Learning XML

Erik T. Ray

O'REILLY®

Изучаем XML

Эрик Рэй

Санкт-Петербург
2001

Эрик Рэй

Изучаем XML

Перевод С. Маккавеева

Главный редактор	*А. Галунов*
Зав. редакцией	*Н. Макарова*
Научный редактор	*Е. Морозов*
Редактор	*В. Овчинников*
Корректура	*И. Воробьева*
Верстка	*Н. Гриценко*

Рэй Э.
Изучаем XML. – Пер. с англ. – СПб: Символ-Плюс, 2001. – 408 с., ил.
ISBN 5-93286-023-5

Данное издание посвящено расширяемому языку разметки XML – перспективному и мощному инструменту, обеспечивающему гибкий способ создания самодокументируемых документов и совместного использования как формата, так и данных в Интернете. Рассмотрены история, современное состояние и задачи XML, фундаментальные вопросы. Для начинающих разработчиков излагаются основы техники создания документов XML, понятия элементов, атрибутов, сущностей и пространств имен XML. Профессионалам адресованы сложные вопросы – трансформации, моделирование документов, тонкая настройка шаблонов, XML-программирование, использование ссылок и каскадных таблиц стилей.

В книге на примерах показано, как эффективно использовать XML путем форматирования и преобразования XML-документов с тем, чтобы они могли обрабатываться броузерами, базами данных и т. д. Материал сопровождается ссылками на реальные проекты. В приложениях описаны ресурсы Интернета, книги и стандарты, имеющие отношение к XML. В книгу включен глоссарий.

ISBN 5-93286-023-5
ISBN 0-596-00046-4 (англ)

Издательство «Символ-Плюс». 193148, Санкт-Петербург, ул. Пинегина, 4,
тел. (812) 324-5353, edit@symbol.ru. Лицензия ЛП N 000054 от 25.12.98.

Подписано в печать 21.08.2001. Формат 70x100 1/16 . Бумага офсетная.
Печать офсетная. Объем 25,5 печ. л. Тираж 3000 экз. Заказ N
Отпечатано с диапозитивов в ФГУП «Печатный Двор» им. А. М. Горького
Министерства РФ по делам печати, телерадиовещания
и средств массовых коммуникаций.
197110, Санкт-Петербург, Чкаловский пр., 15.

Оглавление

Предисловие

Со времени своего появления в конце 90-х годов XML (Extensible Markup Language – расширяемый язык разметки) стал источником бурного потока новых акронимов, стандартов и правил, заставивших часть Интернет-сообщества задуматься, а действительно ли все это так необходимо. В конце концов, HTML использовался уже в течение ряда лет, способствуя созданию совершенно новой экономики и культуры, так стоит ли менять хорошую вещь? На самом деле, XML создан не для того, чтобы заменить собой то, что уже имеется в Сети, он призван заложить более прочные и гибкие основы. Это не имеющий предшествующих аналогов проект ряда организаций и фирм, направленный на создание информационной структуры XXI века, лишь намеком на которую явился HTML.

Чтобы понять важность этого проекта, мы должны расстаться с некоторыми мифами. Во-первых, несмотря на свое название, XML не является языком разметки. Это, скорее, средство для создания, формирования и применения языка разметки. Это обстоятельство должно разъяснить и второе существующее заблуждение – о том, что XML заменит собой HTML. В действительности, HTML должен оказаться поглощенным XML и стать XHTML – более четкой версией себя самого. И это лишь начало, поскольку XML сделает возможным создание сотен новых языков для описания всех типов приложений и документов.

Процесс стандартизации будет иметь важное значение в ходе этой информационной революции. Собственно XML является попыткой установить порядок в неконтролируемой разработке конкурирующих технологий и патентованных языков, которая грозит расколоть Сеть. XML создает игровую площадку, обеспечивающую прекрасное взаимодействие структурированной информации с приложениями, максимально увеличивая ее доступность и не жертвуя при этом богатством выразительности.

Энтузиазм, с которым XML был воспринят сообществом Интернета, открыл двери для многочисленных родственных стандартов. В число новых друзей XML вошли таблицы стилей для вывода и преобразования, надежные методы для связывания ресурсов, средства обработки и запроса данных, средства проверки ошибок и принудительного структурирования, а также множество средств разработки. Эти новые приложения обеспечат XML долгую и плодотворную жизнь в качестве

предпочтительного инструментария для работы со структурированной информацией.

Конечно, XML пока молод, и многие его братья и сестры еще не вышли из младенческого возраста. Некоторые обсуждаемые в данной книге вопросы являются почти умозрительными, поскольку их спецификации все еще представляют собой рабочие проекты. Однако всегда полезно как можно раньше вступить в игру, чтобы не быть застигнутым врасплох позднее. А уж тем, кто участвует в разработках для Интернета или в управлении информацией, просто необходимо знать XML.

Эта книга призвана дать читателю возможность с высоты птичьего полета взглянуть на ландшафт XML, который начинает обретать формы. Чтобы получить наибольшую пользу от книги, следует иметь некоторое знакомство со структурированной разметкой, например, с HTML или TEX, а также с такими понятиями World Wide Web, как гипертекстовые ссылки и представление данных. Однако освоить концепции XML могут не только разработчики. Мы сосредоточимся на теории и практике создания документов, не слишком вникая в подробности, касающиеся разработки приложений или приобретения программных инструментов. Сложности программирования с использованием XML оставлены для других книг, а быстрые изменения, происходящие в отрасли, гарантируют, что нечего и надеяться угнаться за новейшим программным обеспечением для XML. Тем не менее, представленная здесь информация может послужить хорошей отправной точкой для движения в любом выбранном направлении работы с XML.

Что есть в этой книге

Книга состоит из следующих глав.

Глава 1 «Введение» является обзором XML и некоторых наиболее распространенных случаев его применения. Это «трамплин» для оставшейся части книги, знакомящий с основными понятиями, детальное разъяснение которых будет дано в последующих главах.

Глава 2 «Разметка и основные понятия» описывает базовый синтаксис XML, закладывая основы понимания приложений и технологий XML.

Глава 3 «Соединение ресурсов с помощью ссылок» показывает, как создавать простые ссылки между документами и ресурсами, что является важной стороной XML.

Глава 4 «Представление: создание конечного продукта» вводит понятие таблиц стилей с помощью языка Cascading Style Sheets – каскадных таблиц стилей.

Глава 5 «Модели документов: более высокий уровень контроля» рассказывает о DTD – определениях типа документа и знакомит со схемой XML. Это основные технологии, обеспечивающие качество и полноту документов.

Глава 6 «Трансформация: изменение назначения документов» показывает, как создать таблицу стилей для преобразования XML из одного вида в другой.

Глава 7 «Поддержка многоязычности» знакомит с возможностями XML, обеспечивающими доступность документов и их локализацию, включая Unicode, кодировки символов и поддержку языков.

Глава 8 «XML-программирование» знакомит с созданием программного обеспечения для обработки XML.

Кроме того, есть два приложения и глоссарий.

Приложение A «Ресурсы» содержит библиографию ресурсов, из которых можно больше узнать о XML.

Приложение B «Таксономия стандартов» перечисляет технологии, связанные с XML.

Глоссарий разъясняет термины, используемые в книге.

Обозначения, используемые в книге

Элементам книги иногда придается особый внешний вид, чтобы отделить их от обычного текста. Вот как они выглядят:

Курсив

Используется для ссылок на книги и статьи, команд, адресов электронной почты, URL, выделения текста и первого упоминания терминов.

`Моноширинный шрифт`

Применяется в литералах, константах, листингах программ и XML-разметке.

`Моноширинный курсив`

Служит для выделения переменных и параметров, которые должны быть заменены значениями, введенными пользователем.

`Моноширинный полужирный шрифт`

Предназначен для выделения части кода, обсуждаемой в данный момент.

Примеры

Примеры, имеющиеся в этой книге, можно бесплатно загрузить с сайта книги: *http://www.oreilly.com/catalog/learnxml*.

Комментарии и вопросы

Информация в данной книге проверена по мере сил автора и группы подготовки, однако какие-то возможности могли измениться (а в текст могли вкрасться ошибки!). Пожалуйста, сообщите о найденных ошибках или своих предложениях для будущих изданий по адресу:

O'Reilly & Associates
101 Morris Street
Sebastopol, CA 95472
800-998-9938 (из США или Канады)
707-829-0515 (международные или местные)
707-829-0104 (FAX)

Для этой книги поддерживается веб-страница, на которой представлены ошибки, примеры и дополнительные сведения. Ее можно найти по адресу:

http://www.oreilly.com/catalog/learnxml

Для того чтобы задать технический вопрос или сообщить свои комментарии по поводу этой книги, напишите по адресу:

bookquestions@oreilly.com

Подписаться на один или более наших списков рассылки можно на сайте:

http://elists.oreilly.com

Дополнительные сведения о наших книгах, конференциях, программном обеспечении и сети O'Reilly Network можно получить на сайте:

http://www.oreilly.com

Благодарности

Эта книга никогда не появилась бы на свет без помощи первоклассных редакторов Энди Орэма (Andy Oram), Лори Петрицки (Laurie Petrycki), Джона Познера (John Posner) и Эллен Сивер (Ellen Siever); производственного персонала, а именно Клина Гормэна (Colleen Gorman), Эмили Квил (Emily Quill) и Эллен Траутмэн-Цейг (Ellen Troutman-Zaig); блестящих рецензентов Джеффа Лиггетта (Jeff Liggett), Джона Юделла (Jon Udell), Анны-Марии Вадува (Anne-Marie Vaduva), Энди Орэма (Andy Oram), Норма Уэлша (Norm Walsh) и Джессики П. Хекман (Jessica P. Hekman); моих уважаемых коллег Шерила Авруча (Sheryl Avruch), Клиффа Дайера (Cliff Dyer), Джейсона Макинтоша

(Jason McIntosh), Ленни Мюлнера (Lenny Muellner), Бена Солтера (Benn Salter), Майка Сьерры (Mike Sierra) и Фрэнка Уиллисона (Frank Willison); благодарю также Стэфена Спейнаура (Stephen Spainhour) за помощь в написании приложений и Криса Мейдена (Chris Maden) за энтузиазм и знания, понадобившиеся для того, чтобы начать этот проект.

Я бесконечно благодарен своей жене Джаннин Бестайн (Jeannine Bestine) за терпение и поддержку; своей семье (мама1: Берджит (Birgit), мама2: Хелен (Helen), папа1: Эл (Al), папа2: Буч (Butch), а также Эду (Ed), Элтону (Elton), Жан-Полю (Jon-Paul), бабушке и деду Бестайн (Bestine), Мэр (Mare), Маргарет (Margaret), Джин (Gene) и Лиэнн (Lianne)) за непрерывные потоки любви и еды; моим домашним птичкам Estero, Zagnut, Milkyway, Snickers, Punji, Kitkat и Chi Chu; моим большим друзьям Derrick Arnelle, Mr. J. David Curran, Sarah Demb, Chris «800» Gernon, John Grigsby, Andy Grosser, Lisa Musiker, Benn «Nietzsche» Salter и Greg «Mitochondrion» Travis; вдохновлявшим меня героям и знаменитостям: Лори Андерсону (Laurie Anderson), Айзеку Азимову (Isaac Asimov), Вернеру фон Брауну (Wernher von Braun), Джеймсу Берку (James Burke), Альберту Эйнштейну (Albert Einstein), Махатме Ганди (Mahatma Gandhi), Чаку Джонсу (Chuck Jones), Миямото Мусаси (Miyamoto Musashi), Ральфу Нейдеру (Ralph Nader), Райнеру Марии Рильке (Rainer Maria Rilke) и Оскару Уайлду (Oscar Wilde); особая благодарность горчице Weber, сделавшей мои сэндвичи чрезвычайно вкусными.

1

- *Что такое XML?*
- *Истоки XML*
- *Задачи XML*
- *XML сегодня*
- *Создание документов*
- *Просмотр документов XML*
- *Тестирование XML*
- *Трансформация*

Введение

XML (Extensible Markup Language – расширяемый язык разметки) является инструментарием для хранения данных, конфигурируемым транспортным средством для информации любого рода, развивающимся и открытым стандартом, воспринятым всеми – от банкира до вебмастера. Всего за несколько лет он в равной степени захватил воображение экспертов в области технологий и специалистов в промышленности. В чем же причина его успеха?

Краткий перечень возможностей XML говорит сам за себя:

- XML позволяет хранить и упорядочивать информацию почти любого рода в формате, приспособленном к потребностям пользователя.
- Будучи открытым стандартом, XML не связан с судьбой какой-либо отдельной компании или с конкретным программным обеспечением.
- Используя Unicode в качестве стандартного набора символов, XML поддерживает внушительное число различных систем письма и символов, от скандинавских рунических символов до китайских идеографов Хань.
- XML предоставляет несколько способов проверки качества документа путем применения синтаксических правил, внутренней проверки ссылок, сравнения с моделями документов и типов данных.
- Благодаря простому и понятному синтаксису, а также однозначной структуре, XML легко читается и анализируется, как человеком, так и программами.
- XML легко сочетается с таблицами стилей для создания документов, оформленных в любом требуемом стиле. Чистота информационной структуры не служит помехой изменениям оформления.

И все это появляется в тот момент, когда мир готов перейти на новый уровень связанности. Объем доступной нам информации поражает, но доступ к ней может быть затруднен существующими технологическими ограничениями. Многие предприятия борются за то, чтобы быть представленными в Интернете и открыть каналы обмена данными, но встречают трудности из-за несовместимости с уже существующими у них системами хранения и обработки данных. Поддержка программ с открытым исходным кодом привела к взрыву в разработке программного обеспечения, обострившему потребность в единообразном интерфейсе для обмена информацией. XML был разработан для решения всех этих проблем и должен стать смазкой в колесах информационной инфраструктуры.

В этой главе ландшафт XML представлен под широким углом. Рассказано, как работает XML и как взаимодействуют между собой все части. Это послужит основой для будущих глав, в которых более подробно рассматриваются детали таблиц стилей, преобразований и моделей документов. Дойдя до конца книги, читатель получит хорошее представление о том, как XML помогает решать задачи управления информацией, и о том, куда следует двинуться дальше.

Что такое XML?

Ответить на это вопрос не просто. На первом уровне XML является протоколом хранения и управления информацией. На следующем уровне – это семейство технологий, с помощью которых можно осуществлять все – от оформления документов до фильтрации данных. И, наконец, на самом высоком уровне – это философия обработки информации, которая призвана обеспечить максимальную полезность и гибкость данных путем придания им наиболее чистой и структурированной формы. Полное понимание XML затрагивает все эти уровни.

Начнем с анализа первого уровня XML: способа хранения информации и управления ей с помощью разметки. Эта общая схема упаковки данных является необходимой основой для следующего уровня, на котором XML становится действительно увлекательным: сопутствующие технологии, такие как таблицы стилей, преобразования и языки разметки «сделай сам». Понимание основ разметки, документов и представления поможет читателю с наибольшей выгодой использовать XML и предоставляемые им средства.

Разметка

Обратите внимание, что, несмотря на свое название, XML не является языком разметки: это набор правил для создания языков разметки. Каково же точное определение языка разметки? *Разметка* (*markup*) – это информация, добавляемая в документ и в некоторых отношениях

усиливающая его смысл, поскольку определяет его части и их соотношение друг с другом. Например, при чтении газеты можно различать статьи по промежуткам между ними и их положению на странице, а также по применению различных шрифтов для названий и заголовков. Разметка действует сходным образом, но вместо пустых промежутков использует символы. *Язык разметки* (*markup language*) – это способ именования и выделения частей документа при помощи набора символов, размещаемых в тексте документа.

Разметка важна в электронных документах, потому что они обрабатываются компьютерными программами. Если в документе нет меток или границ, программа не будет знать, как отличить один участок текста от другого. Фактически, программе придется работать со всем документом как с единым целым, что резко ограничивает возможность каких-либо интересных обработок содержимого. Газета без промежутков между статьями и с единственным стилем текста была бы большим и неинтересным куском текста. Возможно, кому-то и удалось бы определить, где кончается одна статья и начинается другая, но это потребовало бы большой работы. Компьютерная программа не смогла бы сделать даже этого, поскольку обладает лишь элементарными возможностями поиска по шаблону.

К счастью, разметка дает решение таких проблем. Вот пример разметки XML, встроенной во фрагмент текста:

```
<message>
  <exclamation>Hello, world!</exclamation>
  <paragraph>XML is <emphasis>fun</emphasis> and
    <emphasis>easy</emphasis> to use.
    <graphic fileref="smiley_face.pict"/></paragraph>
</message>
```

В этом отрывке имеются следующие символы разметки, или *теги*:

- Теги `<message>` и `</message>` отмечают начальную и конечную точку всего фрагмента XML.
- Теги `<exclamation>` и `</exclamation>` окружают текст `Hello, world!`.
- Теги `<paragraph>` и `</paragraph>` окружают более крупную область текста и тегов.
- Несколько тегов `<emphasis>` и `</emphasis>` помечают отдельные слова.
- Тег `<graphic fileref="smiley_face.pict"/>` помечает место в тексте для вставки графического изображения.

По этому примеру можно понять систему: некоторые теги действуют как ограничители, помечая начало и конец области, а другие помечают некоторую точку в тексте. Даже простой документ, представленный здесь, содержит много информации:

Границы

Участок текста начинается в одном месте и заканчивается в другом. Теги `<message>` и `</message>` определяют начало и конец совокупности текста и разметки, помеченной как `message`.

Роли

Каково назначение некоторой области текста в документе? Здесь теги `<paragraph>` и `</paragraph>` помечают некоторый текст как абзац, а не список, заголовок или лимерик.

Позиции

Участок текста чему-то предшествует и зачем-то следует. Абзац появляется после текста, отмеченного тегами `<exclamation>` (восклицание), который, вероятно, и будет выводиться соответствующим образом.

Вложенность

Текст `fun` находится внутри элемента `<emphasis>`, расположенного внутри `<paragraph>`, который вложен в `<message>`. Такая структура элементов учитывается программным обеспечением обработки XML, которое может по-разному обрабатывать содержимое в зависимости от расположения последнего. Например, заголовок может выводиться шрифтом разного размера в зависимости от того, к чему он относится – к газете или к статье.

Связи

Участок текста может быть связан с ресурсом, находящимся в другом месте. Например, тег `<graphic fileref="smiley_face.pict"/>` создает связь (ссылку) между фрагментом XML и файлом с именем *smiley_ face.pict*. Цель состоит в том, чтобы импортировать из файла графические данные и отобразить их в этом фрагменте.

В XML информационную ценность представляют как содержание документа, так и его разметка. Разметка позволяет программам определять функции и границы частей документа. Содержание (обычный текст) – это то, что важно для читателя, но оно должно быть представлено осмысленным образом. XML помогает компьютеру форматировать документ так, чтобы он был более понятен человеку.

Документы

Услышав слово *документ*, многие, вероятно, представляют себе последовательность слов, разбитых на параграфы, разделы и главы, образующие вместе некоторую запись для чтения человеком, например, книгу, статью, очерк. Но в XML документ представляет собой нечто большее: это основная единица информации XML, составленная из элементов и разметки в упорядоченном пакете. Документ может содержать текст, например, рассказ или статью, но это не обязательно. С

тем же успехом он может содержать базу данных с числами или некую абстрактную структуру, представляющую молекулу или уравнение. На практике, одним из наиболее перспективных приложений XML является обеспечение формата для обмена данными между приложениями. Запомните, определение документа XML может быть значительно шире нашего традиционного понимания документа.

Документ состоит из частей, называемых *элементами*. Элементы вкладываются друг в друга, как маленькие ящички – в более крупные, придавая форму и маркируя содержимое документа. На верхнем уровне находится элемент, называемый *элементом документа* (*document element*) или *корневым элементом* (*root element*), в котором содержатся остальные элементы. Ниже приведены короткие примеры документов.

Язык математической разметки (MathML) кодирует уравнения. Физикам хорошо известно уравнение закона тяготения Ньютона: $F = GMm / r^2$. Следующий документ представляет это уравнение.

```
<?xml version="1.0"?>
<math xmlns="http://www.w3.org/TR/REC-MathML/">
  <mi>F</mi>
  <mo>=</mo>
  <mi>G</mi>
  <mo>&InvisibleTimes;</mo>
  <mfrac>
    <mrow>
      <mi>M</mi>
      <mo>&InvisibleTimes;</mo>
      <mi>m</mi>
    </mrow>
    <apply>
      <power>
      <mi>r</mi>
      <mn>2</mn>
      </power>
    </apply>
  </mfrac>
</math>
```

Обратите внимание: одно приложение может использовать эти данные для наглядного вывода уравнения, а другое – для того, чтобы решать его. Это признак мощи XML.

В документах XML можно также хранить графические изображения. Для вычерчивания контурных рисунков переменного размера применяется язык масштабируемой векторной графики – Scalable Vector Graphics (SVG). В следующем документе определен рисунок с тремя фигурами (прямоугольником, окружностью и многоугольником):

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg
```

```
    PUBLIC "-//W3C//DTD SVG 20001102//EN"
    "http://www.w3.org/TR/2000/CR-SVG-20001102/DTD/svg-20001102.dtd">
<svg>
  <desc>Three shapes</desc>
  <rect fill="green" x="1cm" y="1cm" width="3cm" height="3cm"/>
  <circle fill="red" cx="3cm" cy="2cm" r="4cm"/>
  <polygon fill="blue" points="110,160 50,300 180,290"/>
</svg>
```

Эти примеры основываются на уже определенных языках разметки, но пользователь, имеющий специализированное приложение, может создать собственный язык, основанный на XML. В следующем документе названия элементов выдуманы (что совершенно допустимо в XML) для кодирования простого сообщения:

```
<?xml version="1.0"?>
<message>
  <exclamation>Hello, world!</exclamation>
  <paragraph>XML is <emphasis>fun</emphasis> and
    <emphasis>easy</emphasis> to use.
    <graphic fileref="smiley_face.pict"/></paragraph>
</message>
```

Документ – не то же самое, что файл. *Файл* – это набор данных, рассматриваемый операционной системой как единое целое. Это называется *физической* структурой. Документ XML может располагаться в одном или нескольких файлах, причем некоторые из них могут находиться на разных машинах. В XML используется специальная разметка для интеграции содержимого разных файлов в один объект, который можно охарактеризовать как *логическую* структуру. Благодаря тому, что документ не ограничен одним файлом, XML позволяет создавать документ из частей, которые могут располагаться где угодно.

Моделирование документов

Мы уже говорили, что XML не является языком, а служит спецификацией для создания языков разметки. Как же создавать язык, основываясь на XML? Есть два способа. Первый из них называется *свободным форматом* (*freeform*) XML. В этом режиме действуют некоторые минимальные правила образования и применения тегов, но последним при этом можно давать любые имена, которые могут следовать в любом порядке. Это, в некотором роде, похоже на выдумывание собственных слов, но с использованием знаков препинания. Если документ удовлетворяет минимуму правил XML, он называется *корректным* (*well-formed*) и квалифицируется как «хороший XML».

Однако полезность свободного формата XML ограничена. Поскольку нет ограничений на применяемые теги, то нет и спецификации, которая служила бы инструкцией по использованию такого языка. Конеч-

но, можно постараться употреблять теги единообразно, но существует вероятность ошибки при вводе имени тега, а программное обеспечение воспримет его как часть пользовательского языка свободного формата. Едва ли удастся обнаружить ошибку раньше, чем программа считает данные и некорректно обработает их, заставив разработчика ломать голову в поисках ошибки. Контроль качества может быть устроен гораздо лучше.

XML, к счастью, предоставляет пользователю способ описать язык вполне определенным образом. Он называется *моделированием документов* (*document modelling*), поскольку требует создания спецификации, излагающей правила, которым должен отвечать документ. В сущности, это модель, с которой разработчик может сравнить конкретный документ (называемый *экземпляром документа* (*document instance*)), для того чтобы узнать, действительно ли в документе представлен определенный им язык, т. е. проверить, соответствует ли документ «своей» спецификации языка. Эта проверка носит название *проверки действительности или семантической корректности* (*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.

Чаще всего документы моделируются с помощью *определения типа документа* (*Document Type Definition, DTD*). Это набор правил, или *объявлений* (*declarations*), указывающих, какие теги можно использовать и что они могут содержать. В начале документа располагается ссылка на DTD, которая объявляет желание автора проверить действительность документа.

Развивается также новый стандарт моделирования документов, известный как *схема XML* (*XML Scheme*). Схемы используют фрагменты XML, называемые *шаблонами* (*templates*), чтобы показать, как должен выглядеть документ. Преимущество применения схем в том, что они сами являются разновидностью XML, поэтому редактировать их можно с помощью тех же самых средств, которые используются для редактирования документов. Они также предоставляют более строгую проверку типов данных, позволяющую обнаруживать ошибки как в применении тегов, так и в содержании.

Язык разметки, созданный с помощью правил XML, называется *приложением* (*application*) XML, или, иногда, *типом документа*. Существуют сотни общедоступных приложений XML для кодирования чего угодно – от пьес или стихов до содержимого каталогов. Вполне возможно, что читатель найдет такое, которое будет удовлетворять его потребностям, но если это не удастся, он сможет создать свое собственное приложение.

Представление

Представление (*presentation*) описывает, как должен выглядеть документ, подготовленный для просмотра человеком. Например, в приведенном выше примере «Hello, world!» можно потребовать, чтобы содержимое тега <exclamation> выводилось на печать шрифтом Times Roman размером 32 пункта. Такая информация о стиле не является частью документа XML. Автор XML назначает стили в отдельном месте, обычно в документе, называемом *таблицей стилей* (*stylesheet*).

Можно разработать такой язык разметки, в котором объединены информация о стиле и «чистая» разметка. Одним из примеров служит HTML. В нем правильно определены такие элементы, как заголовки (тег <title>) и параграфы (тег <p>), но в то же время применяются, например, теги <i> (использовать курсив) и <pre> (отключить удаление пробельных символов), которые описывают внешний вид, а не функцию составляющих частей документа. В XML такие теги не одобряются.

Может показаться, что это не очень существенно, но такое разделение стиля и значения представляется важным в XML. Документы, содержащие стилистическую разметку, трудно переназначить или преобразовать в новые форматы. Например, представьте себе документ, в котором фразы на иностранном языке помечены как выводимые курсивом, и таким же образом помечены просто выделенные фразы, например:

```
<programlisting><example>Goethe once said, <i>Lieben ist wie
Sauerkraut</i>. I <i>really</i> agree with that
statement.</example></programlisting>
```

Теперь, для того чтобы выделить полужирным шрифтом все выделенные фразы, но сохранить курсив для фраз на иностранном языке, придется вручную изменить все теги <i>, которые представляют выделенный текст. Правильнее было бы помечать слова или фразы в зависимости от их смысла, например:

```
<programlisting><example>Goethe once said, <foreignphrase>Lieben
ist wie Sauerkraut</foreignphrase>. I <emphasis>really</emphasis>
agree with that statement.</example></programlisting>
```

Теперь информация о стиле для каждого тега содержится в таблице стилей, а не включается в тег. Чтобы изменить вывод выделенных фраз с курсива на полужирный шрифт, надо отредактировать только одну строку в таблице стилей, а не искать и изменять каждый тег. Основной принцип, лежащий в основе этой философии, состоит в том, что можно иметь столько различных тегов, сколько видов информации есть в документе. В языке, основывающемся на стилях, таком как HTML, предоставляется меньше возможностей для выбора, а различные типы данных могут отображаться одним и тем же стилем.

Хранение стилей вне документа увеличивает возможности представления, поскольку разработчик не связан каким-то одним словарем стилей. Возможность использования с документом любого количества таблиц стилей позволяет «на лету» создавать новые версии. Один и тот же документ можно просматривать на настольном компьютере, выводить на печать, просматривать на переносном устройстве или даже читать вслух с помощью синтезатора речи, не изменяя при этом ни буквы в исходном тексте, а просто применяя другую таблицу стилей.

Обработка

Когда компьютерная программа считывает документ XML и что-то с ним делает, это называется *обработкой* (*processing*) XML. Поэтому всякую программу, которая может читать и обрабатывать документы XML, называют *процессором XML* (*XML processor*). Примерами процессоров XML служат программы контроля действительности документов, веб-броузеры, редакторы XML, информационные системы и системы архивации – возможности бесконечны.

Самый простой процессор XML читает XML-документы и преобразует их во внутреннее представление, которое могут использовать другие программы или процедуры. Такой процессор называется *синтаксическим анализатором* (*parser*) и является важной составной частью любой программы обработки XML. Анализатор превращает поток символов из файлов в осмысленные участки информации, называемые *лексемами* (*tokens*). Лексемы интерпретируются как события, управляющие программой, или образуют в памяти временную структуру (*представление в виде дерева*), с которой может работать программа.

На рис. 1.1 показаны три стадии синтаксического анализа документа XML. Анализатор считывает XML из файлов на компьютере (1). Затем он транслирует поток символов в лексемы, которые могут быть обработаны приложением (2). Кроме того, лексемы могут быть использованы для создания в памяти абстрактного представления документа – дерева объектов (3).

Анализаторы XML известны своей строгостью. Если какой-либо символ разметки находится не на месте или тег записан прописными буквами, а его следовало записать строчными буквами, анализатор должен сообщить об ошибке. Обычно такая ошибка прекращает всякую последующую обработку. Лишь после исправления всех синтаксических ошибок документ признается корректным (well-formed), и обработка может быть продолжена.

Это может показаться излишним. Почему бы анализатору не пренебречь мелкими проблемами, такими как отсутствие замыкающего тега или неверное использование прописных букв в имени тега? В конце концов, анализаторы HTML являются известным примером синтаксической терпимости. Веб-броузеры обычно игнорируют или исправля-

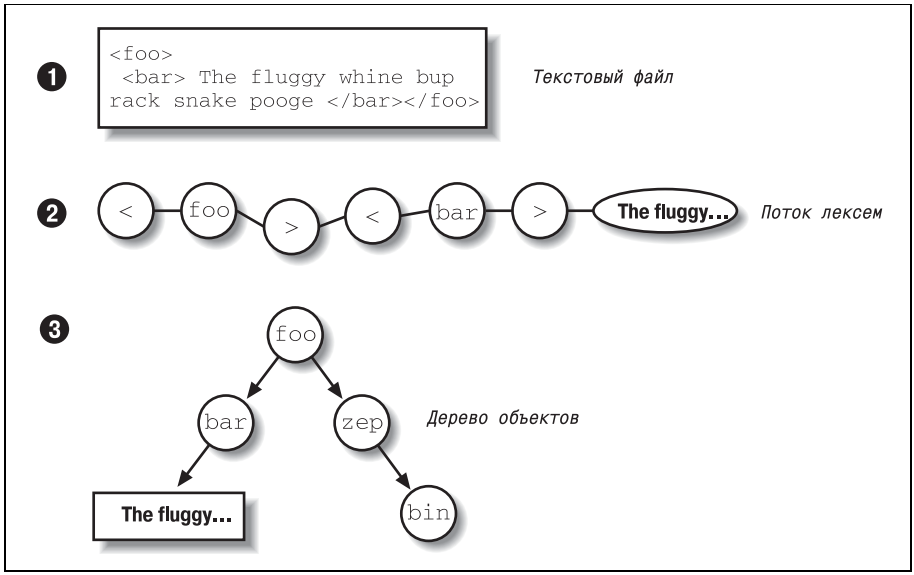


Рис. 1.1. *Три этапа синтаксического анализа документа XML*

ют ошибки без лишнего шума, в результате чего авторы HTML так ничего и не узнают. Однако причиной, по которой XML так строг, является стремление сделать поведение анализаторов XML, обрабатывающих документ, настолько предсказуемым, насколько это возможно.

Это может показаться противоречащим интуиции, но если поразмыслить, вполне оправдано. XML предназначен для повсеместного применения и должен всегда работать одинаково. Если анализатор не предупреждает о синтаксической ошибке, то его вполне можно сравнить с механиком, забывшим гаечный ключ в моторе, и позднее, при обработке документа с помощью другой программы, пользователю придется потрудиться, доискиваясь до ошибки. Вот почему придирчивый анализ XML сокращает количество неудач и проблем с совместимостью в дальнейшем.

Истоки XML

Двадцатый век стал в истории человечества беспрецедентным веком информации. Университеты производят большое количество книг и статей, содержание средств массовой информации богаче, чем когда-либо ранее, и даже исследование космического пространства дает больше данных о вселенной, чем мы в состоянии обработать. Организация всех этих знаний представляет нетривиальную задачу.

Первые электронные форматы были в большей мере ориентированы на описание того, как должна выглядеть информация (представлением),

чем на ее структуру и смысл. Два ранних языка форматирования – troff и TEX – делали фантастическую работу по форматированию печатных документов, но в них не было заложено никакого представления о структуре. Поэтому документы можно было лишь просматривать на экране или выводить на печать. Не так просто было писать программы для поиска и извлечения информации, создавать электронные перекрестные ссылки или использовать документы в разных приложениях.

Общая система кодирования (*generic coding*), использующая описательные теги вместо кодов форматирования, в конечном счете, решила эту проблему. Первой организацией, серьезно исследовавшей эту идею, стала ассоциация Graphic Communications Association (GCA). В конце 1960-х годов в проекте «GenCode» были разработаны способы кодирования различных типов документов с помощью общих тегов и сборки документов из отдельных частей.

Следующим крупным достижением стал обобщенный язык разметки Generalized Markup Language (GML) – проект, осуществленный в IBM. Разработчики GML Чарльз Гольдфарб (Charles Goldfarb), Эдвард Мошер (Edward Mosher) и Рэймонд Лори (Raymond Lorie)[1] рассматривали его как решение проблемы кодирования документов, используемых в различных информационных подсистемах. В документах, закодированных с применением этого языка разметки, можно было осуществлять редактирование, форматирование и поиск при помощи различных программ, поскольку его теги основывались на содержании. IBM, которая публикует очень много технических руководств, широко использовала GML, доказав жизнеспособность общей системы кодирования.

SGML и HTML

Вдохновленный успехом GML, Комитет по обработке информации Национального института стандартизации США (ANSI) собрал группу разработчиков для разработки стандартного языка описания текста на основе GML, поставив Гольдфарба во главе проекта. Был учтен также опыт комитета GCA GenCode. В конце 1970-х и начале 1980-х группа опубликовала рабочие проекты и в конечном итоге создала *стандартный обобщенный язык разметки* (*Standard Generalized Markup Language, SGML*), явившийся кандидатом на промышленный стандарт (GCA 101-1983). Он вскоре был принят Министерством обороны и Налоговой службой США.

В последующие годы SGML получил большое распространение. В 1985 году в Великобритании начались встречи Международной группы

[1] Забавный факт: акроним GML оказывается также инициалами этих трех изобретателей.

пользователей SGML. Вместе с CGA они пропагандировали использование SGML в Европе и Северной Америке. Расширяя границы распространения SGML, проект «Electronic Manuscript» ассоциации американских издателей (AAP) поощрял применение SGML для кодирования универсальных документов, таких как книги и журналы. Министерство обороны США разработало приложения для SGML в группе компьютеризованной поддержки закупок и тылового обеспечения (Computer-Aided Acquisition and Logistic Support, CALS),[1] в том числе популярный тип документа для форматирования таблиц, получивший название CALS Tables. И, наконец, увенчав это успешное начало, Международная организация по стандартизации ISO ратифицировала стандарт SGML (ISO 8879:1986).

SGML проектировался как гибкая и всеохватывающая схема кодировки. Как и XML, это, по сути, инструментарий для разработки специализированных языков разметки. Но SGML значительно больше, чем XML, с более свободным синтаксисом и массой таинственных параметров. Он настолько гибок, что программное обеспечение для его обработки является сложным и дорогим, и его полезность ограничивается крупными организациями, которые могут позволить себе как программное обеспечение, так и затраты на сопровождение сложных сред SGML.[2]

Революция в обобщенном кодировании произошла в начале 1990-х, когда Тим Бернерс-Ли (Tim Berners-Lee) и Андерс Берглунд (Anders Berglund), сотрудники европейской лаборатории физики элементарных частиц CERN, разработали язык разметки гипертекста Hypertext Markup Language (HTML). CERN участвовала в проекте SGML с начала 1980-х годов, когда Берглунд разработал издательскую систему для тестирования SGML. Бернерс-Ли и Берглунд создали тип документа SGML для гипертекстовых документов, который был компактным и эффективным. Этот язык разметки сделал простым написание программного обеспечения и еще проще – кодирование документов. HTML вышел за пределы лаборатории и отправился завоевывать мир.

Однако в некоторых отношениях HTML был шагом в обратном направлении. Чтобы достичь простоты, которая сделала его действительно полезным, пришлось пожертвовать некоторыми принципами обобщенного кодирования. Например, для любых задач использовался один тип документа, из-за чего приходилось производить перегрузку

---

[1] CALS – это стратегия достижения эффективного создания, обмена и использования цифровых данных для систем вооружения и оборудования Министерства обороны США. Более подробную информацию можно найти на странице CALS по адресу *http://navysgml.dt.navy.mil/cals.html*. – *Примеч. науч. ред.*

[2] Существует несколько свободных приложений для обработки SGML, например, программа Jade Джеймса Кларка и несколько основанных на ней приложений. – *Примеч. науч. ред.*

тегов вместо того, чтобы определять теги для конкретных задач. Во-вторых, многие теги определяли исключительно представление. Упрощенная структура затрудняла обнаружение конца одного раздела и начала другого. Сегодня многие документы в кодировке HTML настолько интенсивно используют теги, ориентированные на представление, что их сложно использовать в разных приложениях. Тем не менее, HTML стал блестящим шагом в развитии Web и гигантским скачком в развитии для языков разметки, поскольку, благодаря ему, весь мир заинтересовался электронной документацией и ссылками между документами.

Стремление к идеалам обобщенной схемы кодирования породило попытки адаптировать SGML к Web, или, скорее, адаптировать Web к SGML, но это оказалось слишком сложным. SGML слишком велик, чтобы втиснуть его в маленький броузер. Требовался не такой крупный язык, который сохранил бы обобщенность SGML, и это послужило причиной появления расширяемого языка разметки – Extensible Markup Language (XML).

Задачи XML

Подстегиваемая неудовлетворенностью существующими стандартными и нестандартными форматами, группа компаний и организаций, назвавшая себя World Wide Web Consortium (W3C), начала в середине 1990-х годов работу над языком разметки, который объединил бы в себе гибкость SGML и простоту HTML. Философия разработки XML, которой придерживаются члены W3C, была воплощена в нескольких принципах, описываемых в следующих разделах.

Языки разметки для конкретных применений

XML не определяет какие-либо элементы разметки, а указывает, как создавать собственные элементы. Иными словами, вместо создания элементов общего назначения, скажем, параграфа, в надежде, что он может подойти в любой ситуации, разработчики XML возложили эту задачу на пользователя. Поэтому, если он хочет, чтобы элемент назывался `<segmentedlist>`, `<chapter>` или `<rocketship>`, это его право. Создавайте собственный язык разметки, чтобы выразить свою информацию наилучшим образом. Или воспользуйтесь набором тегов, созданных кем-то другим.

Это означает, что может существовать неограниченное число языков разметки и должен существовать способ предотвратить крах программ, которые пытаются все их читать. Наряду с предоставляемой свободой творчества, XML предполагает выполнение авторами некоторых правил. Если собственные элементы создаются ими определенным образом и при этом соблюдаются все синтаксические правила, то

документ считается корректным, и его может читать любой процессор XML. Так и получается, что и волки сыты, и овцы целы.

Однозначная структура

Правила XML жестко трактуют все, что касается структуры. Документ должен иметь такую разметку, чтобы не существовало двух способов интерпретации имен, порядка и иерархии элементов. Это значительно уменьшает число ошибок и сложность кода. Программы не должны ни о чем «догадываться» или пытаться исправлять синтаксические ошибки, как это часто делают броузеры HTML, потому что различные процессоры XML должны давать одинаковые результаты.

Конечно, это затрудняет написание хорошей разметки XML. Автор должен проверить синтаксис документа с помощью синтаксического анализатора, чтобы оставить минимум ошибок для последующих обрабатывающих программ, обеспечить защиту целостности данных и непротиворечивость результатов.

Помимо элементарной синтаксической проверки, можно создать собственные правила, которым должен соответствовать внешний вид документа. Спецификацией структуры документа является DTD. Схема XML может ограничивать типы данных, допустимых внутри элементов (например, даты, числа, имена). Возможности проверки ошибок и контроля структуры очень велики.

Хранение представления в другом месте

Для того чтобы достичь наибольшей гибкости формата вывода документа, следует размещать информацию о стиле за пределами документа и хранить ее отдельно. XML позволяет делать это с помощью таблиц стилей, содержащих сведения о форматировании. У такого подхода есть много преимуществ:

- Можно использовать одни и те же стили в разных документах.
- Если требуется изменить параметры стиля, то можно сделать исправление в одном месте, и это повлияет на все документы.
- Можно менять таблицы стилей в зависимости от задачи, например, печатать с помощью одной таблицы и создавать веб-страницы с помощью другой.
- Содержание и структура документа остаются целыми вне зависимости от изменения представления. Невозможно испортить документ, делая что-то с его представлением.
- Содержимое документа не загромождается словами, относящимися к стилю (изменение шрифта, разрядка, цвет и т. д.). В результате документ легче читать и сопровождать.

- В отсутствие информации о стиле можно выбирать имена, точно отражающие назначение элементов, а не называть их согласно тому, как они должны выглядеть. Это облегчает редактирование и преобразование документов.

Простота использования

Чтобы получить широкое распространение, XML должен быть простым. Люди не хотят изучать сложную систему лишь для того, чтобы создать документ. XML интуитивен, элегантен и легко читается. Он позволяет разработать собственный язык разметки, удовлетворяющий нескольким логичным правилам. Это маленькое подмножество SGML, из которого выкинуто то, что не требуется большинству пользователей.

Простота также благоприятствует разработке приложений. Если писать программы, обрабатывающие файлы XML, просто, появится большее число более дешевых программ. Правила XML строги, но они делают усилия по анализу и обработке файлов более предсказуемыми, а потому значительно более легкими.

Простота ведет к изобилию. Можете представлять себе XML как своего рода ДНК для многих различных способов выражения информации. Таблицы стилей для определения внешнего вида и преобразования структуры документа можно писать на языке под названием XSL, основанном на XML. Еще одним видом XML являются схемы моделирования документов. Такая вездесущесть означает, что одни и те же средства можно применять для редактирования и обработки во многих различных технологиях.

Интенсивная проверка ошибок

Некоторые языки разметки столь «снисходительны» в отношении синтаксиса, что ошибки проходят незамеченными. А если в файле накапливаются ошибки, то он начинает вести себя не так, как ожидалось: его внешний вид в броузере становится непредсказуемым, информация может быть потеряна, а программы могут вести себя неожиданным образом или завершаться аварийно при попытке открыть этот файл.

В спецификации XML сказано, что документ не является корректным, если не удовлетворяет ряду минимальных синтаксических требований. Анализатор XML является верным сторожевым псом, охраняющим документы от проникновения ошибок. Он проверяет правильность написания имен элементов, обеспечивает герметичность границ, сообщает об объектах, находящихся не на своем месте, и испорченных ссылках. Кому-то может не нравиться такая строгость и, для того чтобы довести документ до стандарта, возможно, придется потрудиться,

но в итоге это стоит усилий. Долговечность и полезность документа будут обеспечены.

XML сегодня

Сейчас XML является официальной рекомендацией и его текущая версия 1.0. Самую последнюю рекомендацию можно найти на веб-сайте World Wide Web Consortium по адресу *http://www.w3.org/TR/REC-xml-19980210*.[1]

Дела этой молодой технологии развиваются хорошо. Интерес к ней проявляется в росте количества сопутствующих технологий, возникающих, как грибы после дождя, в степени внимания, проявляемого средствами массовой информации (см. приложение А «Ресурсы»), а также в быстром росте имеющихся приложений и инструментов XML.

Скорость развития XML захватывает дух, и, для того чтобы уследить за самыми яркими из многочисленных звезд в этой галактике, необходимо прилежно трудиться. Чтобы помочь читателю сориентироваться в происходящем, ниже описывается процесс стандартизации и его результаты.

Процесс стандартизации

Стандарты – это смазка в ступицах колес торговли и коммуникаций. Они описывают все – от форматов документов до сетевых протоколов. Самый хороший стандарт – тот, который является *открытым*, т. е. не управляется какой-либо одной компанией и не является ее собственностью. Другой тип стандарта, *патентованный* (*proprietary*), может изменяться без уведомления, не требует участия общества и часто приносит выгоду владельцу патента, получающему плату за лицензию и вводящему произвольные ограничения.

К счастью, XML является открытым стандартом, которым управляет W3C, придав ему статус формальной рекомендации – документа, описывающего, что собой представляет данный стандарт и как должен использоваться. Однако рекомендация не является строго обязательной. Отсутствует процесс сертификации, нет лицензионного соглашения и нет никакого наказания, кроме общественного осуждения, для тех, кто реализует XML некорректно.

В некотором смысле рекомендация, не накладывающая жестких ограничений, полезна, поскольку проведение стандартов в жизнь требует времени и ресурсов, которые не хочет тратить никто из членов консор-

[1] Перевод данного стандарта на русский можно найти по адресу *http://www.online.ru/it/helpdesk/xml01.htm*. – *Примеч. науч. ред.*

циума. Она также позволяет разработчикам создавать собственные расширения или частично работающие реализации, которые достаточно хорошо выполняют большую часть работы. Нет, однако, никакой гарантии, что вся работа сделана хорошо, и это недостаток данного подхода. Например, выработка стандарта каскадных таблиц стилей затянулась на годы, поскольку производители броузеров не озаботились его полной реализацией. Тем не менее, процесс стандартизации является в целом демократичным и проводится в интересах общества, что обычно хорошо.

W3C принял на себя роль неофициальной кузни Web. Он основан в 1994 г. рядом организаций и компаний со всего мира, вложивших капиталы в Web и ставящих своей долгосрочной целью исследовать и поощрять доступные и самые лучшие веб-технологии для важных областей применения. Они содействуют устранению хаоса конкурирующих полусырых технологий путем издания технических документов и рекомендаций как для производителей программного обеспечения, так и для пользователей.

Все рекомендации, попадающие на веб-сайт W3C, должны пройти долгий и мучительный процесс предложений и пересмотров, прежде чем быть окончательно ратифицированными Рекомендательной комиссией этой организации. Рекомендация начинает действовать как проект, или *активность* (*activity*), когда кто-то посылает директору W3C формальное предложение, или *информирующий пакет* (*briefing package*). В случае одобрения активности для начала разработки создается собственная рабочая группа со своим уставом. Группа быстро утверждает такие детали, как закрепление руководящих постов, расписание встреч участников, а также создание необходимых списков рассылки и веб-страниц.

Через регулярные промежутки времени группа издает отчеты о достигнутых результатах, помещаемые на общедоступную веб-страницу. Такие *рабочие наброски* (*working drafts*) не обязательно представляют готовую работу или свидетельствуют о достижении согласия между членами группы, но скорее являются отчетом о ходе работ по проекту. В конечном счете наступает момент, когда проект можно представлять на суд общественности. Тогда он становится *кандидатом в рекомендации* (*candidate recommendation*).

Когда кандидат в рекомендации появляется на свет, сообществу предлагается рассмотреть его и прокомментировать. Эксперты в данной области приводят свои соображения. Разработчики реализуют компоненты предложенной технологии и тестируют их, выявляя при этом проблемы. Поставщики программного обеспечения формулируют дополнительные требования к функциональности. Наконец, наступает последний срок представления комментариев, и рабочая группа возвращается к работе, производя пересмотры и изменения.

Если директор считает, что группа создала нечто ценное, что может быть предложено миру, он утверждает кандидата в рекомендации в качестве *предлагаемой рекомендации* (*proposed recommendation*). После этого она должна быть подвергнута тщательному изучению Рекомендательного совета и, возможно, еще немного пересмотрена, прежде чем окончательно превратиться в рекомендацию.

Для завершения всего процесса могут потребоваться годы, и до выхода окончательной рекомендации не следует ничего воспринимать как истину в последней инстанции. Все может измениться за одну-единственную ночь, когда будет представлен новый набросок, и не один разработчик погорел на том, что реализовал эскизные детали рабочего наброска и обнаружил затем, что окончательная рекомендация совсем иная. Конечным пользователям также следует проявлять осторожность. Понадеявшись, что нужная функция будет включена в реализацию, можно обнаружить, что в последнюю минуту она была изъята из списка функций.

Неплохо время от времени посещать веб-сайт консорциума W3C (*http://www.w3.org*). Там можно найти новости и информацию о разрабатываемых стандартах, ссылки на учебники и указатели на программные средства. Они перечислены, наряду с другими полезными ресурсами, в приложении А «Ресурсы».

Сопутствующие технологии

Формально XML представляет собой набор правил для создания собственных языков разметки, а также чтения и написания документов на языке разметки. Это и само по себе полезно, но есть и другие спецификации, которые могут его дополнить. Например, *каскадные таблицы стилей* (*Cascading StyleSheets, CSS*) служат языком для задания внешнего вида документов XML и также имеют формальную спецификацию, написанную W3C.

В данной книге представлены некоторые наиболее важные собратья XML. Основные сведения о них приведены в приложении В «Таксономия стандартов», а некоторые из них мы рассмотрим более подробно. Основными категориями являются следующие:

Базовый синтаксис

В эту группу входят стандарты, обеспечивающие основы функциональности XML. В их составе спецификация собственно XML, пространства имен (способ объединения нескольких типов документов), XLinks (язык для связывания документов) и другие.

Приложения XML

В эту категорию попадают некоторые полезные, производные от XML языки разметки, в том числе XHTML (XML-совместимая вер-

сия языка гипертекста HTML) и MathML (язык математической разметки).

Моделирование документов

В эту категорию входят языки обеспечения структурности для определений типа документа (DTD) и схем XML.

Адресация данных и запросы

Для нахождения документов и данных в них существуют такие спецификации, как XPath (которая описывает пути к данным внутри документов), XPointer (способ описания местонахождения файлов в Интернете) и язык запросов XML (XML Query Language, или XQL) – язык доступа к базам данных.

Стиль и преобразование документов

К этой группе принадлежат языки, предназначенные для описания представления и для преобразования документов в новые форматы, в том числе язык таблиц стилей XML (XML Stylesheet Language XSL), язык преобразования XSL (XSL Transformation Language XSLT), расширяемый язык таблиц стилей для форматирования объектов (Extensible Stylesheet Language for Formatting Objects XSL-FO) и каскадные таблицы стилей (Cascading Style Sheets CSS).

Программирование и инфраструктура

В этой обширной категории содержатся интерфейсы для доступа и обработки информации, закодированной в XML, в том числе *объектная модель документов* (*Document Object Model, DOM*), общий интерфейс программирования; язык описания содержимого документов XML Information Set; стандарт XML Fragment Interchange, описывающий разбиение документов на части для передачи по сети; программный интерфейс обработки данных XML – Simple API for XML (SAX).

Создание документов

Из всего многообразия применяемых программных инструментов самым важным, вероятно, является средство создания документов, или редактор. Средство создания документов определяет среду, в которой, в основном, будет создаваться содержание, а также производиться обновление, а возможно, и просмотр документов XML. Как верный молоток плотника, редактор XML всегда должен быть под рукой.

Есть много средств для написания XML – от простого текстового редактора до роскошных процессоров, позволяющих отображать документ с применением стилей и скрывающих теги от пользователя. XML полностью открыт: никто не обязан работать с каким-либо конкретным инструментом. Если кому-то надоел один редактор, он может пе-

рейти на другой, и документы будут работать точно так же, как и до этого.

Тем, кто привык к суровым условиям работы, приятно будет узнать, что XML можно легко писать в любом текстовом редакторе или текстовом процессоре, позволяющем сохранять данные в простом текстовом формате. Microsoft Notepad, vi в Unix и SimpleText на Apple – все они могут создавать законченные документы XML, а все теги и символы XML содержат символы стандартной клавиатуры. Восхитительно логичная структура XML в сочетании с разумным использованием пробелов и комментариев позволяют некоторым разработчикам не испытывать никаких неудобств, изготавливая целые документы с помощью текстового редактора.

Конечно, можно избавить себя от утомительной работы по разметке, если захотеть этого. В отличие от обычного текстового, специализированный редактор XML может более четко представлять разметку, выделяя теги цветом, либо полностью скрыть разметку и применить таблицу стилей, чтобы выводить части документа с собственными стилями шрифтов. Такие редакторы могут предоставлять специальные механизмы интерфейса пользователя для обработки разметки XML, например, редакторы атрибутов и перетаскивание элементов.

Обязательной функцией в развитых системах создания XML становится автоматическая проверка структуры. Такое средство редактирования предохраняет от синтаксических и структурных ошибок во время написания и редактирования, отказываясь добавлять элемент, не принадлежащий заданному контексту. Некоторые редакторы предлагают меню допустимых элементов. Такая технология идеальна для строго структурированных приложений, например, для заполнения форм или ввода данных в базу.

Навязывая правильную структуру, этот инструмент может также мешать работе. Многие авторы вырезают и вставляют фрагменты документов, экспериментируя с их расположением. При этом часто временно нарушаются структурные правила, что заставляет автора останавливаться и выяснять, почему было отвергнуто перемещение фрагмента, а на это уходит время, которое можно было потратить на создание содержимого. Решить такую головоломку непросто: с одной стороны – выгоды отсутствия ошибок в содержимом, а с другой – препятствия, затрудняющие творческий процесс.

Высококачественная среда инструмента создания документов XML является конфигурируемой. Автор, разработав некоторый тип документа, должен иметь возможность настроить редактор так, чтобы он поддерживал структуру, проверял допустимость и предоставлял набор допустимых элементов, из которых можно осуществлять выбор. Должна существовать возможность создания макросов для автоматизации часто повторяющихся при редактировании действий и назначения клавиш клавиатуры этим макросам. Интерфейс должен быть эр-

гономичным и удобным, обеспечивающим использование комбинаций клавиш вместо многочисленных щелчков мышью для выполнения каждой задачи. Средство редактирования должно поддерживать определение собственных свойств отображения, будь то большой шрифт с цветом или маленький шрифт с тегами.

Конфигурируемость иногда вступает в конфликт с другим важным свойством: легкостью сопровождения. Если есть редактор, который чудесно форматирует текст (например, выводя заголовки крупным и жирным шрифтом и отделяя их от параграфов), это означает, что кто-то должен написать и поддерживать таблицу стилей. Некоторые редакторы снабжены весьма удобным интерфейсом редактирования таблицы стилей, позволяющим работать со стилями элементов почти так же легко, как создавать шаблон в текстовом процессоре. Дополнительные заботы иногда связаны с принуждением к соблюдению правильности структуры, поскольку может потребоваться создать определение типа документа (DTD) с чистого листа. Подобно таблице стилей, DTD сообщает редактору, как обрабатывать элементы, и о допустимости их в различных контекстах. Автор принимает решение о том, что стоит проделать дополнительную работу, если в результате сокращаются время на проверку ошибок и количество жалоб от будущих пользователей.

Инструментарий XML

Рассмотрим некоторые программы, используемые при написании XML. Запомните, пользователь не привязан к какому-то одному конкретному инструменту, поэтому он может экспериментировать, пока не найдет то, что ему больше подходит. Найдя подходящий инструмент, постарайтесь овладеть им в совершенстве. Он должен быть удобным, как перчатка, в противном случае, использование XML может стать мучительным испытанием.

Текстовые редакторы

Текстовые редакторы – это экономичные средства работы с XML. Они выводят все одним шрифтом (хотя некоторые поддерживают выделение цветом), не могут отделить разметку от содержимого и обычно кажутся довольно скучными тем, кто привык к графическим текстовым процессорам. Другими словами, хорошие текстовые редакторы принадлежат к наиболее мощным средствам обработки текста.

Текстовые редакторы не собираются вымирать. Где вы найдете такой простой для изучения и в то же время такой мощный редактор, как vi? В каком текстовом процессоре есть такой встроенный язык программирования, как в Emacs? Здесь описываются эти текстовые редакторы:

vi

Принадлежит к пантеону Unix. Его текстовый интерфейс может показаться примитивным по сегодняшним стандартам GUI, но у vi имеется легион преданных пользователей, благодаря которым он продолжает жить. Есть несколько настраиваемых разновидностей vi, которые можно научить распознавать теги XML. Версии *vim* и *elvis* имеют режимы, которые могут сделать редактирование XML более приятным занятием благодаря выделению тегов разными цветами, созданию отступов и некоторым другим удобным способам преобразования текста.

Emacs

Emacs – это текстовый редактор «с головой». Он был создан в рамках задачи обеспечить мир свободным высококачественным программным обеспечением, которую поставил Фонд свободного программного обеспечения (Free Software Foundation, *http://www.fsf.org*). Emacs остается любимцем пишущих на компьютерах в течение десятилетий. Он поставляется со встроенным языком программирования, многими утилитами обработки текста и модулями, которые можно добавить, чтобы настроить Emacs для редактирования XML, XSLT и DTD. Что следует иметь непременно, так это разработанный Леннартом Стаффлином (Lennart Stafflin) модуль *psgml* (его можно загрузить с *http://www.lysator.liu.se/~lenst/*), который дает Emacs возможность выделять теги цветом, структурировать текст и проверять действительность документа.

Графические редакторы

Подавляющее большинство пользователей компьютеров пишет свои документы в графических редакторах (текстовых процессорах), которые предоставляют меню, возможность редактирования с помощью перетаскивания, выделения текста мышью и так далее. Они также предоставляют форматированный вывод, иногда называемый представлением WYSIWYG (что видишь, то и получаешь). Чтобы сделать XML привлекательным для большинства пользователей, нужны редакторы XML, с которыми легко работать.

Первые графические редакторы для структурированных языков разметки основывались на SGML, «дедушке» XML. Поскольку SGML больше и сложнее, редакторы SGML дороги, трудны в сопровождении и недоступны по цене большинству пользователей. Но XML дал жизнь новому поколению редакторов, которые проще и доступнее. Все перечисленные здесь редакторы поддерживают проверку и принудительное соблюдение структуры:

Arbortext Adept

Arbortext, ветеран в области электронной издательской деятельности, предоставляет одну из лучших программных сред для

редактирования XML. *Adept*, первоначально являвшийся системой создания документов SGML, был доработан для XML. Редактор поддерживает вывод с использованием таблиц стилей FOSI (см. раздел «Таблицы стилей» далее в этой главе) со встроенным интерфейсом назначения стилей. Возможно, лучшей из его характеристик является наличие интерфейса пользователя с возможностью полного управления сценариями, что позволяет писать макросы и осуществлять интеграцию с другим программным обеспечением.

На рис. 1.2 показано рабочее окно Adept. Обратите внимание на представленную слева иерархическую схему, отображающую документ в виде графа. В этом представлении можно сворачивать, открывать и перемещать элементы, что служит альтернативой обычному интерфейсу форматированного содержимого.

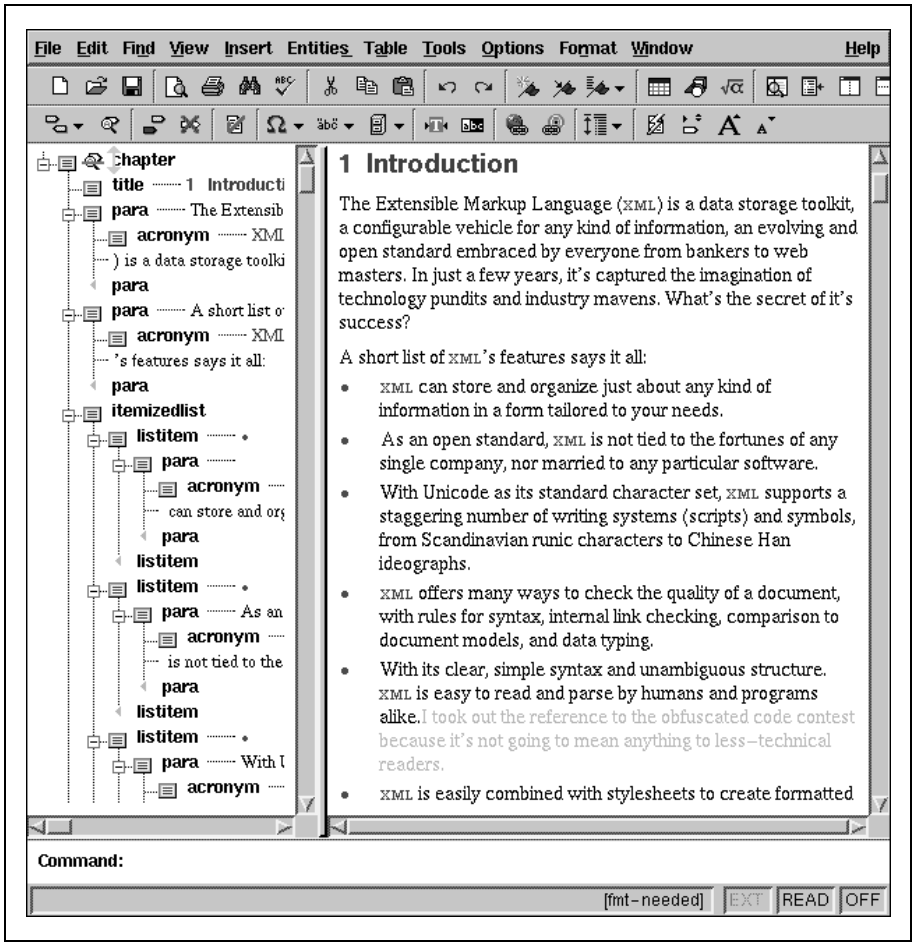


Рис. 1.2. *Редактор Adept*

Adobe FrameMaker+SGML

FrameMaker является высококачественным издательским инструментом редактирования и верстки. Первоначально он поставлялся с собственным языком разметки под названием MIF. Однако когда в мире наметилась тенденция перехода на SGML, а позднее XML, как на универсальный язык разметки, ей последовал и FrameMaker. В настоящее время имеется расширенный пакет под названием *FrameMaker+SGML*, который может читать и писать документы SGML и XML. Он позволяет также осуществлять взаимные преобразования с собственным форматом, обеспечивая сложное форматирование и высококачественный вывод.

SoftQuad XMetaL

Этот графический редактор работает только на PC под Windows, но более доступен по цене и прост в установке, чем два предыдущие. *XMetaL* использует для форматированного вывода таблицу стилей CSS.

Conglomerate

Conglomerate является бесплатным графическим редактором. Некоторые шероховатости и отсутствие исчерпывающей документации не умаляют значение того факта, что его создатели поставили перед собой честолюбивую задачу объединить в будущем редактор с архивной базой данных и трансформирующим механизмом для преобразования документов в форматы HTML и TEX.

Просмотр документов XML

Создавая документ XML, автор, вероятно, хочет показать его другим людям. Одним из способов сделать это является вывод XML на экран, подобно тому, как веб-страница выводится веб-броузером. Можно выводить XML непосредственно с помощью таблицы стилей либо преобразовать его в другой язык разметки (например HTML), который проще форматируется. Альтернативой выводу на экран является создание печатного экземпляра документа. Наконец, реже встречаются, но все же важны форматы «просмотра» в виде азбуки Брайля и звуковой (синтезированной речи).

Как уже говорилось, в XML нет неявных определений стилей. Это значит, что одного документа XML обычно недостаточно для получения форматированного результата. Есть, однако, несколько исключений:

Представление в виде иерархической схемы

Можно вывести в виде схемы структуру и содержание любого документа XML. Например, Internet Explorer 5 выводит в таком виде документ XML (но не XHTML), если не указана таблица стилей. На рис. 1.3 показано типичное представление в виде схемы (outline).

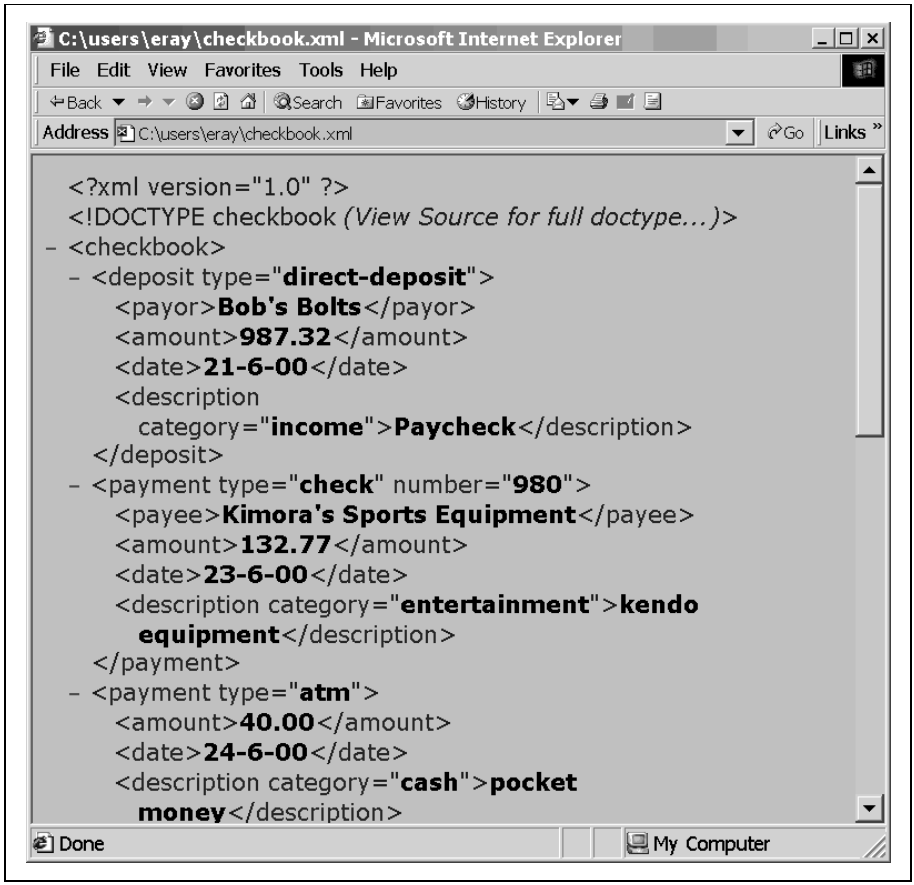


Рис. 1.3. *Представление в Internet Explorer в виде схемы*

XHTML

XHTML (версия HTML, которая удовлетворяет правилам XML) является языком разметки, в котором неявно заданы стили для элементов. Поскольку HTML появился прежде XML и таблиц стилей, то документы HTML автоматически форматируются веб-броузерами и позволяют обойтись без использования таблиц стилей. Не так уж редко документы XML преобразуют в XHTML, чтобы просматривать их в броузере как форматированные документы.

Специализированные программы просмотра

Некоторые языки разметки трудно или невозможно вывести с использованием какой бы то ни было таблицы стилей, и единственным способом просмотра форматированного документа становится применение специальной программы. Например, Chemical Markup Language представляет молекулярные структуры, которые можно просмотреть только специальной программой, например, *Jumbo*.

Таблицы стилей

Таблицы стилей – это главный способ превращения документа XML в форматированный документ, предназначенный для просмотра. Существует несколько типов таблиц стилей, каждая со своими сильными и слабыми сторонами:

Каскадные таблицы стилей (Cascading Style Sheets, CSS)

CSS является простым языком таблиц стилей. Большинство вебброузеров в той или иной степени поддерживает таблицы стилей CSS, однако ни в одном из них нет пока полной поддержки, а в общих функциях между броузерами имеются существенные различия. Таблицы стилей CSS не предназначены для вывода сложных структур, которые встречаются на печатных страницах, но в большинстве случаев их оказывается достаточно.

Расширяемый язык таблиц стилей (Extensible Stylesheet Language, XSL)

Разработка XSL консорциумом W3C продолжается, и когда-нибудь этот тип таблиц стилей может оказаться предпочтительным для документов XML. В то время как CSS использует простое отображение элементов в стили, XSL больше напоминает язык программирования – с рекурсией, шаблонами и функциями. Его форматирующие качества должны значительно превосходить возможности CSS. Однако из-за своей сложности он может остаться вне основного русла развития и применяться только в наиболее развитых издательских системах.

Язык семантики и спецификации стиля документа (Document Style Semantics and Specification Language, DSSSL)

Этот сложный язык форматирования был разработан для форматирования документов SGML и XML, но он труден в изучении и реализации. DSSSL проложил дорогу для XSL, в котором заимствованы и упрощены многие из его идей форматирования.

Пример правил выходного формата (Formatting Output Specification Instances, FOSI)

В качестве раннего «компаньона» SGML этот язык таблиц стилей использовался правительственными организациями, в том числе Министерством обороны. Некоторые фирмы, например, Arbortext и Datalogics, применяли его в своих издательских системах SGML/XML, но, по большей части, FOSI не получил широкой поддержки в частном секторе.

Патентованные языки таблиц стилей

Раздражаемые медленным развитием стандартов или неадекватностью технологий таблиц стилей своим потребностям, некоторые компании разработали собственные языки таблиц стилей. Например, XyEnterprise, давний пионер электронной издательской де-

ятельности, использует собственный язык стилей под названием XPP, который вставляет в содержимое документа макросы обработки. Хотя такие языки могут обеспечивать высококачественный вывод, их можно применять только с одним продуктом.

Броузеры общего назначения

Полезно иметь средство просмотра XML для вывода своих документов, а для текстового документа пользователю понадобится лишь средство просмотра общего назначения. Ниже приводится перечень некоторых веб-броузеров, позволяющих просматривать документы:

Microsoft Internet Explorer (IE)

В настоящее время Microsoft IE – наиболее распространенный веб-броузер. Версия 5.0 для Macintosh стала первым броузером общего назначения, производящим анализ документов XML и выводящим их с помощью каскадных таблиц стилей. Он может также проверять действительность документов, сообщая об ошибках в построении и типе документа, что является хорошим способом проверки документов.

OperaSoft Opera

Этот небольшой, но мощный броузер представляет собой компактную и быструю альтернативу таким броузерам, как Microsoft IE. Он может производить синтаксический анализ документов XML, но поддерживает только CSS Level 1 и частично CSS Level 2.

Mozilla

Mozilla – это проект с открытыми исходными текстами, посвященный разработке полнофункционального броузера, который поддерживает стандарты Сети и выполняется одинаково хорошо на всех основных платформах. Он основывается на коде Netscape Navigator, который компания Netscape сделала общедоступным. И Mozilla, и Navigator версии 6 происходят из одного и того же проекта разработки и строятся вокруг одного и того же ядра вывода под кодовым названием «Gecko». Navigator версии 6 и недавние выпуски Mozilla могут анализировать XML и выводить документы с помощью таблиц стилей CSS.

Amaya

Amaya – это демонстрационный броузер с открытыми исходными текстами, разработанный W3C. В текущей версии 4.1 поддерживается HTML 4.0, XHTML 1.0, HTTP 1.1, MathML 2.0 и CSS.

Конечно, не всегда характеристики оказываются такими замечательными, какими их представляет рекламная шумиха. Во всех перечисленных броузерах есть проблемы ограниченности поддержки таблиц стилей, ошибок реализации и отсутствия функций. Иногда эти

проблемы можно списать на преждевременность выпуска версий, которые не были еще полностью проверены, но иногда их корни лежат глубже.

Не будем вникать в подробности ошибок и замеченных недостатков, но те, кого это интересует, могут поучаствовать в соответствующих обсуждениях, проводимых на сайтах новостей и веб-форумах. Глен Дейвис (Glen Davis), сооснователь Web Standards Project, написал статью для XML.com, озаглавленную «A Tale of Two Browsers» (История двух броузеров) (*http://www.xml.com/pub/a/98/12/2Browsers.html*). В ней он сравнивает поддержку XML и CSS в двух броузерах-тяжеловесах, Internet Explorer и Navigator, и вскрывает несколько поразительных проблем. Проект Web Standards Project (*http://www.webstandards.org*) поощряет применение таких стандартов, как XML и CSS, и организует общественный протест против некорректной и неполной реализации этих стандартов.

Тестирование XML

Контроль качества – важная функция XML. Если XML должен стать универсальным языком, который одинаково работает везде и всегда, стандарты для целостности данных должны быть высоки. Создание документа XML от начала и до конца без ошибок в синтаксисе разметки почти невозможно, а любая из них может заставить процессор XML споткнуться, приводя к непредсказуемым результатам. К счастью, существуют средства тестирования документов и диагностики ошибок.

На первом уровне проверки определяется, можно ли отнести документ к корректным (well-formed). В документах, которые не проходят этот тест, обычно имеются простые ошибки, такие как неправильно введенный тег или отсутствующий символ-разделитель. *Средство контроля корректности* (*well-formedness checker*), или *синтаксический анализатор* (*parser*) – это программа, которая выискивает такие ошибки и сообщает имя файла и номер строки в нем, где встретилась такая ошибка. Редактируя документ XML, используйте средство контроля корректности, чтобы проверить, не осталось ли где-нибудь нарушенной разметки; если анализатор обнаружит ошибки, исправьте их и проведите повторную проверку.

Разумеется, контроль корректности не может обнаружить такие ошибки, как пропуск состава действующих лиц в пьесе или имени автора в написанном очерке. Такие ошибки являются не синтаксическими, а контекстными. Следовательно, средство проверки сообщит пользователю, что документ корректен, и он обнаружит свою ошибку, когда будет слишком поздно.

Решение заключается в применении *средства проверки действительности модели документа* (*validator*), или *проверяющего анализа-*

mopa (*validating parser*). Проверяющий анализатор идет дальше, чем средства проверки корректности, обнаруживая ошибки, которые автор мог не заметить, такие как отсутствующие элементы или их неправильный порядок. Как говорилось выше, модель документа является описанием правил структурирования документа: какие элементы должны быть включены в документ, что могут содержать элементы, и в каком порядке они встречаются. Проверяющий анализатор, будучи применен для проверки наличия в документах контекстных ошибок, становится мощным средством контроля качества.

В следующем листинге приведен пример вывода проверяющего анализатора, обнаружившего в документе несколько ошибок:

