлов DOM

В таблице 6.2 представлен пример применения некоторых общих свойств узлов.

Таблица 6.2 – Общие свойства узлов DOM

childNodes	Множество NodeList всех дочерних узлов, которые не являются атрибутами, данного узла	FirstNode = Element.childNodes(0);
dataType	Тип данных этого узла (применительно только к определенным типам узлов Attribute)	AttributeType = Attribute.dataType;
.firstChild	Первый дочерний узел данного узла, не являющийся атрибутом	FirstChildNode = Element.firstChild;
lastChild	Последний дочерний узел данного узла, не являющийся атрибутом	LastChildNode = Element.lastChild;
nextSibling	Следующий узел на том же уровне данного узла	NextElement = Element.nextSibling;
nodeName	Имя данного узла	ElementName = Element.nodeName;
nodeType	Цифровой код, указывающий на тип данного узла	NodeTupeCode = Node.nodeType;
nodeTypeString	Строка, содержащая тип данного узла, строчными буквами (например, "element" или "attribute")	NodeTypeString = Node.nodeTypeString;
nodeValue	Значение данного узла (или null, если он не содержит значения)	AttributeValue = Attribute.nodeValue;
ownerDocument	Корневой узел Document документа, содержащего данный узел	Document = Node.ownerDocument;
parentNode	Узел, для которого данный узел является дочерним (не действует для узла Attribute)	ParentElement = Element.parentNode;
previousSibling	Предыдущий узел на том же уровне данного узла	PreviousElement = Element.previousSibling;
text	Все текстовое содержимое данного узла и всех подчиненных узлов Element	AllCharacterData = Element.text;

Несколько замечаний к свойствам узлов:

1. Свойство будет иметь значение null, если данное свойство неприменимо к определенному узлу. Например, если узел представляет XML-компонент, который не имеет атрибутов (например, узел Document или Comment), его свойство attributes будет иметь значение null. Если узел представляет XML-компонент, который не имеет типа дан-

ных (тип данных имеют только определенные атрибуты), его свойство dataType будет иметь значение null. Если узел не имеет дочернего узла, не являющегося атрибутом, его свойство firstChild будет иметь значение null. Если узел относится к типу, который не имеет значений (например, узел Document или Element), его свойство nodeValue также будет иметь значение null.

2. Каждый узел обладает набором свойств, которые позволяют вам перемещаться в иерархии узла, т. е. получать доступ к другим узлам от текущего узла. Например, в ранее рассмотренном листинге документа. Если переменная Document содержит корневой узел Document, используя строку *alert(Document.childNodes(1).node Value);* получим отображение содержимого комментария, расположенного в начале документа (комментарий в DOM представляет второй дочерний узел узла Document (начиная от нулевого)). Строка вызовет отображение сообщения, содержащего текст *"Имя файла: Inventory Dom.xml"*.

6.3. Использование узла Document

Узел ***Document*** является шлюзом к XML-документу. Через него осуществляется один из способов доступа к другим узлам. Для того чтобы получить к нему доступ, необходимо связать XML-документ с HTML-страницей. Самый простой способ сделать это через фрагмент данных, который создается через HTML-элемент с именем XML.

Например, следующий элемент BODY HTML-страницы содержит фрагмент данных, который связывает XML-документ, хранящийся в файле Book.xml:

```
<BODY>
  <XML ID="dsoBook" SRC="Book.xml"></XML>
  <!-- другие элементы отображаемой части страницы ... -->
</BODY>
```

Идентификатор ID (назначенный фрагменту данных, указывает на DSO (Data Source Objects) документа) фрагмента данных на HTML-странице предписывает Internet Explorer создать как DSO (представленный непосредственно через ID фрагмента данных), так и DOM (доступ к которой осуществляется через член XML Document DSO):

```
Document = dsoBook.XMLDocument;
```

Сценарий, который вставляется в заголовок HTML документа, первой инструкцией получает узел Document, который представляет весь документ и формирует корневой элемент иерархии узлов DOM. Он делает это через член XMLDocument DSO:

Document = dsolab5.XMLDocument;

Член *XMLDocument* содержит корневой объект DOM, известный как узел Document (Document node). После включения данного фрагмента кода, можно использовать свойства и методы узла Document для доступа к другим объектам DOM.

В табл. 6.3 представлены некоторые свойства и методы, предоставляемые узлами Document.

Таблица 6.3 – Свойства и методы, предоставляемые узлами Document

Свойство узла Document	Описание	Пример
documentElement	Узел Element, представляющий корневой элемент (в нашем примере BOOK)	RootElement=Document.do cu-mentElement;
ondataavailable	Если вы присвоите этому свойству имя функции, которую вы написали, функция будет вызываться в момент доступности данных XML	Document.ondataavailable= =MyDataAvailableHandler; (Функция MyDataAvailable-Handler будет вызываться, когда станут доступными данные XML)
onreadystate-change	Если вы присвоите этому свойству имя функции, которую вы написали, функция будет вызываться всякий раз, когда изменяется свойство readyState узла Document (об этом свойстве см. далее в этой таблице)	Document.onreadystate-change=MyReadyState-Handler; (Функция MyReadyState-Handler будет вызываться всякий раз при изменении свойства readyState узла Document)

Продолжение табл. 6.3

Свойство узла Document	Описание	Пример
parseError	Объект, который содержит информацию о любых ошибках, возникающих в процессе обработки документа	ErrorCode = Document. parseError.errorCode;
readyState	Текущий статус загрузки и обработки XML-документа. Может принимать одно из следующих числовых значений: 0: неинициализирован; 1: загружается; 2: загружен; 3: интерактивный режим; 4: завершение	if (Document.readyState=4) /* обработка данных… */
url	URL XML-документа	URL = Document.url;
Метод узла Document	Описание	Пример
nodeFromID (id-value)	Возвращает узел, представляющий элемент, чей атрибут типа ID имеет указанное значение	Element = Document.node-FromID("S021");

Например:

Document = dsoBook.XMLDocument;

Document.documentElement.childNodes(2).text

Document содержит узел Document в основании (корне) иерархии узлов DOM.

Свойство *documentElement* представляет собой свойство узла Document (специфическое для данного узла свойство), оно представляет корневой элемент XML-документа – в нашем примере: BOOK.

Свойство *childNodes* является свойством узла Element для корневого элемента. Оно содержит множество всех дочерних узлов корневого узла Element, не являющихся атрибутами. В нашем примере оно содержит узлы Element для пяти дочерних XML-элементов: TITLE, AUTHOR, BINDING, PAGES и PRICE. Выражение *childNodes(2)* ссылается на третий из этих дочерних узлов, а именно – на элемент AUTHOR.

Выражение *Document.childNodes(2)* используется для получения доступа к узлу корневого элемента. Преимущество использования свойства documentElement узла Document заключается в том, что его значение не зависит от положения корневого элемента внутри XML-документа. Например, если бы Вы удалили комментарий в начале документа либо если бы добавили объявление типа документа, выражение Document.childNodes(2) больше не представляло бы корневой элемент.

Свойство *text* является свойством узла, возвращаемого выражением childNodes(2). Оно показывает весь текст, содержащийся в этом узле, а также текст, принадлежащий любому подчиненному узлу Element. В нашем примере AUTHOR не имеет подчиненных элементов, поэтому свойство text содержит только собственно текст элемента AUTHOR, "Марк Твен".

6.4. Использование объекта NodeList

Свойство childNodes узла содержит набор дочерних узлов текущего узла, не являющихся атрибутами. Доступ к дочерним узлам-атрибутам осуществляется через свойство attributes узла. Определенный тип набора, который содержит свойство childNodes, носит название объекта ***NodeList***.

Чтобы извлечь определенный дочерний узел из объекта NodeList, можно обратиться к его методу ***item***, указав при этом индекс дочернего узла, который нужно получить (индексы отсчитываются с нуля). Например, обращение к следующему методу позволяет получить первый дочерний узел, принадлежащий узлу Element:

FirstNode = Element.childNodes.item(0);

Однако, поскольку item является методом по умолчанию объекта NodeList, его можно опустить:

FirstNode = Element.childNodes(0);

В табл. 6.4 представлены свойство и методы, поддерживаемые групповым объектом NodeList.

Таблица 6.4 – Свойство и методы, поддерживаемые групповым объектом NodeList

Свойство NodeList	Описание	Пример
Length	Количество узлов, содержащихся в наборе	NodeCount=Element.childNodes.length;
Методы NodeList	Описание	Пример
item (индекс, отсчитываемый с 0) (метод по умолчанию)	Возвращает узел в соответствии с заданным вами индексом, при этом 0 соответствует первому узлу	SecondChild=Element.childNodes.item(1); или SecondChild=Element.childNodes(1);
reset()	Устанавливает внутренний указатель на позицию перед первым узлом в наборе, чтобы последующий вызов nextNode возвращал первый узел	Element.childNodes.reset();
nextNode()	Возвращает следующий узел в наборе в соответствии с позицией внутреннего указателя	Element.childNodes.reset(); FirstNode = Element.childNodes.nextNode();

6.5. Извлечение символьных данных элемента

Свойство *text* элемента используется для получения символьных данных текущего элемента и текстовых значений всех подчиненных элементов. Его хорошо использовать для извлечения символьных данных элемента в том случае, если элемент не имеет дочерних элементов.

Например:

```
<?xml version="1.0" encoding="windows-1251" ?>
<!-- Имя файла: Book.xml -->
<TITLE>Моби Дик
  <SUBTITLE> или Белый кит </SUBTITLE>
</TITLE>
```

Для извлечения символьных данных элемента *SUBTITLE* можно использовать следующий оператор:

Document.documentElement.childNodes(0).text;

Для элемента <TITLE> свойство text вернет весь текст (Моби Дик, или Белый кит).

Для получения только символьных данных элемента TITLE необходимо осуществить доступ к дочернему узлу Text. Для этого используется свойство *nodeValue*. Как рассматривалось в таблице свойств, для узла Element свойство nodeValue имеет значение null. Если элемент содержит символьные данные (например, кроме дочерних элементов), то они хранятся в его дочернем узле Text и их можно получить через свойство nodeValue узла Text.

Например, для извлечения символьных данных элемента TITLE можно использовать оператор *Element.firstChild.nodeValue.*

Поскольку символьные данные элемента TITLE (его узел Text) предшествуют подчиненному элементу, они представляются первым дочерним узлом, и их можно извлечь с помощью свойства *firstChild.*

Если символьные данные элемента смешаны с дочерними элементами, комментариями или инструкциями по обработке, каждый отдельный блок символьных данных представляется собственным дочерним узлом Text.

Например:

<ITEM>

Раздел № 1 / узел Text */*

<SUB-ITEM>подраздел</SUB-ITEM>

текст / узел Text */*

</ITEM>

Элемент ITEM имеет три дочерних узла, расположенных в следующем порядке: узел Text, представляющий первый блок символьных данных; узел ELEMENT, представляющий дочерний элемент SUB-ITEM; еще один узел Text, представляющий второй блок символьных данных.

В табл. 6.5 представлены полезные свойство и метод, поддерживаемые узлами Text.

Таблица 6.5 – Свойство и метод, поддерживаемые узлами Text

Свойство узла Text	Описание	Пример
Length	Количество символов в тексте узла	CharacterCount = Text.length;
Метод узла Text	Описание	Пример
substringData (char-offset, num-chars)	Возвращает строку, содержащую заданное число символов из текстового содержимого узла, начиная с указанной параметром смещения позиции	Substring = Text.substringData (2, 3); // возвращает третий, четвертый и пятый символы из содержимого элемента Text

6.6. Использование других способов доступа к элементам

Мы рассмотрели доступы к узлам Element с использованием иерархической структуры и свойств childNodes или firstChild, lastChild, previousSibling, nextSibling и parentNode, при которых осуществляется переход от одного узла к другому.

Другим способом доступа к XML-элементам является использование свойства *getElementsByTagName*, которое позволяет извлечь все элементы, имеющие определенное имя типа (например, TITLE). Этот метод может использоваться для узла Document и для узла Element.

Если обращаться к методу для узла Document, он возвращает набор узлов Element для всех элементов в документе, обладающих заданным именем типа.

Например:

NodeList = Document.getElementsByTagName("BOOK");

позволяет получить группу узлов для всех элементов в документе, обладающих именем BOOK.

Если обращаться к методу getElementsByTagName для узла Element, он возвращает набор узлов для всех соответствующих элементов, которые являются подчиненными для узла Element.

Например:

NodeList = Element.getElementsByTagName("AUTHOR");

Если в качестве параметра метода getElementsByTagName указать "*", то метод вернет:

1) набор всех элементов в документе, если вызвать метод для объекта Document;

2) набор всех подчиненных элементов, если вызвать метод для объекта Element.

В табл. 6.6 представлены полезные методы, поддерживаемые узлами Element.

Таблица 6.6 – Методы, поддерживаемые узлами Element

Метод узла Element	Описание	Пример
getAttribute (имя-атр)	Возвращает значение атрибута элемента с заданным именем	AttValue = Element. getAttribute ("InStock");
getAttributeNode (имя-атр)	Возвращает узел Attribute, представляющий атрибут элемента с заданным именем	Attribute = Element.get Attribute Node ("InStock");
getElementsBy TagName (имя-типа)	Возвращает набор Node-List узлов Element для всех подчиненных элементов элемента с заданным именем. Если указано "*", возвращает узлы для всех подчиненных элементов	AuthorElementCollection = =Element.getElementsBy TagName ("AUTHOR");

Метод getElementsByTagName предоставляет узлы Element в виде группового объекта NodeList.

Например:

```
text="<timetable>";
text=text+"<lesson>";
text=text+"<timeFrom>08.00</timeFrom>";
text=text+"<subject>Deutsch</subject>";
text=text+"<teacher>Borisova</teacher>";
text=text+"</lesson>";
text=text+"/<timetable>";
txt = Document.getElementsByTagName ("subject") [0]. childNodes[0]. nodeValue;
for (i=0; i < NodeList.length; ++i)
  alert(NodeList(i).text);
```

HTML-страница из Листинга 1 демонстрирует использование метода getElementsByTagName для узла Document.