```
% nsgmls -sv /usr/local/sp/pubtext/xml.dcl book.xml
/usr/local/prod/bin/nsgmls:I: SP version "1.3.3"
/usr/local/prod/bin/nsgmls:ch01.xml:54:13:E: document type does not
allow element "itemizedlist" here
/usr/local/prod/bin/nsgmls:ch01.xml:57:0:W: character "<" is the first
character of a delimiter but occurred as data
/usr/local/prod/bin/nsgmls:ch01.xml:57:0:E: character data is not
allowed here
```

В первом сообщении сказано, что <itemizedlist> (маркированный список) появился там, где его не должно быть (в данном случае, внутри параграфа). Это пример контекстной ошибки, которая не будет замечена средством проверки корректности документа. Вторая ошибка указывает, что специальный символ разметки (<) был обнаружен среди символов содержимого, а не в теге разметки. Это синтаксическая ошибка, которую обнаружит и средство проверки корректности документа.

Большинство лучших проверяющих анализаторов распространяется бесплатно. Дополнительную информацию можно найти в отличной статье Майкла Классена (Michael Classen), сравнивающей большинство распространенных анализаторов (*http://webreference.com/xml/column22*). Несколько часто используемых проверяющих анализаторов описано ниже:

Xerces

Создан в рамках проекта Apache XML Project (теми же людьми, благодаря которым у нас есть веб-сервер Apache). Xerces является проверяющим анализатором и существует в двух вариантах: на Java и C++. Он поддерживает DTD и более новый стандарт XML Schema для моделей документов.

nsgmls

Создан продуктивным разработчиком Джеймсом Кларком (James Clark). *nsgmls* является быстрым и многофункциональным бесплатно распространяемым проверяющим анализатором. Изначально написан для анализа документов SGML, но также совместим с XML.

XML4J и XML4C

Разработанные alphaWorks R&D Labs в IBM, эти мощные проверяющие анализаторы написаны на Java и C++, соответственно.

Трансформация

Это может прозвучать неправдободобно, но преобразование документов является важной частью XML. *Трансформация* XML (*transformation*) есть процесс, который перестраивает документ в новый вид. Результатом по-прежнему является XML, но он может в корне отличаться от оригинала. Что-то вроде кухонного комбайна для информации.

Одной из целей преобразования является перенос документа из одного приложения XML в другое. Допустим, что некто написал документ в созданном им приложении XML. Документ нельзя просматривать в старых броузерах, понимающих только HTML, но можно преобразовать его в XHTML. При этом сохраняется все содержание, но изменяется разметка, что позволяет просматривать документ даже в броузерах, поддерживающих только HTML.

Трансформацию можно также использовать для фильтрации документа, при которой сохраняется только часть оригинала. Можно создавать выдержки или сводки по документу, например, для подытоживания расходов в чековой книжке или печати названий разделов книги при создании ее оглавления.

Документы преобразуются с помощью *расширяемого языка стилей для трансформаций* (*Extensible Style Language for Transformations, XSLT*). Инструкции XSLT по преобразованию записываются в документе, похожем на таблицу стилей, а затем для получения результата используется процессор трансформации (transformation engine).

Процессоры трансформации

Вот некоторые полезные процессоры трансформации, которые можно применять с XSLT для преобразования документов XML:

XT

Простой и быстрый процессор, написанный на Java Джеймсом Кларком. Примеры в главе 6 «Трансформация: изменение назначения документов» были написаны с использованием XT. К сожалению, некоторое время назад автор прекратил поддержку XT, но будем надеяться, что кто-нибудь поднимет этот факел, не дав ему погаснуть.

Xalan

Созданный в рамках проекта Apache XML Project, Xalan является свободно распространяемым продуктом.

2

- *Анатомия документа*
- *Элементы: строительные блоки XML*
- *Атрибуты: дополнительная сила элементов*
- *Пространства имен: расширьте ваш словарь*
- *Сущности: символы-заместители содержания*
- *Разная разметка*
- *Корректные документы*
- *Как наилучшим образом использовать разметку*
- *Приложение XML: DocBook*

Разметка и основные понятия

Возможно, это самая важная глава в книге, т. к. в ней описываются фундаментальные строительные блоки всех производных от XML языков: элементы, атрибуты, сущности и инструкции обработки. В ней разъясняется, что такое документ и в чем смысл утверждения о правильности построения документа или его состоятельности. Овладение этими понятиями является необходимым предварительным условием понимания многих технологий, приложений и программного обеспечения, связанных с XML.

Откуда у нас такое знание деталей синтаксиса XML? Все они описаны в поддерживаемом W3C техническом документе, рекомендации XML (*http://www.w3.org/TR/REC-xml*). Читать его нелегко, и большинству пользователей XML он не потребуется, но каждый может ознакомиться с первоисточником. Те, кого интересует процесс стандартизации и смысл всего этого жаргона, могут взглянуть на интерактивную аннотированную версию рекомендации на *http://www.xml.com/axml/testaxml.htm*, созданную Тимом Бреем (Tim Bray).[1]

[1] На странице *http://www.w3.org/TR/2000/REC-xml-20001006* сайта W3C есть ссылка на перевод этой спецификации: *http://www.online.ru/it/helpdesk/xml01.htm*.

Анатомия документа

В примере 2.1 приведен крохотный фрагмент XML. Давайте рассмотрим его.

Пример 2.1. Небольшой документ XML

```
<?xml version="1.0"?>
<time-o-gram pri="important">
  <to>Sarah</to>
  <subject>Reminder</subject>
  <message>Don't forget to recharge K-9
    <emphasis>twice a day.</emphasis>
    Also, I think we should have his
    bearings checked out. See you soon
    (or late). I have a date with
    some <villain>Daleks</villain>...
  </message>
  <from>The Doctor</from>
</time-o-gram>
```

Это глупый пример, но совершенно приемлемый XML. XML разрешает именовать части как угодно, в отличие от HTML, в котором можно использовать лишь предопределенные имена тегов. XML безразличен к тому, как будет использоваться документ, как тот будет выглядеть в результате форматирования и даже к значению имен элементов. Важно только, соответствует ли он основным правилам разметки, описанным в данной главе. Из этого, однако, не следует, что организационные вопросы не имеют значения. Нужно выбирать имена элементов, имеющие смысл в контексте документа, а не произвольные, как знаки зодиака. Это принесет пользу в первую очередь вам и тем, кто будет работать с вашим приложением XML.

Этот пример, как и весь XML, состоит из содержимого, перемежаемого символами разметки. Угловые скобки (< >) и заключенные в них имена называются *тегами* (*tags*). Теги разграничивают и помечают части документа, а также добавляют другую информацию, которая помогает определить структуру. Текст между тегами является содержимым документа, необработанной информацией, которая может быть телом сообщения, заголовком или полем данных. Разметка и содержимое дополняют друг друга, создавая информационный объект, в котором поделенные на части и помеченные данные заключены в удобный пакет.

Хотя XML спроектирован так, что его можно считать относительно доступным для чтения человеком, он не предназначен для создания законченного документа. Иными словами, нельзя рассчитывать, что произвольный документ с тегами XML при открытии его в броузере бу-

дет красиво отформатирован.[1] Назначением XML в действительности является обеспечение такого способа хранения содержимого, чтобы в сочетании с другими ресурсами, например, таблицей стилей, документ становился законченным по стилю и отделке продуктом.

Объединение таблицы стилей с документом XML с целью создания форматированного вывода будет рассмотрено в главе 4 «Представление: создание конечного продукта». А пока просто представим себе, как может выглядеть документ в результате применения простой таблицы стилей. В частности, так, как показано в примере 2.2.

Пример 2.2. Памятная записка, отформатированная с помощью таблицы стилей

TIME-O-GRAM
Priority: important
To: Sarah
Subject: Reminder

Don't forget to recharge **K-9 twice a day**.
Also, I think we should have his bearings checked out.
See you soon (or late). I have a date with some *Daleks*...

From: The Doctor

Вывод этого примера в данный момент является чисто умозрительным. При помощи другой таблицы стилей можно форматировать ту же самую записку иным образом. Она могла бы изменить порядок элементов, скажем, выведя строку «From:» выше тела сообщения, либо ограничить ширину тела сообщения 20 символами. И даже «пойти еще дальше», используя различные шрифты, выводя рамку вокруг сообщения, заставляя некоторые части мерцать – все, что угодно. Прелесть XML в том, что он не накладывает ограничений на представление документа.

Посмотрим внимательнее на разметку, чтобы разобраться в ее структуре. Как показано на рис. 2.1, теги разметки разделяют записку на части, представленные на диаграмме прямоугольниками, содержащими другие прямоугольники. В первом прямоугольнике содержится специальный пролог объявления, в котором представлена управляющая информация документа (мы вернемся к нему через мгновение). Остальные прямоугольники называются *элементами* (*elements*). Они выступают в качестве контейнеров и меток текста. Самый большой элемент с

[1] Некоторые броузеры, например Internet Explorer 5.0, пытаются работать с XML осмысленным образом, часто путем вывода его в виде иерархической схемы, которая понятна человеку. Конечно, это выглядит значительно лучше, чем сплошной текст, но все же не совсем то, чего следует ожидать от законченного документа. Таблица должна быть похожа на таблицу, абзац должен быть блоком текста и т. д. Один только XML не может передать броузеру такую информацию.

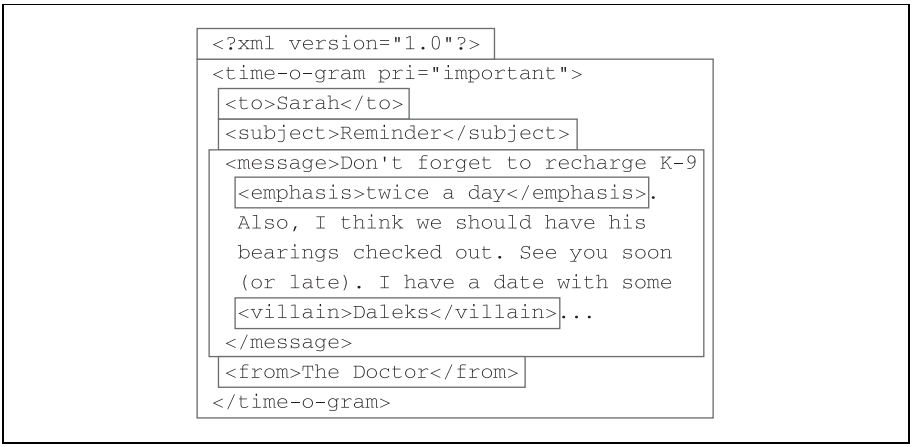


***Рис. 2.1.** Элементы документа памятной записки*

меткой <time-o-gram> окружает все остальные элементы и действует как пакет, содержащий все остальные подразделы. Внутри него находятся специальные элементы, представляющие отдельные функциональные части документа. Глядя на эту диаграмму, можно отметить, что главными частями <time-o-gram> являются адресат (<to>), отправитель (<from>), заголовок сообщения (<subject>) и тело сообщения (<message>). Последняя часть сложнее всего, и в ней совмещены элементы и текст. Поэтому данный пример показывает, что даже простой документ XML может таить в себе несколько структурных уровней.

Представление в виде дерева

Элементы разделяют документ на составные части. В них может содержаться текст, другие элементы или и то, и другое. Рис. 2.2 раскрывает иерархию элементов в нашей памятной записке. Эта диаграмма, называемая *деревом* из-за своего ветвистого вида, является удобным представлением для обсуждения связей между частями документа. Черные прямоугольники представляют семь элементов. Верхний элемент (<time-o-gram>) называется *корневым элементом* (*root element*). Его часто называют также *элементом документа* (*document element*), потому что он заключает в себе все остальные элементы и таким образом определяет границы документа. Прямоугольники в конце цепочки элементов называются *листьями* (*leaves*) и представляют фактическое содержание документа. Все объекты на рисунке, к которым или из которых ведет стрелка, являются *узлами* (*nodes*).

Мы пока не упомянули еще об одной части обсуждаемого рисунка: прямоугольнике слева, помеченном pri. Он был внутри тега <time-o-gram>, но здесь мы видим его ответившимся от элемента. Это особый вид содержимого, называемый *атрибутом* и предоставляющий до-

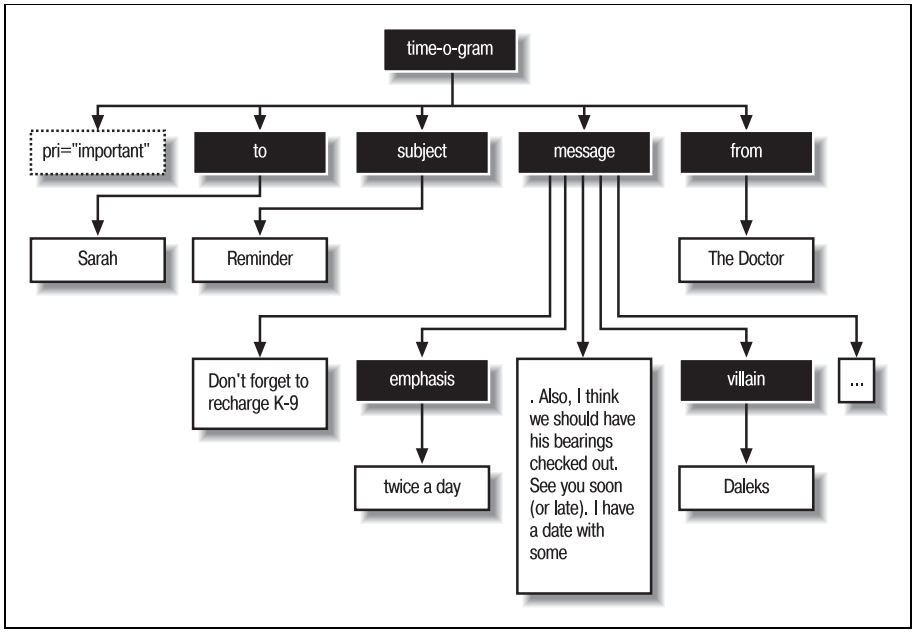


Рис. 2.2. *Диаграмма памятной записки в виде дерева*

полнительную информацию об элементе. Как и у элемента, у атрибута есть метка (`pri`) и некоторое содержание (`important`). Можно представлять его как пару имя/значение, содержащуюся в теге элемента `<time-o-gram>`. Атрибуты используются, в основном, для модификации поведения элемента, а не для хранения его данных. Например, при дальнейшей обработке в верхней части документа можно большими буквами вывести: «High Priority».

Теперь разовьем метафору дерева дальше и представим себе диаграмму как некое родословное дерево, в котором каждый узел является родителем или потомком (или тем и другим) других узлов. Заметьте, однако, что в отличие от родословного дерева у каждого элемента XML только один родитель. С этой точки зрения мы видим, что корневой элемент (старый, седой `<time-o-gram>`) является предком всех остальных элементов. Дочерними для него являются четыре элемента, находящиеся непосредственно под ним. У тех, в свою очередь, есть дочерние элементы, и так будет продолжаться, пока не будут достигнуты бездетные узлы-листья, содержащие текст документа, или какие-либо пустые элементы. Элементы, имеющие одного родителя, называются элементами одного уровня, или узлами-братьями (siblings).

Каждый узел дерева можно рассматривать как корень меньшего поддерева. Поддеревья обладают такими же свойствами, как обычные деревья, а вершина каждого поддерева является предком для всех потомков, расположенных ниже него. В главе 6 «Трансформация: изме-

нение назначения документов» будет показано, что документ XML можно легко обрабатывать, разбивая его на мелкие поддеревья, и получать результат, пересобирая их. На рис. 2.3 показаны некоторые поддеревья нашего примера `<time-o-gram>`.

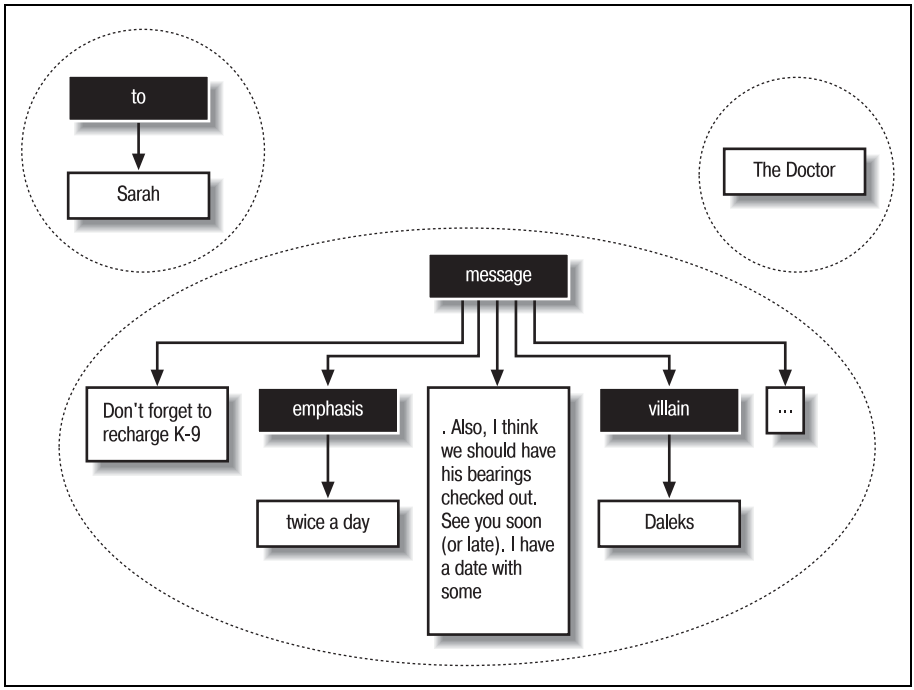


Рис. 2.3. *Некоторые поддеревья*

И на этом 10-минутный обзор XML закончен. Сила XML заключается в его простоте. В оставшейся части главы мы поговорим о деталях разметки.

Пролог документа

Каким-то образом необходимо показать миру, что наш документ размечен в XML. Предоставить это компьютерной программе означает искать себе неприятностей. Многие языки разметки выглядят похоже, и, когда в эту смесь добавляются новые версии, становится трудно отличить один от другого. В особенности это касается документов в World Wide Web, где используются буквально сотни различных форматов файлов.

Верхняя часть документа XML украшена особой информацией, называемой *прологом документа* (*document prolog*). В простейшем случае пролог просто сообщает, что это документ XML, и объявляет используемую версию XML:

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

Но пролог может содержать и дополнительную информацию, которая содержит такие детали, как используемое определение типа документа, объявления особых частей текста, кодировка текста и команды для процессоров XML.

Рассмотрим, из чего состоит пролог, а затем более пристально изучим каждую часть. На рис. 2.4 показан документ XML. Вверху находится объявление XML (1). За ним следует объявление типа документа (2), устанавливающее связь с определением типа документа (3), находящимся в отдельном файле. Затем следует ряд объявлений (4). Совместно эти четыре части составляют пролог (6), хотя не в каждом прологе есть все четыре части. Наконец, корневой элемент (5) содержит оставшуюся часть документа. Этот порядок нельзя изменить: если есть объявление XML, оно должно находиться в первой строке; если есть объявление типа документа, оно должно предшествовать корневому элементу.

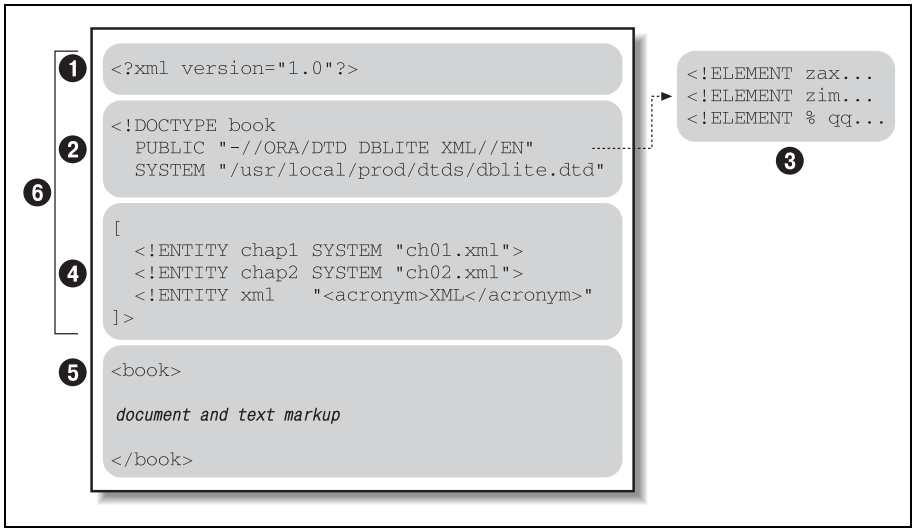


***Рис. 2.4.** Документ с прологом и корневым элементом*

Рассмотрим более тщательно пролог нашего документа <time-o-gram>, который показан ниже в примере 2.3. Обратите внимание, т. к. мы изучаем пролог более детально, то номера в данном примере не совпадают с номерами на рис. 2.4.

Пример 2.3. Пролог документа

```
<?xml version="1.0" encoding="utf-8"?>                    1
<!DOCTYPE time-o-gram                                     2
    PUBLIC "-//LordsOfTime//DTD TimeOGram 1.8//EN"        3
```

Пример 2.3. Пролог документа (продолжение)

```
    "http://www.lordsoftime.org/DTDs/timeogram.dtd"                    4
[                                                                      5
    <!ENTITY sj "Sarah Jane">                                          6
    <!ENTITY me "Doctor Who">
]>                                                                     7
```

1. Объявление XML описывает некоторые самые общие свойства документа, указывая процессору XML на необходимость использования синтаксического анализатора XML для интерпретации данного документа.
2. *Объявление типа документа* (*document type declaration*) описывает тип корневого элемента, в данном случае `<time-o-gram>`, и (в строках 3 и 4) указывает на *определение типа документа* (*Document Type Definition, DTD*), которое управляет структурой разметки.
3. Идентифицирующий код, называемый *открытым идентификатором* (*public identifier*), задает используемое DTD.
4. *Системный идентификатор* (*system identifier*) указывает местонахождение DTD. В данном примере системным идентификатором является URL.
5. Это начало *внутреннего подмножества* (*internal subset*), в котором находятся специальные объявления.
6. Внутри данного внутреннего подмножества есть два *объявления сущностей* (*entity declarations*).
7. Пролог завершается символами `]` и `>`, закрывающими внутреннее подмножество () и объявление типа документа, соответственно.

Каждый из этих терминов более подробно описывается ниже в данной главе.

Объявление XML

Объявление XML возвещает процессору XML, что данный документ размечен в XML, и в виде схемы представлено на рис. 2.5. Объявление начинается с разделителя, состоящего из пяти символов `<?xml` (1), за которым следует некоторое количество определений свойств (2), каждое из которых состоит из имени свойства (3) и его значения, заключенного в кавычки (4). Объявление завершается закрывающим разделителем из двух символов `?>` (5).

Существуют три свойства, значения которых могут быть установлены:

`version` (версия)

Устанавливает номер версии. В настоящий момент есть только одна версия XML, поэтому данное значение всегда 1.0. Однако, когда будут одобрены новые версии, это свойство станет указывать процес-

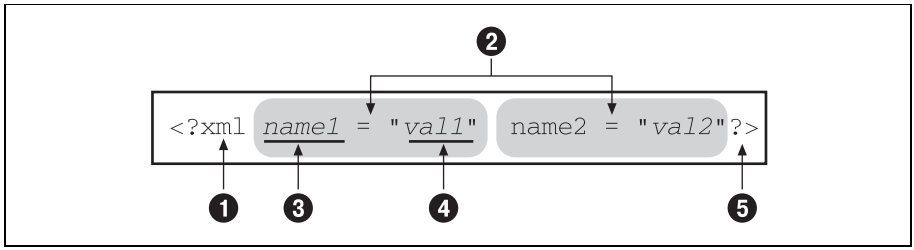


***Рис. 2.5.** Синтаксис объявления XML*

сору XML, которая из версий должна использоваться. Это свойство всегда должно быть определено в прологе.

`encoding` **(кодировка)**

Задает кодировку символов, использованную в документе, например, `US-ASCII` или `iso-8859-1`. При использовании набора символов, отличного от стандартных латинских символов ASCII[1] (например, японской катаканы или кириллицы), следует задать это свойство. В противном случае можно его опустить. О кодировках символов рассказано в главе 7 «Поддержка многоязычности».

`standalone` **(автономность)**

Сообщает процессору XML, есть ли другие файлы, которые нужно загружать. Например, это свойство устанавливается в `no`, если есть внешние сущности (смотрите раздел «Сущности: символы-заместители содержания» далее в этой главе) или DTD, загружаемое в дополнение к основному файлу документа. Если о файле известно, что он может обрабатываться самостоятельно, установите `standalone="yes"`, что увеличит скорость загрузки. Более подробно об этом параметре рассказано в главе 5 «Модели документов: более высокий уровень контроля».

Вот некоторые примеры правильно построенных объявлений XML:

```
<?xml version="1.0"?>
<?xml version='1.0' encoding='US-ASCII' standalone='yes'?>
<?xml version = '1.0' encoding= 'iso-8859-1' standalone ="no"?>
```

Любое из заданий свойств является необязательным, но следует указать хотя бы номер версии на тот случай, если что-либо существенно изменится в будущих версиях спецификации XML. Имена параметров должны задаваться в нижнем регистре, а все значения обязательно заключаются в двойные или одинарные кавычки.

---

[1] В оригинале UTF-8, но это может внести путаницу. Согласно стандарту XML, если в документе используются символы кодировки ASCII, которая является подмножеством UTF-8, то указывать кодировку не обязательно. – *Примеч. науч. ред.*

Объявление типа документа

Вторая часть пролога содержит объявление типа документа.[1] Здесь можно задавать разные параметры, такие как объявления сущностей, DTD, используемое для проверки действительности документа, и имя корневого элемента. Указывая DTD, мы просим, чтобы синтаксический анализатор сравнил экземпляр документа с моделью документа, и этот процесс называется *проверкой действительности*. Проверка действительности на самом деле не является обязательной, но она полезна, если требуется, чтобы документ соответствовал предсказуемым шаблонам и включал в себя необходимые данные. Подробные сведения о DTD и проверке действительности можно найти в главе 5 «Модели документов: более высокий уровень контроля».

Синтаксис объявления типа документа показан на рис. 2.6. Объявление начинается с литеральной строки `<!DOCTYPE` (1). Затем идет корневой элемент (2), который является первым элементом XML, появляющимся в документе, и содержит остальную часть документа. Если с документом используется DTD, следующим должен быть включен URI для DTD (3), чтобы процессор XML мог его найти. Затем следует внутреннее подмножество (5), границами которого являются квадратные скобки (4 и 6). Объявление заканчивается закрывающей угловой скобкой `>`.

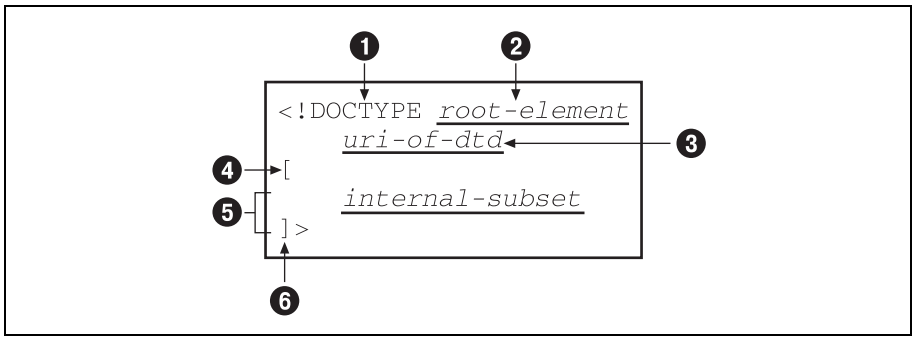


Рис. 2.6. *Синтаксис объявления типа документа*

Внутреннее подмножество (internal subset) предназначено для размещения различных объявлений, используемых в документе, как было показано на рис. 2.4. Эти объявления могут содержать определения сущностей и части DTD. Внутреннее подмножество – единственное место в самом документе, где можно расположить эти объявления.

[1] Не путайте это понятие с определением типа документа, DTD, которое представляет собой совокупность параметров, описывающих тип документа, и может использоваться многими экземплярами документа такого типа.

Внутреннее подмножество используется для дополнения или замены объявлений, которые находятся во *внешнем подмножестве* (*external subset*). Внешнее подмножество является совокупностью объявлений, находящихся вне документа, например, в DTD. URI, задаваемый в объявлении типа документа, указывает на файл, содержащий эти внешние объявления. Внутреннее и внешнее подмножество не являются обязательными. Описание внутренних и внешних подмножеств представлено в главе 5 «Модели документов: более высокий уровень контроля».

Элементы: строительные блоки XML

Элементы суть части документа. Можно разделить документ на части, которые будут выводиться различным образом или использоваться поисковым механизмом. Элементы могут быть контейнерами, содержащими смесь текста и других элементов. Следующий элемент содержит только текст:

```
<flooby>This is text contained inside an element</flooby>
```

А этот элемент содержит как текст, так и другие элементы:

```
<outer>this is text<inner>more
text</inner>still more text</outer>
```

Некоторые элементы являются пустыми, и информация, которую они несут, определяется положением и атрибутами. Вот пустой элемент внутри данного примера:

```
<outer>an element can be empty: <nuttin/></outer>
```

На рис. 2.7 показан синтаксис элемента-контейнера. Впереди него находится открывающий тег (1), состоящий из угловой скобки (<), за которой следует имя (2). Открывающий тег может содержать атрибуты (3), которые разделяются пробельными символами, а оканчивается он

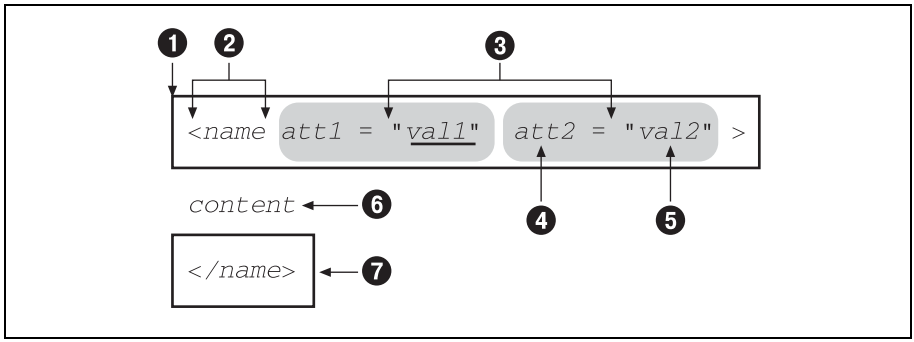


Рис. 2.7. *Синтаксис элемента-контейнера*

закрывающей угловой скобкой (>). Атрибут определяет свойство элемента и состоит из имени (4), соединяемого знаком равенства (=) со значением, заключенным в кавычки (5). У элемента может быть любое количество атрибутов, но никакие два атрибута не могут иметь одинаковых имен. За открывающим тегом следует содержимое элемента (6), за которым, в свою очередь, следует закрывающий тег (7). Закрывающий тег состоит из открывающей угловой скобки, косой черты, имени элемента и закрывающей скобки. У закрывающего тега нет атрибутов, а имя элемента должно в точности совпадать с именем открывающего тега.

Как показано на рис. 2.8, пустой элемент (не имеющий содержания) состоит из единственного тега (1), начинающегося с открывающей угловой скобки (<), за которой следует имя элемента (2). Затем следует некоторое количество атрибутов (3), каждый из которых состоит из имени (4) и значения в кавычках (5), а заканчивается элемент косой чертой (/) и закрывающей угловой скобкой.

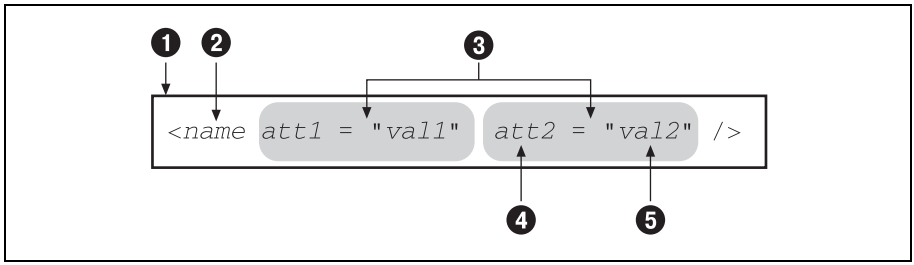


***Рис. 2.8.** Синтаксис пустого элемента*

Имя элемента должно начинаться с буквы или символа подчеркивания и может содержать любое число букв, цифр, дефисов, точек и символов подчеркивания.[1] Имена элементов могут содержать латинские символы с акцентами, буквы таких алфавитов, как кириллица, греческий, иврит, арабский, тайский, хирагана, катакана и деванагари, а также китайские, японские и корейские идеограммы. Символ двоеточия применяется при использовании пространств имен, как описывается ниже в разделе «Пространства имен: расширьте ваш словарь». Поэтому избегайте его в именах элементов, не использующих пространства имен. Пробел, символы табуляции, перевода строки, равенства и кавычки служат разделителями имен элементов, имен атрибутов и значений атрибутов, поэтому они также недопустимы в именах. Вот некоторые допустимые имена элементов: `<Bob>`, `<chapter.title>`,

[1] На практике следует избегать слишком длинных имен элементов на случай, если процессор XML не может обрабатывать имена, длина которых превышает некоторое значение. Конкретное значение назвать трудно, но, вероятно, это имена длиннее 40 символов.

<THX-1138> и даже <_>. В именах XML учитывается регистр, поэтому <Para>, <para> и <pArA> представляют три разные элемента.

Между открывающей угловой скобкой и именем элемента не должно быть пробела, но лишние пробелы в любом другом месте тега элемента допустимы. Благодаря этому можно разбивать элемент на несколько строк для улучшения читаемости, например:

```
<boat
  type="trireme"
><crewmember   class="rower">Dronicus Laborius</crewmember    >
```

Есть два правила, касающиеся расположения открывающего и закрывающего тегов:

- Закрывающий тег должен помещаться после открывающего тега.
- Открывающий и закрывающий теги элемента должны иметь одного и того же родителя.

Чтобы понять второе правило, представьте себе элементы как контейнеры, которые могут находиться как внутри друг друга, так и отдельно. При этом ни один из них невозможно переместить через стенку другого, не сломав ее. Вот почему следующий пример с перекрывающимися элементами работать не будет:

```
<a>Don't <b>do</a> this!</b>
```

А такое размещение элементов нормально, они не перепутаны:

```
<a>No problem</a><b>here</b>
```

В содержимом все, что не является элементом, представляет собой текст, или *символьные данные* (*character data*). В тексте может присутствовать любой символ из набора, указанного в прологе. Однако некоторые символы необходимо представлять особым образом, чтобы не ввести в заблуждение анализатор. Например, левая угловая скобка (<) зарезервирована для тегов элементов. Непосредственное включение ее в содержимое вызывает неопределенную ситуацию: как понять, является она началом тега XML или просто данными? Вот пример:

```
<foo>x < y</foo>     ой!
```

Чтобы разрешить этот конфликт, нужно использовать специальный код для такого символа. Левую угловую скобку можно представить кодом < (соответствующий код для правой угловой скобки – >). Поэтому приведенный выше пример можно переписать так:

```
<foo>x &lt; y</foo>
```

Такая подстановка известна как *ссылка на сущность* (*entity reference*). Мы расскажем о сущностях и ссылках на сущности в разделе «Сущности: символы-заместители содержания» данной главы.

В XML сохраняются все символы, в том числе символы пробела, табуляции и перевода строки, в отличие от таких языков программирования, как Perl и C, в которых пробельные символы (whitespace characters), в сущности, игнорируются. В языках разметки, подобных HTML, несколько последовательных пробелов свертываются броузером в один, а строки могут разбиваться в любом месте в соответствии с требованиями форматирования. Напротив, по умолчанию XML сохраняет все пробелы.

XML – это не HTML

Тем, у кого есть некоторый опыт написания документов HTML, следует обратить особое внимание на правила XML для элементов. Сокращения, которые прощает HTML, в XML не допускаются. В число некоторых важных отличий, которые следует принимать во внимание, входят следующие:

- В именах элементов XML учитывается регистр. HTML позволяет писать теги на любом регистре.
- В XML элементы-контейнеры требуют как открывающий, так и закрывающий теги. В HTML в некоторых случаях можно опустить закрывающий тег.
- Перед правой (закрывающей) угловой скобкой в пустых элементах XML требуется ставить косую черту – `<example/>`), тогда как в HTML используется одиночный открывающий тег без косой черты в конце.
- В элементах XML пробельные символы рассматриваются как часть содержимого и сохраняются, если только явно не указано обратное. Но в HTML для большинства элементов лишние пробелы и переносы строк отбрасываются при форматировании содержимого в броузере.

В отличие от многих элементов HTML, элементы XML основываются строго на функциональности, а не на представлении. Не следует предполагать, что какой-либо стиль форматирования и представления основывается только на разметке. Наоборот, в XML представление определяется таблицами стилей, которые являются отдельными документами, отображающими элементы в стили.

Атрибуты: дополнительная сила элементов

Иногда требуется передать об элементе больше информации, чем могут выразить его имя и содержимое. Атрибуты позволяют более ясно описывать детали элементов, например, для того чтобы дать элементу уникальную метку, по которой его можно легко найти, или описать свойство элемента, скажем, адрес файла, на который указывает ссылка. При помощи атрибута можно описать какую-либо сторону поведения элемента или создать подтип. Например, в нашем послании `<time-o-gram>` ранее в этой главе атрибут `pri` был использован для указания того, что послание имеет высокую срочность. Как показано на рис. 2.9, атрибут состоит из имени атрибута (1), знака равенства (2) и значения в кавычках (3).

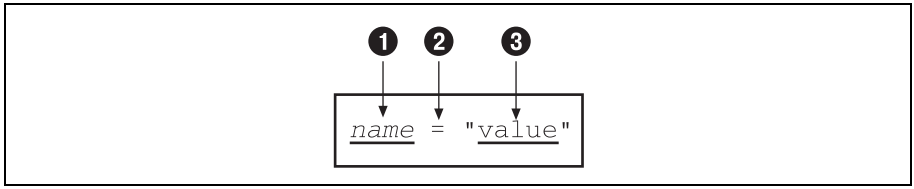


***Рис. 2.9.** Синтаксис атрибутов*

У элемента может быть любое число атрибутов, но каждый из них должен обладать уникальным именем. Вот элемент с тремя атрибутами:

```
<kiosk music="bagpipes" color="red" id="page-81527">
```

Атрибуты разделяются пробелами. Они должны всегда задаваться после имени элемента, но порядок задания может быть любым. Значения должны быть заключены в одинарные (') или двойные (") кавычки. Если само значение содержит кавычки, оно заключается в кавычки другого типа. Вот пример:

```
<choice test='msg="hi"'/>
```

При желании можно заменять кавычки сущностями `&apos;` для одинарной кавычки и `&quot;` для двойной:

```
<choice test='msg=&quot;hi&quot;'/>
```

Для каждого элемента любой атрибут можно задать только один раз, поэтому следующая запись недопустима:

```
<!-- Ошибка -->
<team person="sue" person="joe" person="jane">
```

Вот некоторые альтернативы. Используем один атрибут для хранения всех значений:

```
<team persons="sue joe jane">
```

Определим несколько атрибутов:

```
<team person1="sue" person2="joe" person3="jane">
```

А вот как можно применять элементы:

```
<team>
  <person>sue</person>
  <person>joe</person>
  <person>jane</person>
</team>
```

Если используется DTD, то значения атрибутов можно ограничить определенными типами. Одним из типов является `ID`, который сообщает XML, что его значение является уникальным идентифицирующим кодом элемента. Никакие два элемента в документе не должны иметь одинаковые `ID`. Другой тип, `IDREF`, служит ссылкой на `ID`. Покажем, как можно использовать эти типы. Пусть где-то в документе есть элемент с атрибутом `ID`-типа:

```
<part id="bolt-1573">...</part>
```

В каком-то другом месте есть элемент, который на него ссылается:

```
<part id="nut-44456">
  <description>This nut is compatible with <partref
  idref="bolt-1573"/>.</description>...
```

Если вместе с документом используется DTD, то можно присвоить тип `ID` или `IDREF` конкретным атрибутам, и тогда анализатор XML будет требовать соблюдения синтаксиса значения, а также предупреждать о случаях, когда `IDREF` указывает на несуществующий элемент или значение `ID` не уникально. Подробнее об этих атрибутах мы поговорим в главе 3 «Соединение с ресурсами с помощью ссылок».

Другой способ, которым DTD может ограничивать значения атрибутов, заключается в создании множества допустимых значений. Пусть, например, потребовался атрибут с именем `day`, который может принимать одно из семи значений: `Monday`, `Tuesday`, `Wednesday`, `Thursday`, `Friday`, `Saturday` или `Sunday`. DTD сообщит анализатору XML, что нужно отвергать любые значения, не принадлежащие этому списку, то есть `day="Halloween"` недопустимо. Более подробно типы атрибутов рассмотрены в главе 5 «Модели документов: более высокий уровень контроля».

Зарезервированные имена атрибутов

Некоторые имена атрибутов зарезервированы рабочей группой XML для специальных целей – для использования в XML, и начинаются с префикса `xml:`. Имена `xml:lang` и `xml:space` определены для XML вер-

сии 1.0. Другие два имени, `xml:link` и `xml:attribute`, определены в XLink, еще одном стандарте, дополняющем XML и определяющем способы связи элементов между собой. Вот описание этих особых имен атрибутов:

`xml:lang`

Классифицирует элемент по языку, на котором написано содержание. Например, запись `xml:lang="en"` означает, что содержимое элемента является текстом на английском языке. Это полезно для создания текста, зависимого от условия и выбираемого процессором XML исходя из того, какой язык предпочитает пользователь. Мы вернемся к этой теме в главе 7 «Поддержка многоязычности».

`xml:space`

Указывает, должны ли пробельные символы сохраняться в содержимом элемента. Если ему присвоено значение `preserve`, то любой процессор XML при выводе документа должен учитывать все символы перевода строки, пробела и табуляции в содержимом элемента. При значении `default` процессор может свободно обращаться с пробельными символами (т. е. устанавливать свои значения по умолчанию). Если атрибут `xml:space` опущен, то по умолчанию процессор сохраняет пробельные символы. Поэтому при желании сократить количество пробельных символов в элементе следует установить атрибут `xml:space="default"` и использовать процессор XML, для которого действием по умолчанию является удаление лишних пробельных символов.

`xml:link`

Сообщает процессору XLink, что элемент является элементом ссылки. Сведения об использовании этого элемента приводятся в главе 3 «Соединение ресурсов с помощью ссылок».

`xml:attribute`

Помимо `xml:link` XLink использует ряд других имен атрибутов. Но чтобы избежать возможных конфликтов в случае применения этих атрибутов для других целей, XLink определяет атрибут `xml:attribute`, который позволяет «переназначить» эти специальные атрибуты. Это означает, что можно сказать: «Когда XLink ищет атрибут с именем `title`, я хочу, чтобы он использовал вместо него атрибут `linkname`». Этот атрибут более подробно также обсуждается в главе 3.

Пространства имен: расширьте ваш словарь

Что произойдет, если потребуется включить элементы или атрибуты из другого типа документа? Например, включить в документ XML уравнение, закодированное на языке MathML. К сожалению, нельзя использовать с одним документом несколько DTD, но никто не говорит, что в XML наличие DTD обязательно. Если можно обойтись без DTD (а большинство броузеров терпят документы без них), воспользуйтесь другим средством XML, которое называется пространствами имен.

Пространство имен (*namespace*) – это группа имен элементов и атрибутов. Можно объявить, что элемент существует в некотором пространстве имен и что он должен проверяться относительно DTD этого пространства. Добавляя префикс пространства имен к имени элемента или атрибута, мы сообщаем анализатору о том, из какого пространства имен оно происходит.

Представьте себе, например, что английский язык разделен на пространства имен, соответствующих областям понятий. Выберем две таких области, скажем, hardware («железки») и food (еда). Тема hardware содержит такие слова, как hammer и bolt, а тема food содержит слова fruit и meat. В обоих пространствах имен есть слово nut, но в каждом из контекстов оно имеет свое значение, несмотря на то, что пишется одинаково в обоих случаях. В сущности, это два разных слова с одинаковыми именами, но как это можно описать, не вызывая конфликта пространств имен?

Такая же проблема может возникнуть в XML, если два объекта XML в различных пространствах имен имеют одинаковые имена, что приводит к неоднозначности в определении их происхождения. Решение заключается в том, чтобы указать для каждого элемента или атрибута пространство имен, к которому они принадлежат, в виде префикса.

Синтаксис таких *полных имен элементов* (*qualified names*) показан на рис. 2.10. Префикс пространства имен (1) присоединяется через двоеточие (2) к локальному имени элемента или атрибута (3).

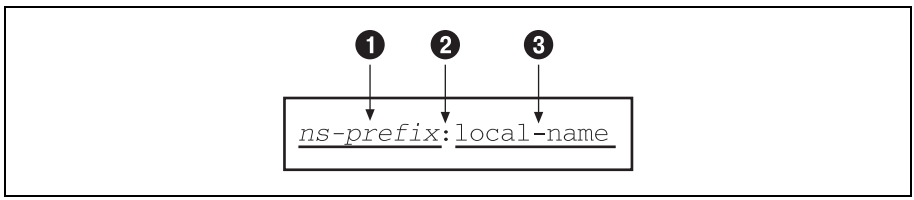


Рис. 2.10. *Синтаксис полного имени*

На рис. 2.11 показано, как обращаться к элементу `nut`, чтобы использовать версии из пространств имен `hardware` и `food`.

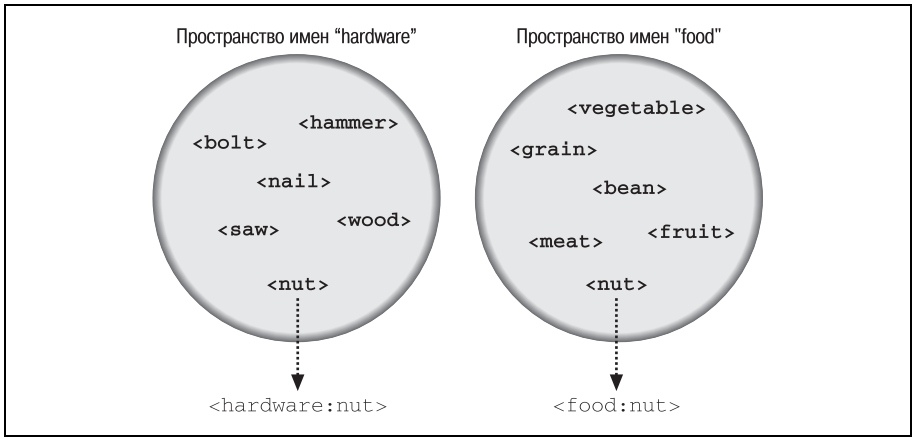


***Рис. 2.11.** Задание пространства имен элемента префиксами*

Пространства имен полезны не только для предотвращения конфликтов имен. В более общем случае они помогают процессору XML выделять различные группы элементов для разных способов обработки. Если вернуться к примеру с MathML, то элементы из пространства имен MathML должны обрабатываться не так, как обычные элементы XML. Броузер должен знать, когда ему входить в «режим математических уравнений», а когда остаться в «обычном режиме XML». Пространства имен имеют решающее значение для переключения режимов броузера.

В другом примере язык преобразований XSLT (описываемый в главе 6 «Трансформация: изменение назначения документов») использует пространства имен, чтобы различать объекты XML, являющиеся данными, и объекты, представляющие собой инструкции обработки данных. Элементы и атрибуты, являющиеся инструкциями, имеют префикс пространства имен `xsl:`. Все, что не имеет префикса пространства имен, рассматривается при преобразовании как данные.

Пространство имен нужно объявить в документе прежде, чем оно будет использовано. Объявление имеет вид атрибута внутри элемента. Все потомки этого элемента входят в это пространство имен. На рис. 2.12 показан синтаксис объявления пространства имен. Оно начинается с ключевого слова `xmlns:` (1), предупреждающего анализатор XML о том, что это объявление пространства имен. Затем следуют двоеточие, префикс пространства имен (2), знак равенства и, наконец, URL, заключенный в кавычки (3).

Например:

```
<part-catalog
    xmlns:bob="http://www.bobco.com/">
```

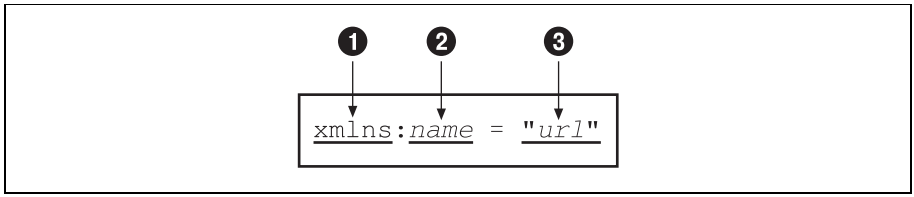


***Рис. 2.12.** Синтаксис объявления пространства имен*

Те, кому не нравится префикс `bob`, могут использовать любое другое имя, удовлетворяющее правилам именования элементов. В частности, `b`, `bobs-company` или `wiggledy.piggledy` являются допустимыми именами. Следует остерегаться использования таких префиксов, как `xml`, `xsl` или других имен, зарезервированных в XML и связанных с ним языках.

Значением атрибута `xmlns:` является URL, обычно принадлежащий организации, поддерживающей пространство имен. Однако процессору XML не нужно ничего делать с URL. Даже документ, на который указывает URL, не обязательно должен существовать. Задание этого URL является формальностью, предоставляющей дополнительные сведения о пространстве имен, например, о его владельце и используемой версии.

Любой элемент в документе может содержать объявление пространства имен. Чаще всего объявления располагаются в корневом элементе, но это не является обязательным требованием. Может оказаться полезным ограничить видимость пространства имен некоторой внутренней областью документа, объявив пространство имен в каком-нибудь более глубоко вложенном элементе. В таких случаях пространство имен применяется только к этому элементу и его потомкам.

Вот 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>March 4, 2001</myns:date>
    <myns:subject>Effects of Caffeine on Psychokinetic
     Ability</myns:subject>
    <myns:abstract>The experiment consists of a subject, a can of
     caffeinated soda, and a goldfish tank. The ability to make a
     goldfish turn in a circle through the power of a human's mental
     control is given by the well-known equation:

     <eq:formula xmlns:eq="http://www.mathstuff.org/">
       <eq:variable>P</eq:variable> =
       <eq:variable>m</eq:variable>
       <eq:variable>M</eq:variable> /
       <eq:variable>d</eq:variable>
     </eq:formula>
```

```
      where P is the probability it will turn in a given time interval,
      m is the mental acuity of the fish, M is the mental acuity of
      the subject, and d is the distance between
      fish and subject.</myns:abstract>
      ...
  </myns:experiment>
</myns:journal>
```

Можно объявить одно из пространств имен как используемое по умолчанию, опустив двоеточие (:) и имя в атрибуте `xmlns`. Элементы и атрибуты из пространства имен по умолчанию не требуют префикса, что приводит к более ясной разметке:

```
<?xml version="1.0"?>
<journal xmlns="http://www.psycholabs.org/mynamespace/">
  <experiment>
    <date>March 4, 2001</date>
    <subject>Effects of Caffeine on Psychokinetic Ability</subject>
    <abstract>The experiment consists of a subject, a can of
     caffeinated soda, and a goldfish tank. The ability to make a
     goldfish turn in a circle through the power of a human's mental
     control is given by the well-known equation:

     <eq:formula xmlns:eq="http://www.mathstuff.org/">
       <eq:variable>P</eq:variable> =
       <eq:variable>m</eq:variable>
       <eq:variable>M</eq:variable> /
       <eq:variable>d</eq:variable>
     </eq:formula>

     where P is the probability it will turn in a given time interval,
     m is the mental acuity of the fish, M is the mental acuity
     of the subject, and d is the distance between
     fish and subject.</myns:abstract>
     ...
  </experiment>
</journal>
```

Пространства имен могут стать источником головной боли, если используются вместе с DTD. Было бы прекрасно, если бы анализатор игнорировал все элементы или атрибуты из другого пространства имен, чтобы действительность документа проверялась по DTD без представления о пространстве имен. К несчастью, дело обстоит не так. Чтобы использовать пространство имен с DTD, нужно переписать DTD так, чтобы оно «знало» об элементах в этом пространстве имен.

Другая проблема, порождаемая применением пространств имен, состоит в том, что они не импортируют DTD или другого рода информацию об используемых автором документа элементах и атрибутах. Поэтому на самом деле можно создать собственные элементы, добавить префикс пространства имен, и анализатор ничего не узнает. В результате пространства имен оказываются менее полезными для тех, кто хочет наложить на свои документы ограничения в виде согласованности с DTD.

По этим и некоторым другим причинам пространства имен служат предметом споров среди проектировщиков XML. Что произойдет в будущем, пока не ясно, но необходимы какие-то действия, направленные на преодоление разрыва между принудительным соблюдением структуры и пространствами имен.

Сущности: символы-заместители содержания

Теперь, когда основные части разметки XML определены, остается еще один компонент, который нужно рассмотреть. *Сущность* (*entity*) является заместителем (placeholder) содержания, которую можно однажды объявить и многократно использовать почти в любом месте документа. Семантически это ничего не добавляет к разметке. Скорее это удобный способ облегчить написание, сопровождение и чтение XML.

Сущности могут использоваться по различным причинам, но они всегда устраняют неудобства. Они выполняют все – от замены символов, которые невозможно ввести с клавиатуры, до отметки места, в которое должен быть импортирован файл. Можно определять собственные сущности, заменяющие повторяющийся текст, например, название компании или стандартный юридический текст. Сущности могут содержать одиночные символы, строки текста или даже куски разметки XML. Без сущностей XML был бы значительно менее полезен.

Можно, например, определить сущность `w3url`, представляющую URL для W3C. Если ввести эту сущность в документ, она будет заменена текстом *http://www.w3.org* /.

На рис. 2.13 показаны основные типы сущностей – *параметрические* и *общие* – и их роли. *Параметрические сущности* используются только в DTD, поэтому мы опишем их в главе 5. В данном разделе мы сосредоточимся на другом типе, общих сущностях. *Общие сущности* (*general entities*) являются заместителями для любого содержимого, которое присутствует на уровне корневого элемента документа XML или внутри него.

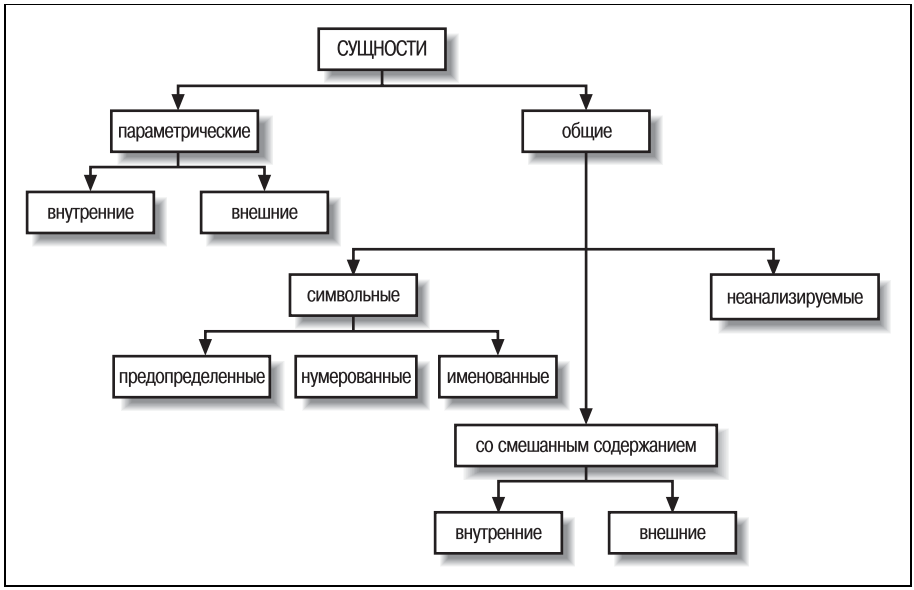


Рис. 2.13. *Таксономия сущностей*

Сущность состоит из имени и значения. Приступая к обработке документа, анализатор XML сначала считывает ряд *объявлений* (*declarations*), а некоторые из них определяют сущности, связывая имя со значением. Значением может быть все – от отдельного символа до файла с разметкой XML. Когда анализатор просматривает документ XML, он находит *ссылки на сущности* (*entity references*), являющиеся специальными маркерами, основанными на именах сущностей. Для каждой ссылки на сущность анализатор ищет в таблице, расположенной в памяти, то, чем маркер должен быть заменен. Ссылку на сущность он заменяет соответствующим текстом или разметкой и возобновляет анализ с точки, находящейся перед местом замены, поэтому новый текст тоже анализируется. Все ссылки на сущности внутри текста замены тоже подвергаются замене, и этот процесс повторяется столько раз, сколько требуется.

На рис. 2.14 показано, что есть два вида синтаксиса ссылок на сущности. Первый, состоящий из амперсанда (`&`), имени сущности и точки с запятой (`;`), применяется для общих сущностей. Второй синтаксис, отличающийся использованием знака процента (`%`) вместо амперсанда, предназначен для параметрических сущностей.

Ниже приведен пример документа, в котором объявлены три общие сущности, причем ссылки на них есть в тексте:

```
<?xml version="1.0"?>
<!DOCTYPE message SYSTEM "/xmlstuff/dtds/message.dtd"
[
```

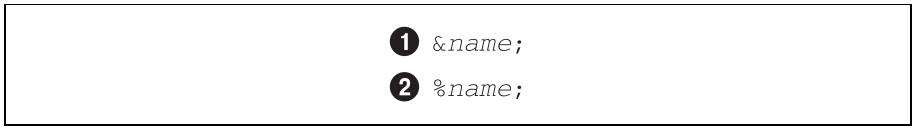


***Рис. 2.14.** Синтаксис ссылок на сущности*

```
    <!ENTITY client "Mr. Rufus Xavier Sasperilla">
    <!ENTITY agent "Ms. Sally Tashuns">
    <!ENTITY phone "<number>617-555-1299</number>">
]>
<message>
<opening>Dear &client;</opening>
<body>We have an exciting opportunity for you! A set of
ocean-front cliff dwellings in Pi&#241;ata, Mexico have been
renovated as time-share vacation homes. They're going fast! To
reserve a place for your holiday, call &agent; at &phone;.
Hurry, &client;. Time is running out!</body>
</message>
```

Сущности `&client;`, `&agent;` и `&phone;` объявляются во внутреннем подмножестве этого документа, а ссылки на них расположены в элементе `<message>`. Четвертая сущность, `&#241`, является нумерованной символьной сущностью, представляющей символ ñ. Ссылка на эту сущность есть, но она не объявляется. В объявлении нет необходимости, поскольку нумерованные символьные сущности неявно определены в XML как ссылки на символы в текущем наборе символов (подробнее о наборах символов рассказано в главе 7 «Поддержка многоязычности»). Анализатор XML просто заменяет сущность правильным символом.

После разрешения сущностей предыдущий пример выглядит так:

```
<?xml version="1.0"?>
<!DOCTYPE message SYSTEM "/xmlstuff/dtds/message.dtd">
<message>
<opening>Dear Mr. Rufus Xavier Sasperilla</opening>
<body>We have an exciting opportunity for you! A set of
ocean-front cliff dwellings in Picata, Mexico have been
renovated as time-share vacation homes. They're going fast! To
reserve a place for your holiday, call Ms. Sally Tashuns at
<number>617-555-1299</number>.
Hurry, Mr. Rufus Xavier Sasperilla. Time is running out!</body>
</message>
```

Все сущности (кроме предопределенных) должны быть объявлены раньше, чем они использованы в документе. Есть два места, где их можно определить: во внутреннем подмножестве, что идеально для локальных сущностей, и во внешнем DTD, что более подходит для сущностей, совместно используемых в различных документах. Если анализатор находит сущность, которая не была определена неявным (предоп-

ределенная сущность) или явным образом, он не может вставить в документ текст замены, потому что не знает, чем нужно заменить сущность. Если возникает такая ошибка, документ не считается корректным (well-formed).

Символьные сущности

Сущности, содержащие единственный символ, естественно, называются *символьными сущностями* (*character entities*). Они разделяются на несколько групп:

Предопределенные символьные сущности

Некоторые символы нельзя использовать в тексте документа XML, поскольку они конфликтуют со специальными разделителями разметки. Например, угловые скобки (< >) предназначены для разграничения тегов элементов. В спецификации XML описаны следующие *предопределенные символьные сущности* (*predefined character entities*) для безопасного использования этих символов:

Имя	Значение
amp	&
apos	'
gt	>
lt	<
quot	"

Нумерованные символьные сущности

XML поддерживает Unicode, огромный набор символов с десятками тысяч различных знаков, букв и идеограмм. Разработчик должен иметь возможность использовать в своем документе любой символ Unicode. Проблема в том, как ввести нестандартный символ с клавиатуры, имеющей менее 100 клавиш, и как отобразить его в чисто текстовом редакторе. Одно из решений – использовать *нумерованные символьные сущности* (*numbered character entities*), т. е. сущности, имена которых имеют вид `#n`, где *n* является числом, представляющим позицию символа в наборе символов Unicode.

Число в имени сущности может иметь десятичный или шестнадцатеричный формат. Например, «с» с седилем (ç) на нижнем регистре (`&#231;`) является 231-м символом Unicode. Эту букву можно представить в десятичном виде как `&#231;` и в шестнадцатеричном как `&#xe7;`. Как видите, шестнадцатеричная версия отличается тем, что число имеет префикс x. Диапазон символов, которые можно представить таким образом, простирается от нуля до 65536. Более

подробно наборы символов и кодировки будут обсуждаться в главе 7 «Поддержка многоязычности».

Именованные символьные сущности

Проблема нумерованных символьных сущностей в том, что их трудно запомнить: всякий раз, сталкиваясь со специальным символом, приходится справляться в таблице. Легче запоминаются мнемонические имена сущностей. Такие *именованные символьные сущности* (*named character entities*) используют для ссылок легко запоминаемые имена, например, `&THORN;`, обозначающее заглавный символ «торн» (Þ) исландского языка.

В отличие от предопределенных и нумерованных символьных сущностей, именованные символьные сущности нужно объявлять. На практике они формально не отличаются от остальных общих сущностей. Тем не менее, полезно делать различие между ними, потому что большие группы таких сущностей определены в модулях DTD, которые разработчики могут использовать в своих документах. Примером служит ISO-8879, стандартизованный набор именованных символьных сущностей, включающий латинское, греческое, скандинавское и кириллическое письмо, математические символы и другие полезные символы для европейских документов.

Сущности со смешанным содержанием

Значения сущностей, конечно, не ограничиваются отдельными символами. Более общие *сущности со смешанным содержанием* (*mixed-content entities*) имеют значения неограниченной длины и могут содержать как разметку, так и текст. Эти сущности распадаются на две категории: внутренние и внешние. Для *внутренних сущностей* (*internal entities*) заменяющий текст определяется в объявлении сущности; для *внешних сущностей* (*external entities*) он находится в другом файле.

Внутренние сущности

Внутренние сущности со смешанным содержанием чаще всего применяются для замены часто повторяемых фраз, имен или стандартного текста. Ссылки на сущность не только легче вводить, чем длинные фрагменты текста, они также улучшают точность и облегчают сопровождение, поскольку достаточно изменить сущность в одном случае, и результат отразится сразу везде. Это иллюстрируется следующим примером:

```
<?xml version="1.0"?>
<!DOCTYPE press-release SYSTEM "http://www.dtdland.org/dtds/reports.dtd"
[
  <!ENTITY bobco "Bob's Bolt Bazaar, Inc.">
]>
```

```
<press-release>
<title>&bobco; Earnings Report for Q3</title>
<par>The earnings report for &bobco; in fiscal
quarter Q3 is generally good. Sales of &bobco; bolts increased 35%
over this time a year ago.</par>
<par>&bobco; has been supplying high-quality bolts to contractors
for over a century, and &bobco; is recognized as a leader in the
construction-grade metal fastener industry.</par>
</press-release>
```

Сущность `&bobco;` встречается в документе пять раз. Если требуется что-нибудь изменить в названии компании, изменение нужно сделать только в одном месте. Например, для того чтобы название появлялось внутри элемента `<companyname>`, просто отредактируйте объявление сущности:

```
<!ENTITY bobco
  "<companyname>Bob's Bolt Bazaar, Inc.</companyname>">
```

Включая в объявления сущностей разметку, следите за тем, чтобы не использовать предопределенные символьные сущности (например, `<` и `>`). Анализатор умеет читать разметку как значение сущности, поскольку значение цитируется внутри объявления сущности. Исключения составляют сущность символа кавычки `&quot;` и символа одиночной кавычки `&apos;`. Если они вступят в конфликт с разделителями значения в объявлении сущности, то используйте предопределенные сущности, например, если значение заключено в двойные кавычки и требуется, чтобы оно содержало двойные кавычки.

Сущности могут содержать ссылки на другие сущности, если эти последние объявлены ранее. Следите за тем, чтобы не включать ссылки на сущность, которая объявляется, т. к. тогда будет создана рекурсивная схема, и анализатор может зациклиться. Некоторые анализаторы отслеживают зацикленные ссылки, но они тем не менее являются ошибкой.

Внешние сущности

Иногда может понадобиться создать сущность для такого большого объема смешанного содержимого, что непрактично заключать его внутрь объявления сущности. В таком случае следует использовать *внешние сущности* (*external entity*), для которых текст замены находится в другом файле. Внешние сущности полезны для импортирования содержимого, которое совместно используют многие документы или которое изменяется слишком часто, чтобы его можно было хранить внутри документа. Они позволяют также разбить большой монолитный документ на меньшие части, которые можно редактировать последовательно и которые занимают меньше места при передаче по сети. На рис. 2.15 показана возможность импортирования в документ фрагментов XML и текста.