Листинг 1 GetElements.htm:

```
<!-- Имя файла: GetElements.htm -->
<HTML>
<HEAD>
  <TITLE>Element Finder</TITLE>
  <SCRIPT LANGUAGE="JavaScript">
    function ShowElements()
    {
      /* проверка ввода пользователем названия элемента в поле
        'Element name': */
      if (ElementName.value == "")
      {
        ResultDiv.innerText = "<You must enter an element "
          + "name into 'Element name' box.>";
        return;
       }
      /* создание списка элементов NodeList с именами, анало-
гичными введенному: */
      Document = dsoXML.XMLDocument;
      NodeList =
        Document.getElementsByTagName(ElementName.value);
      /* запись XML указателя для каждого найденного элемента
в ResultHTML: */
      ResultHTML = "";
      for (i=0; i < NodeList.length; ++i)
        ResultHTML += NodeList(i).xml + "\n\n";
      /* назначение сохраненных результатов в свойстве
innerText элемента DIV: */
      if (ResultHTML == "")
        ResultDiv.innerText = "<no matching elements found>";
      else
        ResultDiv.innerText = ResultHTML;
    }
  </SCRIPT>
```

```
</HEAD>
<BODY>
  <XML ID="dsoXML" SRC="Inventory.xml"></XML>
  <H2>Find Elements by Element Name</H2>
  Element name: <INPUT TYPE="Text" ID="ElementName">
 
  <BUTTON ONCLICK="ShowElements()"> Show Elements
</BUTTON>
  <HR>
  <DIV ID=ResultDiv></DIV>
</BODY>
</HTML>
```

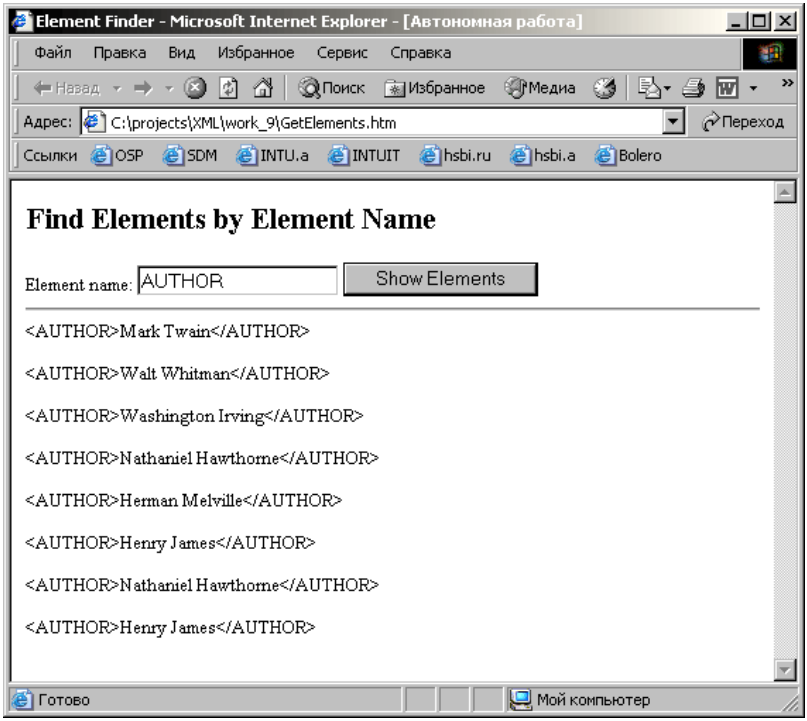


Рисунок 6.2 – Использование метода getElementsByTagName для узла Document

Страница отображает поле ввода INPUT типа Text, которое дает вам возможность ввести имя элемента. Когда вы щелкаете мышью на

кнопке Show Elements, вызывается функция ShowElements сценария, которая использует метод getElementsByTagName для узла Document, чтобы найти и отобразить XML-разметку для всех элементов в документе, носящих введенное вами имя элемента (если они имеются). Заметим, что сценарий использует свойство xml каждого из узлов Element для отображения содержимого XML-разметки элемента. Страница изначально связана с документом Inventory.xml, хотя вы можете отредактировать фрагмент данных, чтобы отобразить элементы из другого XML-документа. На рис. 6.2 показано как Internet Explorer 5 отобразит страницу после того, как вы ввели в поле ввода INPUT имя AUTHOR и щелкнули мышью на кнопке Show Elements.

6.7. Доступ и отображение значений атрибутов в XML-документе

Атрибут, который содержится в XML-элементе, представляется дочерним узлом Attribute. Однако, как известно, нельзя обратиться к дочернему узлу Attribute с использованием свойств childNodes, firstChild или lastChild, которые используются для доступа к дочерним узлам других типов. Вместо этого вам потребуется воспользоваться свойством *attributes* узла Element.

Изменим немного наш пример:

```
<?xml version="1.0" encoding="windows-1251" ?>
<!-- Имя файла: Inventory Dom.xml -->
<INVENTORY>
  <BOOK Binding="бумажная обложка" InStock="yes" Review="***">
    <TITLE>Приключения Гекльберри Финна</TITLE>
    <AUTHOR Born="1835">Марк Твен</AUTHOR>
    <PAGES>298</PAGES>
    <PRICE>$5.49</PRICE>
  </BOOK>
    <BOOK Binding="твердый переплет" InStock="no" >
    <TITLE>Моби Дик</TITLE>
    <AUTHOR Born="1819">Герман Мелвилл</AUTHOR>
    <PAGES>724</PAGES>
    <PRICE>$9.95</PRICE>
  </BOOK>
```

</INVENTORY>

Элементы BOOK в этом документе имеют от двух до трех атрибутов. Элемент сценария *Document.documentElement.childNodes(0)* получает узел для первого элемента BOOK (предполагается, что Document содержит узел Document).

Групповой объект NamedNodeMap представляет набор узлов Attribute данного элемента Element. Свойство attributes данного узла Element содержит набор NamedNodeMap узлов Attribute всех атрибутов, принадлежащих данному элементу.

В нашем примере *NamedNodeMap=Document.documentElement. childNodes(0).attributes* набор NamedNodeMap – это набор узлов Attribute всех атрибутов, принадлежащих первому элементу BOOK.

Набор объектов атрибутов, объединненый свойством attributes, похож на групповой объект NodeList, предоставляемого свойством узла childNodes.

В табл. 6.7 показаны свойство и некоторые полезные методы, предоставляемы групповым объектом NamedNodeMap.

Таблица 6.7 – Свойство и методы, предоставляемые NamedNodeMap

Свойство NamedNode Map	Описание	Пример
length	Количество узлов, содержащихся в наборе	AttributeCount = =Element.attributes.length;
Метод NamedNode Map	Описание	Пример
getNamedItem (имя-атр)	Возвращает узел, который носит заданное имя	Attribute = Element.attributes. getNamedItem ("Binding");
item(индекс, отсчитываемый от нуля) (метод по умолчанию)	Возвращает узел в заданной индексом позиции (0 соответствует первому узлу)	SecondAttribute = Element. attributes.item(1); или SecondAttribute = Element.attributes(1);

Продолжение табл. 6.7

Метод NamedNode Map	Описание	Пример
reset()	Устанавливает внутренний указатель на позицию перед первым узлом в наборе, так что последующий вызов nextNode возвращает первый узел	Element.attributes.reset();
nextNode()	Возвращает следующий узел в наборе в соответствии со значением внутреннего указателя	Element.attributes.reset(); FirstAttribute = Element.attributes.nextNode();

Например:

NamedNodeMap = Document.documentElement.childNodes (0). attributes;

for (i=0; i<NamedNodeMap.length; ++i)

alert("node name: " + NamedNodeMap(i).nodeName + "\n"

+ "node value:" + NamedNodeMap(i).nodeValue);

Свойство *length* объекта NamedNodeMap и установленный по умолчанию метод *item* используются для перемещения внутри набора и извлечения отдельных узлов *Attribute*. Фрагмент сценария отображает имя и значение каждого атрибута для первого элемента BOOK рассматриваемого документа. Каждая пара имя – значение отображается в окне сообщения – предупреждения. Свойство nodeName узла Attribute содержит имя атрибута, в то время как свойство nodeValue содержит значение атрибута. На самом деле узел Attribute имеет дочерний узел Text, который содержит значение атрибута. Однако этот узел практически не нужен, поскольку можно легко получить значение атрибута из свойства nodeValue узла Attribute.

Например:

NamedNodeMap = Document.documentElement.childNodes (0). attributes;

alert(NamedNodeMap.getNamedItem("Binding").nodeValue);

Фрагмент кода сценария отображает значение атрибута *Binding* первого элемента BOOK в рассматриваемом документе.

Примечание. DOM использует узлы Attribute для представления не только атрибутов, но и нескольких типов других компонентов XML, которые состоят из пар имя-значение, а именно:

- имя и значение в инструкции по обработке (например, version="1.0" в XML-объявлении);
- ключевое слово SYSTEM, за которым следует системный литерал в объявлении типа документа, объявлении внешнего примитива либо в объявлении нотации;
- ключевое слово NDATA, за которым следует имя нотации в объявлении неразбираемого примитива.

Возьмем в качестве примера XML-документ из Листинга 1:

for (i=0; i < NodeList.length; ++i)

alert(NodeList(i).text); // отображает текстовое содержимое всех узлов Element в объекте NodeList, возвращенное методом getElementsByTagName.

HTML-страница из Листинга 1 демонстрирует использование метода getElementsByTagName для узла Document. Страница отображает поле ввода INPUT типа Text, которое дает вам возможность ввести имя элемента. Когда вы щелкаете мышью на кнопке Show Elements, вызывается функция ShowElements сценария, которая использует метод getElementsByTagName для узла Document, чтобы найти и отобразить XML-разметку для всех элементов в документе, носящих введенное вами имя элемента (если они имеются).

Заметим, что сценарий использует свойство xml каждого из узлов Element для отображения содержимого XML-разметки элемента. Страница изначально связана с документом Inventory.xml, хотя вы можете отредактировать фрагмент данных, чтобы отобразить элементы из другого XML-документа.

Свойство *innerText* определено в Dynamic HTML (DHTML), который поддерживается Internet Explorer. В DHTML каждый HTML-элемент имеет набор свойств, которые можно использовать для установки или получения различных характеристик элемента через код сценария. Свойство innerText устанавливает или получает текстовое содержимое элемента.

Символьные данные элемента FAMILY, которые получены из выражения справа от знака равенства ("Иванов"), присваиваются свойству innerText HTML-элемента SPAN, имеющему идентификатор family:

XML-файл:

```
<?xml version="1.0" encoding="windows-1251"?>
<!--Имя файла:InventoryDM.xml-->
<INVENTORY>
  <BOOK Binding="твердая обложка">
    <TITLE>Моби Дик</TITLE>
    <AUTHOR Born="1819">Герман Мелвилл </AUTHOR>
    <PAGES>724</PAGES>
    <PRICE>76,8</PRICE>
  </BOOK>
  <BOOK Binding="мягкий переплет">
    <TITLE>Приключения Гекльберри Финна</TITLE>
    <AUTHOR Born="1835">Марк Твен </AUTHOR>
    <PAGES>298</PAGES>
    <PRICE>65,8</PRICE>
  </BOOK>
</INVENTORY>
```

HTML-файл:

```
<HTML>
<HEAD>
 <TITLE>Список литературы</TITLE>
 <SCRIPT LANGUAGE="JavaScript" FOR="window" EVENT="ONLOAD">
  Document = dsoXML.XMLDocument;
 title. innerText = Document. documentElement. lastChild. childNodes(0). text;
```

```
NodeList= Document.getElementsByTagName("AUTHOR");
ResultHTML="";
for(i=0;i<NodeList.length;++i)
ResultHTML+=NodeList(i).xml+"\n\n";
/* запись XML указателя для каждого найденного элемента в ResultHTML: */
Result.innerText=ResultHTML;
/* назначение сохраненных результатов в свойстве innerText элемента DIV: */
  </SCRIPT>
</HEAD>
<BODY>
 <XML ID="dsoXML" SRC="Inventory.xml"></XML>
 <H2>Список литературы:</H2>
<SPAN> Заголовок:</SPAN>
<SPAN ID="title"> </SPAN>
<div ID=Result> </div>
</BODY>
</HTML>
```

Задания к лабораторной работе 5
"Отображение XML-документов с использованием XSL-таблиц стилей"

1. Используя расширяемые таблицы стилей, отобразите XML-документ в браузере Microsoft Internet Explorer:

а) создайте простой XML-документ, описывающий список книг:

Файл dema2.xml:

```
<?xml version="1.0"?>
<!-- File name: Dema2.xml -->
<?xml-stylesheet type="text/xsl" href="dema2.xsl"?>
<INVENTORY>
<BOOK>
  <TITLE>Моби Дик</TITLE>
  <AUTHOR>
    <FIRSTNAME>Герман</FIRSTNAME>
    <LASTNAME>Мелвилл</LASTNAME>
```

```
  </AUTHOR>
  <BINDING>твердая обложка</BINDING>
  <PAGES>724</PAGES>
  <PRICE>$9.95</PRICE>
</BOOK>
<BOOK>
  <TITLE>Приключения Гекльберри Финна</TITLE>
  <AUTHOR>
    <FIRSTNAME>Марк</FIRSTNAME>
    <LASTNAME>Твен</LASTNAME>
  </AUTHOR>
  <BINDING>твердая обложка</BINDING>
  <PAGES>564</PAGES>
  <PRICE>$15.25</PRICE>
</BOOK>
<BOOK>
  <TITLE>Будденброки</TITLE>
  <AUTHOR>
    <FIRSTNAME>Томас</FIRSTNAME>
    <LASTNAME>Манн</LASTNAME>
  </AUTHOR>
  <BINDING>твердая обложка</BINDING>
  <PAGES>1089</PAGES>
  <PRICE>$85.25</PRICE>
</BOOK>
<BOOK>
  <TITLE>Молодые годы короля Генриха IV</TITLE>
  <AUTHOR>
    <FIRSTNAME>Генрих</FIRSTNAME>
    <LASTNAME>Манн</LASTNAME>
  </AUTHOR>
  <BINDING>твердая обложка</BINDING>
  <PAGES>1089</PAGES>
  <PRICE>$85.25</PRICE>
</BOOK>
</INVENTORY>
```

б) создайте файл расширяемого языка таблицы стилей:

Файл dema2.xsl:

```
<?xml version="1.0" encoding="windows-1251"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/
1999/XSL/Transform">
<xsl:template match="/">
<table border="1">
<tr bgcolor="#CCCCCC">
<td align="center"> <strong> Название </strong></td>
<td align="center"> <strong> Имя автора </strong></td>
<td align="center"> <strong> Фамилия автора </strong></td>
<td align="center"> <strong> Обложка </strong></td>
<td align="center"> <strong> Цена </strong></td>
<td align="center"> <strong> Количество страниц </strong>
</td>
</tr>
<xsl:for-each select="INVENTORY/BOOK" >
<xsl:sort order="descending" select="PAGES" data-type="number"
/>
<tr bgcolor="#F5F5F5">
<td> <xsl:value-of select= "TITLE"/>
</td>
<td align="right"> <xsl:value-of select=
"AUTHOR/FIRSTNAME"/>
</td>
<td> <xsl:value-of select= "AUTHOR/LASTNAME"/>
</td>
<td align="center"> <xsl:value-of select= "BINDING"/>
</td>
<td align="center"> <xsl:value-of select= "PRICE"/>
</td>
<td align="center"> <xsl:value-of select= "PAGES"/>
</td>
</tr>
</xsl:for-each>
</table>
```

</xsl:template>

</xsl:stylesheet>

в) документ в браузере Internet Explorer должен иметь следующий вид:

Название	Имя автора	Фамилия автора	Обложка	Цена	Количество страниц
Будденброки	Томас	Манн	твердая обложка	$85.25	1089
Молодые годы короля Генриха IV	Генрих	Манн	твердая обложка	$85.25	1089
Моби Дик	Герман	Мелвилл	твердая обложка	$9.95	724
Приключения Гекльберри Финна	Марк	Твен	твердая обложка	$15.25	564

2. Используя расширяемую таблицу стилей, создать и отобразить документ с переменным числом XML-элементов для предметных областей, заданных вариантов в виде таблицы.

3. Варианты предметных областей, создаваемых XML-документов:

Вариант 1:	библиографическое описание списка литературы
Вариант 2:	описание фильмов видеотеки
Вариант 3:	список сотрудников организации
Вариант 4:	список моделей мобильных телефонов
Вариант 5:	список студентов факультета
Вариант 6:	список изучаемых дисциплин

4. Отсортируйте созданные списки по первому полю.

5. Добавьте фильтр содержимого некоторого поля созданного XML-документа по условию.

ТЕМА 7

ОТОБРАЖЕНИЕ XML-ДОКУМЕНТОВ С ИСПОЛЬЗОВАНИЕМ XSL-ТАБЛИЦ СТИЛЕЙ

7.1. Основы использования XSL-таблиц стилей

Еще одним методом отображения XML-документов в браузере Microsoft Internet Explorer 5 является использование расширяемого языка таблицы стилей ***XSL (Extensible Stylesheet Language)***.

Подобно каскадной таблице стилей (CSS), XSL-таблица стилей связывается с XML-документом и сообщает браузеру, как отображать данные XML (без посредничества HTML-страницы). XSL-таблица стилей – более мощный и гибкий инструмент для отображения XML-документов, чем CSS-таблица:

1. С помощью XSL-таблицы стилей можно не только задать формат для каждого элемента XML, как с помощью CSS-таблицы, но и обеспечить средства контроля над выводимыми данными.

2. XSL-таблицы позволяют сортировать и фильтровать данные XML, добавлять информацию, добавлять в документ сценарии.

3. В отличие от таблиц каскадных стилей, если связать с XML-документом более одной XSL-таблицы стилей, браузер использует первую таблицу и игнорирует все остальные. Если связать с XML-документом и CSS-таблицу, и XSL-таблицу стилей, браузер использует только XSL-таблицу стилей.