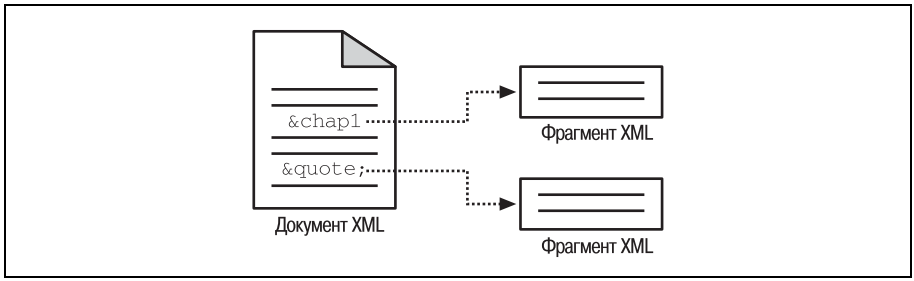


Рис. 2.15. *Использование внешних сущностей для импорта XML и текста*

Внешние сущности фактически разбивают документ на несколько физических частей. Однако для процессора XML важно только, чтобы части собирались в совершенное целое. Это значит, что все части, находящиеся в разных местах, по-прежнему должны удовлетворять правилам корректности (well-formedness rules). Анализатор XML собирает из всех кусочков один логический документ; при правильной разметке физическое разделение его не должно влиять на значение документа.

Внешние сущности являются механизмом связывания. Они соединяют части документа, которые могут находиться на других системах, совсем в другой части Интернета. Отличие от обычных ссылок XML (XLinks) состоит в том, что для внешних ссылок процессор XML должен вставлять заменяющий текст на этапе синтаксического анализа. Информацию о других типах ссылок можно найти в главе 3 «Соединение ресурсов с помощью ссылок».

Внешние сущности всегда должны быть объявлены, чтобы анализатор знал, где искать текст замены. В следующем примере в документе объявлены три внешние сущности `part1`, `part2` и `part3`, в которых хранится его содержание:

```
<?xml version="1.0"?>
<!DOCTYPE longdoc SYSTEM "http://www.dtds-r-us.com/generic.dtd"
[
  <!ENTITY part1 SYSTEM "p1.xml">
  <!ENTITY part2 SYSTEM "p2.xml">
  <!ENTITY part3 SYSTEM "p3.xml">
]>
<longdoc>
  &part1;
  &part2;
  &part3;
</longdoc>
```

Этот процесс проиллюстрирован на рис. 2.16. Файл на вершине пирамиды содержит объявления документа и ссылки на внешние сущности, поэтому можно назвать его «главным файлом». Другие файлы яв-

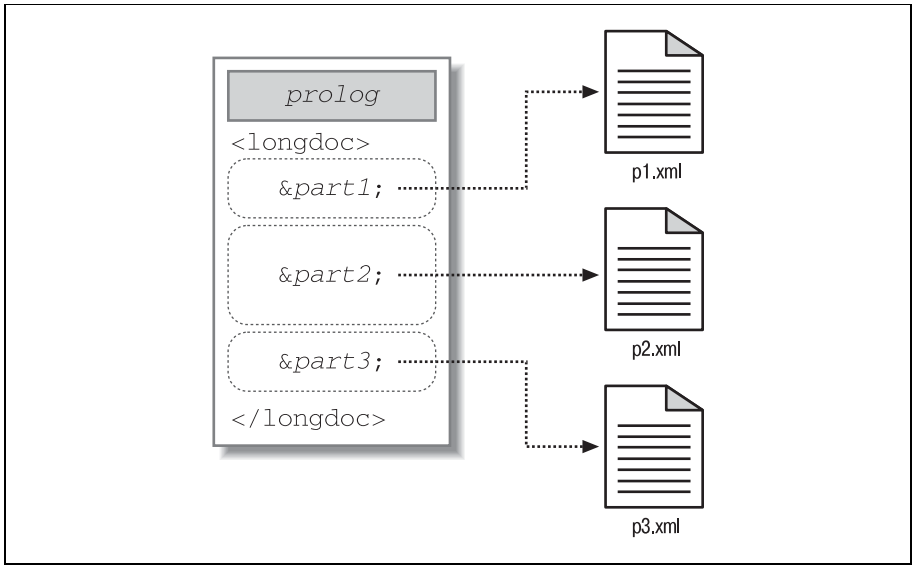


***Рис. 2.16.** Составной документ*

ляются поддокументами, частями XML, которые сами по себе не являются документами. Включение пролога документа в каждый поддокумент было бы ошибкой, поскольку в результате мы не получили бы единого логического документа.

Поскольку поддокументы это лишь фрагменты документов и в них отсутствуют собственные прологи документов, нельзя проводить проверку действительности для каждого из них в отдельности (хотя они могут являться корректными документами без прологов документа). Действительность главного файла проверять можно, поскольку его части автоматически импортируются анализатором, когда он встречает внешние сущности. Обратите, однако, внимание, что если в поддокументе есть синтаксическая ошибка, то она будет импортирована в документ в целом. Внешние сущности не защищают разработчика от ошибок анализа и не избавляют от необходимости проверки действительности.

В приведенном только что синтаксисе объявления внешних сущностей указано ключевое слово `SYSTEM`, за которым следует строка, заключенная в кавычки и содержащая имя файла. Эта строка называется *системным идентификатором* (*system identifier*) и используется для идентификации ресурса по адресу. Строка в кавычках это фактически URL, поэтому можно включать файлы, находящиеся в любом месте Интернета. Например:

```
<!ENTITY catalog SYSTEM "http://www.bobsbolts.com/catalog.xml">
```

Системный идентификатор имеет тот же недостаток, что и все URL: если ресурс, на который указывает ссылка, перемещен, ссылка разрывается. Чтобы обойти эту проблему, можно использовать открытый идентификатор (public identifier) в объявлении сущности. Теоретически, открытый идентификатор должен выдержать любое изменение адреса и все равно доставить правильный ресурс. Например:

```
<!ENTITY faraway PUBLIC "-//BOB//FILE Catalog//EN"
    "http://www.bobsbolts.com/catalog.xml">
```

Конечно, для этого процессор XML должен уметь использовать открытые идентификаторы и находить каталог, в котором они отображаются в фактические адреса. Кроме того, нет гарантии, что каталог не устарел. Ошибка может возникнуть по многим причинам. Вероятно, поэтому открытый идентификатор должен сопровождаться системным идентификатором (в данном случае, `"http://www.bobsbolts.com/catalog.xml"`). Если по какой-то причине процессор XML не может обработать открытый идентификатор, он осуществляет откат к системному идентификатору. Большинство имеющихся сегодня веб-броузеров не могут работать с открытыми идентификаторами, поэтому резервирование является, вероятно, хорошей идеей.

Неанализируемые сущности

Последним видом сущностей, обсуждаемых в этой главе, являются *неанализируемые сущности* (*unparsed entity*). Сущности этого вида хранят содержание, которое не должно анализироваться, поскольку оно не является текстом и может вызвать замешательство анализатора. Неанализируемые сущности применяются для импорта графики, звуковых файлов и других несимвольных данных.

Объявление неанализируемой сущности похоже на объявление внешней сущности, но в конце его находится некоторая дополнительная информация. Например:

```
<?xml version="1.0"?>
<!DOCTYPE doc [
  <!ENTITY mypic SYSTEM "photos/erik.gif" NDATA GIF>
]>
<doc>
  Here's a picture of me:
  &mypic;
</doc>
```

Отличие этого объявления от объявления внешней сущности в том, что за данными о системном пути расположено ключевое слово `NDATA`. Это ключевое слово указывает анализатору, что содержимое сущности имеет особый формат, или *нотацию* (*notation*), отличный от обычного анализируемого смешанного содержимого. За ключевым словом `NDATA`

следует *идентификатор нотации* (*notation identifier*), в котором указан формат данных. В данном случае сущность является графическим файлом в формате GIF, поэтому соответствующим идентификатором служит `GIF`.

Идентификатор нотации должен быть объявлен в отдельном *объявлении нотации* (*notation declaration*), что является сложной операцией, описываемой в главе 5 «Модели документов: более высокий уровень контроля». Нотации, такие как GIF и другие, не являются встроенными элементами XML, и процессор XML может не знать, что с ними делать. Во всяком случае, анализатор не станет бездумно загружать содержимое сущности и пытаться его анализировать, что в некоторой мере защищает от ошибок.

Разная разметка

Элементы, атрибуты, пространства имен и сущности являются самыми важными объектами разметки, но на них история не заканчивается. Другие объекты разметки, в том числе комментарии, инструкции обработки и разделы CDATA, различными способами скрывают содержимое от анализатора, позволяя включать специализированную информацию.

Комментарии

Комментарии – это заметки в документе, не интерпретируемые анализатором. Если разные люди работают над одними и теми же файлами, то такие данные могут оказаться очень ценными. Их можно использовать для указания назначения файлов и их разделов, чтобы облегчить перемещение по сложному документу, или просто для связи друг с другом. Поэтому в XML есть особый вид разметки, называемый *комментарием*. Синтаксис комментариев показан на рис. 2.17.

Комментарий начинается с четырех символов: открывающей угловой скобки, восклицательного знака и двух черточек (1). Завершается он двумя черточками и закрывающей угловой скобкой (3). Между этими ограничителями находится содержимое, которое должно игнориро-

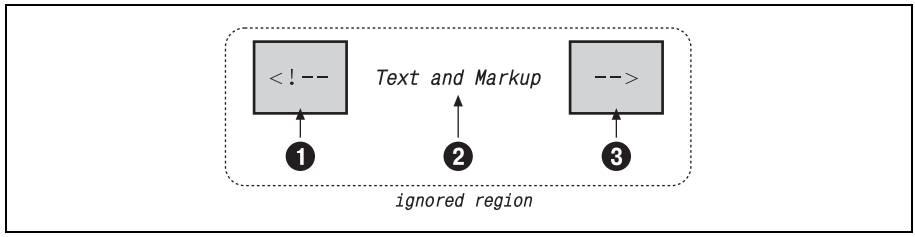


***Рис. 2.17.** Синтаксис комментариев*

ваться (2). В комментарий можно поместить почти любой желаемый текст, включая пробелы, перевод строки и разметку. Однако, поскольку два дефиса подряд (- -) используются для того, чтобы указывать анализатору, где начинается и кончается комментарий, их нельзя произвольно располагать в комментарии. Это означает, что для создания хорошо заметной линии вместо дефисов нужен другой символ, например, знак равенства (=) или символ подчеркивания (_):

```
Хорошо:  <!--============================================================-->
Хорошо:  <!--____________________________________________________________-->
Хорошо:  <!-- - - - - - - - - - -  - - - - - - - - - - - - - - - - - - -->
Плохо:   <!------------------------------------------------------------->
Плохо:   <!--                -- Так не делайте! --                      -->
```

Комментарии можно помещать в любое место документа, но не перед объявлением XML и не внутри тегов; анализатор XML полностью их игнорирует. Поэтому фрагмент XML

```
<p>The quick brown fox jumped<!-- test -->over the lazy dog.
The quick brown <!-- test --> fox jumped over the lazy dog. The<!-

test

-->quick brown fox
jumped over the lazy dog.</p>
```

после удаления анализатором комментариев превращается в следующий:

```
<p>The quick brown fox jumpedover the lazy dog.
The quick brown  fox jumped over the lazy dog. Thequick brown fox
jumped over the lazy dog.</p>
```

Поскольку комментарии могут содержать разметку, с их помощью иногда «выключают» части документа. Это ценно, если требуется временно удалить какой-то раздел, сохранив его в файле для дальнейшего использования. В следующем примере закомментирована область документа:

```
<p>Our store is located at:</p>
<!--
<address>59 Sunspot Avenue</address>
-->
<address>210 Blather Street</address>
```

Применяя данный прием, следите за тем, чтобы не закомментировать комментарии, то есть не создавать вложенных комментариев. Содержащиеся в них двойные дефисы вызовут сообщение анализатора об ошибке, когда он доберется до внутренних комментариев.

Разделы CDATA

Тем, кто часто пользуется разметкой символов в тексте, может показаться утомительной работа с предопределенными сущностями `<`, `>`, `&`. Их сложнее вводить с клавиатуры и, в целом, они трудно читаются в разметке. Однако есть другой способ ввода большого количества запрещенных символов: раздел CDATA.

CDATA служит акронимом для «character data», что просто значит «не разметка» (символьные данные). В сущности, мы сообщаем анализатору, что данный раздел документа не содержит разметки и должен рассматриваться как обычный текст. Единственное, что нельзя включать внутрь раздела CDATA, это завершающий ограничитель (`]]>`). Чтобы все же включить его, нужно прибегнуть к предопределенной сущности и записать как `]]>`.

Синтаксис раздела CDATA показан на рис. 2.18. Раздел CDATA начинается с состоящего из девяти символов ограничителя `<![CDATA[` (1) и заканчивается ограничителем `]]>` (3). Содержимое раздела (2) может содержать символы разметки (`<`, `>` и `&`), но они игнорируются процессором XML.

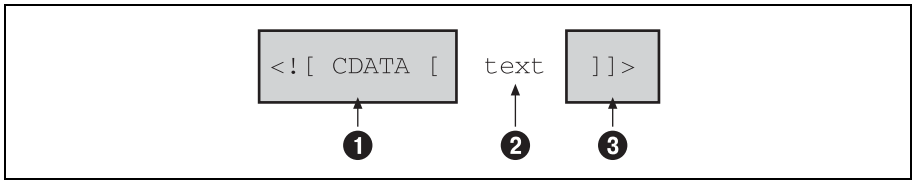


***Рис. 2.18.** Синтаксис раздела CDATA*

Вот пример раздела CDATA:

```
<para>Then you can say <![CDATA[if (&x < &y)]]> and be done
with it.</para>
```

Разделы CDATA удобнее всего применять для областей большого размера, скажем, маленькой компьютерной программы. Если часто использовать их для небольших фрагментов текста, документ станет трудно читаемым, поэтому больше подойдут ссылки на сущности.

Инструкции обработки

Информация, касающаяся представления, должна, по возможности, храниться вне документа. Однако бывают случаи, когда иного выбора нет, например, если в документе требуется запомнить номера страниц для создания указателя. Такая информация применима только для конкретного процессора XML и может быть неуместной или вводящей в заблуждение для других. Для такого рода данных предписывается ис-

пользовать *инструкции обработки* (*processing instruction*). Это контейнер для данных, предназначенных определенному процессору XML.

Инструкция обработки (PI) состоит из двух частей: ключевого слова получателя и некоторых данных. Анализатор передает эти инструкции на следующий уровень обработки. Если обработчик инструкций распознает ключевое слово получателя (target), он может решить обработать данные, в противном случае данные отбрасываются. При этом создатель документа определяет, каким образом данные участвуют в обработке.

На рис. 2.19 показан синтаксис PI. Инструкция обработки начинается с двухсимвольного ограничителя (1), которым являются открывающая угловая скобка и вопросительный знак (<?), затем следует *получатель* (2), необязательная строка символов, являющаяся секцией данных PI (3), и закрывающий ограничитель (4), состоящий из вопросительного знака и закрывающей угловой скобки (?>).

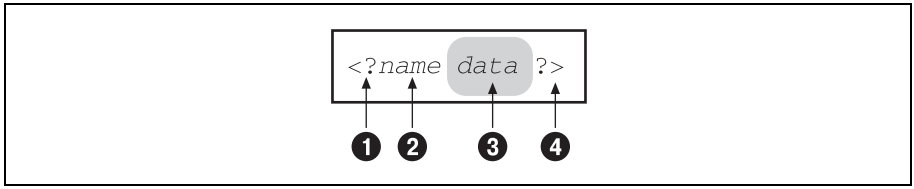


Рис. 2.19. *Синтаксис инструкции обработки*

«Однако», – скажет читатель, – «PI выглядят очень похоже на объявление XML». И будет прав: объявление XML можно рассматривать как инструкцию обработки для всех процессоров XML,[1] которая объявляет всем общие сведения о документе.

Получатель является ключевым словом, по которому процессор XML определяет, предназначены эти данные ему или другому приложению. Ключевое слово не обязательно имеет определенное значение вроде названия использующего его программного обеспечения. Инструкцию обработки могут выполнять несколько программ, и одна программа может принимать несколько инструкций обработки. В качестве аналогии можно привести объявление, в котором сказано, что «Вечеринка переместилась в оранжерею», и те, кого это интересует, последуют туда, а остальные – нет.

Инструкция обработки может содержать любые данные, кроме символов ?>, интерпретируемых как закрывающий разделитель. Вот некоторые примеры допустимых PI:

[1] Этот синтаксический прием позволяет легко обрабатывать документы XML с помощью старых системам SGML: они просто рассматривают объявление XML как некую инструкцию обработки, которая не игнорируется только процессорами XML.

```
<?flubber pg=9 recto?>
<?thingie?>
<?xyz stop: the presses?>
```

Если строки данных нет, ключевое слово получателя может само выступать в качестве данных. Хорошим примером служит принудительный перенос строки. Представьте себе, что есть длинный заголовок раздела, не умещающийся в ширину страницы. Не полагаясь на автоматическое средство форматирования (formatter), которое перенесет заголовок в произвольном месте, мы потребуем переноса в определенном месте.

Вот как будет выглядеть принудительный перенос строки:

```
<title>The Confabulation of Branklefitzers <?lb?>in a Portlebunky
Frammins <?lb?>Without Denaculization of <?lb?>Crunky Grabblefooties
</title>
```

Корректные документы

XML предоставляет пользователям мощные возможности выбора собственных типов элементов и изобретения собственных грамматик для создания индивидуальных языков разметки. Но такая гибкость представляет опасность для анализаторов XML, если отсутствуют некоторые минимальные правила, которые их защищают. Анализатор, предназначенный для какого-то одного языка разметки, такой как броузер HTML, может разрешать некоторую небрежность в разметке, т. к. набор тегов невелик и веб-страница не очень сложна. Поскольку процессоры XML должны быть готовы к работе с любыми видами языков разметки, необходим набор базовых правил.

Эти правила являются очень простыми ограничениями на синтаксис. Все теги должны использовать правильные ограничители; закрывающий тег должен находиться после открывающего тега; элементы не могут перекрываться – и так далее. Документы, которые удовлетворяют этим правилам, называются *корректными* (*well-formed*). Ниже перечислены некоторые из этих правил.

Первое правило гласит, что у элемента, содержащего текст или элементы, должны иметься открывающий и закрывающий теги.

Правильно

```
<list>
  <listitem>soupcan</listitem>
  <listitem>alligator</listitem>
  <listitem>tree</listitem>
</list>
```

Неправильно

```
<list>
  <listitem>soupcan
  <listitem>alligator
  <listitem>tree
</list>
```

В теге пустого элемента перед закрывающей скобкой должна стоять косая черта (/).

Правильно

```
<graphic filename="icon.png"/>
```

Неправильно

```
<graphic filename="icon.png">
```

Все значения атрибутов должны заключаться в кавычки.

Правильно

```
<figure filename="icon.png"/>
```

Неправильно

```
<figure filename=icon.png/>
```

Элементы не могут перекрываться.

Правильно

```
<a>A good <b>nesting</b>
example.</a>
```

Неправильно

```
<a>This is <b>a poor</a>
  nesting scheme.</b>
```

Отдельные символы разметки не могут появляться в анализируемом содержимом. В их число входят <,]]> и &.

Правильно

```
<equation>5 &lt; 2</equation>
```

Неправильно

```
<equation>5 < 2</equation>
```

Последнее правило определяет, что имена элементов могут начинаться только с букв и символов подчеркивания и могут содержать только буквы, цифры, дефисы, точки и символы подчеркивания. Пространства имен задаются с помощью двоеточия.

Правильно

```
<example-one>
<_example2>
<Example.Three>
```

Неправильно

```
<bad*characters>
<illegal space>
<99number-start>
```

Зачем все эти правила?

Веб-разработчики, «собаку съевшие» на HTML, заметят, что синтаксические правила XML значительно более строги, чем в HTML. К чему все эти сложности с корректностью документов? Нельзя ли сделать анализаторы достаточно сообразительными, чтобы они могли разобраться самостоятельно? Возьмем, к примеру, требование наличия закрывающих тегов для каждого элемента-контейнера. В HTML закрывающие теги иногда опускаются, а принятие решения о том, где заканчивается элемент, предоставляется броузеру:

```
<body>
  <p>This is a paragraph.
  <p>This is also a paragraph.
</body>
```

В HTML это допускается, поскольку никакой неоднозначности в отношении элемента <p> не возникает. HTML не позволяет, чтобы <p> находился внутри другого <p>, поэтому ясно, что они находятся на одном уровне. Все анализаторы HTML обладают встроенным знанием HTML, которое называется *грамматикой*. В XML, где грамматика не является чем-то незыблемым, может возникнуть неоднозначность:

```
<blurbo>This is one element.
<blurbo>This is another element.
```

Находится ли второй элемент <blurbo> на том же уровне, что и первый, или является его потомком? Выяснить это невозможно, поскольку о модели содержимого этого элемента ничего не известно. XML не требует наличия DTD, в котором определяется грамматика, поэтому анализатор тоже не может дать ответ. Анализаторам XML приходится работать в отсутствие грамматики, поэтому разработчики должны правильно строить документы, компенсируя эту «расслабленность» анализаторов.

Как наилучшим образом использовать разметку

В настоящее время поставщики программного обеспечения все чаще заявляют, что их продукты являются «XML-совместимыми». Звучит впечатляюще, но есть ли повод для восторга? Конечно, корректный (well-formed) XML обеспечивает некоторые минимальные стандарты качества; однако этим дело далеко не кончается. XML сам по себе яв-

ляется не языком, а набором правил для разработки языков разметки. Поэтому, пока не станет ясно, какого рода язык создали поставщики для своих продуктов, следует относиться к таким заявлениям с осторожным оптимизмом.

Правда заключается в том, что многие языки разметки, построенные на основе XML, отвратительны. Часто разработчики не слишком продумывают структуру данных документа, и в результате их разметка выглядит, как те же файлы неорганизованных исходных данных с различными тегами. Хороший язык разметки имеет продуманную конструкцию, разумно использует контейнеры и атрибуты, понятно именует объекты и имеет логичную иерархическую структуру.

Вот показательный пример. Хорошо известная издательская программа может выводить свои данные в виде XML. Однако полезность этого ограничена из-за серьезной проблемы: иерархическая структура очень плоская. Нет секций или разделов, которые могли бы содержать параграфы и более мелкие разделы; все параграфы находятся на одном и том же уровне, а заголовки разделов суть просто украшенные параграфы. Сравните это с таким языком XML, как DocBook (см. раздел «Приложение XML: DocBook» ниже в этой главе), в котором вложенные элементы используются для представления связей, а именно, проясняют, что некоторые области текста находятся внутри конкретных разделов. Такая информация важна для установки стилей в таблицах стилей или выполнения преобразований.

Другой язык разметки применяется для кодирования маркетинговой информации, размещаемой в электронных книгах. Недостатком его конструкции является невразумительность и бесполезность схемы именования элементов. Элементы, предназначенные для хранения такой информации, как ISBN или заглавие документа, названы `<A5>`, `<B2>` или `<C1>`. Эти имена никак не связаны с назначением элементов, а имена `<isbn>` и `<title>` были бы легко понятны.

Элементы – это первая забота в хорошем языке разметки. Они могут нести много информации различными способами:

Тип

Имя, находящееся внутри открывающего и закрывающего тегов элемента, отличает его от других типов и указывает программам XML, как его обрабатывать. Имена должны отражать назначение элемента в документе и быть читаемыми как для человека, так и для машины. Выбирайте имена по возможности более описательные и узнаваемые, например, `<model>` или `<programlisting>`. Придерживайтесь соглашения по использованию только букв нижнего регистра и избегайте чередования регистров (например, `<OrderedList>`), т. к. люди часто забывают, когда какой регистр нужно выбирать. Не поддавайтесь желанию использовать общие типы элементов, которые могут иметь практически любое содержание. А

всякого, кто выбирает бессмысленные имена вроде <XjKnpl> или <J-9>, следует вывести на улицу и забросать пончиками.

Содержимое

Содержимое элемента может включать в себя символы, элементы или то и другое вперемешку. Элементы внутри смешанного содержимого модифицируют символьные данные (например, помечая слово, которое должно быть выделено) и называются *внутритекстовыми* (*inline*). Другие элементы применяются для деления документа на части и часто называются *составными элементами* (*components*) или *блоками* (*blocks*). В символьных данных обычно имеют значение пробельные символы, в отличие от HTML и других языков разметки.

Позиция

Важна позиция одного элемента внутри другого. Порядок элементов всегда сохраняется, поэтому можно описать последовательность объектов, такую как нумерованный список. Можно использовать элементы, часто без содержимого, чтобы пометить место в тексте, например, для вставки графики или сноски. Участок текста можно отметить при помощи двух элементов, если применение одного элемента с этой целью вызывает неудобства.

Иерархия

Предки элемента тоже могут нести информацию. Например, содержимое элемента <title> (заголовок) форматируется по-разному в зависимости от нахождения его внутри элемента <chapter> (глава), элемента <section> (раздел) или элемента <table> (таблица) с использованием разных гарнитур и кеглей. Таблицы стилей могут использовать сведения о родительских элементах, принимая решение о выборе способа обработки элемента.

Пространство имен

С помощью пространств имен элементы можно разбить на категории соответственно их источнику или назначению. Например, в XSLT элементы пространства имен xsl используются для управления процессом преобразования, тогда как другие элементы служат просто данными для создания дерева результата. Некоторые веббраузеры могут обрабатывать документы, использующие несколько пространств имен, например, в Amaya на страницах HTML поддерживаются уравнения MathML. В обоих случаях пространство имен помогает процессору XML решить, как обрабатывать элементы.

Второе, что следует принимать во внимание в хорошем языке разметки, это атрибуты. Не злоупотребляйте ими, поскольку они загромождают разметку, но все-таки используйте, если в них есть нужда. Атрибут передает специфическую информацию об элементе, которая помогает указать на его роль в документе. Он не предназначен для хране-

ния содержимого. Иногда трудно решить, что именно использовать: атрибут или дочерний элемент. Руководствоваться можно примерно следующими принципами.

Используйте элемент, если:

- Содержимое не ограничивается несколькими словами. Некоторые анализаторы XML ограничивают максимально допустимое количество символов, которое может содержать атрибут. Кроме того, длинные значения атрибутов тяжело читаются.
- Имеет значение порядок. Порядок атрибутов в элементе игнорируется, но порядок элементов имеет значение.
- Информация является частью содержимого документа, а не просто параметром для настройки поведения документа. В случае, если процессор XML не может обработать какой-то документ (вероятно, потому, что не полностью поддерживает таблицу стилей), атрибуты не выводятся, в то время как содержимое элемента выводится «как есть». В этом случае по крайней мере документ можно прочесть (если использован элемент, а не атрибут).

Выбирайте атрибут, если:

- Информация «тонко» модифицирует элемент, влияя на обработку, но не являясь частью содержимого. Например, когда нужно задать особый вид маркера для маркированного списка:

```
<bulletlist bullettype="filledcircle">
```

- Желательно ограничить возможные значения. С помощью DTD можно гарантировать, что атрибут будет принадлежать предопределенному множеству значений.
- Информация является уникальным идентификатором или ссылкой на идентификатор в другом элементе. XML предоставляет особые механизмы для проверки идентификаторов в атрибутах, чтобы обеспечить целостность ссылок. Подробнее этот тип ссылок рассмотрен в разделе «Создание внутренних ссылок с помощью ID и IDREF» главы 3.

Инструкции обработки следует применять как можно реже. Обычно они содержат управляющую информацию, не относящуюся к какому-либо одному элементу, а используемую каким-то конкретным процессором XML. Например, PI позволяют запомнить, в каких местах необходимо делать разрыв страницы при выводе на принтер, но в варианте документа для Интернета это бесполезно. Неправильно слишком полагаться в языке разметки на инструкции обработки.

Несомненно, вы столкнетесь как с хорошими, так и с плохими примерами разметки XML, но не обязательно повторять самому чужие ошибки. Старайтесь как можно лучше продумать собственную конструкцию.

Приложение XML: DocBook

Приложение XML (*application*) – это язык разметки, полученный на основе правил XML, что не следует путать с программными приложениями XML, которые в данной книге называются *процессорами* XML. Приложение XML часто является самостоятельным стандартом, и его DTD общедоступно. Одним из таких приложений является DocBook, язык разметки технической документации.

DocBook – это большой язык разметки, состоящий из нескольких сотен элементов. Он был разработан консорциумом компаний и организаций для решения обширного множества задач технической документации. DocBook достаточно гибок, чтобы кодировать все – от одной странички руководства до многотомных изданий. В настоящее время DocBook имеет широкую базу пользователей, включая разработчиков программ с открытым исходным текстом (open source developers) и издателей. Подробности о стандарте DocBook можно найти в приложении B «Таксономия стандартов».

Пример 2.4 представляет документ DocBook, в данном случае это руководство по эксплуатации промышленного изделия. (Фактически, в нем используется DTD с названием «Barebones DocBook», сходная с версией DocBook, которая описана в главе 5, но значительно уступающая ей по размеру). По тексту примера расставлены числовые маркеры, соответствующие комментариям, имеющимся в конце.

Пример 2.4. Документ DocBook

```
<?xml version="1.0" encoding="utf-8"?> (1)
<!DOCTYPE book SYSTEM "/xmlstuff/dtds/barebonesdb.dtd" (2)
[
  <!ENTITY companyname "Cybertronix">
  <!ENTITY productname "Sonic Screwdriver 9000">
]>

<book> (3)
  <title>&productname; User Manual</title> (4)
  <author>Indigo Riceway</author>

  <preface id="preface">
    <title>Preface</title>

    <sect1 id="about">
      <title>Availability</title>

<!-- Note to author: maybe put a picture here? -->

      <para> (5)
The information in this manual is available in the following forms:
      </para>
```

Пример 2.4. Документ DocBook (продолжение)

```
      <itemizedlist> (6)
        <listitem><para>
Instant telepathic injection
        </para></listitem>
        <listitem><para>
Lumino-goggle display
        </para></listitem>
        <listitem><para>
Ink on compressed, dead, arboreal matter
        </para></listitem>
        <listitem><para>
Cuneiform etched in clay tablets
        </para></listitem>
      </itemizedlist>

      <para>
The &productname; is sold in galactic pamphlet boutiques or wherever
&companyname; equipment can be purchased. For more information, or
to order a copy by hyperspacial courier, please visit our universe-wide
Web page at <systemitem (7)
role="url">http://www.cybertronix.com/sonic_screwdrivers.html</systemitem>.
      </para>
    </sect1>

    <sect1 id="disclaimer"
      <title>Notice</title>
      <para>
While <emphasis>every</emphasis> (8)
effort has been taken to ensure the accuracy and
usefulness of this guide, we cannot be held responsible for the
occasional inaccuracy or typographical error.
      </para>
    </sect1>
  </preface>


  <chapter id="intro"> (9)
    <title>Introduction</title>

    <para>
Congratulations on your purchase of one of the most valuable tools in
the universe! The &companyname; &productname; is
equipment no hyperspace traveller should be without. Some of the
myriad tasks you can achieve with this device are:
    </para>

    <itemizedlist>
      <listitem><para>
Pick locks in seconds. Never be locked out of your tardis
```

Пример 2.4. Документ DocBook (продолжение)

```
again. Good for all makes and models including Yale, Dalek, and
Xngfzz.
      </para></listitem>
      <listitem><para>
Spot-weld metal, alloys, plastic, skin lesions, and virtually any
other material.
      </para></listitem>
      <listitem><para>
Rid your dwelling of vermin. Banish insects, rodents, and computer
viruses from your time machine or spaceship.
      </para></listitem>
      <listitem><para>
Slice and process foodstuffs from tomatoes to brine-worms. Unlike a
knife, ther is no blade to go dull.
      </para></listitem>
    </itemizedlist>
    <para>
Here is what satisfied customers are saying about their &companyname;
&productname;:
    </para>

    <comment> (10)
Should we name the people who spoke these quotes?  --Ed.
    </comment>

    <blockquote>
      <para>
<quote>It helped me escape from the prison planet Garboplactor VI. I
wouldn't be alive today if it weren't for my Cybertronix 9000.</quote>
      </para>
    </blockquote>

    <blockquote>
      <para>
<quote>As a bartender, I have to mix martinis <emphasis>just
right</emphasis>. Some of my customers get pretty cranky if I slip
up. Luckily, my new sonic screwdriver from Cybertronix is so accurate,
it gets the mixture right every time. No more looking down the barrel
of a kill-o-zap gun for this bartender!</quote>
      </para>
    </blockquote>


  </chapter>

  <chapter id="controls">
    <title>Mastering the Controls</title>

    <sect1>
```

Пример 2.4. Документ DocBook (продолжение)

```
      <title>Overview</title>

      <para>
<xref linkend="controls-diagram"/> is a diagram of the parts of your
&productname;.
      </para>

      <figure id="controls-diagram"> (11)
        <title>Exploded Parts Diagram</title>
        <graphic fileref="parts.gif"/>
      </figure>

      <para>
<xref linkend="controls-table"/> (12)
lists the function of the parts labeled in the diagram.
      </para>

      <table id="controls-table"> (13)
        <title>Control Descriptions</title>
        <tgroup cols="2">
          <thead>
            <row>
              <entry>Control</entry>
              <entry>Purpose</entry>
            </row>
          </thead>
          <tbody>
            <row>
              <entry>Decoy Power Switch</entry>
              <entry><para>
Looks just like an on-off toggle button, but only turns on a small
flashlight when pressed. Very handy when your &productname; is misplaced
and discovered by primitive aliens who might otherwise accidentally
injure themselves.
              </para></entry>
            </row>
            <row>
              <entry><emphasis>Real</emphasis> Power Switch</entry>
              <entry><para>
An invisible fingerprint-scanning capacitance-sensitive on/off switch.
              </para></entry>
            </row>

.
.
.

            <row>
              <entry>The <quote>Z</quote> Twiddle Switch</entry>
              <entry><para>
```

Пример 2.4. Документ DocBook (продолжение)

```
We're not entirely sure what this does. Our lab testers have had
various results from teleportation to spontaneous
liquification. <emphasis role="bold">Use at your own risk!</emphasis>
              </para></entry>
            </row>
          </tbody>
        </tgroup>
      </table>

      <note>
        <para>
A note to arthropods: Stop forcing your inflexible appendages to adopt
un-ergonomic positions. Our new claw-friendly control template is
available.
        </para>
      </note>

      <sect2 id="power-sect">
        <title>Power Switch</title>

        <sect3 id="decoy-power-sect">
          <title>Why a decoy?</title>

          <comment>
Talk about the Earth's Tunguska Blast of 1908 here.
          </comment>
        </sect3>
      </sect2>
    </sect1>

    <sect1>
      <title>The View Screen</title>
      <para>
The view screen displays error messages and warnings, such as a
<errorcode>LOW-BATT</errorcode> (14)
(low battery) message.<footnote> (15)

        <para>
The advanced model now uses a direct psychic link to the user's
visual cortex, but it should appear approximately the same as the more
primitive liquid crystal display.
        </para>

</footnote> When your &productname; starts up, it should
show a status display like this:
      </para>

                  (16)
      <screen>STATUS DISPLAY
```

Пример 2.4. Документ DocBook (продолжение)

```
BATT: 1.782E8 V
TEMP: 284 K
FREQ: 9.32E3 Hz
WARRANTY: ACTIVE</screen>
    </sect1>

    <sect1>
      <title>The Battery</title>
      <para>
Your &productname; is capable of generating tremendous amounts of
energy. For that reason, any old battery won't do. The power source is
a tiny nuclear reactor containing a piece of ultra-condensed plutonium
that provides up to 10 megawatts of power to your device. With a
half-life of over 20 years, it will be a long time before a
replacement is necessary.
      </para>
    </sect1>
  </chapter>
</book>
```

Ниже следуют комментарии к примеру 2.4:

1. Объявление XML указывает, что этот файл содержит документ XML, соответствующий версии 1.0 спецификации XML, и следует использовать набор символов UTF-8 (подробнее о наборах символов сказано в главе 7). Свойство `standalone` не задано, поэтому будет использоваться значение по умолчанию «no».

2. Это объявление типа документа выполняет три задачи. Во-первых, оно сообщает нам, что корневым элементом будет `<book>`. Во-вторых, оно связывает с документом DTD, имеющее адрес */xmlstuff/dtds/barebonesdb.dtd*. В-третьих, оно объявляет две общие сущности во внутреннем подмножестве объявлений документа. Эти сущности будут использоваться в тех местах документа, где нужно указать название компании и изделия. Если в будущем изменится название изделия или компания будет продана, автору понадобится только изменить значения в объявлениях сущностей.

3. Элемент `<book>` является корнем документа – элементом, в котором хранится все содержание. Он начинает иерархию, в которую входят предисловие `<preface>` и глава `<chapter>`, затем идут какие-то разделы с метками `<sect1>`, затем `<sect2>` и т. д. до уровня параграфов и списков. В примере показаны только две главы `<chapter>`, но в реальном документе за ними последуют другие главы со своими разделами и параграфами и т. д.

4. Обратите внимание, что все основные компоненты (предисловие, главы, разделы) начинаются с элемента `<title>`. Это пример использования элемента в различных контекстах. В форматированном экземпляре этого документа заголовки на различных уровнях

будут выводиться по-разному: одни крупными, а другие – нет. Таблица стилей будет использовать информацию об иерархии (т. е. о том, кто является предком этого <title>), чтобы определить способ его форматирования.

5. Примером блочного элемента служит <para>. Блочность означает, что он начинается с новой строки и содержит смесь символьных данных и элементов, которая ограничена прямоугольной областью.

6. Этот элемент начинает маркированный список элементов. Если бы это был нумерованный список (например, <orderedlist>, а не <itemizedlist>), нам не пришлось бы вставлять номера в содержимое. Это сделала бы за нас программа форматирования XML, одновременно сохранив порядок элементов списка <listitem> и автоматически сгенерировав номера в соответствии с настройками таблицы стилей. Это еще один пример элемента (<listitem>), который поразному обрабатывается в зависимости от того, внутри какого элемента он находится.

7. Этот элемент <systemitem> является примером внутритекстового элемента, модифицирующего текст внутри потока. В данном случае он помечает свое содержимое как URL ресурса в Интернете. Процессор XML может использовать эту информацию для применения стиля (заставляя содержимое элемента выглядеть иначе, чем окружающий текст) или, на некоторых носителях, например, на экране компьютера, для превращения элемента в ссылку, по которой пользователь может щелкнуть, чтобы просмотреть ресурс.

8. Вот еще один внутритекстовый элемент, на этот раз кодирующий свой текст как требующий выделения, возможно, с помощью полужирного шрифта или курсива.

9. Элемент <chapter> имеет атрибут ID, поскольку может потребоваться добавить перекрестную ссылку на него где-нибудь в тексте. Перекрестная ссылка является пустым элементом вида:

```
<xref linkend="idref"/>
```

где *idref* является значением ID элемента, на который производится ссылка. В данном случае это может быть <xref linkend="chapt-1"/>. При форматировании документа этот элемент перекрестной ссылки будет заменен текстом, например «Chapter 1, 'Introduction'».

10. Этот блочный элемент содержит комментарий, предназначенный для редактора. В соответствии с форматированием, он будет выделен, например, более светлым оттенком. При подготовке книги к печати будет применена другая таблица стилей, которая запретит печать элементов <comment>.

11. Данный элемент <figure> содержит графический элемент и его заголовок. Элемент <graphic> является ссылкой (см. главу 3) на гра-

фический файл, который процессор XML должен будет импортировать для вывода.

⓬ Вот пример использования перекрестной ссылки на элемент <table> (атрибут linkend и атрибут ID элемента <table> совпадают). Это ссылка ID-IDREF, описываемая в главе 3. Программа форматирования заменит элемент <xref> текстом, например «Table 2-1». Теперь, если снова прочесть предложение и подставить текст вместо элемента перекрестной ссылки, оно станет осмысленным. Одна из причин, по которой вместо того, чтобы просто написать «Table 2-1», используют такой элемент перекрестной ссылки, состоит в том, что если таблица будет перемещена в другую главу, программа форматирования автоматически обновит текст.

⓭ Пример того, как в DocBook размечается таблица[1] с восемью строками и двумя колонками. Первая строка, находящаяся в <thead>, является заголовком таблицы.

⓮ Элемент <errorcode> является внутритекстовым тегом, но в данном случае он не означает особого форматирования (хотя при желании можно форматировать его специальным образом). Он помечает объект особого вида: код ошибки, используемый в компьютерной программе. В DocBook очень много особых компьютерных терминов, например, <filename>, <function> и <guimenuitem>, которые применяются как внутритекстовые элементы.

Мы хотим особо пометить такие элементы, т. к. весьма вероятно, что кому-нибудь понадобится искать в книге объекты определенного вида. Пользователь всегда может ввести ключевое слово в поисковый механизм, который найдет соответствия, но если удастся ограничить поиск только содержимым элементов <errorcode>, то с большей вероятностью будут найдены только относящиеся к делу совпадения, а не омонимы в неправильном контексте. Например, ключевое слово string встречается во многих языках программирования и может относиться к чему угодно – от части имени метода до типа данных. Поиск его в книге по Java вернет буквально сотни совпадений, поэтому для сужения поиска можно указать, что термин содержится внутри некоторого элемента, например <type>.

---

[1] На самом деле элемент <table> и все элементы внутри него основываются на другом приложении, модели таблиц CALS, являющейся более старым стандартом Министерства обороны США. Это гибкая структура для определения многих типов таблиц с диапазонами, заголовками, сносками и прочими полезными вещами. DTD DocBook импортирует DTD таблиц CALS, поэтому оно становится частью DocBook. Часто встречаются вещи, уже реализованные кем-то раньше, поэтому есть смысл импортировать их в свой проект, а не изобретать велосипед (при условии, что они общедоступны и заслуживают доверия).

⓯ Здесь мы вставили сноску. Элемент <footnote> действует одновременно как контейнер для текста и маркер, помечая некоторое место для специальной обработки. Когда документ форматируется, эта точка становится местом вставки символа сноски, например, звездочки (*). Содержимое сноски перемещается куда-то в другое место, возможно, в нижнюю часть страницы.

⓰ Элемент <screen> задан для того, чтобы сохранить все пробельные символы (символы пробелов, табуляции, перевода строки), т. к. в компьютерных программах часто содержатся дополнительные пробелы, облегчающие чтение. XML сохраняет пробельные символы во всех элементах, если не указано обратное. DocBook дает указание процессорам XML выбрасывать дополнительные пробелы во всех элементах, кроме нескольких, чтобы при форматировании документа в абзацы не содержали лишних пробелов и правильно выравнивались, тогда как в выдаче на экран и листингах программ дополнительные пробелы сохранялись.

Это был беглый взгляд на DocBook в действии. Дополнительные сведения об этом популярном приложении XML можно найти в описании, приведенном в приложении B.

- *Введение*
- *Задание ресурсов*
- *XPointer: перемещение по дереву XML*
- *Введение в XLinks*
- *Приложение XML: XHTML*

3

Соединение ресурсов с помощью ссылок

В общих чертах, *ссылка* (*link*) – это связь между двумя или более ресурсами. *Ресурс* (*resource*) может представлять собой самые разнообразные вещи: текстовый документ, написанный, например, на XML, двоичный файл (графический или звуковой) или даже сервис (скажем, канал новостей или редактор электронной почты), или компьютерную программу, динамически генерирующую данные (например, поисковую систему или интерфейс базы данных).

Чаще всего одним из этих ресурсов является документ XML. Так, чтобы включить в текст картинку, можно создать ссылку из документа на файл, эту картинку содержащий. Обнаружив эту ссылку, процессор XML найдет графический файл и отобразит его, используя информацию, содержащуюся в ссылке. Другим примером ссылки является соединение двух документов XML. Такая ссылка позволяет процессору XML отображать содержимое второго ресурса автоматически или по требованию пользователя.

Введение

С помощью ссылок можно создать целую систему взаимосвязанных информационных элементов, позволяющую увеличить ценность документа, как показано на рис. 3.1. Ссылки, изображенные на этой диаграмме, называются *простыми ссылками* (*simple links*), поскольку в них участвуют только два ресурса, по крайней мере один из которых представляет собой документ XML, и они являются однонаправленными. Вся информация для ссылки такого рода располагается внутри од-

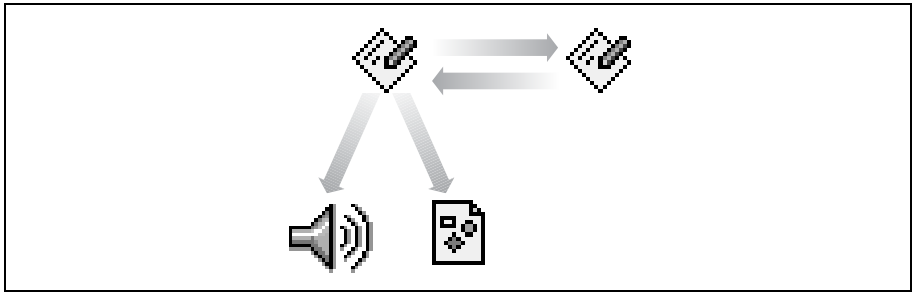


Рис. 3.1. *Группа ресурсов, соединенных ссылками*

ного элемента XML, действующего как один конец ссылки. В приведенных ранее примерах – импортирования графики и связывания двух документов XML – участвуют простые ссылки.

Более сложные ссылки могут объединять много ресурсов, а информация о ссылке может храниться в месте, не связанном с фактическим документом, на который указывает ссылка. Например, главная страница веб-сайта может определять сложную структуру навигации, что позволяет отказаться от объявления на каждой странице ссылок на другие страницы. Такая абстракция облегчает поддержку сложной системы страниц, поскольку вся информация о конфигурации существует в одном файле.

В данной книге мы сосредоточимся только на простых ссылках. Это вызвано тем, что спецификация функционирования сложных ссылок (входящая в XLink) все еще развивается и процессоров XML, которые могут их обрабатывать, не много. Однако многое можно осуществить и с помощью простых ссылок, не дожидаясь, пока будет достигнуто большее согласие относительно сложных ссылок. Например, можно делать следующее:

- Разбивать документ на несколько файлов и использовать ссылки для их соединения. Это позволяет нескольким людям одновременно работать над документом, а если разбиваются большие файлы, то тем самым уменьшается нагрузка на канал связи.

- Обеспечивать перемещение между частями документа, создавая меню важных адресов, оглавление или предметный указатель при помощи ссылок.

- Цитировать документы, находящиеся где угодно в Интернете, используя ссылки для их получения и отображения.

- Импортировать данные или текст и выводить их в документе, применяя ссылки для отображения рисунков, результатов работы программ или фрагментов других документов.

- Создавать мультимедийные презентации. Можно создавать ссылки на анимацию или звуковые клипы, чтобы включить их в свою презентацию.
- Запускать события на машине пользователя, например: создавать сообщения электронной почты, запускать программы чтения телеконференций или открывать мультимедийные каналы. Ссылка может содержать информацию о том, какое приложение должно быть использовано для обработки ресурса. Если такой информации нет, процессор XML может использовать свои настройки или системную таблицу, которая ставит в соответствие типам ресурсов (например, типам MIME) имеющиеся программные приложения.

На рис. 3.2 показана простая ссылка, состоящая из двух ресурсов, соединенных стрелкой. *Локальный* ресурс является источником ссылки, наделенным всей информацией для ее инициирования. *Удаленный* ресурс является целью ссылки. Цель – это пассивный участник, не вовлекаемый непосредственно в установку ссылки, хотя у нее может иметься идентифицирующая метка, к которой может присоединяться ссылка. Связь между ресурсами называется *дугой* (*arc*), представленной здесь в виде стрелки, показывающей, что один конец инициирует соединение с другим. Такая схема используется также в HTML для импорта изображений и создания гипертекстовых ссылок.

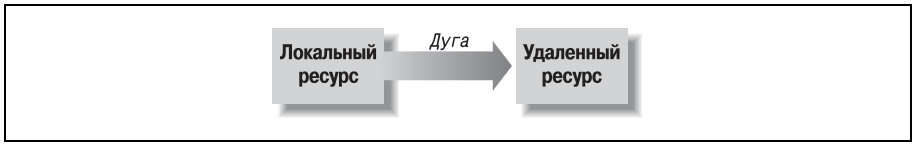


Рис. 3.2. *Простая ссылка*

Простая ссылка имеет следующие характеристики:

- В ссылке участвуют два ресурса: локальный, содержащий информацию о ссылке, и удаленный. Локальный ресурс должен располагаться в документе XML.
- Ссылка определяет *цель* (*target*), идентифицирующую удаленный ресурс.
- Режим действия ссылки определяется несколькими параметрами, выражаемыми атрибутами элемента ссылки, которые мы обсудим позднее. Вот эти параметры:
  - *Приведение ссылки в действие* (*actuation*) описывает, как она запускается. Оно может быть автоматическим, как в случае графики, импортируемой в документ, или оно может потребовать взаимодействия с пользователем, например, когда последний щелкает по гипертекстовой ссылке, указывая тем самым броузеру осуществить переход по ней.

  - Ссылка может выполнять различные действия с удаленным ресурсом, например, вставить содержимое в формате локального документа или фактически заменить локальный документ удаленным ресурсом.
- Со ссылкой может быть связана некоторая информация, например, метка или краткое описание.

Рассмотрим пример. Допустим, нужно импортировать в документ графическое изображение. Ссылка объявляется в элементе, обычно находящемся в том месте, куда требуется поместить картинку. Например:

```
<image
   xmlns:xlink="http://www.w3.org/1999/xlink"
   xlink:type="simple"
   xlink:href="figs/monkey.gif"
   xlink:show="embed"
/>
```

Первый атрибут устанавливает пространство имен с именем `xlink`, которое будет использоваться в качестве префикса для всех специальных атрибутов, описывающих ссылку. Следующий атрибут, `xlink:type`, объявляет тип ссылки как `simple`, сообщая процессору XML о том, что данный элемент определяет простую ссылку. Без этого атрибута все остальные атрибуты могут обрабатываться некорректно. Затем идет атрибут `xlink:href`, содержащий URL для получения графического файла. Наконец, атрибут `xlink:show` указывает, как должна обрабатываться ссылка – в данном случае файл должен быть загружен сразу, а его содержимое выведено в данной точке документа. Обратите также внимание, что у данного конкретного элемента ссылки нет содержимого, поскольку для загрузки ресурса не требуются действия пользователя.

В качестве другого примера рассмотрим такую ссылку:

```
<doclink
   xmlns:xlink="http://www.w3.org/1999/xlink"
   xlink:type="simple"
   xlink:href="anotherdoc.xml"
   xlink:show="replace"
   xlink:actuate="onRequest"
>click here</doclink>
for more info about stuff.
```

Разница здесь в том, что ресурс имеет тип документа XML и вместо автоматической загрузки и встраивания в документ, как в предыдущей ссылке, он заменит текущую страницу по запросу пользователя. Атрибуты `xlink:show` и `xlink:actuate` управляют стилем вывода и методом приведения в действие, соответственно. Другое различие заключается в том, что у этого элемента есть содержимое, которое можно использовать в схеме активации ссылки, например так, как действует гипер-

текстовая ссылка в броузере HTML: посредством выделения текста и превращения его в область, реагирующую на щелчок.

Задание ресурсов

Чтобы создать ссылку на объект, нужно его идентифицировать. Обычно это делается с помощью строки символов, называемой *универсальным идентификатором ресурса* (*uniform resource identifier, URI*). Существует две основные категории URI: первая уникальным образом идентифицирует ресурс по его адресу, а вторая дает ресурсу уникальное имя и использует таблицу, расположенную где-то в системе и ставящую в соответствие имена и физические адреса.

URI начинается со *схемы* (*scheme*) – краткого имени, указывающего способ идентификации объекта. Часто это протокол связи, например HTTP или FTP. За ней следуют двоеточие (:) и строка данных, однозначно идентифицирующая ресурс. Какой бы ни была схема, она должна однозначно идентифицировать ресурс.

В следующих разделах оба типа URI описаны более подробно.

Задание ресурса по адресу

Тип URI, с которым знакомо большинство – это *универсальный локатор ресурса* (*uniform resource locator, URL*), который относится к первой категории: он использует адрес для прямого указания ресурса. URL действует подобно адресу письма, в котором указаны страна, штат или область, адрес дома на улице и необязательный номер квартиры. Каждая дополнительная порция информации в адресе сужает поиск, пока он не будет сведен к одному месту; таким образом, почтовый адрес служит хорошим уникальным идентификатором.

Аналогичным образом URL использует номенклатуру компьютерных сетей. Эта информация может содержать доменное имя компьютера, путь в его файловой системе[1] и любую другую специфическую для системы информацию, помогающую найти ресурс. URL начинается со схемы, идентифицирующей используемый метод адресации или протокол связи. Определено много схем, в том числе передача гипертекста (HTTP), передача файлов (FTP) и другие. Например, в HTTP адрес

[1] Не требуется, чтобы часть URL, содержащая путь, представляла собой реальный путь в файловой системе. В некоторых схемах используется совершенно другой тип маршрута, например, иерархия ключевых слов. Но в наших примерах говорится о путях; они гораздо чаще используются для нахождения файлов в файловой системе.

URL, используемый для определения местонахождения веб-документов, выглядит так:

```
http:// address/path
```

Остальные части URL в HTTP следующие:

address

Адрес машины. Самый распространенный способ адресации машины – при помощи *доменного имени* (*domain name*), которое содержит ряд имен сетевых уровней, разделенных точками. Например, `www.oreilly.com` является доменным именем веб-сервера O'Reilly & Associates. Этот сервер находится в домене `com` – домене верхнего уровня для коммерческих сетей. Более точно, это часть поддомена `oreilly` для сети O'Reilly на машине, идентифицируемой как `www`.

path

Путь к ресурсу на машине. В компьютерной системе могут находиться многие тысячи файлов. Универсальная система адресации файлов в системе использует строку, называемую *путем* (*path*), в котором последовательно вложенные друг в друга каталоги разделяются косой чертой.[1] Например, путь */documents/work/sched.html* определяет местонахождение файла с именем *sched.html* в подкаталоге *work* основного каталога *documents*.

Вот несколько примеров URL:

```
http://www.w3c.org/Addressing/
ftp://ftp.fossil-hunters.org/pub/goodsites.pdf
file://www.laffs.com/clownwigs/catalog.txt
```

URL можно расширить так, чтобы он содержал дополнительную информацию. Идентификатор фрагмента, дописываемый в конец URL через символ решетки (`#`), ссылается на местоположение внутри файла. Его можно использовать только с некоторыми видами ресурсов, например, с документами HTML и XML. Идентификатор фрагмента должен быть объявлен внутри целевого файла в атрибуте. В HTML он называется *якорем* (*anchor*) и использует элемент `a` следующим образом:

```
<a name="ziggy">
```

В XML можно использовать атрибут ID в любом элементе:

```
<section id="ziggy">
```

[1] В разных системах определены свои правила представления внутренних путей. Например, в MS-DOS используется обратная косая черта (\), а на Macintosh – двоеточие (:). В URL разделитель пути – всегда обычная косая черта (/).

Чтобы сослаться на любой из этих элементов, нужно просто дописать к URL идентификатор фрагмента:

```
http://cartoons.net/buffoon_archetypes.htm#ziggy
```

Можно также передавать аргументы в программы, добавляя к URL вопросительный знак (?), за которым следуют аргументы, разделенные амперсандами (&). Например, при открытии следующего URL вызывается программа *clock.cgi*, которой передаются два параметра: `zone` (часовой пояс) и `format` (формат выдачи):

```
http://www.tictoc.org/cgi-bin/clock.cgi?zone=gmt&format=hhmmss
```

Описывавшиеся до сих пор URL были *абсолютными*, т. е. представленными в виде полной записи. Это утомительный способ записи URL, но существует и его сокращенный вариант. В каждом абсолютном URL есть *базовая* составляющая, содержащая информацию о системе и пути, которую, кроме того, можно выразить как URL. Например, базовым URL для *http://www.oreilly.com/catalog/learnxml/index.html* будет *http://www.oreilly.com/catalog/learnxml/*. Если базовая часть URL целевого ресурса ссылки такая же, как у локального ресурса, то можно воспользоваться *относительным* (*relative*) URL. Он представляет собой абсолютный URL без начальной части.

Ниже в таблице приведены некоторые примеры URL. Адреса первой и второй колонок эквивалентны. Допустим, что исходным URL является *http://www.oreilly.com/catalog/learnxml/index.html*.

Относительный URL	**Абсолютный URL**
www.oreilly.com/catalog/learnxml/desc.html	*http://www.oreilly.com/catalog/learnxml/desc.html*
../../	*http://www.oreilly.com/catalog/*
errata/	*http://www.oreilly.com/catalog/learnxml/errata/*
/	*http://www.oreilly.com/*
/catalog/learnxml/desc.html	*http://www.oreilly.com/catalog/learnxml/desc.html*

Полезно использовать относительные URL при любой возможности. В результате не только сокращается объем ввода с клавиатуры, но и сохраняется работоспособность ссылок при переносе группы взаимосвязанных документов в другое место, поскольку при этом изменяется только базовый URL.

В некоторых случаях желательно установить базовый URL явным образом. Возможно, процессор XML оказывается недостаточно «сообразительным» или требуется задать ссылки на большое количество файлов, находящихся в другом месте. Атрибут `xml:base` применяется для установки базового URL, используемого по умолчанию со всеми относительными URL в области видимости этого атрибута, каковой является все поддерево элемента, в котором он задан. Например:

```
<?xml version="1.0"?>
<html>
  <head>
    <title>Book Information</title>
  </head>
  <body>
    <ul xml:base="http://www.oreilly.com/catalog/learnxml/">
      <li><a href="index.html">Main page</a></li>
      <li><a href="desc.html">Description</a></li>
      <li><a href="errata/">Errata</a></li>
    </ul>
    <p xml:base="http:www.coolbooks.com/reviews/">
      There's also a <a href="lxml.html">review of
      the book</a> available.
    </p>
  </body>
</html>
```

Независимо от того, где находится этот документ, его ссылки всегда указывают на одно и то же место, потому что информация о базовом URL жестко закодирована.

Задание ресурсов по имени

Схема поиска ресурсов основывается на том, что ресурсы остаются в одном и том же месте. Если изменяется адрес целевого ресурса, ссылка разрывается. К сожалению, это происходит постоянно. Файлы и машины перемещаются, переименовываются или вообще удаляются. Когда это происходит, ссылки на соответствующие ресурсы становятся негодными, и требуется обновление исходного документа. Для решения этой проблемы предложена иная схема адресации: имена ресурсов.

Идея, лежащая в основе схем именования ресурсов, заключается в том, что уникальные имена не должны меняться, куда бы ни перемещался объект. Например, типичный американский гражданин обладает девятизначным номером социального страхования (SSN), который присваивается ему (ей) однажды и до конца жизни. Все другие детали могут измениться, например, номер водительского удостоверения, домашний адрес, даже фамилия, но SSN останется тем же. Будет

ли человек жить в Портленде, Сент-Льюисе или Уолла-Уолла, SSN попрежнему укажет на него.

Схемы поиска ресурсов, не зависящие от местоположения, решают проблему разрыва ссылок, так почему же они не используются чаще? Несомненно, более удобно ввести в броузере одно-два ключевых слова и гарантированно оказаться в нужном месте, даже если адрес ресурса изменился. Однако такие схемы все еще внове и недостаточно разработаны, в противоположность более распространенным схемам прямой адресации. Сетевые адреса работают, потому что все компьютерные системы обрабатывают их одинаковым образом, с использованием IP-адресации, встроенной в стек TCP/IP операционной системы компьютера. Для реализации схемы с именованием ресурсов необходимо средство, ставящее в соответствие уникальным именам изменяющиеся адреса (возможно, в файле конфигурации), т. е. требуется программное обеспечение, способное осуществлять поиск адресов.

В одной из распространенных схем именования ресурсов для XML используется идентификатор, называемый *формальным открытым идентификатором* (*formal public identifier, FPI*)[1] и представляющий собой строку текста, в которой описаны некоторые характеристики ресурса. Взятая в совокупности, эта информация образует идентифицирующую метку. FPI обычно можно встретить в объявлениях типа документа (см. главу 2 «Разметка и основные понятия») и объявлениях сущностей (см. главу 5 «Модели документов: более высокий уровень контроля»).

Синтаксис FPI показан на рис. 3.3. FPI начинается с символа *registered* (1), обозначающего статус регистрации идентификатора: знак «плюс» означает, что он зарегистрирован и является общественно признанным, знак «минус» означает отсутствие регистрации, а ISO означает принадлежность ISO. За символом следуют разделитель, состоящий из двух символов косой черты (2), и *owner-id* (идентификатор владельца) (3), являющийся короткой строкой, идентифицирующей того, кто владеет сущностью, представленной FPI, или сопровождает ее.[2] После еще одного разделителя указывается *class* (класс открытого

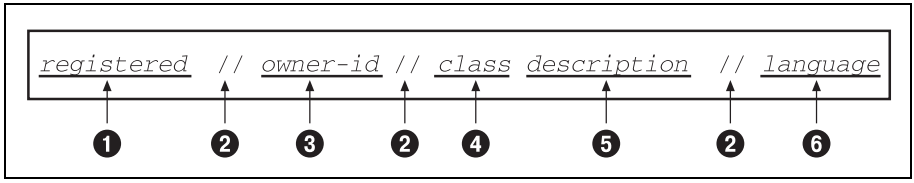


Рис. 3.3. *Синтаксис формального открытого идентификатора*

[1] Официальный стандарт ISO: ISO-8879.

[2] Заметьте, что если владелец идентификатора не зарегистрирован, он может не быть уникальным.

текста) (4), описывающий тип ресурса, представленного FPI (например, DTD для определения типа документа). За классом открытого текста следуют пробел и *description* (краткое описание ресурса) (5), например, его название или назначение. Идентификатор завершается еще одним разделителем и двухбуквенным кодом языка *language*, если это применимо к данному ресурсу (6).

Рассмотрим следующий пример, в котором формальный открытый идентификатор принадлежит незарегистрированному владельцу указанного DTD на английском языке:

```
❶ ❷ ❸                    ❹
-//ORA//DTD DocBook Lite XML 1.1//EN
```

❶ Знак «минус» (-) означает, что организация, поддерживающая FPI, формально не зарегистрирована общественным органом, например ISO.

❷ Учреждением, ответственным за сопровождение данного документа, является ORA (сокращение от O'Reilly & Associates).

❸ DTD указывает, что документ, на который осуществляется ссылка, является определением типа документа. За типом следует короткое текстовое описание, DocBook Lite XML 1.1, в которое входят название объекта, номер версии и другие особенности.

❹ Двухбуквенный код языка EN указывает, что основным языком документа служит английский. Коды языков определены в ISO-639.

Для завершения ссылки процессору XML нужно знать, как получить по FPI физический адрес ресурса. Механизм, позволяющий это сделать, обычно использует поиск имени в таблице, называемой *каталогом* (*catalog*). Как правило, это файл, находящийся на вашей машине и содержащий колонку FPI и колонку системных путей к ресурсам. Каталоги, используемые для поиска адресов по FPI, формально описаны группой OASIS в технической резолюции 9401:1997, которую можно найти в документе на *http://www.oasis-open.org/html/a401.htm*. Электронная форма для разрешения FPI имеется на *http://www.ucc.ie/cgi-bin/public*.

В XML нельзя использовать FPI в объявлении сущности самостоятельно. За ним всегда должен следовать системный идентификатор (ключевое слово SYSTEM, за которым идет системный путь или URL в кавычках). Разработчики XML решили, что было бы рискованно полагаться на процессоры XML в получении физического адреса по открытому идентификатору, и что поэтому следует добавить подсказку. Это снижает ценность открытого идентификатора, но является, вероятно, правильной идеей, по крайней мере до тех пор, пока FPI не станут применяться более широко.

Создание внутренних ссылок с помощью ID и IDREF

До сих пор мы говорили о том, как идентифицируются ресурсы в целом, но это лишь то, что лежит на поверхности. Пользователя может интересовать специальный участок данных, находящийся глубоко внутри документа. Но как найти один элемент среди тысяч однотипных? Один простой способ состоит в том, чтобы пометить его. Атрибуты `ID` и `IDREF`, описываемые ниже, позволяют поместить на элемент метку, которая и позволяет впоследствии на него ссылаться.

ID: уникальные идентификаторы элементов

В Соединенных Штатах в качестве уникального идентификатора часто используется номер социального обеспечения (SSN). Ни у каких двух людей в стране не может быть одного и того же девятизначного SSN (в противном случае, один из них, видимо, занимается чем-то недозволенным). Никто не станет обращаться к своей подруге по ее SSN: «Послушай, 456-02-9211, можно мне взять твою машину?». Но учреждениям, например, правительству или страховым компаниям, удобно использовать это число в учетных записях, т. к. оно гарантирует, что никакие два человека не будут спутаны по ошибке. В таком же духе XML предоставляет специальный маркер элемента, гарантирующий соответствие одному и только одному элементу в документе.

Этот маркер имеет вид атрибута. Атрибуты могут быть различного типа, и одним из них является `ID`. Если в DTD определено, что атрибут имеет тип `ID` (подробно о DTD рассказано в главе 5), то атрибут приобретает особое значение для анализатора XML. Значение атрибута рассматривается как *уникальный идентификатор* – строка символов, которую нельзя использовать в каком-либо другом атрибуте `ID` в документе, например:

```
<sandwich lbl="blt">Bacon, lettuce, tomato on rye</sandwich>
<sandwich lbl="ham-n-chs">Ham and swiss cheese on roll</sandwich>
<sandwich lbl="turkey">Turkey, stuffing,
    cranberry sauce on bulky roll</sandwich>
```

У всех этих трех элементов есть атрибут `lbl`, тип которого определен в DTD как `ID`. Их значениями являются неповторяющиеся строки символов, отличных от пробела. Назначение двум или более атрибутам `lbl` одного и того же значения является ошибкой. На самом деле, никакие два атрибута типа `ID` не могут иметь одинаковые значения, даже если у них разные имена.

Задумаемся над этим на минуту. Кажется, что требование отличия идентификаторов является довольно строгим. Для чего нужна проверка анализатором совпадения атрибутов? Чтобы в дальнейшем уберечься от массы проблем, которые могут возникнуть, когда идентифи-

каторы станут использоваться в качестве конечных точек для ссылок. В простой двусторонней ссылке требуется задать одну и только одну цель. Если бы для одного и того же идентификатора их оказалось две или более, возникла бы неопределенная ситуация, в которой нельзя предсказать, куда приведет ссылка.

Проблема неоднозначных меток элементов часто возникает в HTML. Чтобы создать метку в документе HTML, нужен якорь: элемент <A> с атрибутом NAME, которому присвоена некоторая строка символов. Например:

```
<A NAME="beginning_of_the_story">
```

Если теперь по ошибке создать две метки <A> с одинаковым значением, у HTML не возникнет проблем. Броузер не жалуется, и ссылка работает. Беда в том, что неизвестно, куда она приведет. Возможно, ссылка соединится с первым вхождением имени, а возможно, и нет. В спецификации HTML не сказано, как нужно поступать. Не исключено, что веб-дизайнер или автор будут рвать на себе волосы, пытаясь определить, почему ссылка не приводит туда, куда нужно.

Поэтому, благодаря своей строгости, XML уберегает нас от путаницы в дальнейшем. Проверив действительность документа, мы будем знать, что все идентификаторы уникальны и со ссылками все в порядке, т. е. их цели могут быть найдены. В этом состоит роль IDREF, как мы увидим ниже.

Каким элементам назначить идентификаторы, зависит от разработчика, но следует проявлять некоторую сдержанность. Может возникнуть соблазн дать каждому элементу свой собственный идентификатор в слабой надежде, что когда-нибудь потребуется соединиться с ним, но лучше отметить только главные элементы. Например, в книге следует дать идентификаторы главам, разделам, рисункам и таблицам, которые часто являются целями для ссылок в тексте, но большинству внутритекстовых элементов давать идентификаторы не требуется.

Следует также внимательно отнестись к синтаксису меток. Постарайтесь придумать легко запоминающиеся имена, имеющие отношение к контексту, например «vegetables-rutabaga» или «intro-chapter». Можно использовать иерархическую структуру имен, отражающую фактическую структуру документа. Назначать идентификаторам такие значения, как «k3828384» или «thingy», плохо, потому что почти невозможно запомнить их и понять, что они означают. По возможности лучше не использовать числа, так как при необходимости что-нибудь поменять местами такие идентификаторы, как «chapter-13», оказываются не очень хороши.

IDREF: гарантированные целые ссылки

XML предоставляет другой, особый вид атрибута, называемый IDREF. Как предполагает его название, это ссылка на ID, находящийся в том же документе. В XML нет способа описать отношение между элементом, на который указывает ссылка, и элементом, осуществляющим ссылку. Можно сказать лишь о существовании *некоторого* отношения, определенного в таблице стилей или в приложении, выполняющем обработку. Ценность этого механизма может показаться ограниченной, но на самом деле он крайне прост, эффективен и позволяет связать два или более элемента, не прибегая к сложной структуре XLink, описанной в разделе «Введение в Xlinks» ниже в этой главе.

Есть и другая выгода. Как мы видели, атрибутам ID гарантируется уникальность в пределах документа. Для атрибутов IDREF гарантируется иное: любое значение ID, на которое ссылается IDREF, должно существовать в том же документе. Если ссылка на ID разорвана, анализатор сообщает об этом, и разработчик может исправить положение, прежде чем дать жизнь своему документу.

Где могут применяться ID и IDREF? Вот краткий перечень областей:

- Перекрестные ссылки на части книги, такие как таблицы, рисунки, главы и приложения
- Предметные указатели и оглавления документов, состоящих из многих разделов
- Элементы, которые обозначают диапазон и могут появляться в другом элементе, например, статьи предметного указателя, относящиеся к интервалу из нескольких страниц
- Ссылки на сноски и врезки
- Перекрестные ссылки внутри объектно-ориентированной базы данных, физическая структура которой может не соответствовать логической структуре

Например, в документе может быть несколько сносок с одинаковым текстом. В следующем примере элемент <footnoteref> ссылается на <footnote>, в результате чего при обработке документа он наследует текст целевого элемента:

```
<para>The wumpus<footnote id="donut-warning">
Do not try to feed this animal donuts!</footnote>
lives in caves and hunts unsuspecting computer nerds. It is related
to the jabberwock<footnoteref idref="donut-warning"/>,
which prefers to hunt its prey in the open.</para>
```

В применении IDREF есть тонкость – надо знать, на что ссылаться. Например, нужно сослаться на главу, чтобы включить ее название в выводимый текст. На что следует указать: на название главы или на сам элемент главы? Обычно лучше ссылаться на самый общий элемент, со-

ответствующий смыслу ссылки, в данном случае, главу. Позднее можно изменить решение и опустить название главы, выведя вместо него номер главы или какой-либо другой атрибут. Предоставьте таблице стилей заботу о том, как найти информацию, необходимую для представления. В разметке нужно сосредотачиваться на смысле.