4. Если не связать XML-документ ни с CSS-таблицей, ни с XSL-таблицей стилей, Internet Explorer отобразит документ с помощью встроенной XSL-таблицы, которая используется по умолчанию. Эта таблица стилей отображает исходный XML-текст в виде дерева с возможностью свертывания/развертывания уровней.

5. XSL-таблицы являются более сложными для понимания, чем CSS-таблицы. Это новая технология имеет меньшую степень поддержки – меньшую степень поддержки среди браузеров.

Для того чтобы отобразить XML-документ с использованием XSL-таблицы стилей, необходимо:

1. Создать файл XSL-таблицы стилей.

XSL является приложением XML, т.е. XSL-таблица представляет собой корректно сформированный XML-документ, который отвечает правилам XSL. Подобно любому XML-документу ее можно создать с помощью текстового редактора. В последующих разделах рассказывается, как создавать различные типы XSL-таблиц стилей.

2. Связывание XSL-таблицы стилей с XML-документом.

Для того чтобы связать XSL-таблицу стилей с XML-документом, необходимо включить в документ инструкцию по обработке xml-stylesheet, которая обычно добавляется в пролог XML-документа, вслед за объявлением XML:

<?xml-stylesheet type="text/xsl" href=" заключенный в кавычки URL файла таблицы стилей"?>

Хотя можно связать XSL-таблицу стилей с использованием полного URL, таблица стилей при этом должна размещаться на том же домене, что и XML-документ, с которым она связывается.

XSL-таблица стилей включает один или несколько шаблонов, каждый из которых содержит информацию для отображения в определенной ветви элементов в XML-документе.

Каждая XSL-таблица стилей должна иметь корневой элемент – элемент верхнего уровня, который содержит все остальные элементы:

<xsl:stylesheet
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<!-- один или несколько элементов шаблона …-->
</xsl:stylesheet>

Элемент *Документ* xsl:stylesheet служит:

- хранилищем других элементов;
- идентифицирует документ как XSL-таблицу стилей;
- все XSL-элементы принадлежат пространству имен xsl. Необходимо предварять имя каждого XSL-элемента префиксом xsl:, пространство имен объявляется в начальном теге элемента xsl:stylesheet.

Корневой элемент *xsl:stylesheet* XSL-таблицы стилей должен содержать один или несколько шаблонов элементов или шаблонов.

Браузер использует шаблон для отображения определенной ветви элементов в иерархии XML-документа, с которым связывается таблица стилей.

7.2. Использование одного шаблона XSL

Шаблоны имеют следующую форму:

```
<xsl:template match="/">
  <!-- дочерние элементы... -->
</xsl:template>
```

Браузер использует каждый шаблон для отображения определенной ветви элементов в иерархии XML-документа.

Атрибут ***match*** шаблона элемента указывает на определенную ветвь. Значение атрибута match носит название образца (pattern). Если значение атрибута match равно слешу ("/"), то он представляет корневой элемент всего XML-документа. Тогда шаблон содержит инструкции для отображения всего XML-документа.

Каждая XSL-таблица стилей должна содержать один и только один шаблон с атрибутом match, который имеет значение "/".

Можно включить один или несколько дополнительных шаблонов с инструкциями для отображения определенных подчиненных ветвей в структуре XML-документа; каждая из них должна иметь образец, отвечающий определенной ветви.

Корневой образец ("/") не представляет элемент Документ (или корневой элемент) XML-документа. Он представляет весь документ, для которого элемент Документ является дочерним, т.е. он аналогичен корневому узлу Document в объектной модели документа DOM.

XSL-шаблон содержит два вида XML-элементов:

1. XML-элементы, представляющие HTML-разметку. Каждый из элементов, представляющих HTML-разметку, должен быть корректно сформированным XML-элементом, а также стандартным HTML-элементом. Нельзя использовать HTML-конструкции, которые не являются корректно сформированным XML, такие, как элементы, состоящие только из начального тега. Например, чтобы задать элемент перевода строки в HTML, нельзя просто ввести <BR>, как это делается для HTML-страницы. Вместо этого необходимо использовать корректно сформированный тег пустого XML-элемента, <BR/>.

Браузер просто копирует каждый HTML-элемент непосредственно на выход HTML, который воспринимает и отображает их.

Например:

<H2>Описание книг</H2>

<SPAN STYLE="font-style:italic">Author: </SPAN>

*
*

2. *XSL-элементы*. XSL-элементы в шаблоне не копируются на выход HTML. Они содержат инструкции по выбору и модификации данных XML либо используются для выполнения других задач. Браузер отличает XSL-элемент от элемента, представляющего HTML, так как к нему в качестве префикса добавлено описание пространства имен xsl:.

Например, XSL-элемент *value-of* добавляет текстовое содержимое определенного XML-элемента (текст, заключенный между тегами), а также любых его дочерних элементов, в формате HTML.

Если использовать атрибут select XSL-элемента value-of, то можно выбрать образец, который указывает путь к местонахождению элемента в иерархии документа XML.

Каждый контекст внутри XSL-таблицы стилей относится к текущему элементу.

Вы указываете определенный XML-элемент заданием образца, который присваиваете атрибуту select элемента value-of. XML-элемент в образце задается с помощью оператора пути. Оператор пути аналогичен неполному пути к файлу, задающему местонахождение файла относительно текущей рабочей папки. Текущим элементом ("текущей папкой") является значение атрибута match самого шаблона.

Например, файл xml-документа:

<?xml version="1.0"?>

<!-- File name: XslDemo01.xml -->

<?xml-stylesheet type="text/xsl" href="XslDemo01.xsl"?>

<BOOK>

<TITLE>Моби Дик</TITLE>

<AUTHOR>

<FIRSTNAME>Герма</FIRSTNAME>

<LASTNAME>Мелвилл</LASTNAME>

</AUTHOR>

```
    <BINDING>твердая обложка</BINDING>
    <PAGES>724</PAGES>
    <PRICE>$9.95</PRICE>
  </BOOK>
```

XSL-таблица стилей, имеющая только один шаблон элемента <xsl:template match="/">, имеет следующую форму:

```
  <?xml version="1.0"?>
  <!-- File name: XslDemo01.xsl -->
  <xsl:stylesheet
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:template match="/">
      <H2>Описание книг</H2>
      <SPAN STYLE="font-style:italic">Автор: </SPAN>
      <xsl:value-of select="BOOK/AUTHOR"/><BR/>
      <SPAN STYLE="font-style:italic">Заголовок: </SPAN>
      <xsl:value-of select="BOOK/TITLE"/><BR/>
      <SPAN STYLE="font-style:italic">Цена: </SPAN>
      <xsl:value-of select="BOOK/PRICE"/><BR/>
      <SPAN STYLE="font-style:italic">Обложка: </SPAN>
      <xsl:value-of select="BOOK/BINDING"/><BR/>
      <SPAN STYLE="font-style:italic">Количество страниц:
</SPAN>
      <xsl:value-of select="BOOK/PAGES"/>
    </xsl:template>
  </xsl:stylesheet>
```

В примере шаблон относится к корневому элементу всего документа (посредством установки атрибута match="/"), текущим "элементом" для данного шаблона является корневой элемент документа (текущий элемент не обладает соответствующим литералом (названием), а является родителем элемента *Документ*). Внутри этого шаблона оператор пути BOOK/AUTHOR указывает на элемент AUTHOR, вложенный в элемент BOOK, вложенный в корневой элемент документа.

Если опустить атрибут select для XSL-элемента value-of, элемент будет осуществлять вывод текстового содержимого плюс текстовое содержимое всех дочерних элементов текущего элемента. В нашем примере, поскольку текущим является корневой элемент, пропуск ат-

рибута select приведет к выводу всех символьных данных XML-документа.

Порядок элементов value-of в шаблоне определяет порядок, в котором браузер отображает эти элементы.

Схема формирования браузером блока HTML-разметки для документа XML и таблицы стилей XSL представлена на рис. 7.1.

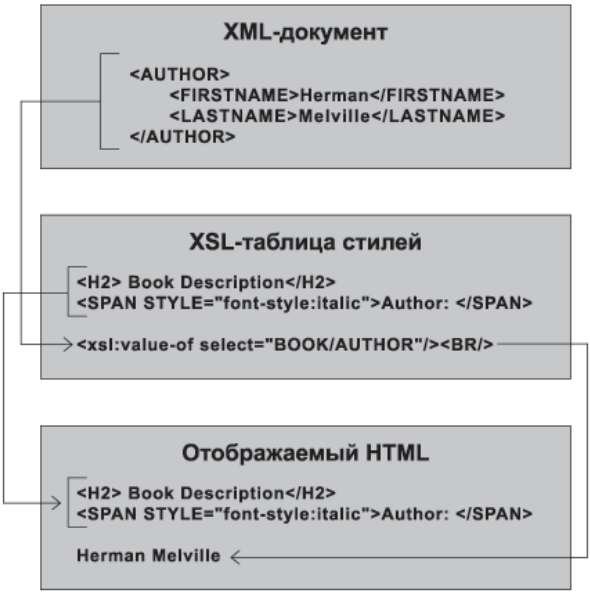


Рисунок 7.1 – Схема формирования блока HTML-разметки

Из этой простой таблицы стилей можно увидеть, что XSL-таблица стилей является гораздо более гибкой, чем CSS, которая всегда отображает элементы в том порядке, в котором они следуют в документе.

7.3. Отображение переменного числа элементов

Мы рассмотрели пример, где XML-документ содержал только один элемент BOOK. В случае если документ содержит несколько элементов BOOK, такая методика не подойдет, она отобразит только один из элементов.

Чтобы отобразить все отвечающие образцу элементы, следует использовать XSL-элемент *for-each*, который вызывает повторный вывод для каждого из содержащихся в XML-файле элементов. Элемент for-each выполняет две основные задачи:

1) осуществляет вывод блока элементов, содержащихся внутри элемента for-each, повторяя его для каждого XML-элемента в документе, отвечающего образцу, присвоенному атрибуту select элемента for-each;

2) задает текущий элемент, устанавливаемый атрибутом select элемента for-each следующим образом:

файл xsl:

```
<?xml version="1.0" encoding="windows-1251"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
<table border="1">
<tr bgcolor="#CCCCCC">
<td align="center"> <strong> Название </strong></td>
<td align="center"> <strong> Имя автора </strong></td>
<td align="center"> <strong> Фамилия автора </strong></td>
<td align="center"> <strong> Обложка </strong></td>
<td align="center"> <strong> Цена </strong></td>
<td align="center"> <strong> Количество страниц </strong> </td>
</tr>
<xsl:for-each select="INVENTORY/BOOK" >
<tr bgcolor="#F5F5F5">
<td> <xsl:value-of select= "TITLE"/> </td>
<td align="right"> <xsl:value-of select= "AUTHOR/FIRSTNAME"/ > </td>
<td> <xsl:value-of select= "AUTHOR/LASTNAME"/> </td>
<td align="center"> <xsl:value-of select= "BINDING"/> </td>
<td align="center"> <xsl:value-of select= "PRICE"/> </td>
<td align="center"> <xsl:value-of select= "PAGES"/> </td>
</tr>
</xsl:for-each>
```

```
</table>
</xsl:template>
</xsl:stylesheet>
```
Для XML-документа:
```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="dema2.xsl"?>
<INVENTORY>
<BOOK>
  <TITLE>Моби Дик</TITLE>
  <AUTHOR>
    <FIRSTNAME>Герман</FIRSTNAME>
    <LASTNAME>Мелвилл</LASTNAME>
  </AUTHOR>
  <BINDING>твердая обложка</BINDING>
  <PAGES>724</PAGES>
  <PRICE>$9.95</PRICE>
</BOOK>
<BOOK>
  <TITLE>Приключения Гекльберри Финна</TITLE>
  <AUTHOR>
    <FIRSTNAME>Марк</FIRSTNAME>
    <LASTNAME>Твен</LASTNAME>
  </AUTHOR>
  <BINDING>твердая обложка</BINDING>
  <PAGES>564</PAGES>
  <PRICE>$15.25</PRICE>
</BOOK>
<BOOK>
  <TITLE>Будденброки</TITLE>
  <AUTHOR>
    <FIRSTNAME>Томас</FIRSTNAME>
    <LASTNAME>Манн</LASTNAME>
  </AUTHOR>
  <BINDING>твердая обложка</BINDING>
  <PAGES>1089</PAGES>
  <PRICE>$85.25</PRICE>
```

</BOOK>
<BOOK>
<TITLE>Молодые годы короля Генриха IV</TITLE>
<AUTHOR>
<FIRSTNAME>Генрих</FIRSTNAME>
<LASTNAME>Манн</LASTNAME>
</AUTHOR>
<BINDING>твердая обложка</BINDING>
<PAGES>1089</PAGES>
<PRICE>$85.25</PRICE>
</BOOK>
</INVENTORY>

С помощью элемента for-each выполняется цикл по одному разу для каждого элемента BOOK, найденного в элементе *Документ* с именем INVENTORY. Образец, присваиваемый атрибуту select элемента for-each, работает точно так же, как образец, присваиваемый атрибуту select элемента value-of;.

Аналогично, внутри элемента for-each каждый дочерний элемент может быть выбран путем задания образца, содержащего только имя элемента, например:

<xsl:value-of select="TITLE"/>

В результате выводятся данные из всех элементов BOOK, найденных в документе, независимо от того, сколько этих элементов содержит документ.

Если элемент xsl:sort присутствует в элементе xsl:for-each, то он всегда должен стоять сразу после элемента xsl:for-each. В нем используются два атрибута: атрибут *order* – способ сортировки (ascending по возрастанию или по убыванию descending) и атрибут *select* – имя поля, по которому производится сортировка. Если нам нужно отсортировать по первому элементу, как в данном примере, то вместо "AUTHOR/LASTNAME" можно было поставить точку ".", для других элементов нужно указывать его имя. На самом деле атрибутов может быть пять – *select, lang, data-type, order* и *case-order*.

Для того чтобы сортировка выполнялась в числовой последовательности, в элемент xsl:sort добавили атрибут data-type =”number”.

Фильтрация строк таблицы. Условие фильтра добавляется в отдельный элемент *xsl:if*:

```
<xsl:if test="PAGES>1000">
<tr bgcolor="#F5F5F5">
<td> <xsl:value-of select= "TITLE"/> </td>
<td align="right"> <xsl:value-of select= "AUTHOR/FIRSTNAME"/> </td>
<td> <xsl:value-of select= "AUTHOR/LASTNAME"/> </td>
<td align="center"> <xsl:value-of select= "BINDING"/> </td>
<td align="center"> <xsl:value-of select= "PRICE"/> </td>
<td align="center"> <xsl:value-of select= "PAGES"/> </td>
</tr>
</xsl:if>
```

7.4. Использование нескольких шаблонов

Другой способ отображения повторяющихся XML-элементов состоит в создании отдельного шаблона для каждого элемента с последующим вызовом этого шаблона с использованием XSL-элемента *apply-templates*:

```
<?xml version="1.0" encoding="windows-1251"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
<H2> Инвентаризация книг </H2>
<xsl:apply-templates select= "INVENTORY/BOOK"/>
</xsl:template>
<xsl:template match="BOOK">
<SPAN STYLE="font-style:italic"> Название: </SPAN>
    <xsl:value-of select="TITLE"/><BR/>
<SPAN STYLE="font-style:italic"> Автор: </SPAN>
    <xsl:value-of select="AUTHOR"/><BR/>
<SPAN STYLE="font-style:italic"> Цена: </SPAN>
    <xsl:value-of select="PRICE"/><BR/>
<SPAN STYLE="font-style:italic"> Обложка: </SPAN>
    <xsl:value-of select="BINDING"/><BR/>
<SPAN STYLE="font-style:italic"> Количество страниц </SPAN>
```

```
        <xsl:value-of select="PAGES"/><BR/>
    <BR/>
    <BR/>
    </xsl:template>
    </xsl:stylesheet>
```

Рассматриваемая в примере таблица стилей содержит два шаблона. Один из них содержит инструкции для отображения всего документа (путем установки match="/", указывающей на корневую часть документа). Все XSL-таблицы стилей требуют наличия такого шаблона. Другой шаблон содержит инструкции для отображения элемента BOOK (шаблон с установкой match="BOOK"). Сначала браузер обрабатывает шаблон, соответствующий корневой части элемента:

```
<xsl:template match="/">
   <H2>Book Inventory</H2>
   <xsl:apply-templates select="INVENTORY/BOOK" />
</xsl:template>
```

XSL-элемент apply-templates сообщает браузеру, что для каждого элемента BOOK внутри корневого элемента INVENTORY он должен обрабатывать шаблон, отвечающий элементу BOOK, т. е. шаблон, для атрибута match которого установлено значение "BOOK". Таблица стилей включает следующий шаблон, отвечающий элементу BOOK:

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

Поскольку этот шаблон отвечает элементу BOOK, элемент BOOK является текущим элементом в контексте шаблона. В связи с этим доступ к дочерним элементам BOOK осуществляется посредством образца, содержащего только имя элемента, как в нашем примере:

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

Примечание. Если вы не укажете атрибут select для элемента apply-templates, браузер обрабатывает соответствующий шаблон (если он имеется) для каждого дочернего элемента текущего элемента. В рассматриваемом примере элемента apply-templates единственным дочерним элементом для текущего элемента (корневая часть документа) является элемент INVENTORY, который не имеет соответствующего шаблона. Таким образом, если вы опустите атрибут select, никакие данные не будут выведены.

ТЕМА 8

ВВЕДЕНИЕ В МОДЕЛЬ RDF

8.1. Выражение семантики данных.

8.2. Модель данных RDF. RDF-граф.

8.3. Типы сущностей RDF.

8.4. Синтаксис RDF/XML.

8.1. Выражение семантики данных

Практически параллельно с работами по стандартизации XML основатель WWW Консорциума Тим Бернерс-Ли сформулировал новое понятие *Semantic Web* – то, каким он видит будущее глобальной сети, и инициировал исследования в этом направлении. В основе предполагаемого им будущего лежит способность машин не только читать, но и понимать содержание Интернет-ресурсов, причем достигнуть этого, по мнению Бернерса-Ли, мы должны не через создание программ искусственного интеллекта, моделирующих деятельность человека, а через использование средств выражения семантики данных и их связей.

Первым шагом в разработке такого Интернета, в котором программы смогут автоматически анализировать содержание Интернет-ресурсов (разработке Semantic Web), явилась разработка языка, позволяющего описывать семантику данных Интернет-ресурсов, ***Resource Description Framework (RDF)*** (структура (конструкция) описания ресурсов).

Неотъемлемой характеристикой любого ресурса *Сети* является сопровождающая его информация. Эту "сверхинформацию", или информацию об информации (о ресурсе) принято называть метаданными. Под ***метаданными*** понимают машинопонятную (машиночитаемую) информацию о веб-ресурсах и других сущностях.

Поскольку метаданные – это данные, на практике в *Сети* существует 3 способа передачи/получения метаданных:

1) метаданные хранятся и передаются внутри документа (тег HEAD в HTML, данные о документе MS Word);

2) сообщение метаданных происходит средствами протокола HTTP (GET, POST или PUT) передачи;

3) метаданные хранятся в каком-то другом документе.

Метаданные ресурса представляют набор независимых высказываний (утверждений).

Наиболее распространенной формой высказывания является следующая модель:

Ресурс – атрибут – значение

Ресурс – это объект, о котором фиксируется высказывание; ***атрибут*** – некоторое именованное свойство; ***значение*** представляет некоторое значение из области значений атрибута:

E-mail - Date - 01.01.2006

E-mail - From – Vasya

8.2. Модель данных RDF. RDF-граф

Основополагающим для RDF является понятие модели данных. Модель данных представляет собой утверждение, включающее ресурс, именованное свойство и его значение.

В RDF ресурсы предоставляются с помощью идентификаторов ресурсов, которые позволяют идентифицировать ресурсы. Под ***ресурсом*** понимают все, чему можно присвоить некоторый идентификатор URI (даже произвольный предмет из мира вещей, не относящийся к Интернету), поскольку URI является уникальным, то он позволяет идентифицировать каждый отдельный ресурс. Значение URI несущественно для RDF, так как используется всего лишь как идентификатор. В более узком смысле ресурс – это все, что можно получить, переходя по ссылке URL (HTML-страниц, PDF-документов и т.п.).

Язык RDF базируется на XML. Формат XML является всего лишь способом хранения RDF в файле.

RDF модель данных можно представить себе как граф, в котором узлы соединяются посредством различных отношений:

1. Например, допустим, каждый из узлов представляет какого-то человека. Каждый человек может быть связан с другим человеком, все люди являются детьми, родителями, супругами, служащими. Каждая связь отмечается названием отношения.

2. Другим типом отношения являются физические свойства узла. Например, имя или возраст человека. Эти отношения отмечаются как ‘имя’ и ‘возраст’ вместо ‘ребенок’ или ‘родитель’.

В RDF существуют два общих типа узлов:

1. *Ресурсы.* Ресурс можно представить как объект в языке программирования.

2. *Литералы.* Литерал – это реальное значение, например имя ‘Sandra’ или число ‘7’. Литерал можно представить как строку, используемую в языке программирования.

Например, человек является ресурсом, а его имя – литералом.

Мы можем добавлять отношения между двумя ресурсами или между ресурсом и литералом. Эти отношения часто называют ***тройными отношениями (тройниками),*** или ***дугами***.

Базовой структурной единицей RDF является коллекция троек (или ***триплетов “triples”***), каждая из которых состоит из двух ресурсов и отношения или ресурса, отношения и литерала.

Триплеты можно записать в следующем виде:

<http://www.xulplanet.com/rdf/people/Sandram> -> name -> Sandra отношение указывает, что существует ресурс <http://www.xulplanet.com/rdf/people/Sandra>, который имеет свойство ‘name’ со значением ‘Sandra’;

<http://www.xulplanet.com/rdf/people/Sandra> -> gender -> female отношение указывает, что тот же ресурс имеет свойство ‘gender’ со значением ‘female’;

<http://www.xulplanet.com/rdf/people/Sandra> -> sibling -> <http://www.xulplanet.com/rdf/people/Kevin>

отношение указывает, что тот же ресурс имеет свойство ‘sibling’, значением которого является другой ресурс <http://www.xulplanet.com/rdf/people/Kevin>;

<http://www.xulplanet.com/rdf/people/Kevin> -> gender -> male

отношение определяет значение свойства ‘gender’ ресурса <http://www.xulplanet.com/rdf/people/Kevin>.

Элементы, заключенные в угловые скобки, являются ресурсами, а элементы, не заключенные в угловые скобки, – литералами.

URI является всего лишь идентификатором. RDF не проверяет что это такое, и, тем более, не проверяет, существует ли такой адрес. Но если один и тот же URI используется в нескольких местах, все они указывают на один и тот же ресурс.

Набор триплетов называется ***RDF-графом***. Левая часть RDF-графа называется ***‘субъект’***, а правая – ***‘объект’***, или ***‘цель’***. Субъект всегда является ресурсом, а объект может быть как ресурсом, так и литералом. Невозможно описать отношения между литералами (т.е. левая часть отношения не может являться литералом). Дугами выступают предикаты (или свойства, или отношения). Направление дуги, соответствующей предикату в данной тройке (S,P,O), всегда выбирается так, чтобы дуга вела от субъекта к объекту, как представлено на рис. 8.1.

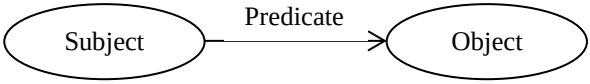


Рисунок 8.1 – RDF-граф

Предикаты – это тоже ресурсы, и можно создать отношения между предикатом и чем-то еще. То есть каждое свойство в RDF само является ресурсом и может иметь свои собственные атрибуты, что обычно используется для определения свойств, описывающих смысл предиката.

Для обеспечения уникальности имен свойства (предикаты) подобно названиям элементов в XML придерживаются концепции пространства имен. Это означает, что не существует атрибута “цвет” (свойства «друг») как такового, а существует “цвет” в каком-то уникальном пространстве. В другом пространстве может существовать свое одноименное свойство. Для обеспечения уникальности предикаты также идентифицируются с помощью URI. Для каждого предиката необходимо использовать уникальный URI. Например, для ‘sibling’ можно применить такой URI:

http://www.xulplanet.com/rdf/people/sibling:

http://www.xulplanet.com/rdf/people/Sandra ->

http://www.xulplanet.com/rdf/people/sibling ->

http://www.xulplanet.com/rdf/people/Kevin.

Графическое представление RDF. Тот факт, что значением свойства может быть некоторый ресурс, превращает модель данных из дерева, которым является XML-разметка, в ориентированный граф. Вершинами этого графа являются субъекты и объекты, а дугами – именованные свойства. Свойство, в свою очередь, может быть субъек-

том некоторого утверждения. При графическом изображении RDF ресурсы принято рисовать овалами, а литералы – прямоугольниками.

Например:

“Ora Lassila is the creator of the resource http://www.w3.org/Home/Lassila

8.3. Типы сущностей RDF

Для указания к какому классу сущностей относится ресурс в RDF используются типы. Ресурсы и литералы – это наиболее общие типы RDF. Но могут быть использованы более точные типы. Например, ресурсу Sandra может быть присвоен тип ‘Person’.

Как и другие свойства, типы также определяются с помощью тройных отношений:

<http://www.xulplanet.com/rdf/people/Sandra> -> rdf:type -> <http://xmlns.com/wordnet/1.6/Person>

Ресурс <http://xmlns.com/wordnet/1.6/Person> используется для представления персоны. Свойство “type” является одним из общезначимых, относящихся к пространству имен, задаваемому непосредственно спецификацией RDF, т.е. предикат ‘type’ встроен в RDF. Данное свойство позволяет указать класс описываемого ресурса. Конструкция *rdf:type* является упрощенной (префикс ‘rdf:’ используется для указания пространства имен RDF (подобно XML)), так как полным именем предиката является 'http://www.w3.org/1999/02/22-rdf-syntax-ns#type'. Это сделано только для упрощения примера – предикат всегда именуется полностью, с указанием пространства имен.

RDF имеет большое количество встроенных типов для представления списков сущностей. Например, можно создать несколько тройных отношений с одними и теми же субъектами и предикатами:

<http://www.xulplanet.com/rdf/people/Sandra> -> name -> Sandra

<http://www.xulplanet.com/rdf/people/Sandra> -> name -> Sandy

Здесь один и тот же ресурс имеет тройные отношения с одним и тем же предикатом, но с разными целями. В этом примере предполага-

ется присвоение нескольких имен, предназначенных для обозначения ников.

RDF не упорядочивает имена, поэтому все они эквивалентны. Но в некоторых случаях может оказаться полезным поместить набор значений в определенном порядке. Для этого в RDF есть встроенный механизм: RDF предикат, пронумерованный с помощью знака подчеркивания, используется для указания на элемент списка. Например, rdf:_1 используется для указания на первый элемент списка (используется пространство имен RDF).

Для того чтобы в RDF эти нумерованные предикаты распознавались каким-то специальным образом (а эти предикаты являются всего лишь обычными предикатами), необходимо использовать специальные типы для списков. Доступны следующие списочные типы:

- *rdf:Seq*: упорядоченный список, элементы размещаются в указанном порядке (sequence);
- *rdf:Bag*: неупорядоченный список;
- *rdf:Alt*: список альтернативных значений, из которых используется только одно.

Назначение списочных типов производится точно так же, как и любых других типов:

<http://www.xulplanet.com/rdf/people/Karen> -> rdf:type -> <http://www.w3.org/1999/02/22-rdf-syntax-ns#Seq>

Здесь для целевого значения используется полное имя пространства имен RDF. Теперь, когда Karen имеет тип rdf:Seq, мы можем добавить трех ее детей. Но так как ресурс Karen имеет тип rdf:Seq, он не относится к типу Person. Мы могли бы обойти эту проблему, связав с Karen второй тип, но лучшим способом является использование второго ресурса, замещающего список детей Karen. Ресурс Karen будет иметь тип Person, а список детей Karen будет иметь тип rdf:Seq:

<http://www.xulplanet.com/rdf/people/Karen> -> rdf:type -> <http://xmlns.com/wordnet/1.6/Person>

<http://www.xulplanet.com/rdf/people/Karen> -> children -> <http://www.xulplanet.com/rdf/people/KarensKids>

<http://www.xulplanet.com/rdf/people/KarensKids> -> rdf:type -> <http://www.w3.org/1999/02/22-rdf-syntax-ns#Seq>

<http://www.xulplanet.com/rdf/people/KarensKids> -> rdf:_1 -> <http://www.xulplanet.com/rdf/people/Sandra>

<http://www.xulplanet.com/rdf/people/KarensKids> -> rdf:_2 -> <http://www.xulplanet.com/rdf/people/Kevin>

<http://www.xulplanet.com/rdf/people/KarensKids> -> rdf:_3 -> <http://www.xulplanet.com/rdf/people/Jack>

Здесь мы присвоили ресурсу Karen тип Person и связали его с ресурсом <http://www.xulplanet.com/rdf/people/KarensKids> с помощью предиката ‘children’. Вместо “привязывания” трех ресурсов, описывающих детей Karen, непосредственно к ресурсу Karen, мы связали их с дополнительным ресурсом, который имеет тип rdf:Seq. В результате, Karen имеет трех детей, но и имеет уникальный тип Person.

В RDF нам не требуется указывать подробный URI для <http://www.xulplanet.com/rdf/people/KarensKids>. Поскольку подобный шаблон используется в большинстве случаев, RDF позволяет использовать пустые узлы или анонимные ресурсы. RDF API позволяет создавать такие узлы, и обычно в этих случаях генерируется случайный URI. Технически они не имеют URI, но вы можете манипулировать ими как ресурсами и добавлять или удалять тройные отношения, связанные с ними.

Во всех приведенных ранее примерах мы определяли только тройные отношения. Каждая строка имеет субъект, предикат и объект. Даже при определении типов и списков мы использовали тройные отношения для их определения. RDF является только списком тройных отношений, или, другими словами, списком отношений между сущностями.

Интересной особенностью RDF является возможность объединения списков тройных отношений из разных источников. Поскольку порядок не имеет значения, эффект будет таким же, как если бы мы определили все эти отношения в одном источнике с самого начала. Допустим, некоторый источник предоставляет следующее тройное отношение:

<http://www.xulplanet.com/rdf/people/KarensKids> -> rdf:_4 -> <http://www.xulplanet.com/rdf/people/Wendy>

После объединения этого отношения с ранее приведенными примерами окажется, что Karen имеет четырех детей, а не трех. Это

очень важная особенность RDF – возможность комбинировать или объединять данные из различных источников.

Некоторые выводы по RDF:

- RDF (Resource Description Framework) – язык для описания ресурсов способом, «понятным» компьютеру на семантическом уровне.
- Ресурс – любая (физическая или абстрактная) сущность, имеющая уникальный идентификатор URI.
- RDF официальная рекомендация консорциума W3C с 10/02/2004 (http://www.w3.org/RDF).
- Утверждения, которые задаются тройками:

<субъект, предикат, объект>

Субъект – некоторый ресурс (идентификатор ресурса).

Предикат – свойство ресурса (или отношение с другим ресурсом).

Объект – значение свойства ресурса (или отношения).

- Свойства также являются ресурсами и имеют URI.
- Так как URI могут быть довольно длинными, то в форматах, используемых для представления RDF, они обычно сокращаются, используя перенятый из XML механизм «пространств имен». Имена берут свои свойства из пространства имен, которые указываются в виде префикса.

8.4. Синтаксис RDF/XML

Существует общий формат XML для хранения RDF, который называется ***RDF/XML***. Основа RDF/XML-файла выглядит подобно следующему примеру:

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:people="http://www.xulplanet.com/rdf/people/">
</rdf:RDF>
```

В примере объявляется два пространства имен, одно из которых является пространством имен RDF, а второе, ‘people’, предназначено для собственных целей. Можно использовать свое собственное пространство имен или одно из множества других, созданных для различных целей.

Отношение между ресурсом и литералом можно записать с помощью одного тега Description из пространства имен RDF, с двумя атрибутами:

1) атрибут *about* используется для указания субъекта отношения;

2) для указания предиката используют атрибут с именем, представляющим имя предиката с указанием пространства имен; значение данного атрибута представляет объект триплета.

Например:

http://www.xulplanet.com/rdf/people/Sandra> -> name -> Sandra

Можно добавить это отношение в файл:

<?xml version="1.0"?>

<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" xmlns:people="http://www.xulplanet.com/rdf/people/">

<rdf:Description rdf:about="http://www.xulplanet.com/rdf/people /Sandra" people:name="Sandra"/>

</rdf:RDF>

Совместно с указанием пространства имен этот атрибут “разворачивается” в значение http://www.xulplanet.com/rdf/people/name, которое является предикатом тройного отношения.

Таким образом, чтобы определить тройное отношение, мы пользуемся преимуществом указания пространства имен и присваиваем значения цели атрибутам с теми же именами, что и у предикатов.

RDF/XML имеет множество альтернативных способов для определения одних и тех же сущностей.

В RDF порядок записи отношений не важен: можно помещать теги или атрибуты в любом порядке:

<rdf:Description rdf:about="http://www.xulplanet.com/rdf/people/ Sandra"

people:name="Sandra"/>

<rdf:Description rdf:about="http://www.xulplanet.com/rdf/people/ Sandra"

people:gender="female"/>

Необязательно записывать тег *Description* каждый раз; можно использовать один тег с несколькими атрибутами. Например, следующее описание дает аналогичные результаты:

<rdf:Description rdf:about="http://www.xulplanet.com/rdf/people/Sandra"

people:name="Sandra"

people:gender="female"/>

Предикат можно определять как атрибутом, так и тегом:

<rdf:Description rdf:about="http://www.xulplanet.com/rdf/people/Sandra">

<people:name>Sandra</people:name>

</rdf:Description>

Здесь атрибут предиката был заменен тегом. Оба способа дают одинаковый результат, но по некоторым причинам может оказаться предпочтительней пользоваться одним из них.

Использование атрибутов дает более короткую запись и полезно при определении большого количества предикатов.

Использование тегов более удобно:

- при длинных значениях для лучшей читаемости,
- а также в случаях, когда значения содержат кавычки
- теги позволяют использовать один и тот же предикат несколько раз:

<rdf:Description rdf:about="http://www.xulplanet.com/rdf/people/Sandra">

<people:name>Sandra</people:name>

<people:name>Sandy</people:name>

</rdf:Description>

В этом примере определены два тройных отношения с одним и тем же ресурсом и предикатом, но с разными целевыми значениями. Мы видели эти отношения в предыдущем разделе. Мы не можем записать эти отношения с помощью атрибутов только потому, что тег не может содержать один и тот же атрибут дважды.

При необходимости можно совмещать обе формы записи. В следующем примере определены три отношения с использованием смешанной формы записи:

<rdf:Description rdf:about="http://www.xulplanet.com/rdf/people/Sandra"

people:name="Sandra">

<people:name>Sandy</people:name>

<people:gender>female</people:gender>

</rdf:Description>

Отношения между ресурсами. Может потребоваться определить отношения между ресурсами, подобные следующему:

<http://www.xulplanet.com/rdf/people/Sandra> -> *sibling* -> *<http://www.xulplanet.com/rdf/people/Kevin>*

Нельзя использовать атрибуты для отношений такого типа, так как атрибуты могут использоваться только для значений литералов.

При определинии отношений между ресурсами предикат записывается в виде тега, а объект определяется значением атрибута *resource*:

<rdf:Description rdf:about="http://www.xulplanet.com/rdf/people /Sandra" people:name="Sandra">

<people:sibling rdf:resource="http://www.xulplanet.com/rdf/people /Kevin"/>

</rdf:Description>

Здесь мы используем атрибут resource для указания того, что ресурс Sandra имеет свойство sibling, значением которого является другой ресурс <http://www.xulplanet.com/rdf/people/Kevin>.

Этот атрибут находится в пространстве имен RDF и используется для определения целевых ресурсов, тогда как текст, заключенный в тег, используется для определения литералов. Обратите внимание на схожесть этой формы записи с формой записи атрибута about. Атрибут about используется для субъектов – левой части тройных отношений, а атрибут resource используется для объектов (целей) – правых частей тройных отношений.

В предыдущем примере имя Сандры определялось с помощью атрибута. Если потребуется добавить какие-то отношения для Кевина, нужно будет воспользоваться другим тегом Description:

<rdf:Description rdf:about="http://www.xulplanet.com/rdf/people/ Sandra" people:name="Sandra">

<people:sibling rdf:resource="http://www.xulplanet.com/rdf/people/ Kevin"/>

</rdf:Description>

<rdf:Description rdf:about="http://www.xulplanet.com/rdf/people/Kevin" people:name="Kevin"/>

Второй тег Description используется для объявления другого тройного отношения, для другого ресурса субъекта, так как атрибут about имеет другое значение.

Существует другая форма записи, которой можно воспользоваться, например, для описания отношения между двумя людьми в одном файле. Вместо записи второго тега Description отдельно можно поместить его внутрь первого тега:

<rdf:Description rdf:about="http://www.xulplanet.com/rdf/people/Sandra" people:name="Sandra">

<people:sibling>

<rdf:Description rdf:about=http://www.xulplanet.com/rdf/people/Kevin people:name="Kevin"/>

</people:sibling>

</rdf:Description>

В результате будет определен тот же набор тройных отношений. Но вместо направления свойства sibling на ресурс и описания его отдельно, в данном примере он включается непосредственно в ресурс. Не пользуемся атрибутом resource, так как ресурс указан внутри атрибута about. Уровень вложенности информации при таком способе записи не ограничен.

Если вы удалите атрибут about из предыдущего отношения, как это сделано в следующем фрагменте:

<rdf:Description
rdf:about="http://www.xulplanet.com/rdf/people/Sandra">

<people:sibling>

<rdf:Description people:name="Kevin"/>

</people:sibling>

</rdf:Description>,

то в результате будет образован почти такой же граф, в котором Кевин все еще остается братом Сандры, но не имеет URI. Поэтому будет создан пустой (анонимный) узел, не имеющий URI, но все еще являющийся узлом RDF-графа. Используя RDF API, вы можете получить ссылку на узел с помощью его URI, но не сможете получить пу-

стой узел таким способом, так как он не имеет URI. Вам потребуется изменить направление поиска, чтобы найти такой узел.

Определение типов в xml/rdf. Так как определение типа узла осуществляется посредством тройного отношения так же, как и для другой информации в RDF, можно использовать ту же форму для определения типа узла.

Например, для представления следующего отношения:

<http://www.xulplanet.com/rdf/people/Sandra> -> rdf:type -> <http://xmlns.com/wordnet/1.6/Person>

Это отношение в формате RDF/XML:

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:people="http://www.xulplanet.com/rdf/people/">
<rdf:Description rdf:about=http://www.xulplanet.com/rdf/people/
Sandra>
<rdf:type resource="http://xmlns.com/wordnet/1.6/Person"/>
</rdf:Description>
</rdf:RDF>
```

Здесь нет каких-либо отличий от определения других тройных отношений. Используем предикат rdf:type, объектом которого является тип Person.

Но так как типы часто используются в RDF, существует короткая форма записи:

1) добавляется пространство имен wordnet="http://xmlns.com/wordnet/1.6/;

2) тег Description заменен тегом, определяющим непосредственно тип;

3) в такой записи, если есть тег с атрибутом about, то этот тег является типом этого ресурса:

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
        xmlns:wordnet="http://xmlns.com/wordnet/1.6/"
        xmlns:people="http://www.xulplanet.com/rdf/people/">
<wordnet:Person
rdf:about="http://www.xulplanet.com/rdf/people/Sandra"/>
</rdf:qualifiedRDF>
```

Тег *wordnet:Person* будет развернут в значение 'http://xmlns.com/wordnet/1.6/Person'.

Если вы видите тег с атрибутом about, то этот тег является типом этого ресурса. Если используется тег Description, это означает, что тип не известен или не важен. Можно использовать синтаксис вложенных отношений, хотя такой синтаксис может быть трудно читаемым:

```
<wordnet:Person rdf:about="http://www.xulplanet.com/rdf/people/
Sandra"
        people:name="Sandra">
  <people:sibling>
<wordnet:Person rdf:about="http://www.xulplanet.com/rdf/people/
Kevin"
         people:name="Kevin"/>
  </people:sibling>
</wordnet:Person>
```

Здесь мы указываем, что ресурсы Sandra и Kevin относятся к типу Person. Обратите внимание, что определения становятся трудночитаемыми при использовании многоуровневых вложений, так как трудно заметить, какие из тегов являются типом, а какие – предикатом. Хотя они всегда взаимоисключают друг друга.

В этом примере внешний тег является типом, следующий вложенный – предикатом, и следующий за ним – опять типом. Следующий тег должен быть снова предикатом. Такая концепция иногда называется ***чередованием***, так как направление типов и предикатов различны.

Списочные типы. RDF поддерживает ***контейнеры*** – способ организации однотипных фактов.

В обычном факте между объектом и субъектом (или между ресурсом и значением свойства) существует отношение "один к одному". У каждого объекта есть один субъект. В случае контейнеров отношение между фактом и объектом имеет вид "один ко многим", причем минимально возможное число объектов у субъекта равно нулю. Контейнеры RDF являются аналогом списков или массивов в других языках. То, каким образом должны обрабатываться контейнеры, зависит от конкретного приложения.

Контейнер состоит из обрамляющих тегов списочных типов, Seq, Bag и Alt, и тегов, содержащих элементы контейнера.

Например:

<http://www.xulplanet.com/rdf/people/KarensKids> -> rdf:type -> <http://www.w3.org/1999/02/22-rdf-syntax-ns#Seq>

В контейнерах вместо тега Description или имени типа указывается списочный тип rdf:Seq для типа Seq, и аналогичные теги для Bag и Alt:

<rdf:Seq rdf:about="http://www.xulplanet.com/rdf/people/Karens Kids"/>

Определение других типов не имеет каких-либо отличий. В данном случае ресурсу <http://www.xulplanet.com/rdf/people/ KarensKids> присвоен тип rdf:Seq. Естественно, нам требуется добавить детей в список этого ресурса:

<rdf:Seq rdf:about="http://www.xulplanet.com/rdf/people/Karens Kids">

<rdf:_1 rdf:resource="http://www.xulplanet.com/rdf/people/Sandra "/>

<rdf:_2 rdf:resource="http://www.xulplanet.com/rdf/people/Kevin" />

<rdf:_3 rdf:resource="http://www.xulplanet.com/rdf/people/Jack"/>

</rdf:Seq>

Так же, как и для всех отношений, мы добавляем их в ресурс и указываем на три дочерних ресурса. Допустим, Карен имеет 20 детей. Мы записали этот факт, но забыли описать какого-то одного, и нам необходимо добавить этого ребенка и перенумеровать остальных в списке. Но это не требуется, так как допустимо пропускать номера, хотя это и будет выглядеть необычно.

Для разрешения подобных проблем RDF/XML предоставляет специальный тег для автоматического перечисления ***li***:

<rdf:Seq rdf:about="http://www.xulplanet.com/rdf/people/Karens Kids">

<rdf:li rdf:resource="http://www.xulplanet.com/rdf/people/Sandra "/>

<rdf:li rdf:resource="http://www.xulplanet.com/rdf/people/Kevin" />

<rdf:li rdf:resource="http://www.xulplanet.com/rdf/people/Jack"/>

</rdf:Seq>

Вместо использования чисел мы можем воспользоваться тегом li, который обеспечивает автоматическое перечисление. Первый элемент будет помечен числом 1 (на самом деле rdf:_1), второй – 2, третий – 3. Мы можем вставлять детей без перенумерации остальных.

Тег li не используется непосредственно как предикат. С помощью RDF API вы не сможете найти предикаты li, так как все они преобразуются в числовую форму. Этот тег используется только для удобства описания RDF в XML-формате.

Обратите внимание, что тег li имеет атрибут resource. Как всегда, он указывает, что <http://www.xulplanet.com/rdf/people/Sandra> является целью тройного отношения. Так же как и раньше, мы можем вкладывать элементы в этот тег:

<rdf:li>

<wordnet:Person rdf:about="http://www.xulplanet.com/rdf/people/Sandra" people:name="Sandra"/>

</rdf:li>

Этот пример комбинирует различные подходы, описанные выше. Ребенок в списке все еще является ресурсом <http://www.xulplanet.com/rdf/people/Sandra>, но имеет имя и тип.

Таким образом, контейнер состоит из обрамляющего тега <Bag>, <Seq> или <Alt>, а также тегов <li>, в которые заключаются элементы контейнера. Контейнер выглядит следующим образом:

<Description>

<Bag>

*
объект 1
*

*
объект 2
*

*
объект 3
*

</Bag>

</Description>

Контейнер может быть помещен вместо объекта какого-либо факта.

Далее приведен полный пример для описания семьи Карен:

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
        xmlns:wordnet="http://xmlns.com/wordnet/1.6/"
        xmlns:people="http://www.xulplanet.com/rdf/people/">
<wordnet:Person rdf:about="http://www.xulplanet.com/rdf/people/
Karen" people:name= "Karen">
  <people:children>
   <rdf:Seq rdf:about="http://www.xulplanet.com/rdf/people/Karens
Kids">
     <rdf:li>
      <wordnet:Person rdf:about="http://www.xulplanet.com/rdf/
people/Sandra" people:name="Sandra"/>
     </rdf:li>
     <rdf:li>
      <wordnet:Person rdf:about="http://www.xulplanet.com/rdf/
people/Kevin" peo-ple:name="Kevin"/>
     </rdf:li>
     <rdf:li>
      <wordnet:Person rdf:about="http://www.xulplanet.com/rdf/
people/Jack" people:name="Jack"/>
     </rdf:li>
   </rdf:Seq>
  </people:children>
</wordnet:Person>
</rdf:RDF>
```

Здесь мы создали тройные отношения из предыдущего раздела, добавив несколько новых. Всего здесь записано 13 отношений: одно имя для каждого из четырех людей, один тип для каждого человека, один тип для Seq, один – для каждого из трех элементов списка и один для предиката ‘children’.

В качестве простой иллюстрации RDF-документа можно привести описание парка, состоящего из двух автомобилей, для каждого из которых указан год выпуска:

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
```

```
    xmlns:dat="http://description.org/dates#"
xmlns:foo="http://foo.org/car-park#">
     <rdf:Description  rdf:about="http://foo.org/car-park">
     <foo:car  rdf:resource="http://foo.org/car-park/01"
      rdf:type="http://description.org/vehicles#car" dat:year="1999"/>
     <foo:car  rdf:resource="http://foo.org/car-park/02"
      rdf:type="http://description.org/vehicles#car"  dat:year="2001"/>
     </rdf:Description>
    </rdf:RDF>
```

Подключение словаря или онтологии для использования имен ресурсов. ***Глобальные имена***, имеющие всюду один и тот же смысл, всегда имеют вид URI. URI используются для глобальных имен потому, что они позволяют разбить пространство всех возможных имен на блоки, за которыми закреплены владельцы.

В свою очередь, синтаксис глобальных имен тоже делится на две группы:

1. ***URN (Uniform Resource Name)*** позволяет идентифицировать ресурс, но не указывает на его местоположение. Например, для идентификации книг по их ISBN-номеру – urn:isbn:0143034650 и т.п. Другой универсальный тип URI – это TAG; пример такого URI – tag:govtrack.us, 2005:congress/senators/frist.

2. ***URL (Uniform Resource Locator)*** указывает на способ получения доступа к ресурсу. Например: URL для того же самого стандарта выглядит так: http://tools.ietf.org/html/rfc2549.

Наборы URI свойств публикуются в словарях или онтологиях. Объявления пространств имен и показывает подключение некого словаря или онтологии для использования имен ресурсов.

Примерами общеизвестных словарей, снабженных спецификациями и пояснениями для облегчения их использования, являются:

- *FOAF (Friend of a Friend).* Эта онтология служит для описания информации о людях, их интересах, связях с другими людьми. Предоставляет свойства для описания людей: name, homepage, mbox (e-mail), account.
- *SIOc (Semantically-Interlinked Online communities).* Цель этого словаря – описать сообщения в форумах, чатах, блогах и связать эти сообщения между собой и другими публикациями по сходной темати-

ке. Предоставляет свойства для описания онлайновых социальных сетей и их пользователей: follows, has_reply, last_reply_date, moderator_of, subscriber_of.

- *DOAp (Description Of A project*). Этот словарь служит для описания open source программных проектов.

- *WSMO (Web Service Modeling Ontology).* С помощью этого словаря веб-сервисы могут публиковать информацию о себе и о том, как вызвать некоторую функцию сервиса.

И еще множество грамматик, ориентированных на специфические предметные области: страны, товары, географическая информация и т.д.:

```
<rdf:RDF
  xml:lang="ru"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:foaf="http://xmlns.com/foaf/0.1/"
  xmlns:dc="http://purl.org/dc/elements/1.1/">
 <foaf:Person>
    <foaf:name>Юрий Катков</foaf:name>
  <foaf:gender>male</foaf:gender>
  <foaf:birthday>12-25</foaf:birthday>
  <foaf:dateOfBirth>1987-12-25</foaf:dateOfBirth>
  <foaf:img
    rdf:resource="http://cs690.vkontakte.ru/u79863/
a_d7e07143.jpg"/>
</foaf:Person>
</rdf:RDF>
```

В данном примере будет создан пустой (анонимный) узел, не имеющий URI, но являющийся узлом RDF-графа.

- *Дублинское Ядро (Dublin core).* Это словарь, созданный специалистами в области хранения и каталогизации информации, служит для представления сведений о том, кто разработал некоторый документ, кто его опубликовал, когда. Этот словарь используется главным образом для составления каталогов библиотек и архивов. Т.е. предоставляет свойства для описания опубликованных работ. Включает 15 простых элементов: creator (создатель), Language (язык), publisher (из-

датель), Title (название), Date (дата) и 18 квалифицированных свойств: Audience (аудитория), RightsHolder (правообладатель). По адресу http://beshenov.ru/dcmi-terms.html можно ознакомиться со всеми свойствами, используемыми в работе.

Система Дублинского ядра разрабатывалась в штате Огайо в городе Дублин сотрудниками ***Online Computer Library Center (OCLC)*** с 1995 года.

В первоначальном своем виде система предлагала описывать цифровые ресурсы html страниц с помощью встраивания в заголовок документа (head) специальных тегов meta. В качестве значения атрибута name указывается название DC-характеристики, с указанием пространства имен: если основная спецификация, то DC:, и если дополнительная, то DcTERMS:, затем имя самой характеристики, рекомендованное записывать маленькими буквами. Значение характеристики записывается значением атрибута content:

<meta name="DC.subject" content = "dublin core metadata element set">

<meta name="DC.publisher" content="OCLC Online Computer Library Center Inc.">

<meta name="DC.creator" content="Weibel, Stuart L., wei...@oclc.org.">

<meta name="DC.title" content="Dublin Core Element Set Reference Page">

<meta name="DC.date" content="1996-05-28">

<meta name="DC.language" scheme="ISO 639" content="en">

<meta name="DcTERMS.audience" content="Заинтересованные в Semantic Web" />

В стандарте XML/RDF пространства имен Dublin core подобная информация о ресурсе будет записана следующим образом:

<rdf:RDF>
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:dc="http://purl.org/dc/elements/1.1/">
<rdf:description RDF:about= "http://hamlet.org">
<dc:creator> Shakespeare </dc:creator>
<dc:type> play </dc:type>
</rdf:description>

</rdf:RDF>

Все элементы Дублинского ядра можно разбить на три группы:

1) элементы, относящиеся к содержанию ресурса (content);

2) элементы, описывающие цифровой ресурс с точки зрения интеллектуальной собственности (Intellectual Property);

3) элементы, относящиеся к конкретному экземпляру ресурса (Instantiation).

Поэтому Дублинское ядро – развивающаяся система, интересующая специалистов компьютерных, библиотечных и юридических наук.

ТЕМА 9

RDF SCHEMA

9.1. Язык RDFS

***RDFS* (*RDF Schema*)** – язык описания словарей для RDF. RDF определяет формальный способ записи ресурсов, но не объясняет значение предикатов или свойств. Предикат RDF может пониматься как атрибут (субъект – свойство (атрибут) – значение) и как отношение между двумя ресурсами (субъект – отношение – объект (ресурс)).

RDFS придает предикатам и ресурсам дополнительный смысл (семантику), т.е. показывает, как термин должен интерпретироваться. RDFS используется для того, чтобы фиксировать на RDF произвольные высказывания RDFS, позволяет определить словарь терминов и отношения между ними.

RDFS представляет собой модель типизации данных, т.е. показывает, какие типы данных могут быть в документе.

RDF Schema – это просто документ или фрагмент кода, управляющий множеством терминов в другом документе или фрагменте кода.

RDF Schema является семантическим расширением RDF, он определяет классы, свойства и другие ресурсы. За ресурсами RDF схемы в спецификации W3C закреплена семантика. Например, за ресурсом rdfs:subClassOf закреплена семантика наследования.

Все определения RDFS выражены на RDF, поэтому RDFS называется ***«самоописывающимся»***. Новые термины, вводимые RDFS, которые используются при создании словарей, являются ресурсами:

- rdfs:Class (класс);
- rdf:Property (свойство);
- rdf:type (тип);

- rdfs:subClassOf (подкласс);
- rdfs: subPropertyOf (подсвойство);
- rdfs:range (область значений);
- rdfs:domain (область определения).

Система классов и свойств языка описания RDF похожа на систему типов объектно-ориентированных языков программирования. Но RDF отличается от большинства языков ООП, тем что центральным аспектом является определение свойства, а не класса.

9.2. Классы RDFS

Ресурсы могут объединяться в группы, называемые ***классами***. Члены класса называются ***экземплярами класса*** (по типу объектов класса в ООП). Сами классы тоже являются ресурсами и идентифицируются URI. Классы просто сообщают о том, какие вещи в них входят. Объекты могут одновременно принадлежать нескольким классам. При этом необязательно должна присутствовать какая-нибудь иерархическая структура, например, объект может сразу принадлежать нескольким классам: *Человек, АнимационныйОбъект, Животное, ВысокийЧеловек, Друг* и т.д.

Для того чтобы указать, что ресурс является экземпляром класса, используется свойство *rdf:type*.

RDF отделяет класс от множества его экземпляров (т.е. экстенсионала). Два класса с одинаковыми экстенсионалами считаются различными, если они имеют разные наборы свойств (интенсионалы).

Рассмотрим множества A={0, 2, 4, 6, 8}, {x, | x=2k, k=0.4, k – целое}, C – множество неотрицательных четных чисел менее 10.

В этом примере множество A описывается своим экстенсиалом, множества B и C описываются интесионалами, т.е. используются характеристические свойства данного множества.

Основным шагом в любом виде описания процесса является идентфикация вида вещей, которые должны быть описаны. RDF Schema определяет этот тип вещи как классы. Класс в RDF Schema соответствует главному основному концепту типа или категории, что-то подобное взгляду на класс в ООП языках, подобно Java. RDF классы могут быть использованы для представления почти любых катего-

рий вещей, таких как Web страница, люди, тип документа, база данных или абстрактный концепт. Класс описывается RDFS ресурсами *rdfs:Class* и *rdfs:Resource*, и свойствами *rdf:type* и *rdfs:subClassOf*.

Над множеством классов определено отношение «подкласс – надкласс», описываемое RDFS свойством rdfs:subClassOf. Семантика данного свойства в том, что экстенсионал любого подкласса данного класса (C) целиком включается (как множество) в экстенсионал данного класса (C). Т.е., если (i) является экземпляром класса C', а класс C' является подклассом класса C., то (i) является экземпляром класса C.

В спецификации RDFS определены также списки, коллекции и контейнеры ресурсов, текстовые пометки и комментарии.

Свойство в RDFS определяется как пара (домен, диапазон).

Домен RDFS – некоторое множество классов RDFS, к которым данное свойство применимо.

Диапазон определяет допустимое множество ресурсов – значений свойства.

В RDFS описание класса всегда остается открытым (т.е. набор свойств класса определяется вне самого класса). Добавление/удаление свойства интуитивно проще, чем управление множествами классов, обладающих каждый своим индивидуальным набором свойств (как в ООП).

Например, определим свойство «автор» с доменом «Документ» и диапазоном «Человек»:

- класс («Документ»);
- класс («Человек»);
- свойство («автор», «Документ», «Человек»).

В случае появления дополнительной информации о свойствах «Документа» нет необходимости изменять описания класса «Документ». Достаточно добавить новое свойство с соответствующим доменом.

Для предикатов можно задать области определения и значения – т.е. указать, что субъектом в свойстве "совершить преступление" может быть представитель объединения классов "физическое лицо" и "юридическое лицо" (задание области определения), а объектом может выступать любой представитель класса "Преступление".

Кроме этого, RDFS предоставляет нам набор контейнеров для группировки графов: *Sequence, Bag, Alt*.

Если необходимо сделать утверждение об утверждении RDFS, используется реификация, или материализация утверждения. Для этого используется специальный класс *rdf:Statement* и его свойства *rdf:subject rdf:predicate rdf:object*. Каждое утверждение является экземпляром класса rdf:Statement, по его свойствам и их значениям можно однозначно идентифицировать само утверждение.

Первым шагом при создании словаря является создание классов. В RDF Schema ***класс*** – это любой ресурс, который имеет тип rdf:type (свойство), обозначаемое, как ресурс rdfs:Class.

В синтаксисе RDF/XML можно использовать так называемый ***базовый URI***, который указывается атрибутом *xml:base* тега <rdf:RDF…, обычно это идентификатор ресурса, расположенного там же, где и парсер (обработчик данного кода):

http://example.org/schemas/vehicles#MotorVehicle

При наличии базового URI, если в последующих элементах есть атрибут ID, то полное имя этого элемента будет:

базовый_URI#значение_параметра_ID (ссылка на элемент)

Например:

<rdf:RDF xml:base="http://example.org/schemas/vehicles">

<rdf:Description rdf:ID="MotorVehicle">

Полный URI ресурса, о котором идет речь:

http://example.org/schemas/vehicles#MotorVehicle

ID показывает, что данный ресурс уникален в рамках данного URI.

Если словарь будет перемещен или скопирован, можно просто назначить новый базовый URI, предполагая, что вся схема будет расположена в одном месте.

Запись примера RDFS на языке RDF/XML:

<?xml version="1.0"?>

<rdf:RDF

xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
xml:base="http://example.org/schemas/vehicles">

<rdf:Description rdf:ID="MotorVehicle">

<rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema# Class"/>

</rdf:Description>

Классы маркированы типом http://www.w3.org/2000/01/rdf-schema#Class, поэтому фактически можно использовать rdfs:Class, а не rdf:Description, используя тип ресурса в качестве имени атрибута:

<rdfs:Class rdf:ID="MotorVehicle"/>

Запишем на языке XMl/RDF классы ресурса MiniVen:

<rdf:Description rdf:ID="MiniVan">

<rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema# Class"/>

<rdfs:subClassOf rdf:resource="#Van"/>

<rdfs:subClassOf rdf:resource="#PassengerVehicle"/>

</rdf:Description>

Или:

<rdfs:Class rdf:ID="MiniVan">

<rdfs:subClassOf rdf:resource="#Van"/>

<rdfs:subClassOf rdf:resource="#PassengerVehicle"/>

</rdfs:Class>

Полный код графа RDF/XML, показанного выше:

```
<?xml version="1.0"?>
<rdf:RDF
 xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
 xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
 xml:base="http://example.org/schemas/vehicles">
<rdfs:Class rdf:ID="MotorVehicle"/>
<rdfs:Class rdf:ID="PassengerVehicle">
 <rdfs:subClassOf rdf:resource="#MotorVehicle"/>
</rdfs:Class>
<rdfs:Class rdf:ID="Truck">
 <rdfs:subClassOf rdf:resource="#MotorVehicle"/>
</rdfs:Class>
<rdfs:Class rdf:ID="Van">
 <rdfs:subClassOf rdf:resource="#MotorVehicle"/>
</rdfs:Class>
<rdfs:Class rdf:ID="MiniVan">
```

<rdfs:subClassOf rdf:resource="#Van"/>
<rdfs:subClassOf rdf:resource="#PassengerVehicle"/>
</rdfs:Class>
</rdf:RDF>

Итак, первым шагом в определении словаря является создание классов. Как и классы в объектно-ориентированном языке программирования, классы RDFS предоставляют шаблон для создания объектов или, говоря языком RDF, индивидуализированных объектов (individuals).

9.3. Свойства в RDFS

Свойства в RDF Schema описываются с помощью использования RDF класса *rdf:Property* (все свойства в RDF описываются как экземпляры класса rdf:Property) и RDF Schema-свойствами:

- rdfs:domain;
- rdfs:range;
- rdfs:subPropertyOf.

Можно использовать совместно классы и свойства путем использования специфических свойств *dfs:range* and *rdfs:domain*. Rdfs:domain всегда определяет, к какому классу принадлежит субъект триплета, использующий это свойство, а dfs:range всегда определяет, какому классу принадлежит объект триплета, использующий это свойство, и указывает допустимые конкретные значения данного свойства.

Например, если мы хотим указать, что есть класс ex:Person, есть свойство – иметь автором ex:author – и это свойство может иметь значение, представляющее экземпляр класса ex:Person, а обладать данным свойством ‘иметь автором’ могут экземпляры класса ex:Book. Данные утверждения записываются следующим образом:

ex:Person rdf:type rdfs:Class.
ex:Book rdf:type rdfs:Class.
ex:author rdf:type rdf:Property.
ex:author rdfs:range ex:Person.
ex:author rdfs:domain ex:Book.

Любое свойство может иметь ноль, один или более одного диапазона допустимых значений.

Свойство *rdfs:range* может быть использовано для индикации величины свойства (явного описания его типа) как литерала:

ex:age rdf:type rdf:Property.

ex:age rdfs:range xsd:integer.

Например:

```
<rdf:Property rdf:ID="registeredTo">
  <rdfs:domain rdf:resource="#MotorVehicle"/>
  <rdfs:range rdf:resource="#Person"/>
</rdf:Property>
```

Пример использования rang и domain для описания двух свойств ex:registeredTo и ex:rearSeatLegRoom, и нового класса ex:Person, с явным описанием типа *xsd:integer:* используется свойство *ex:registeredTo* любыми экземплярами класса и его значением может быть любой экземпляр класса ex:Person; свойство ex:rearSeatLegRoom применяется только для экземпляров класса ex:PassengerVehicle и определяет количество сантиметров для ног в пространстве заднего сидения:

```
<rdf:Property rdf:ID="registeredTo">
  <rdfs:domain rdf:resource="#MotorVehicle"/>
  <rdfs:range rdf:resource="#Person"/>
</rdf:Property>
<rdf:Property rdf:ID="rearSeatLegRoom">
  <rdfs:domain rdf:resource="#PassengerVehicle"/>
  <rdfs:range rdf:resource="&xsd;integer"/>
</rdf:Property>
<rdfs:Class rdf:ID="Person"/>
<rdfs:Datatype rdf:about="&xsd;integer"/>
```

Схема RDF содержит также группу свойств для аннотаций схем, введения комментариев, меток и т.п. Для введения меток используется свойство *rdfs:lable*, а для введения комментариев: *rdfs:comment*.

9.4. Особенности RDFS

Нет разницы между классами и экземплярами:

ex:Person rdf:type rdfs:Class.

ex:man rdf:type ex:Person.

ex:Petr rdf:type ex:man.

Свойства могут иметь свойства:

ex:hasDaughter rdfs:subPropertyOf ex:hasChild.

ex:hasDaughter rdf:type ex:FamilyProperty.

Нет различия между конструкторами языка и онтологическим словарем, поэтому конструкторы могут применяться к самим себе и друг к другу:

<type, range, Class>

<Property, type, Class>

<type, subPropertyOf, subClassOf>

Домен самих rdfs:range и rdfs:domain – это rdf:Property.

Недостатки RDFS:

- RDFS – слишком бедный язык и недостаточно описывает ресурсы.
- Отсутствуют ограничения, локализующие область определения и область значения. Нельзя сказать, что областью значения свойства hasChild, когда оно применяется к людям, является человек, а когда к слонам, – слон.
- Отсутствуют ограничения существования/кардинальности. Нельзя сказать, что все экземпляры класса людей имеют мать, которая также относится к этому классу, или что человек имеет в точности двух родителей.
- Отсутствуют транзитивные, инверсные или симметричные свойства. Нельзя сказать, что isPartOf является транзитивным свойством, а hasPart является инверсным к isPartOf (обратный).
- Трудно обеспечить поддержку рассуждений.

9.5. Синтаксические особенности нотации 3 (Notation3, N3)

Существует более краткий и, как считается, более простой способ записи моделей RDF (не в спецификации XML), так называемая ***Нотация 3 или N3***, которая разработана с целью быть понятней человеку.

В данной нотации каждое высказывание записывается отдельной строкой и заканчивается символом точки. Строятся высказывания по правилу:

подлежащее сказуемое дополнение . // разделенные пробелами

Термины, которые не имеют дальнейшего определения (литералы), записываются в кавычках, а понятия, которые могут быть раскрыты дополнительно, представленные URI, заключаются в угловые скобки.

После слова *@prefix* записывается объявление пространства имен:

@prefix dc: <http://purl.org/dc/elements/1.1/>.

Для широко известных пространств имен используются следующие префиксы:

@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.

@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.

@prefix owl: < http://www.w3.org/2002/07/owl#>.

@prefix ex: <http://www.example.org/> (или http://www.example.com/).

@prefix xsd: <http://www.w3.org/2001/XMLSchema#>.

После объявления пространства имен оно используется в виде префикса.

Пространство имен. Можно использовать пространство имен по умолчанию:

@prefix :< http://www.example.com/>.

После объявления оно используется в виде пустого префикса, т.е. перед именем ставится двоеточие. # показывает, что вся первая часть URI адреса пропущена.

Триплет в N3 можно записать в виде:

<# Jack > <#child> <#Sandra>.

<#Sandra><#name>Sandra.

При записи нескольких утверждений об одном и том же субъекте свойства субъекта перечисляются через точку с запятой “;”. При записи нескольких утверждений с одним и тем же субъектом и предикатом объекты перечисляются через запятую:

< #Sandra ><#name>Sandra; <# child> <#Kevin>, <#Jack>.

Если используется какой-то термин, которому не присвоен URI, то можно воспользоваться утверждением: «есть некто по имени…». Для этого ставится символ подчеркивания, двоеточие и метка, затем предикат и его значение:

_:a1 http://xyz.org/#name “John”.

Такие конструкции называются анонимными узлами, так как в них нет URI.

Если в предложении говорится об объектах, которые не имеют идентификатора, но имеют свойства, то такие сущности можно представить с помощью квадратных скобок. Квадратные скобки декларируют, что существует нечто, обладающее указанными свойствами, но не дают вам возможности сослаться на это нечто из другого места, неважно, в этом же или в другом документе:

<#pat> <#child> [<#age> 4] , [<#age> 3].

RDF-модель, записанная в стандартном XML-виде (RDF/XML):

```
<rdf:RDF
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:dc="http://purl.org/dc/elements/1.1/">
  <rdf:Description
rdf:about="http://en.wikipedia.org/wiki/Tony_Benn">
    <dc:title>Tony Benn</dc:title>
    <dc:publisher>Wikipedia</dc:publisher>
  </rdf:Description>
</rdf:RDF>
```

В N3 ее можно записать так:

```
@prefix dc: <http://purl.org/dc/elements/1.1/>.
<http://en.wikipedia.org/wiki/Tony_Benn>
  dc:title "Tony Benn" ;
  dc:publisher "Wikipedia" .
```

На языке N3 можно записать и RDFS.

```
:Woman a rdfs:Class;
rdfs:subClassOf :Person .
:sister a rdf:Property.
:sister rdfs:domain :Person;
rdfs:range :Woman.
:Dog rdf:type rdfs:Class.
:Fido a:Dog.
:name rdf:type rdf:Property.
:Fido :name “Fido”.
:Dog rdfs:subclassOf :Animal.
```

Если в созданном словаре существует термин, эквивалент которому уже есть в другом словаре, то рекомендуется использовать этот другой термин или указать свойство эквивалентности между двумя терминами "=":

:Woman = foo:FemaleAdult .

:Title a rdf:Property; = dc:title .

Есть еще более простой синтаксис для RDF, чем N3. Это N-триплеты, в которых не используются префиксы:

http://www.w3.org/TR/2004/REC-rdf-primer20040210/#typedliterals6.4

http://www.w3.org/2000/10/swap/Examples.

Недостатки RDF. Семантический веб основывается на принципе минимальных ограничений: чем меньше правил, тем лучше. Это значит, что семантический веб предоставляет большую свободу высказываний. Благодаря открытости и расширяемости RDF любой пользователь может сказать что угодно (т.е. фиксировать произвольное утверждение), о чем угодно (т.е. о любом ресурсе сети и не только). RDF не запрещает делать бессмысленных утверждений или утверждений, не согласующихся с другими, т.е. нет никакой гарантии целостности и непротиворечивости RDF-описаний.

Но при этом RDF-описания должны соответствовать хотя бы минимальным ограничениям. Например, не должны быть перепутаны люди и документы с их именами и названиями.

Неправильно писать:

http://example.org/ dc:creator “Bob”.

Так как литеральная строка не может создать документ, правильно следует написать:

<http:// example.org/> dc:creator _:b.

_:b foaf:name “Bob”,

т.е. этот example.org был создан кем-то, чье имя “Bob”.

9.6. RSS 1.0: RDF Site Summary

RSS – технология, предназначенная для описания новой информации на сайтах, блогах, описания лент новостей пользователю в удобном для него виде. Для этого используются специальные про-

граммы-агрегаторы или онлайн сервисы, такие как Google Reader, Яндекс Новости и другие.

RSS – документ в определенном формате, чаще всего XML, который может содержать, например, последние новости, снабженные датами, ссылками, фамилиями журналистов и другими данными. Все новые и новые сайты обзаводятся RSS-лентой с новостями, описанием обновлений, недавно добавленных статей.

В разных версиях расшифровка аббревиатуры была разной:

- *Rich Site Summary (RSS 0.9x)* – обогащенная сводка сайта;
- *RDF Site Summary (RSS 0.9 и 1.0)* – сводка сайта с применением инфраструктуры описания ресурсов;
- *Really Simple Syndication (RSS 2.x)* – очень простой сбор сводной информации (не использует RDF и является более простой).

RSS 1.0 является самым широко распространенным RDF приложением в Интернете, например, http://www.w3.org/.

Существуют специальные программы для представления информации, полученной через RSS, в удобном для восприятия виде.

Все RSS приложения на сегодня можно разделить на три группы:

1. *On-line агрегаторы-сайты*, такие как Meerkat and NewsIsFree, http://lenta.yandex.ru/, – это каналы, имеющие тысячи источников, каждый разделен по отдельным темам <items>, а затем все соединены в одну большую группу. На них можно найти последние новости о чем-то конкретном.

2. *Desktop Readers* – специальные утилиты, позволяющие пользователям подписываться на сотни каналов рассылок со своего рабочего стола. Readers обычно обновляют каналы один раз в час (FeedReader, FeedDemon и др.).

3. *Скрипты* позволяют веб-мастерам включать содержимое обновление других сайтов на свои собственные. Первоначально RSS создавалась для этих целей.

Для того чтобы сделать RSS канал на собственном сайте, необходимо сформировать веб-страницу по правилам XML или RDF:

```
<?xml version="1.0" encoding="utf-8"?>
<rdf:RDF xmlns="http://purl.org/rss/1.0/"
   xmlns:dc="http://purl.org/dc/elements/1.1/"
```

```
 xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
<channel rdf:about="http://www.w3.org/2000/08/w3c-synd/ home.
rss">
 <title>The World Wide Web Consortium</title>
 <description> Leading the Web to its Full Potential...
</description>
 <link>http://www.w3.org/</link>
 <dc:date>2002-10-28T08:07:21Z</dc:date>
 <items>
  <rdf:Seq>
 <rdf:li rdf:resource="http://www.w3.org/News/2002#item 164"/>
 <rdf:li rdf:resource="http://www.w3.org/News/2002#item 168"/>
 <rdf:li rdf:resource="http://www.w3.org/News/2002#item 167"/>
  </rdf:Seq>
 </items>
item rdf:about="http://www.w3.org/News/2002#item164">
 <title>User Guidelines </title>
 <description>17 October 2002: </description>
 <link>http://www.w3.org/News/2002#item164</link>
 <dc:date>2002-10-17</dc:date>
</item>
<item rdf:about="http://www.w3.org/News/2002#item168">
 <title>Working Draft </title>
 <description>25 October 2002:</description>
 <link>http://www.w3.org/News/2002#item168</link>
 <dc:date>2002-10-25</dc:date>
</item>
<item rdf:about="http://www.w3.org/News/2002#item167">
 <title> Last Call Working Drafts Published</title>
 <description>24 October 2002: </description>
 <link>http://www.w3.org/News/2002#item167</link>
 <dc:date>2002-10-24</dc:date>
</item>
</channel>
</rdf:RDF>
```

ТЕМА 10

ЯЗЫК OWL

10.1. Особенности языка OWL.

10.2. Структура OWL-онтологии.

10.3. Свойства в OWL.

10.1. Особенности языка OWL

Одним из серии языков, используемых для описания онтологий, является язык ***DAML*** (http://www.daml.org/) – это язык, созданный агентством DARPA (http://www.darpa.mil/) на основе RDF как язык онтологий и логического вывода. Он глубже развивает понятия свойств и классов. Например, добавляется конструкция инверсности: ***daml:inverseOfProperty***, используя которую, можно сказать, что одно свойство является инверсией другого:

:hasName daml:inverseOf :isNameOf.

:Sean :hasName "Sean".

"Sean" :isNameOf :Sean.

Еще одной полезной конструкций языка DAML является конструкция ***daml:UnambiguousProperty***. Если некоторые свойства принадлежат к классу daml:UnambiguousProperty, это обозначает, что, если объекты этого свойства одинаковы, то субъекты эквивалентны. То есть обладать данным свойством может только один конкретный объект, свойство определяет объект:

foaf:mbox rdf:type daml:UnambiguousProperty.

:x foaf:mbox .

:y foaf:mbox .

Означает, что:

:x daml:equivalentTo :y.

«Логический вывод» представляет собой возможность выводить новые данные из данных, которые уже известны. Например, свойство DAML "equivalentTo" позволяет делать выводы из уже известных свойств.

Например, можно сказать про автомобиль, что:

:MyCar de:macht "160KW"

В немецкий процессор семантического веба может быть встроен термин *:macht*, тогда как английский процессор этого не поймет, строка об эквивалентности позволит сделать процессору логический вывод:

de:macht daml:equivalentTo en:power.

Используя данное свойство, можно легко объединять БД, указав в каком-либо документе RDF, например, что “PersonName” одной базы данных эквивалентно “Name” другой базы.

Спецификация обмена онтологиями DAML+OIL, объединив 2 проекта DAML и OIL project (European Commission Ontology Inference Layer), включает в онтологию DAML+OIL:

- заголовки (hearders);
- элементы классов (class elements);
- элементы свойств (property elements);
- экземпляры (instances).

OWL (Ontology Web Language) – язык описания онтологий, расширяющий и объединяющий возможности XML, RDF, RDF Schema и DAML+OIL (с 2004-го года является рекомендацией W3C).

OWL обеспечивает более полную машинную обработку, чем указанные выше языки.

Онтология – описание классов объектов, их свойств и взаимоотношений для определенной предметной области.

OWL – это фактически словарь, расширяющий набор терминов, определенных RDFS, добавляющий большие возможности для описания свойств и классов, например, отношения между классами (непересекаемость, равенство, кардинальность (только один)), перечисляемые классы, больше типов свойств, характеристик свойств (симметрия).

OWL имеет три диалекта (подмножества терминов) в порядке возрастания выразительности: OWL Lite, OWL DL и OWL Full:

1. OWL Lite имеет наименьшую выразительную мощность из всех, но для решения простых задач его может быть достаточно, разработан для представления классификационных иерархий и простых ограничений.

2. OWL DL обладает выразительной мощностью, эквивалентно дескрипторной логике (разрешимой части логики предикатов первого

порядка). Для большинства задач, встречающихся при проектировании онтологии, выразительности вполне достаточно. Этот диалект имеет два важных свойства: полноту (все заключения являются вычислимыми) и разрешимость (все вычисления завершаются в конечное время). В частности, в OWL DL классу запрещено быть экземпляром.

3. OWL Full – наиболее выразительный диалект, эквивалентен RDF. Однако при использовании OWL Full нет никаких гарантий по вычислимости заключений.

Каждый из этих диалектов (кроме Lite) является расширением предыдущего.

10.2. Структура OWL-онтологии

Любая онтология имеет заголовок и тело. OWL-документы (OWL-онтологии) являются RDF-документами, т.е. имеют корневой элемент rdf:RDF.

Заголовок содержит набор объявлений XML namespace, заключенный в открывающийся тег <rdf:RDF>.

Часто в определении типа документа создаются сущности в декларации типа документа (DOCTYPE), представляющие префиксы онтологий:

```
<!DOCTYPE rdf:RDF [
   <!ENTITY vin "http://www.w3.org/TR/2004/REC-owl-guide-20040210/wine#" >
   <!ENTITY food "http://www.w3.org/TR/2004/REC-owl-guide-20040210/food#" > ]>
<rdf:RDF
   xmlns ="&vin;"
   xmlns:vin ="&vin;"
   xml:base ="&vin;"
   xmlns:food="&food;"
   xmlns:owl ="http://www.w3.org/2002/07/owl#"
   xmlns:rdf ="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
   xmlns:xsd ="http://www.w3.org/2001/XMLSchema#">
```

Можно обратиться к элементу, используя пространство имен food:cook, и записать, обращаясь к атрибуту: &food;bread, и это было

бы расшифровано как http://www.w3.org/TR/2004/REC-owl-guide-20040210/food#bread, и всегда можно записать ресурс, используя полный URI.

После определений пространств имен ставится тег owl:Ontology, группирующий метаданные о создаваемой онтологии, включающий информацию о самой онтологии (версия, примечание), импортируемых онтологиях. OWL должен позволять собирать информацию из распределенных источников, так как семантическая сеть по определению распределена, поэтому OWL осуществляет прямой импорт информации из других онтологий:

<owl:Ontology rdf:about="">

<rdfs:comment>An example OWL ontology</rdfs:comment>

<owl:priorVersion rdf:resource="http://www.w3.org/TR/2003/PR-owl-guide-20031215/wine"/>

<owl:imports rdf:resource="http://www.w3.org/TR/2004/REC-owl-guide-20040210/food"/>

<rdfs:label>Wine Ontology</rdfs:label>

Rdf:about="" ссылается на создаваемую онтологию, если значение атрибута "", то названием онтологии служит базовый URI.

Импорт другой онтологии обеспечивает механизм включения, перенося в текущую онтологию весь набор утверждений из другой онтологии.

За заголовком следует *тело онтологии*, содержащее описание классов, свойств и экземпляров.

В OWL введен новый термин ***owl:Class***. Не все классы диалектов DL и Lite являются rdfs классами. Owl:Class является подклассом rdfs:Class.

Каждый индивид OWL является членом класса owl:Thing, т.е. каждый определенный пользователем класс автоматически является подклассом owl:Thing. Наиболее фундаментальное понятие в какой-то области должно соответствовать классам, находящимся в корне различных деревьев иерархии понятий (таксономических деревьев).

Специфичные для данной области корневые классы определяются простым объявлением класса:

<owl:Class rdf:ID="Регион"/>

<owl:Class rdf:ID="ПродуктПитания"/>

После того как был объявлен класс с использованием идентификатора ID, можно ссылаться на класс *Регион*, используя синтаксис *#Регион* внутри данного документа, и *http://www.w3.org/TR/2004/REC-owl-guide-20040210/wine#Регион* в других онтологиях.

Использование синтаксиса *rdf:about=”&ont;#x”* – является ключевым в создании распределенной онтологии.

OWL также определяет пустой класс owl:nothing. Каждый класс является суперклассом owl:nothing.

Для организации классов в таксономию используется свойство rdfs:subClassOf. ***Таксономия*** – основное отношение категоризации или классификации (subsumption) – IS-A. Если X – подкласс Y, то каждый представитель X – также представитель Y:

```
<owl:Class rdf:ID="Напиток">
  <rdfs:subClassOf rdf:resource="#ПродуктПитания" />
  ...
</owl:Class>
```

Определение класса состоит из двух частей: указание названия или ссылка и список ограничений. Каждое из содержащихся в определении класса выражений ограничивает (уточняет) свойства представителей определенного класса. Быть подклассом некоторого другого класса тоже является ограничением.

Эквивалентность классов может быть определена с использованием элемента *owl:equivalentClass*:

```
<owl:Class rdf:ID=”faculty”>
   <owl:equivqlentClass rdf:resource=”#acdemicStaffMember”/>
</owl:Class>
```

Элемент *OWL:disjointWith* (разъединять) сообщает, что классы не пересекаются:

```
<owl:Class rdf:about=”associateProfessor”>
<owl:disjointWith rdf:resource=”#assistantProfessor”/>
<owl:disjointWith rdf:resource=”#profesor”/>
</owl:Class>
```

В OWL установлено большое различие между классом и индивидом. ***Класс*** – это просто название и совокупность свойств, которые описывают набор индивидов. ***Индивиды*** – это реальные объекты. Для

определения индивида достаточно объявить его членом какого-то класса:

```
<owl:Class rdf:ID="Виноград">
 ...
</owl:Class>
<owl:Class rdf:ID="ВинныйВиноград">
 <rdfs:subClassOf rdf:resource="&food;Виноград" />
</owl:Class>
<ВинныйВиноград rdf:ID="ВиноградКаберне Совиньон" />
```

10.3. Свойства в OWL

В OWL выделяют два типа свойств:

1) *свойства-значения*, т.е. свойства-типы данных, отношения между представителями классов и RDF-литералами, или типами данных, определяемых XML Schema (owl:DatatypeProperty);

2) *свойства-объекты*, отношения между представителями двух классов (owl:ObjectProperty), связывают объекты друг с другом. Оба класса свойств являются подклассами класса rdf:Property.

Простейший пример аксиомы свойства:

```
<owl:ObjectProperty rdf:ID="hasParent"/>
```

Все, что постулирует эта аксиома, – существование некоторого свойства "hasParent", связывающего экземпляры owl:Thing друг с другом.

При определении свойств существует множество способов их ограничить. Например:

- определить домен и range (диапазон значений);
- указать, что свойство является подсвойством другого свойства.

Можно использовать и более сложные ограничения:

```
<owl:ObjectProperty rdf:ID="сделаноИзВинограда">
 <rdfs:domain rdf:resource="#Вино"/>
 <rdfs:range rdf:resource="#Виноград"/>
</owl:ObjectProperty>
<owl:DatatypeProperty rdf:ID=”age”>
<rdfs:range rdf:resource =http://www.w3.org/201/XMLSchema#
nonNegativeInteger/>
```

</owl:DatatypeProperty>

OWL использует большинство встроенных типов XML Schema. Ссылки на эти типы осуществляются посредством URI для типов http://www.w3.org/2001/XMLSchema:

xsd:string xsd:normalizedString xsd:boolean
xsd:decimal xsd:float xsd:double
xsd:integer xsd:nonNegativeInteger xsd:positiveInteger
xsd:nonPositiveInteger xsd:negativeInteger
xsd:long xsd:int xsd:short xsd:byte
xsd:unsignedLong xsd:unsignedInt xsd:unsignedShort xsd:unsignedByte
xsd:hexBinary xsd:base64Binary
xsd:dateTime xsd:time xsd:date xsd:gYearMonth
xsd:gYear xsd:gMonthDay xsd:gDay xsd:gMonth
xsd:anyURI xsd:token xsd:language
xsd:NMTOKEN xsd:Name xsd:NCName

Все данные типы и тип rdfs:Literal формируют OWL. Все диалекты поддерживают типы xsd:integer xsd:string. Другие встроенные типы XML Schema могут использоваться в OWL Full:

```
<owl:Class rdf:ID="ГодВинтажа" />
<owl:DatatypeProperty rdf:ID="год">
  <rdfs:domain rdf:resource="#ГодВинтажа" />
  <rdfs:range  rdf:resource="&xsd;positiveInteger"/>
</owl:DatatypeProperty>
```

Свойство год связывает *ГодВинтажа* со значениями положительного целого числа.

Пример, когда домен разрешает быть расположенным в регионе чему угодно, включая сами регионы:

```
<owl:ObjectProperty rdf:ID="расположенВ">
  ...
  <rdfs:domain
rdf:resource="http://www.w3.org/2002/07/owl#Thing" />
  <rdfs:range rdf:resource="#Регион" />
</owl:ObjectProperty>
```

Свойства так же, как классы, могут быть организованы в иерархию:

```
<owl:ObjectProperty rdf:ID="обладаетХарактеристикойВина">
  <rdfs:domain rdf:resource="#Вино" />
```

<rdfs:range rdf:resource="#ХарактеристикаВина" />
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="имеетЦвет">
<rdfs:subPropertyOf
rdf:resource="#обладаетХарактеристикойВина" />
<rdfs:range rdf:resource="#ЦветВина" />
...
</owl:ObjectProperty>

Кроме того, можно установить отношения между свойствами:

- *owl:equivalentProperty* (определяет эквивалентное свойство);
- *owl:inverseOf* (определяет обратное свойство). Область определения и область значения могут наследоваться от инверсного свойства (перестановкой области определения и области значений):

<owl:ObjectProperty rdf:ID=”teaches”>
<rdfs:domain rdf:resource=”#academicStaffMember”/>
<rdfs:range rdf:resource=”#course”/>
<owl:inverseOf rdf:resource=”#isTaughtBy”/>
</owl:ObjectProperty>

- *owl:FunctionalProperty* (определяет однозначное свойство – однозначное отображение домена свойства на диапазон).

Если свойство P помечено как функциональное, тогда для любого x, y и z: P(x,y) и P(x,z) предполагают y = z:

<owl:ObjectProperty rdf:ID="имеетПроизводителя" />
<owl:ObjectProperty rdf:ID="производитВино">
<rdf:type rdf:resource="&owl;InverseFunctionalProperty" />
<owl:inverseOf rdf:resource="#имеетПроизводителя" />
</owl:ObjectProperty>

Рассматривайте элементы диапазона обратно функциональному свойству как определение ключевого поля при построении баз данных. Owl:InverseFunctional предполагает, что элементы диапазона обеспечивают уникальный идентификатор для каждого элемента домена.

В OWL Full мы можем пометить DatatypeProperty как обратно функциональное. Это позволяет нам идентифицировать строку как уникальный ключ. В OWL DL литералы (строковые значения) отделены от owl:Thing, и поэтому OWL DL не разрешает применять InverseFunctional к DatatypeProperty.

КОНТРОЛЬНЫЕ ВОПРОСЫ ПО КУРСУ

1. Раскройте понятие интеллектуальных информационных систем.
2. В чем заключается концепция «семантического веба»?
3. Назовите назначение XML.
4. В чем преимущества языка XML над языком HTML?
5. Перечислите способы отображения XML-документа.
6. Укажите два способа создания языка, основанного на XML.
7. Раскройте понятие и структуру XML документа.
8. Назовите инструкции, литералы, комментарии XML документа.
9. Укажите корневой элемент и содержание элемента.
10. В чем заключается раздел CDATA?
11. Опишите синтаксис задания пустого элемента.
12. Какие вы знаете типы элемента, способы задания атрибутов элемента и содержание элемента XML-документа?
13. Отобразите XML-документ с использованием каскадной таблицы стилей.
14. В чем заключается использование пространства имен?
15. Опишите расширенное имя элемента, идентификатор пространства имен.
16. Используйте пространство имен для вставки элементов HTML в XML-документ.
17. Опишите корректный XML документ.
18. Назовите валидный XML-документ и преимущества использования валидных XML-документов.
19. Перечислите два способа объявления типа документа DTD.
20. Какие есть объявления и формы записи типа элемента?
21. Опишите задание дочернего и смешанного содержимого элемента.
22. Перечислите способы объявления атрибутов.
23. Опишите маркерный тип атрибутов в DTD.
24. В чем заключается перечисляемый тип атрибутов?
25. Какие существуют режимы атрибутов?
26. Назовите объявление внешней и полной схем DTD.
27. Перечислите типы сущностей.
28. Опишите объявление общих внутренних разбираемых примитивов.
29. Охарактеризуйте объявление общих внешних разбираемых примитивов.
30. Назовите объявление общего внешнего неразбираемого примитива.

31. Опишите объявление нотаций.

32. Назовите объявление параметрического внутреннего разбираемого примитива.

33. Охарактеризуйте объявление параметрического внешнего разбираемого примитива.

34. Опишите способы вставки ссылок на примитив в XML-документ.

35. Укажите преимущества объявления типа документа с помощью схем над объявлением с помощью DTD.

36. Назовите основы синтаксиса XML Schema.

37. Опишите объектную модель документа XML DOM.

38. Перечислите основные узлы, доступные в модели DOM для языка XML.

39. Какие есть общие свойства узлов DOM языка XML?

40. Опишите использование узла Document и основные свойства узла Document.

41. В чем заключается связывание XML-документа с HTML-страницей через фрагмент данных, создаваемый HTML-элементом с именем XML?

42. Назовите свойства и методы, предоставляемые объектом NodeList.

43. В чем заключается извлечение символьных данных элемента? Назовите свойства и методы, поддерживаемые узлами Text.

44. Перечислите методы, поддерживаемые узлами Element языка XML.

45. Опишите доступ и отображение значений атрибутов в XML-документе. Укажите свойства и методы, предоставляемые групповым объектом NamedNodeMap.

46. Какие есть преимущества использования для отображения XML-документов XSL-таблицы стилей над таблицей CSS?

47. Отобразите XML-документ с использованием XSL-таблицы стилей.

48. Перечислите виды XML-элементов, включаемые в XSL-шаблон.

49. В чем заключается использование элемента value-of XSL?

50. Назовите пример использования атрибута match элемента template XSL-шаблона.

51. Назовите пример использования элемента for-each XSL.

52. В чем заключается использование атрибута select элементов for-each и value-of?

53. Назовите способы выражения семантики данных. Раскройте понятие метаданных.

54. Назовите основные идеи Semantic Web.

55. Охарактеризуйте модель данных RDF. Раскройте понятие RDF-графа.

56. Опишите типы узлов RDF. Раскройте понятие универсального идентификатора ресурса URI.

57. Какие есть два типа глобальных имен URI?

58. Перечислите типы сущностей RDF и доступные списочные типы.

59. Опишите формат XML для хранения RDF и особенности синтаксиса RDF/XML.

60. В чем заключается использование атрибутов и тегов для определения предиката в RDF/XML?

61. Опишите синтаксис записи отношения между ресурсом и литералом RDF/XML.

62. Опишите синтаксис записи отношения между ресурсами RDF/XML.

63. Укажите два способа записи типа сущности RDF/XML.

64. Опишите синтаксис записи списочных типов RDF/XML.

65. Использование пространств имен в RDF/XML. Пространство имен, используемое по умолчанию.

66. Укажите основные и дополнительные свойства словаря Dublin core и способы использования специальных тегов meta в HTML документе?

67. Охарактеризуйте язык описания словарей для RDF-RDF Schema. Назовите термины, вводимые RDFS.

68. Какие есть особенности, преимущества и недостатки RDFS?

69. Укажите классы RDFS. В чем заключаются особенности экстенсионального и интенсионального объявления класса?

70. Опишите способы объявления свойств в RDFS.

71. Охарактеризуйте синтаксические особенности нотации N3.

72. В чем заключается использование префиксов в нотации N3?

73. Назовите технология RSS 1.0.

74. Опишите язык OWL. Раскройе понятие онтологии.

75. Охарактеризуйте структуру и диалекты OWL-онтологии.

76. В чем заключается использование сущностей и пространств имен в синтаксисе OWL?

77. Опишите содержание заголовка OWL-онтологии.

78. В чем заключается использование классов и индивидов в OWL?

79. Укажите два типа свойств языка OWL.

СПИСОК ЛИТЕРАТУРЫ

1. Башмаков А. И. Интеллектуальные информационные технологии [Текст] / А. И. Башмаков, И. А. Башмаков. – М. : МГТУ им. Баумана, 2005. – 304 с.
2. Гаврилова Т. А. Использование онтологий в системах управления знаниями [Электронный ресурс] / Т. А. Гаврилова. – Режим доступа : http://kmsoft.ru/publications/library/authors/use_ontology_in_suz.html
3. Гапанюк Ю. Е. Введение в XML-технологии [Текст] : учебное пособие / Ю. Е. Гапанюк, Г. И. Ревунков. – М. : МГТУ им. Баумана, 2009.
4. Дейтел П. Как программировать на XML [Текст] / П. Дейтел, П. Садху, Х. Дейтел. – М. : Бином. Лаборатория знаний, 2008. – 944 с.
5. Понятия и абстрактный синтаксис Resource Description Framework (RDF) [Электронный ресурс]. – Режим доступа : http://www.w3.org/TR/rdf-concepts/
6. Рэй Э. Изучаем XML [Текст] : пер. с англ. / Э. Рэй. – СПб. : Символ-Плюс, 2001. – 408 с.
7. Руководство Web Ontology Language (OWL) [Электронный ресурс]. – Режим доступа : http://www.w3.org/2004/OWL/
8. Хабибуллин И. Ш. Самоучитель XML [Текст] / И. Ш. Хабибуллин. – СПб. : БХВ-Петербург, 2003. – 336 с.
9. Шеперд Д. Освой самостоятельно XML за 21 день [Текст] : 2-е изд. / Д. Шеперд. – М. : Издательский дом «Вильямс», 2002. – 432 с.
10. Янг М. XML. Шаг за шагом [Текст] : практ. пособие / М. Янг. – М. : Изд-во Эком, 2000. – 384 с.
11. RDF Schema язык описания словарей RDF [Электронный ресурс]. – Режим доступа : http://www.w3.org/TF/rdf-schema/
12. The Extensible Markup Language (XML) [Электронный ресурс]. – Режим доступа : http://www.w3.org/XML/

СОДЕРЖАНИЕ

Навчальне видання

ХАЙРОВА Ніна Феліксівна
ПЕТРАСОВА Світлана Валентинівна

ІНФОРМАЦІЙНІ ІНТЕЛЕКТУАЛЬНІ СИСТЕМИ ТА СЕМАНТИЧНИЙ ВЕБ

Навчальний посібник
з курсу «Сучасні інтелектуальні системи автоматичної обробки інформації» для студентів спеціальностей «Прикладна лінгвістика» та «Прикладна інформатика»

Російською мовою

Відповідальний за випуск проф. *Н.В. Шаронова*
Роботу до видання рекомендував проф. *О.В. Горілий*
Редактор *О.І. Шпільова*

План 2015 р., поз. 8

Підп.до друку 17.06.2016 р. Формат 60х84 1/16. Папір офісний.
Riso-друк. Гарнітура Таймс. Ум. друк. арк. 9,88 Наклад 100 прим.
Зам. № 120 Ціна договірна

Видавець і виготовлювач
Видавничий центр НТУ «ХПІ».
вул. Фрунзе, 21, м. Харків, 61002
Свідоцтво суб'єкта видавничої справи ДК № 3657 від 24.12.2009 р.