XPointer: перемещение по дереву XML

Последняя часть головоломки с идентификацией ресурсов – это XPointer, который официально называется XML Pointer Language – язык указателей XML. XPointer является особым расширением URL, позволяющим добираться до точек, находящихся далеко в глубине документа XML. Чтобы понять, как работает XPointer, рассмотрим сначала действие его более простого собрата, *идентификатора фрагмента* (*fragment identifier*). Идентификатор фрагмента служит механизмом, который используется ссылками HTML для соединения с конкретными местами в файле HTML. Он присоединяется в конец URL и отделяется от URL символом решетки (#):

```
<a href="http://www.someplace.com/takeme/toyour/leader.html#earthling">
```

В данном примере <a> является элементом ссылки. Слово справа от символа решетки, earthling, расширяет URL, в результате чего, тот указывает на местоположение внутри файла *leader.html*. Ссылка находит свою цель, если файл содержит маркер вида

```
<a name="earthling">
```

В XML эквивалентом идентификатора фрагмента служит указатель XPointer, название которого происходит от рекомендации W3C для расширения URL в ссылках XML. Как и идентификатор фрагмента, XPointer присоединяется к правой части URL с помощью символа решетки:

```
url#XPointer
```

В простейшем случае указатель XPointer действует просто как идентификатор фрагмента, осуществляя связывание с элементом внутри целевого ресурса, имеющим атрибут ID. Однако XPointer более гибок, поскольку его целью может быть *любой* элемент. В отличие от HTML, где целью всегда является элемент <A>, целью XPointer может быть любой элемент с атрибутом типа ID, значение которого совпадает с XPointer.

Это и само по себе удобно, но XPointer на этом не останавливается. Рекомендация XPointer определяет целый язык для адресации любого элемента в документе, даже если у того нет ID. Этот язык является производным от XPath (см. приложение B «Таксономия стандартов») – общей спецификации описания местоположений внутри XML-докумен-

тов, разработанной в соответствии с правилами синтаксиса URL. Он состоит из команд, с помощью которых можно пройти документ шаг за шагом.

Создадим образец документа XML, чтобы показать, как XPointers используются для нахождения элементов. Пример 3.1 является простой схемой, показывающей иерархию служащих небольшой компании. На рис. 3.4 документ представлен в виде дерева.

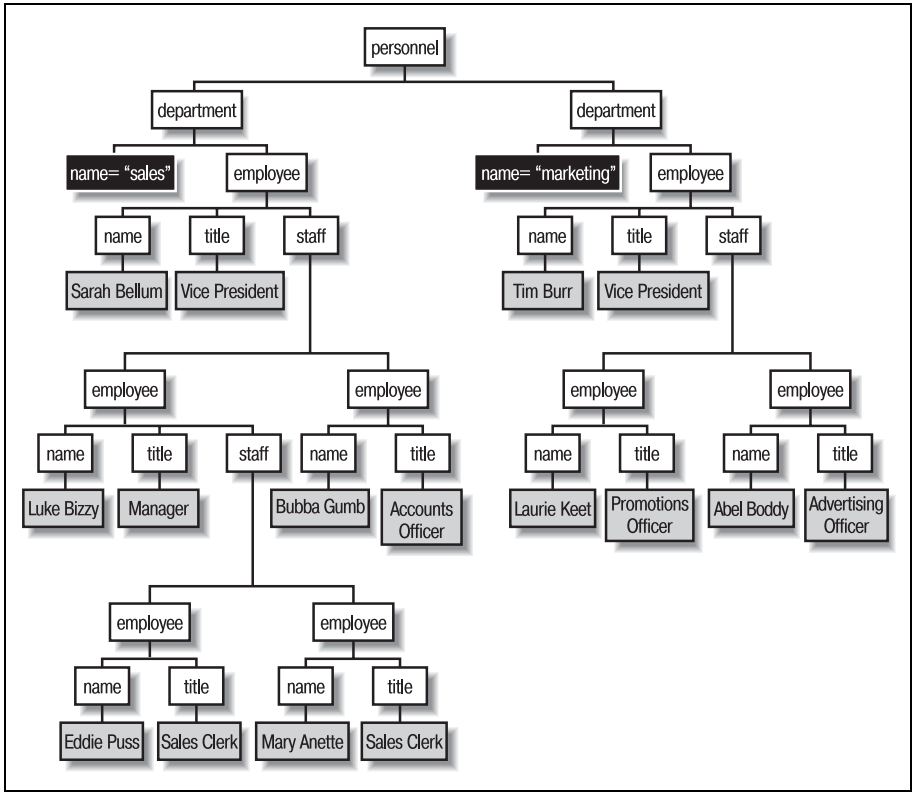


Рис. 3.4. *Представление структуры организации в виде дерева*

Ранее рассматривалось, как находить элемент с атрибутом `ID`. Например, чтобы создать ссылку на элемент из примера 3.1, содержащий отдел продаж, можно использовать указатель XPointer `sales` и найти элемент с атрибутом `ID`, имеющим значение `sales`. В данном примере им окажется первый элемент `<department>`.

Пример 3.1. Схема персонала фирмы Bob's Bolts

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

<personnel>
```

Пример 3.1. Схема персонала фирмы Bob's Bolts (продолжение)

```
  <department id="sales">
    <employee>
      <name>Sarah Bellum</name>
      <title>Vice President</title>
      <staff>
        <employee>
          <name>Luke Bizzy</name>
          <title>Manager</title>
          <staff>
            <employee>
              <name>Eddie Puss</name>
              <title>Sales Clerk</title>
            </employee>
            <employee>
              <name>Mary Anette</name>
              <title>Sales Clerk</title>
            </employee>
          </staff>
        </employee>
        <employee>
          <name>Bubba Gumb</name>
          <title>Accounts Officer</title>
        </employee>
      </staff>
    </employee>
  </department>

  <department id="marketing">
    <employee>
      <name>Tim Burr</name>
      <title>Vice President</title>
      <staff>
        <employee>
          <name>Laurie Keet</name>
          <title>Promotions Officer</title>
        </employee>
        <employee>
          <name>Abel Boddy</name>
          <title>Advertising Officer</title>
        </employee>
      </staff>
    </employee>
  </department>

</personnel>
```

XPointer `sales` **в действительности является сокращенной формой от** `id(sales)`**.** `id()` **представляет собой особый вид терма, который может осуществлять в документе переход к элементу с атрибутом типа** `ID`**,**

совпадающим со строкой в скобках. Он называется *термом абсолютной адресации* (*absolute location term*), поскольку позволяет находить уникальный элемент без помощи других термов. Только у одного элемента может иметься заданный `ID`, и, если такой элемент существует, `id()` его найдет.

Каждый указатель XPointer начинается с абсолютного терма, а затем может расширять его с помощью *термов относительной адресации* (*relative location terms*), которые соединяются вместе точками (`.`). Абсолютный терм начинает поиск в некоторой точке документа, а относительные термы продолжают поиск с этого места и выполняют его шаг за шагом, пока не будет обнаружена требуемая цель. Каждый терм имеет вид:

```
name (args)
```

где *name* является типом терма, а *args* – разделяемым запятыми списком параметров, подробно характеризующих каждый терм.

Например, следующий указатель XPointer начинается с элемента, имеющего атрибут типа `ID` со значением `marketing`, затем перемещается к первому дочернему элементу `<employee>`, затем останавливается на первом находящемся под ним элементе `<staff>`:

```
id(marketing).child(1,employee).child(1,staff)
```

Целью является элемент `<staff>`, его родитель – элемент `<employee>`, а его родитель – элемент `<department>` с атрибутом `id="marketing"`.

В следующих разделах термы абсолютной и относительной адресации описываются более подробно.

Термы абсолютной адресации

XPointer должен начинаться именно с одного терма абсолютной адресации. Все последующие относительные термы расширяют информацию о позиции, содержащуюся в терме абсолютной адресации. Существует четыре типа термов абсолютной адресации: `id()`, `root()`, `origin()` и `html()`.

Как работает терм `id()`, мы уже видели. Он находит в любом месте документа элемент с заданным атрибутом `ID`. Ссылку на `ID` часто лучше всего использовать в качестве абсолютного терма для документов, которые часто изменяются. Даже если содержимое будет реорганизовано, XPointer по-прежнему будет находить элемент.

Абсолютный терм `root()` ссылается на весь документ, заданный базовым URL. Он указывает на абстрактный узел, потомком которого является корневой элемент, а не на сам элемент. Вряд ли кто-нибудь станет использовать `root()` сам по себе, поскольку корневой элемент – не слишком полезная точка для ссылки. Вместо этого вслед за ним лучше

указать цепочку относительных термов. Например, чтобы добраться до отдела маркетинга, можно использовать такой указатель XPointer:

```
root().child(1,personnel).child(2)
```

В то время как `id()` требуется аргумент, устанавливающий `ID`, который нужно искать, `root()` не принимает аргументов. Он всегда указывает на вершину документа, вследствие чего аргументы не нужны.

Терм `origin()` (начало отсчета) является абсолютным термом, определяющим местоположение элемента, из которого инициирована ссылка. Поскольку он ссылается сам на себя, использование его с URL недопустимо. Единственным применением этого терма является соединение исходного элемента с другим элементом того же документа для создания *диапазона* (*range*). Диапазон является особым видом указателя XPointer, содержащим две цепочки термов адресации, соединенные двумя точками. Этот диапазон используется для адресации нескольких элементов с какой-то общей для них задачей. Например:

```
<p>Let's select <range href="root()..origin()">
everything up to this point</range>.
```

Как и `root()`, `origin()` не принимает аргументов.

`html()` служит абсолютным термом для переходных задач. Он применяется с документами HTML, для того чтобы найти первый элемент `<A>`, атрибут `name` которого совпадает со строкой в скобках. Терм `html()` всегда указывает на первое совпадение (в отличие от идентификатора фрагмента HTML, поведение которого при многократных совпадениях не определено).

Термы относительной адресации

Абсолютные термы приводят нас только к тем немногим адресам в документе, которые находятся в самом его начале или помечены с помощью идентификаторов. Чтобы попасть куда-либо еще, нужно использовать термы относительной адресации. Подобно перечню инструкций, которые необходимо дать кому-то, кто должен найти дом по адресу, эти термы проходят документ шаг за шагом, пока не будет достигнуто желаемое место.

Узлы

Вспомните из материала главы 2, что любой документ XML может быть представлен в виде родословного дерева. В рамках этой модели термы относительной адресации, обеспечивающие переходы между узлами и листьями дерева, можно сравнить с ловкими белками, прыгающими с ветки на ветку. В табл. 3.1 перечислены некоторые термы относительной адресации, следующие этой аналогии. Обратите внима-

ние на использование слова *узел* (*node*) вместо слова *элемент*. Узел является в XML общим объектом: элементом, инструкцией обработки или участком текста. *Текущий узел* (*current node*) – это часть дерева, найденная предыдущим термом адресации из цепочки.

Таблица 3.1. Термы относительной адресации XPointer

Терм	Находит
`child()`	Узел из числа непосредственных потомков текущего узла
`descendant()`	Узел из числа потомков текущего узла в порядке «сначала вглубь»
`ancestor()`	Узел из числа предков текущего узла, начиная с `root()`
`following()`	Узел из числа тех, которые заканчиваются после текущего узла
`preceding()`	Узел из числа тех, которые начинаются перед текущим узлом
`fsibling()`	Узел из числа последующих одноуровневых для текущего узла
`psibling()`	Узел из числа предыдущих одноуровневых для текущего узла

Все эти термы принимают от одного до четырех аргументов. Эти аргументы перечислены ниже в том порядке, в котором они задаются:

Номер узла

Если терм адресации соответствует более чем одному узлу, он создает список допустимых узлов. Если нужен только один из них, необходимо указать его с помощью номера. Предположим, например, что у элемента есть три потомка, но нужен только второй из них. Можно указать это с помощью терма `child(2)`. Положительное целое значение производит отсчет в списке допустимых узлов в прямом направлении, в то время как отрицательное – в обратном. Отсчет в обратном направлении полезен, когда нужно найти последний (или предпоследний и т. д.) узел. Альтернативой является использование ключевого слова `all` для выбора всех допустимых узлов.

Тип узла

Этот аргумент указывает, какого типа узлы нужно искать. Если значение является именем или аргумент опущен, считается, что тип является элементом. Для всех остальных типов нужно использовать ключевое слово:

`#text`

Соответствует непрерывным строкам символьных данных.

`#pi`

Соответствует инструкции обработки.

`#comment`

Соответствует комментарию.

`#element` *или* `*`

Соответствует любому элементу независимо от имени.

`#all`

Соответствует любому узлу.

Например, `descendant(1,#all)` соответствует любому узлу, будь он элементом, положительным целым числом, комментарием или строкой текста. Терм `descendant(1,*)` соответствует любому элементу, а `descendant(1,buttercup)` соответствует любому элементу типа `buttercup`.

Имя атрибута

Этот аргумент сужает поиск элементов, требуя для них наличия конкретного атрибута. Аргумент имени атрибута действует только тогда, когда типом узла является элемент. Можно задать имя, если требуется наличие конкретного атрибута, или звездочку (`*`), чтобы принимать любой атрибут (к сожалению, нет возможности указывать более одного атрибута). Если аргумент опущен, то атрибуты при поиске не учитываются. Этот аргумент должен использоваться вместе с аргументом значения атрибута, который описывается следующим.

Например, `ancestor(1,grape)` соответствует любому элементу `<grape>`, независимо от того, есть у него атрибуты или нет. Терм `ancestor(1,grape,vine,*)` соответствует только тем элементам `<grape>`, у которых есть атрибут `vine`, в то время как `ancestor(1,grape,*,*)` соответствует всем элементам `<grape>`, имеющим хотя бы один атрибут.

Значение атрибута

Этот аргумент устанавливает значение атрибута, указанного в аргументе имени атрибута. Можно устанавливать конкретное значение, использовать звездочку, разрешающую любое значение, или указать ключевое слово `#IMPLIED`, означающее, что никакого значения не задано и атрибут является необязательным. Этот аргумент нельзя опускать, если используется аргумент имени атрибута.

Например, терм `preceding(1,fudge,tasty,yes)` соответствует всем элементам, которые выглядят так: `<fudge tasty="yes">`. Терм `preceding(1,fudge,tasty,*)` соответствует элементам `<fudge>` с атрибутом `tasty`, имеющим любое значение, в то время как `preceding(1,fudge,tasty,#IMPLIED)` соответствует элементам `<fudge>`, даже если у них нет атрибута `tasty`.

Мы описали аргументы. Теперь рассмотрим термы относительной адресации подробно:

`child()`

`child()` находит узел среди дочерних узлов текущего узла. В отличие от `descendant()`, `child()` не идет вглубь дальше одного узла, ограничивая область поиска. Если поиск не дал результатов, процессор возвращается быстрее, чем при использовании `descendant()` или `forward()`.

На рис. 3.5 показан путь обхода `child()` в прямом направлении (при передаче положительного номера узла) и в обратном направлении (при передаче отрицательного номера узла). Черный узел показывает исходное местонахождение.

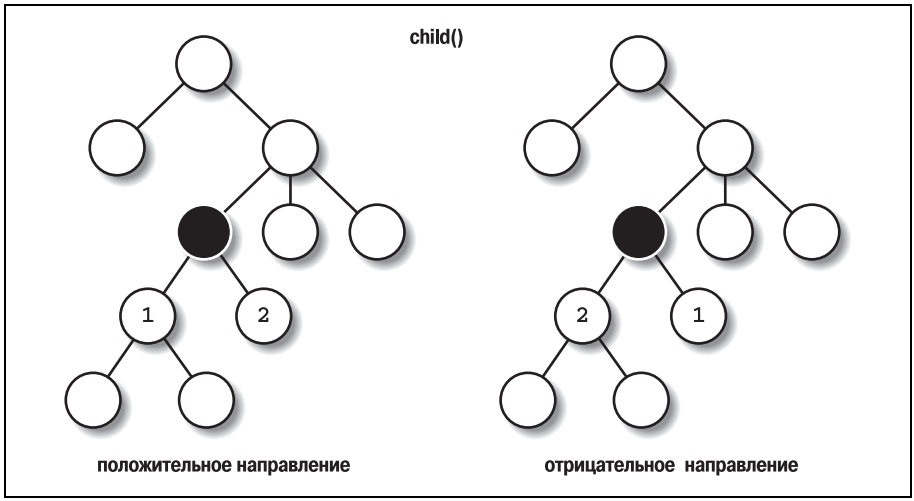


Рис. 3.5. *Путь child( )*

Например, для поиска имени руководителя отдела продаж можно использовать такой указатель XPointer:

```
id(sales).child(1,employee).child(1,name)
```

XPointer допускает следующее синтаксическое сокращение: если терм того же типа, что и предшествующий ему, можно опустить имя второго терма. Поэтому указатель XPointer из предыдущего примера можно сократить до:

```
id(sales).child(1,employee).(1,name)
```

`descendant()`

`descendant()` идет дальше, чем `child()`, осуществляя поиск среди потомков на любую глубину. Однако `descendant()` все же ограничивает поиск поддеревом, находящимся под текущим узлом. Порядок об-

хода определяется как «*сначала вглубь*» (*depth-first*), при этом выбирается зигзагообразный маршрут вниз, пока не будет достигнут лист дерева, после чего выполняется возврат. Поиск с использованием `descendant()` гарантированно проходит каждый узел, произрастающий из текущего. Порядок поиска узла для положительного и отрицательного направлений проиллюстрирован на рис. 3.6.

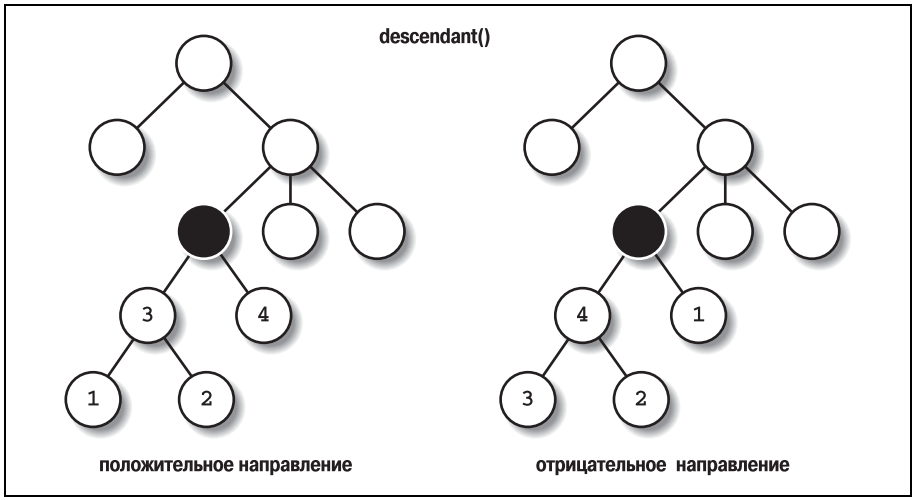


Рис. 3.6. *Путь descendant( )*

Числовой аргумент для `descendant()` более сложен, чем для `child()`. При положительном значении терм выполняет поиск с открывающего тега текущего элемента и производит его по файлу, отсчитывая открывающие теги каждого потомка, пока не достигнет закрывающего тега текущего узла. Для отрицательных значений отсчет начинается с закрывающего тега текущего элемента и осуществляется в обратном направлении, при этом отсчитывается каждый закрывающий тег. В следующем примере `id(start).descendant(4)` находит элемент `item2`, т. к. перед целевым элементом имеется четыре открывающих тега, считая от открывающего тега текущего элемента:

```
<cont id="start">
  <sub1>
    <item1>text</item1>
    <sub2>
      <item2>more text</item2>
    </sub2>
  </sub1>
</cont>
```

Можно упростить пример с `child()`, в котором потребовалось два относительных терма, заменив их одним термом `descendant()`:

```
id(sales).descendant(1,name)
```

В этом примере производится поиск в поддереве под узлом начального элемента (которым является `<department id="sales">`) первого элемента типа `name`.

`following()`

`following()` задает самые слабые ограничения на область поиска: в нее входят все узлы документа, следующие за текущим узлом. Поиск начинается с текущего узла и проходит один за другим все узлы, пока не будет найден соответствующий узел или достигнут конец документа. Рис. 3.7 иллюстрирует порядок прохода узлов в обоих направлениях.

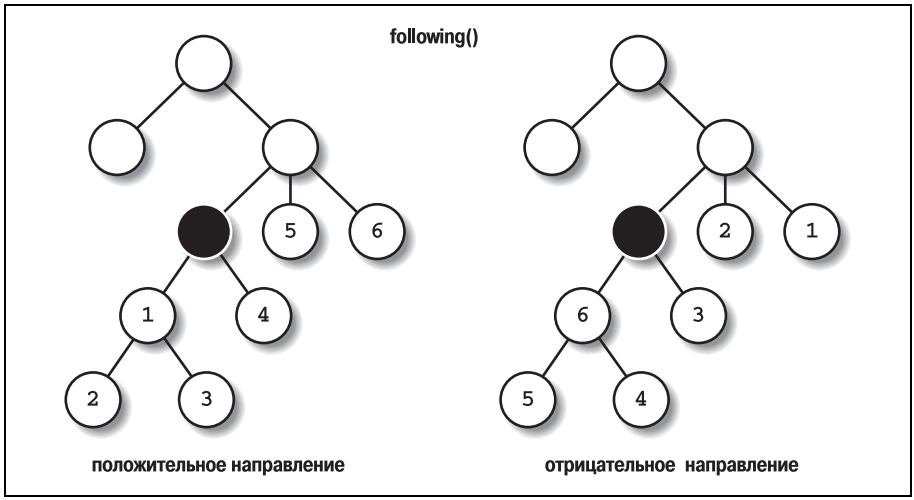


Рис. 3.7. *Путь following( )*

Например, можно найти элемент `<employee>` для Mary A. из элемента `<employee>` для Eddie P. с помощью терма `following(1,employee)`. Из той же начальной точки можно найти элемент `<employee>` для Tim B. с помощью терма `following(3,employee)`.

`preceding()`

`preceding()` работает аналогично `following()`, но с противоположным концом документа – от исходной точки к началу. Направление тоже становится противоположным, поэтому положительное число приводит к перемещению в направлении начала файла, а отрицательное – в направлении исходной точки. Рис. 3.8 показывает порядок поиска узлов в обоих направлениях.

Начав с любого работника, можно найти того, кто находится непосредственно перед ним в схеме, с помощью терма `preceding(1,employee)`. От Laurie K. он находит Tim B., а от Abel B. – Laurie K.

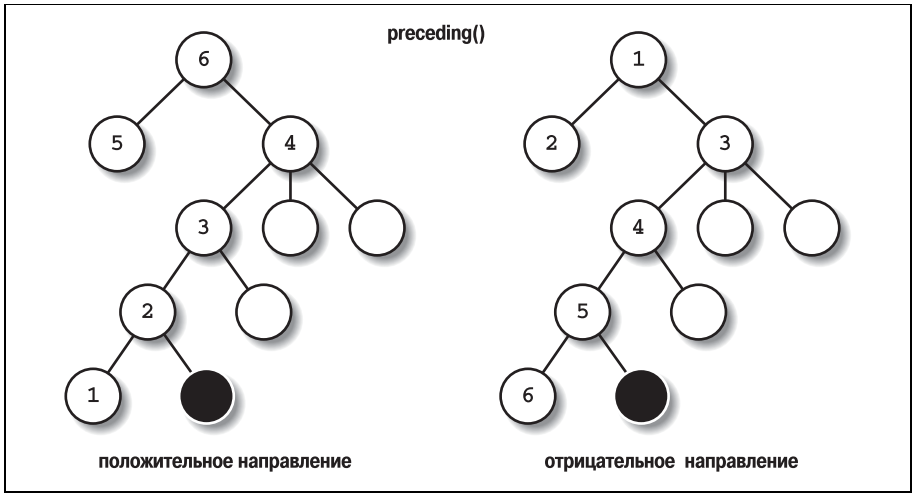


Рис. 3.8. *Путь preceding( )*

`fsibling()`

Это терм ограничивает поиск одноуровневыми узлами, следующими за исходным адресом (можно назвать их младшими братьями). Он находит только те элементы, у которых тот же родитель, что у текущего узла. Подобно `child()`, он обеспечивает маленькую и безопасную зону поиска; однако для применения `fsibling()` требуется некоторое знание структуры документа. Путь поиска узлов показан на рис. 3.9.

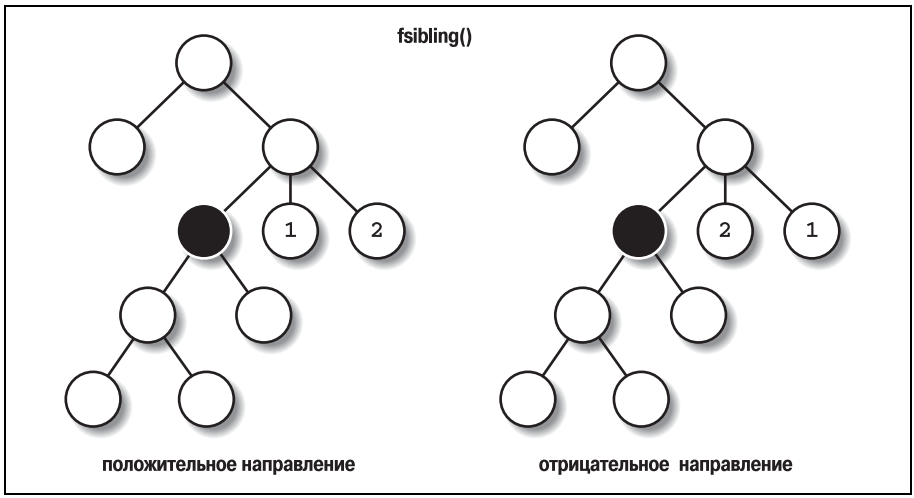


Рис. 3.9. *Путь fsibling( )*

Например, `fsibling(1)` может найти Bubba G., являющегося коллегой Luke B., но `fsibling(2)` возвращается «с пустыми руками».

`psibling()`

`psibling()` ведет себя аналогично `fsibling()`, но выполняет поиск среди одноуровневых элементов, которые предшествуют исходному адресу в контейнере-родителе (старших братьев). Направление также меняется на противоположное. Путь показан на рис. 3.10.

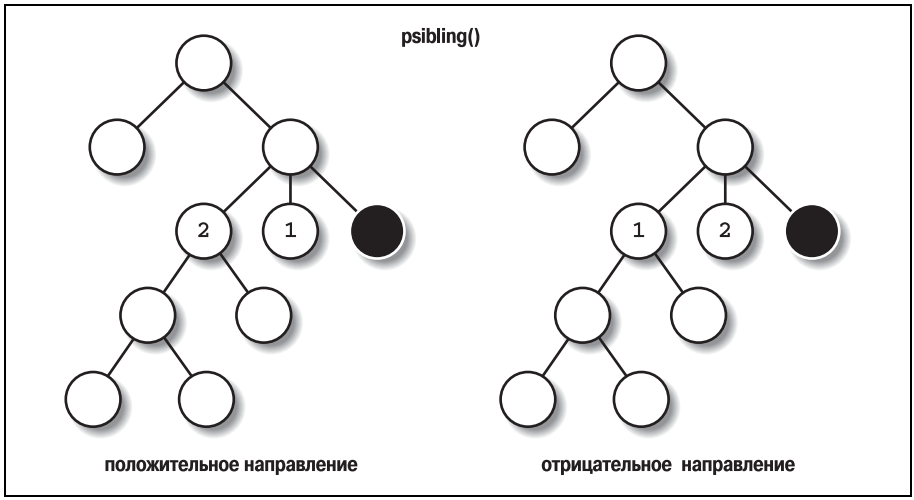


Рис. 3.10. *Путь psibling( )*

`ancestor()`

Терм `ancestor()` работает как специалист по генеалогии, прослеживая предков узла вплоть до `root()`. При положительном первом аргументе `ancestor()` работает, начиная с родителя исходного адреса и заканчивая в `root()`. При отрицательном аргументе поиск начинается с `root()` и завершается в родительском элементе исходного адреса. Порядок, в котором этот терм проходит узлы, отображен на рис. 3.11.

Например, чтобы найти `<department>` для любого служащего в схеме, можно использовать терм `ancestor(1,department)`. Чтобы найти начальника этого служащего (если таковой существует), используйте терм `ancestor(1,employee)`. Обратите внимание, что если начальной точкой является элемент для вице-президента, этот терм поиска найдет ноль узлов и выполнится неудачно.

Существует много способов, которыми можно попасть в одно и то же место. Чтобы найти элемент `<employee>` для Mary A., в данном примере годится любой из следующих способов поиска:

```
root().child(1,personnel).child(1).child(1).child(3).child(1).child(3).
   child(2)
root().child(1,personnel).(1).(1).(3).(1).(3).(2)
root().child(1,personnel).following(1,*,id,'marketing').
```

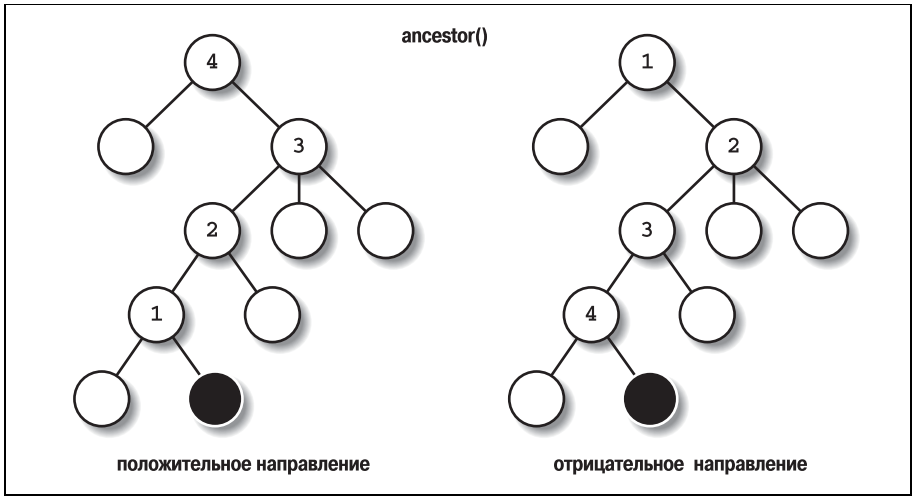


Рис. 3.11. *Путь ancestor( )*

```
    preceding(2,employee)
id(sales).descendant(4,employee)
id(sales).descendant(-2,employee)
```

Строки

Относительные термы, обсуждавшиеся до сих пор, работают только с полными узлами. Даже с ключевым словом `#text` локатор соответствует всему тексту между соседними узлами. И это представляет собой проблему, если требуется найти меньшее подмножество, такое как слово, или более крупную группу текста, например, целый параграф, по которому разбросаны внутритекстовые элементы. В таких ситуациях помогает терм `string()`.

`string()` принимает от двух до четырех аргументов. Они в некоторой степени схожи с аргументами рассмотренных ранее термов относительной адресации. Первый аргумент указывает на экземпляр, а второй – на строку, которую нужно искать. Например, `string(2, "bubba")` находит второе вхождение строки «bubba» по начальному адресу. `string(all, "billy")` находит все вхождения «billy» в узле.

Поиск не ограничен словами. Терм `string(2, "B")` находит вторую букву «B» в строке «Billy-Bob». Поиск чувствителен к регистру, поэтому при замене терма на `string(2, "b")` поиск окажется неудачным, поскольку есть только одна строчная буква «b». XML не поддерживает нечувствительный к регистру поиск, т. к. это потребовало бы принятия решений в зависимости от норм, принятых в разных языках . Например, что считать верхним и нижним регистром в наборах символов для китайского языка?

Другим полезным режимом string() является подсчет общего числа символов. Пустая строка ("") соответствует любому символу. Терм string(23,"") находит точку непосредственно перед двадцать третьим символом в исходном адресе. Это полезно, когда известно, *где* находится что-то, но неизвестно, *что* именно.

Третий и четвертый аргументы определяют позицию и размер возвращаемой подстроки. Например, локатор string(1, "Vasco Da Gama", 6, 2) ищет строку «Vasco Da Gama» и, найдя ее, возвращает «Da» – часть строки, начинающуюся через шесть символов после начала совпадения и имеющую два символа в длину. Это метод действует как условный оператор, сначала находя основную строку, а затем возвращая небольшую ее часть.

Поиск не ограничен рамками искомой строки. Смещение может выходить за границы, минуя при этом оставшийся текст узла. Поиск в тексте «The Ascott Venture» с помощью локатора string(1, "Ascott", 8, 7) находит строку «Venture».

Обратите внимание, что найденный объект не обязательно должен содержать какие-либо символы, это может быть просто позиция. Если установить четвертый аргумент предыдущего поиска равным нулю, то мы найдем позицию в строке непосредственно перед «V». По такой ссылке пользователю может оказаться трудно щелкнуть мышью, но это совершенно допустимый адрес ссылки или точка вставки для блока текста с другой страницы.

Интервалы

Не все, что требуется находить, может быть аккуратно упаковано в виде элемента или участка текста, целиком находящегося в одном элементе. Поэтому указатель XPointer предоставляет способ определения местонахождения двух объектов и всего, что находится между ними. Это достигается при помощи терма span(), имеющего следующий синтаксис:

```
span(XPointer, XPointer)
```

Например, в документе DocBook, в котором находится раздел <sect1> с id="s1", можно создать диапазон от начала первого параграфа до начала второго параграфа следующим образом:

```
id(s1).span(descendant(1,para),descendant(2,para))
```

Введение в XLinks

Правила ссылок в XML определены в стандарте XML Linking Language, или *XLink*. В XML любой элемент может быть использован в качестве элемента ссылки. В этом есть необходимость, т. к. в XML нет предопре-

деленных элементов. Поскольку разработчик может определять собственные элементы, у него должна быть возможность делать один или несколько из них ссылками. Синтаксис и возможности XLinks стимулировались успехами (а в некоторых случаях – неудачами) HTML. Ссылки XLinks совместимы с более старыми ссылками HTML, но предоставляют дополнительную гибкость и функциональность.

Обычно в HTML используется два вида ссылок. Элемент `<A>` создает ссылку, но не переходит по ней автоматически; если пользователь решает перейти по ссылке, текущий документ заменяется документом на другом конце ссылки. Элемент `<IMG>` работает «молча», автоматически ссылаясь на графические данные и импортируя их в документ.

Для сравнения рассмотрим, чем XLinks лучше ссылок HTML:

- Любой элемент XML может быть превращен в ссылку. В HTML лишь несколько элементов могут быть ссылками.
- XLinks может использовать XPointers, чтобы попасть в любую точку внутри документа. Ссылки HTML, указывающие на конкретные места в документе, используют выделенные для них якоря, поэтому автор целевого документа должен предвидеть возможные ссылки и обеспечить наличие якорей.
- XML может использовать XLinks для импорта текста и разметки. В HTML нет способа встраивать текст целевого документа в исходный документ.
- Указатели XPointer могут определять диапазон разметки XML для ссылки на подмножество документа. Ссылка HTML может указывать только на одну точку или файл целиком.

Установка связующего элемента

Любой элемент XML можно преобразовать в ссылку при помощи определенных атрибутов XLink: `type`, `href`, `role`, `title`, `show` и `actuate`. Применяя эти атрибуты, нужно использовать префикс пространства имен, поставленный в соответствие URI XLink. Процессор XML использует пространство имен для интерпретации этих атрибутов как параметров ссылки. Вот некоторые примеры связующих элементов, в которых применяются эти атрибуты:

```
<cite
   xmlns:xlink="http://www.w3.org/1999/xlink"
   xlink:type="simple"
   xlink:href="http://www.books.org/huckfinn.xml"
   xlink:show="new"
   xlink:actuate="onRequest"
>Huckleberry Finn</cite>
<graphic
   xmlns:xlink="http://www.w3.org/1999/xlink"
```

```
    xlink:type="simple"
    xlink:href="figs/diagram39.png"
    xlink:show="embed"
    xlink:actuate="onLoad"
/>
<dataref
    xmlns:xlink="http://www.w3.org/1999/xlink"
    xlink:type="simple"
    xlink:href="http://dataserv.buggs.com/db.xml#entry92"
    xlink:actuate="onLoad"
    xlink:show="embed"
/>
```

Первый пример является ссылкой на книгу, находящуюся где-то в Сети. В следующем примере импортируется графика из локального файла. В третьем извлекается часть информации, находящаяся внутри файла. А то, как будут выглядеть все эти ссылки, определяет обрабатывающее приложение.

Атрибутом, необходимым для любой ссылки XLink, является `type`. Обнаружив это ключевое слово, анализатор определяет, что данный элемент нужно обрабатывать как ссылку. Значение `type` определяет тип XLink: в данном случае – `simple`.

Ссылка XLink типа `simple` должна иметь цель, определяемую с помощью атрибута `href`. `href` получил свое название от атрибута, используемого в HTML для указания элементам `<A>` конечного пункта ссылки, и обеспечивает совместимость документов XML и HTML. Его значением является URI другого конца ссылки; значение может ссылаться на документ в целом либо на точку или элемент в этом документе.

К анализатору XML не предъявляется требование проверки существования удаленных ресурсов в указанном вами месте. URL могут быть неверными, но документ, тем не менее, будет определен как корректный (well-formed) и действительный (valid). Такой подход противоположен по отношению к внутренним ссылкам, описанным ранее, в которых атрибуты `ID` должны быть уникальными, а атрибуты `IDREF` должны указывать на существующие элементы. Причина этого в том, что все внутренние ссылки находятся внутри одного и того же документа, который обычно располагается на одной машине. Поскольку время установления сетевых соединений обычно составляет несколько секунд, всякое требование проверки URL сделало бы синтаксический анализ очень длительной операцией.

Остальные атрибуты не являются обязательными и пока не распространены широко, поскольку спецификация XLink является достаточно новой. Тем не менее, в следующих разделах мы обсудим возможности их использования.

Поведение

Описать характер действия ссылки XLink так же важно, как и то, куда она нацелена. Должен ли процессор XML немедленно перейти по ссылке или подождать, пока этого потребует пользователь? Должен ли он вставить текст и данные в локальный документ или перенести пользователя к целевому ресурсу? Такие сведения предоставляют атрибуты, описываемые в данном разделе.

Атрибут `actuate` указывает, в какое время осуществлять доступ по XLink. Можно указать, что переход по некоторым ссылкам на странице, таким как графика и импортируемый текст, должен происходить во время форматирования страницы. В этом случае данные удаленного ресурса автоматически извлекаются процессором XML, обрабатываются так, как это требуется приложением, и затем оформляются вместе со всем документом. Значение `onLoad` объявляет, что доступ по ссылке должен производиться сразу.

Значение `onRequest` используется для ссылок, доступ к которым оставляется на усмотрение читателя. Такая ссылка остается в скрытом состоянии, пока пользователь не выберет ее, и тогда конечный результат действия ссылки определяется остальными атрибутами. Точный способ, которым пользователь приводит в действие ссылку, не указывается. Читателю документа может потребоваться щелкнуть по управляющему элементу в графическом приложении или ввести команду с клавиатуры в текстовом броузере, или произнести команду вслух в звуковом броузере. Точный способ приведения в действие определяет процессор XML.

Атрибут `show` описывает поведение ссылки после того, как она приведена в действие (автоматически или пользователем) и к ней осуществлен доступ (удаленный ресурс найден и загружен). В этот момент возникает вопрос о том, что делать с данными целевого ресурса. Возможны три варианта:

`embed`

Данные удаленного ресурса следует отобразить в месте нахождения связующего элемента.

`replace`

Просмотр текущего документа следует прекратить и заменить его удаленным документом.

`new`

Броузер должен каким-то образом создать новый контекст, если это возможно. Например, он может открыть новое окно для вывода содержимого удаленного ресурса, не прекращая просмотр локального ресурса.

Вот пример, в котором используются атрибуты поведения:

```
<para>The quote of the day is:</para>
<para>
   <program-call xlink:type="simple"
     xlink:href="bin/quote-o-matic.pl"
     xlink:actuate="onLoad"
     xlink:show="embed"/>
</para>
```

Данная ссылка XLink вызывает программу, которая возвращает текст. Удобно, что нам не нужно объяснять, как это работает, но мы должны объяснить, что происходит с полученными от программы данными. В этом случае они встраиваются в документ и появляются в виде текста. У читателя не возникает мысли, что была вызвана другая программа, потому что вся страница строится одновременно.

В этом примере способом запуска является `onLoad`, однако можно представить себе и применение `onRequest`. В таком случае пользователь мог бы щелкнуть по тексту цитаты (которым может быть «щелкните здесь»), чтобы вывести другую цитату в том же месте. Опять-таки, XML не позволяет себе точно указывать вам, как это должно выглядеть.

Текст описания

XLink предоставляет несколько мест для помещения текста, описывающего ссылку. Такая информация необязательна, но может быть полезной читателям, которым хотелось бы больше узнать о том, на что они смотрят и стоит ли следовать по этой ссылке. Одним из этих мест является содержимое элемента. Взгляните на такую ссылку:

```
A topic related to rockets is
<related
   xmlns:xlink="http://www.w3.org/1999/xlink"
   xlink:type="simple"
   xlink:href="planes.xml"
>Airplanes</related>
```

Роль содержимого в связующем элементе может быть различной. Если у ссылки есть атрибут `actuate="onRequest"`, то содержимое этой ссылки (`Airplanes`) может выступать в качестве реагирующей на щелчок метки, посредством которой пользователь может привести ссылку в действие. С другой стороны, с атрибутом `actuate="onLoad"` содержимое

может быть просто заголовком. Часто элемент, который автоматически загружает свой целевой ресурс, вообще не имеет содержимого.

Атрибут `role` предоставляет способ описания природы или функции удаленного ресурса и типа связи, установленной между документом и этим ресурсом. Значением должен быть URI, но, как и в случае пространств имен, это, скорее, уникальный идентификатор, а не указатель на некоторый требующийся ресурс. Например:

```
<image
   xmlns:xlink="http://www.w3.org/1999/xlink"
   xlink:type="simple"
   xlink:href="images/me.gif"
   xlink:role="http://www.bobsbolts.com/linkstuff/photograph"
/>
```

В данном случае мы описали целевой ресурс как `photograph` (фотография). Это отличает его от других ролей, таких как `cartoon` (мультфильм), `diagram` (диаграмма), `logo` (логотип) или прочих графических элементов `<image>`, которые могут появиться в документе. Одной из причин проведения такого различия является то, что в таблице стилей можно воспользоваться атрибутом `role` и обрабатывать каждую роль особым образом. Например, для фотографий можно сделать большую рамку, для диаграмм – маленькую, а логотипы выводить вообще без рамки.

Атрибут `title` тоже описывает удаленный ресурс, но он предназначен для прочтения людьми, а не для обработки. В случае приведенного выше `<image>` он мог быть подписью к картинке:

```
<image
   xmlns:xlink="http://www.w3.org/1999/xlink"
   xlink:type="simple"
   xlink:href="images/me.gif"
   xlink:role="http://www.bobsbolts.com/linkstuff/photograph"
   xlink:activate="onLoad"
   xlink:title="A picture of me on the beach."
/>
```

Для ссылки, приводимой в действие пользователем и указывающей на другой документ, это может быть названием того документа. Способ использования заголовка программой XML (если тот вообще ей используется) не вполне определен. Эта часть предоставляется процессору XML.

Приложение XML: XHTML

Изучать применение ссылок в реальном мире хорошо на HTML (Hypertext Markup Language) – языке, являющемся основой веб-страниц. Гипертекст – это текст, в который встроены ссылки, соединяю-

щие связанные документы. Благодаря ему, World Wide Web развилась в чрезвычайно успешную коммуникационную среду, какой она ныне является.

HTML предоставляет простую структуру для общих документов, выводимых на экран. В нем есть небольшая группа элементов, выполняющих базовые задачи структурирования без особых украшательств. Имеются управляющие элементы для реализации заголовков (<h1>, <h2> и т. д.), абзацев (<p>), списков (<ul>, <ol>), таблиц (<table>), простых внутритекстовых элементов (, <tt>) и т. д. Этот язык не очень детализирован, но его возможностей достаточно для отображения страниц на экране с целью их просмотра.

Мы собираемся рассмотреть некоторую переформулировку HTML, имеющую название XHTML. Это почти тот же HTML версии 4, но с некоторыми ограничениями, обеспечивающими совместимость с правилами XML. Каждая страница XHTML является законченным документом XML, удовлетворяющим стандарту XML версии 1.0 и совместимым со всеми инструментами общего назначения и процессорами XML. Документы XHTML также совместимы с большинством используемых сегодня броузеров HTML, при условии соблюдения основных принципов, которые буду перечислены ниже.

Применение XHTML вместо обычного HTML дает важные преимущества:

- Поскольку XHTML является стандартом, совместимым с XML, документы XHTML можно использовать с любыми редакторами XML общего назначения, средствами проверки, броузерами или другими программами, предназначенными для работы с документами XML.
- Документы, в которых выполняются более строгие правила XML, являются более ясными, более предсказуемыми и отличаются лучшим поведением в броузерах и программном обеспечении XML.
- В конечном итоге, свойства расширяемости XML благотворно скажутся на XHTML, упрощая добавление новых элементов и функций. Для этого может оказаться достаточным объявить пространство имен или использовать другое DTD.

В настоящее время есть три «сорта» XHTML: строгий (strict), переходный (transitional) и фреймовый (frameset). Вот описание различий между ними:

Строгий XHTML

Это полный разрыв с HTML, при котором за счет упразднения многих элементов устранена сильная зависимость HTML от семантики представления. Необходимо использовать с документом таблицу стилей (CSS), для того чтобы отформатировать его желательным образом. Из всех трех типов этот наиболее близок к XML и прогрессивен.

Переходный XHTML

Для тех, кто хочет, чтобы их страницы были совместимы с более старыми броузерами, не поддерживающими таблицы стилей, в этом виде XHTML сохранены элементы и атрибуты HTML. К примеру, есть настройки шрифтов и цвета.

Фреймовый XHTML

Фреймовый XHTML подобен строгому XHTML, но поддерживает применение фреймов. Наличие самостоятельной версии, работающей с фреймами, значительно упрощает другие версии для тех, кому фреймы не нужны.

Вид XHTML выбирается путем задания DTD в объявлении типа документа. Вот объявление для строгого формата:

```
<!DOCTYPE html
    PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
```

Если DTD установлено на локальной системе, следует изменить часть, относящуюся к системному идентификатору, на соответствующий локальный путь. Использование локального экземпляра DTD может заметно сократить время загрузки документа. Определения типов документа и информационные ресурсы для XHTML поддерживаются W3C (подробности можно найти в приложении B).

Рассмотрим пример 3.2, в котором приведен документ, удовлетворяющий определению строгого XHTML.

Пример 3.2. Образец документа XHTML

```
<?xml version="1.0"?>     (1)
<!DOCTYPE html (2)
    PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html (3)
  xmlns="http://www.w3.org/1999/xhtml" (4)
  xml:lang="en" lang="en"> (5)

  <head>
    <title>Evil Science Institute</title>
  </head>

  <body>
    <h1>Evil Science Institute</h1> (6)
    <p><em>Welcome</em> to Dr. Indigo Riceway's Institute for Evil
                Science!</p> (7)

    <h2>Table of Contents</h2>
    <ol>
      <li><a href="#staff">Meet Our Staff</a></li> (8)
```

Пример 3.2. Образец документа XHTML (продолжение)

```
  <li><a href="#courses">Exciting Courses</a></li>
  <li><a href="#research">Groundbreaking Research</a></li>
  <li><a href="#contact">Contact Us</a></li>
</ol>

<a name="staff" /> (9)
<h2 id="staff">Meet Our Staff</h2>
<dl>
  <dt><a href="riceway.html">Dr. Indigo Riceway</a></dt>
  <dd>
    <img src="images/riceway.gif" width="60" height="80" /> (10)
    Founder of the institute, inventor of the moon magnet and the
    metal-eating termite, three-time winner of Most Evil Genius
    award. Teaches Death Rays 101, Physics, Astronomy, and Criminal
    Schemes.
  </dd>
  <dt><a href="grzinsky.html">Dr. Ruth "Ruthless" Grzinsky</a></dt>
  <dd>
    <img src="images/grzinsky.gif" width="60" height="80" /> (11)
    Mastermind of the Fort Knox nano-robot heist of 2002.
    Teaches Computer Science, Nanotechnology, and Foiling Security
    Systems.
  </dd>
  <dt><a href="zucav.html">Dr. Sebastian Zucav</a></dt>
  <dd>
    <img src="images/zucav.gif" width="60" height="80" />
    A man of supreme mystery and devastating intellect.
    Teaches Chemistry, Poisons, Explosives, Gambling, and
    Economics of Extortion.
  </dd>
</dl>

<a name="courses" />
<h2 id="courses">Exciting Courses</h2>
<p>   Choose from such
  intriguing subjects as</p> (12)
<ul>
  <li>Training Cobras to Kill</li>
  <li>Care and Feeding of Mutant Beasts</li>
  <li>Superheros and Their Weaknesses</li>
  <li>The Wonderful World of Money</li>
  <li>Hijacking: From Studebakers to Supertankers</li>
</ul>

<a name="research" />
<h2 id="research">Groundbreaking Research</h2>
<p>Indigo's Evil Institute is a world-class research facility.
      Ongoing projects include:</p>
```

Пример 3.2. Образец документа XHTML (продолжение)

```
    <h3>Blot Out The Sky</h3>
    <p>A diabolical scheme to fill the sky with garish neon
      advertisements unless the governments of the world agree to pay us
      one hundred billion dollars. Mha ha ha ha ha!</p>

    <h3>Killer Pigeons</h3>
    <p>A merciless plan to mutate and train pigeons to become efficient
      assassins, whereby we can command huge bounties by blackmailing
      the public not to set them loose. Mha ha ha ha ha!</p>

     <h3>Horror From Below</h3>
     <p>A sinister plot so horrendous and terrifying, we dare not
      reveal it to any but 3rd year students and above. We shall only
      say that it will be the most evil of our projects to date!
      Mha ha ha ha ha!</p>

     <a name="contact" />
     <h2 id="contact">Contact Us</h2>
     <p>If you think you have what it takes to be an Evil Scientist,
      including unbounded intellect, inhumane cruelty, and a sincere
      loathing of your fellow man, contact us for an application. Send
      a self-addressed, stamped envelope to:
     </p>
     <address>The Evil Science Institute,
Office of Admissions,
10 Clover Lane,
Death Island,
Mine Infested Waters off the Coast of Sri Lanka</address>

  </body>
</html>
```

Ниже следуют некоторые примечания к этому примеру кода:

1. Объявление XML в данном примере не требуется, но использование его разумно, особенно если предполагается применение кодировки, отличной от UTF-8. К несчастью, некоторые старые броузеры HTML некорректно интерпретируют инструкции обработки (PI) и могут полностью или частично выводить объявление XML.
2. Объявление типа документа необходимо для проверки версии и вида XHTML. Обратите внимание, что нельзя использовать объявления во внутреннем подмножестве: это дезориентирует многие броузеры, понимающие XHTML.
3. Корневым элементом всегда является `<html>`. Заметьте, что в XHTML все элементы без исключения должны полностью записываться в нижнем регистре, в отличие от HTML, где регистр не имеет значения.

❹ Объявление пространства имен по умолчанию тоже обязательно. Пространством имен для всех видов XHTML является `http://www.w3.org/1999/xhtml`.

❺ В переходных документах следует использовать оба элемента – `<lang>` и `<xml:lang>`. Некоторые броузеры не распознают последний, но те, которые это делают, отдают ему предпочтение.

❻ Элемент `<h1>` является примером заголовка раздела. В отличие от DocBook, в котором разделы целиком содержатся в особых элементах типа `<sect1>`, XHTML не поддерживает элементы разделов. Их заменяют элементы, содержащие заголовки, стили которых «сообщают» о начале нового раздела. Это пример того, как информация о представлении проникает в разметку за счет структуры.

❼ Существенным отходом от прежнего HTML является то, что в XHTML все элементы, имеющие содержимое, должны иметь закрывающий тег; ранее иногда можно было обойтись без них. Элемент `<p>` всегда должен иметь открывающий и закрывающий теги, даже если между ними нет никакого содержимого.

❽ Использованный здесь элемент `<a>` является простой ссылкой на позицию в том же документе. Знакомый атрибут `href` содержит идентификатор фрагмента. Остальные атрибуты XLink скрыты, будучи встроенными в определение `<a>` в DTD (в главе 5 мы узнаем, как задавать неявные атрибуты). Эта ссылка приводится в действие пользователем: содержимое элемента выводится особым образом и он превращается в управляющий элемент, который, будучи выбранным пользователем, активизирует ссылку. Режимом действия при активации является немедленный доступ по ссылке и замена текущего документа.

❾ Этот элемент `<a>` использует атрибут `name` для создания цели, на которую указывает ссылка. В XML можно делать ссылки на любые элементы с помощью XPointer. Поэтому данный прием использования специального элемента как выделенной цели ссылки является анахронизмом, но включен сюда для обратной совместимости с более старыми броузерами.

❿ Это пример пустого элемента, в котором закрывающий ограничитель (`/>`) подчиняется законам построения корректных документов XML. Однако мы добавили перед ним пробел, т. к. это помогает некоторым броузерам отличить пустой тег от контейнера. Следует избегать применения синтаксиса элемента-контейнера с элементами, которые не должны иметь содержимое (например `<br></br>`), т. к. это может привести к непредсказуемым результатам.

⓫ `<img>` является еще одним примером связующего элемента, в данном случае – для импорта и вывода графического файла. В отличие от `<a>`, его параметрами являются `actuation="auto"` и `show="embed"`.

Это означает, что когда выводится страница, вся графика импортируется и отображается в потоке документа.

⓬ В XHTML все элементы, кроме <pre>, выбрасывают при форматировании лишние пробельные символы. Программа форматирования выкидывает пробелы в начале и конце содержимого и сжимает серии пробелов до одиночного пробела. Это необходимо, чтобы абзацы выглядели «прилично», несмотря на все пробелы, символы табуляции и перевода строки, использованные для создания отступов и улучшения читаемости XML. В XML все элементы тщательно сохраняют пробельные символы, если только особо не указано, что этого делать не нужно.

Теперь должно стать ясным, что XHTML является очень важным шагом в эволюции HTML. Веб-страницы станут чище и будут совместимы с большим числом броузеров, а также, впервые, с инструментами XML. Удаление настроек стиля из разметки, которое пытается осуществить более строгая версия, вынудит авторов использовать вместо них таблицы стилей. Применение таблиц стилей приведет к ускорению разработки поддержки стилей и более богатому представлению.

В будущем XHTML станет развиваться в направлении модульности, т. е. скоро DTD станут создаваться из взаимозаменяемых частей, называемых *модулями*. Когда это произойдет, в документах HTML можно будет комбинировать наборы элементов с целью конструирования документов почти для любых целей, включая устройства для Интернета, беспроводные устройства, клиенты синтезированной речи и прочие.

Хотим, однако, предупредить, что XHTML не решает всех задач разметки. Его общие элементы могут оказаться недостаточно детализированными для ваших задач, а отсутствие структуры вложенных разделов является препятствием для создания больших и сложных документов. Но в качестве компактного языка общего назначения для размещения страниц в Сети он превосходит всех своих конкурентов.

- *Зачем нужны таблицы стилей?*
- *Обзор CSS*
- *Правила*
- *Свойства*
- *Практический пример*

4

Представление: создание конечного продукта

Таблицы стилей играют важную роль в мире XML, являясь мостом между кристаллизованным, лишенным стиля оформлением информации и конечным продуктом, пригодным к использованию человеком. Они представляют собой подробные инструкции по преобразованию разметки XML в новую форму, например, HTML или PDF.

Зачем нужны таблицы стилей?

Документ XML и таблица стилей дополняют друг друга. Документ является сущностью, или смыслом информации, тогда как таблица стилей описывает форму, которую он принимает (рис. 4.1).Представьте себе применение таблицы стилей к документу, как приготовление блюда согласно поваренной книге. Документ XML представляет собой набор сырых необработанных ингредиентов; таблица стилей служит рецептом, из которого повар узнает, как готовить каждый из ингредиентов и как соединить их вместе. Программное обеспечение, преобразующее XML в другой формат на основе инструкций таблицы стилей, в нашей аналогии соответствует повару. После нарезки, перемешивания и выпекания мы получаем нечто вкусное и съедобное.

Поощрение хороших привычек

Отделение разметки от стиля может показаться лишней трудностью: HTML используется весьма успешно и не требует никаких таблиц стилей. Его элементы обладают внутренними настройками представления, которые используются всеми броузерами для создания прилично

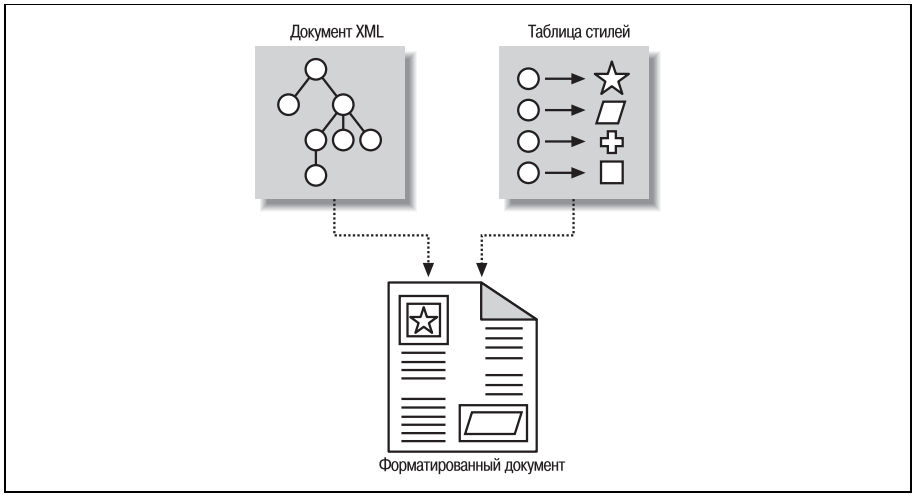


Рис. 4.1. *Таблица стилей помогает создать форматированный документ*

выглядящих страниц. Однако такое разделение имеет следующие преимущества:

- Хранение информации о стиле вне разметки освобождает автора, позволяя ему сосредоточиться на смысле документа и не беспокоиться о его внешнем виде. В HTML выбор элемента основывается как на его функции, так и стиле. Применение таблицы стилей позволяет сохранить чистоту и концентрированность разметки, не замутняя ее деталями внешнего вида.
- Хранение настроек стилей в отдельном документе облегчает работу дизайнера. В то время как автор сосредоточен на создании текста, художник контролирует внешний вид. Одна и та же таблица стилей может использоваться с сотнями документов, гарантируя им единообразный внешний вид и сокращая затраты труда на обновление документов (рис. 4.2).
- Увеличивается выбор вариантов представления документа. Комбинируйте XML с различными таблицами стилей в зависимости от назначения. Например, можно поддерживать несколько размеров отображения: обычный, уменьшенный и увеличенный; можно настроить параметры вывода на принтер, используя при этом мелкий шрифт и высококачественную графику; можно ориентироваться на аудиторию с особыми потребностями, форматируя документ для азбуки Брайля, звукового вывода и неграфического текста. Без информации о стилях документ XML становится действительно независимым от устройств, как показано на рис. 4.3.
- Таблицы стилей можно комбинировать, заменяя их части в зависимости от конкретных потребностей. Например, можно применить таблицу стилей общего назначения, объединив ее с другой, которая осуществляет тонкую настройку стиля для отдельного продукта. Та-

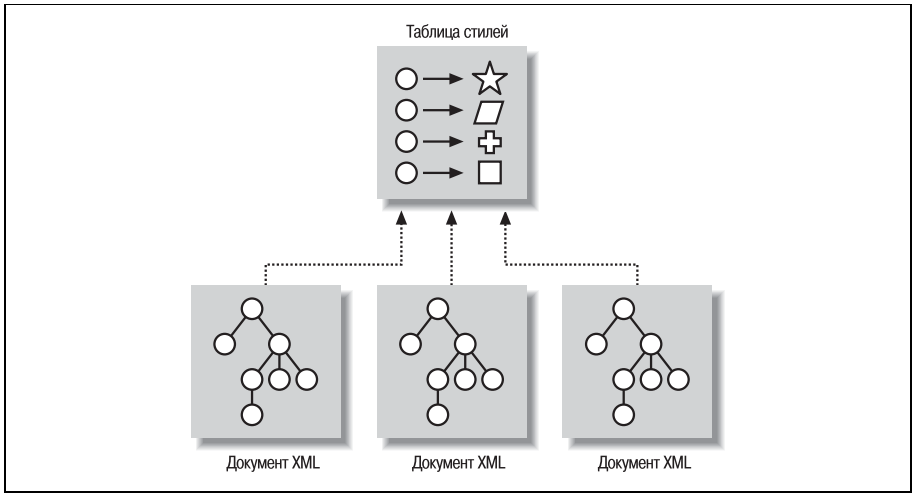


Рис. 4.2. *Одна таблица стилей может использоваться многими документами XML*

кой прием *каскадирования* таблиц стилей позволяет осуществлять слияние стилей из многих различных источников (рис. 4.4).

В некоторых случаях даже конечный пользователь может сказать свое слово по поводу того, как должен выглядеть документ. Допустим, например, что при чтении документа в режиме online шрифт кажется

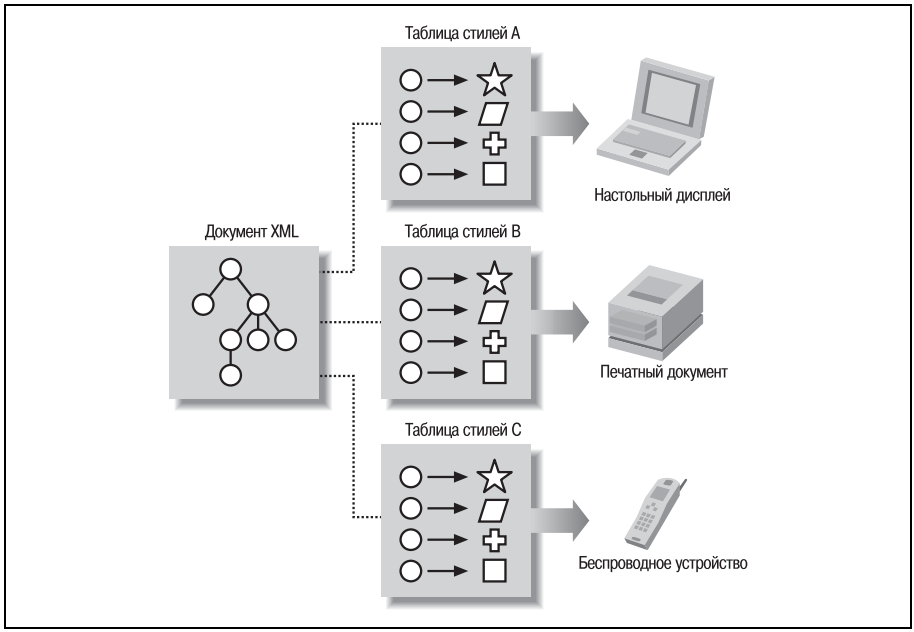


Рис. 4.3. *Комбинирование таблиц стилей для различных задач*

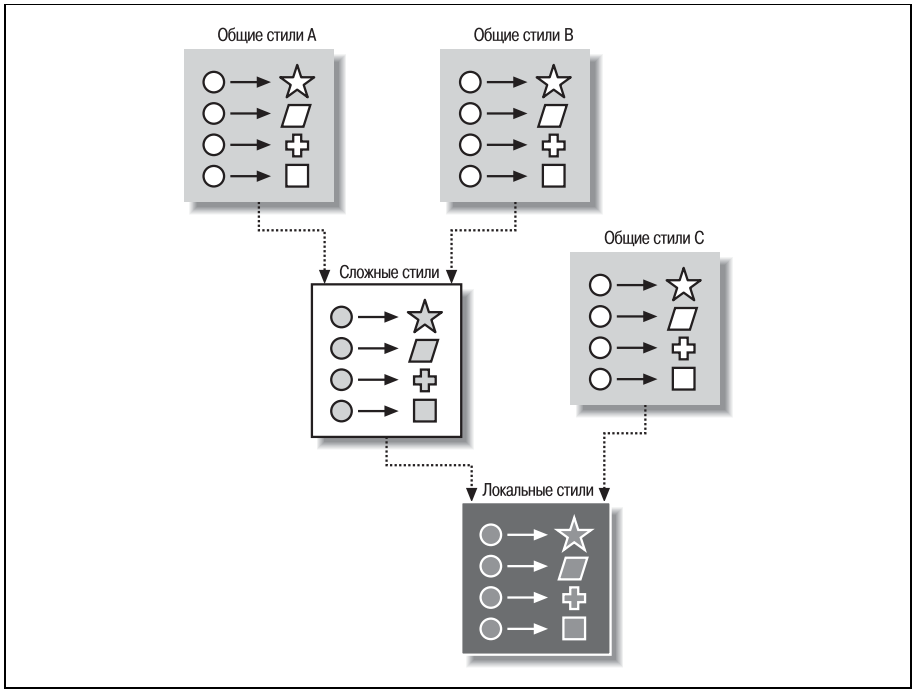


Рис. 4.4. *Каскад таблиц стилей*

слишком мелким. Можно переопределить установку в документе и использовать более крупный шрифт, фактически добавляя в каскад еще одну таблицу стилей, переопределяющую некоторые из свойств.

Электронные документы могут представлять собой источник проблем, поскольку системные ресурсы бывают различными. Набор шрифтов, установленных на компьютере пользователя, не известен разработчику и может в огромной степени влиять на внешний вид документа. Таблицы стилей позволяют проявить гибкость и указать альтернативные шрифты, если первоначально выбранные отсутствуют на машине пользователя.

Изживайте дурные привычки

Таблицы стилей – не новое изобретение, но до недавнего времени они не применялись в Сети широко. Либо из-за того, что обработку таблиц стилей трудно реализовать в броузере, либо потому, что таблицы стилей конфликтуют с программами конкурирующих поставщиков, но многим веб-дизайнерам приходилось довольствоваться встроенным стилем HTML с его ограничениями. Это вызвало к жизни творческие подходы к расширению рамок стилей:

Использование графики вместо текста

Этот прием дает простой способ поместить на веб-страницу практически любой видимый элемент. Если необходимо использовать особый, редко встречающийся шрифт, которого может не оказаться на машине пользователя, можете смастерить текст в своем графическом редакторе и показать его на экране как растровое изображение. Смотрится отлично, но это слабое решение.

Заменив текст графикой, мы сделали его невидимым для какой-либо автоматической обработки. Поисковые механизмы его не найдут, вырезать и вставлять слова становится невозможным, проверить правописание текста нельзя. Документ также жестко привязывается к одному средству представления. Он становится бесполезен для систем, синтезирующих звук или азбуку Брайля, а также для текстовых броузеров, и, кроме того, он медленнее загружается.

Использование побочных эффектов броузера

В любой компьютерной программе, как бы хорошо она ни была написана и тщательно протестирована, есть ошибки. Поэтому представление документа в каждом броузере имеет некоторое своеобразие. Одним из худших применяемых приемов является использование таких странностей для создания особых эффектов представления.

Например, многие разработчики применяют элементы `<table>` в качестве общей структуры расположения. Это нравится разработчикам, поскольку позволяет располагать текст в нескольких колонках, осуществлять строгий контроль ширины и размещать все – от графики до вложенных таблиц. Но для этого ли был изначально предназначен данный элемент?

Другой возможной проблемой является чрезмерный контроль внешнего вида, скажем, применение элементов `<font>` для установки шрифта и размера. Некоторые генерирующие HTML программы (скажем, текстовые процессоры с функцией «Сохранить как HTML») используют тег переноса строки `<br>` после каждой строки, вместо того чтобы позволить броузеру форматировать параграфы.

Такие приемы создают негибкие, зависящие от устройств документы, которые будут по-разному выглядеть в каждом броузере, а могут вообще выглядеть абракадаброй.

«Усовершенствования» производителя

Планы производителей программного обеспечения не всегда совпадают с интересами широкой общественности. Их цель состоит в извлечении прибыли, а потому они хотят, чтобы их продукты использовало возможно большее число людей. Одним из средств достижения этого является создание искусных патентованных функций, блокирующих конкуренцию.

Подобная коллизия имела место в конце 1990-х годов между крупными производителями броузеров (Netscape и Microsoft). Оба титана добивались своих целей, создавая все новые и новые расширения HTML, которые работали только в их броузерах. Организации стандартизации (консорциум W3C), регулирующей HTML, оставалось только наблюдать, как язык (а в результате и Сеть) раскалывался на конкурирующие между собой феоды. На некоторых веб-страницах даже стало возможным прочесть сообщения, гласящие, что «эту страницу лучше всего просматривать с помощью броузера X».

Патентованные технологии

К счастью, народ устал от этого нелепого противостояния («мой броузер лучше, чем твой»), и война между броузерами несколько затихла. Вместо нее разгорелась другая война. Программные расширения проникли в области, куда HTML никогда и не «надеялся» попасть. Java и JavaScript (последний не имеет отношения к Java, а представляет собой просто беспринципное маркетинговое решение) являются языками программирования, встраивающими (довольно грубо) в веб-страницы специальные функции. Пакеты векторной анимации типа Macromedia Flash превращают веб-страницы в маленькие телевизоры, посредством которых можно смотреть мультфильмы и играть в игры.

Это не так уж плохо с *технической* точки зрения. В действительности, электронные технологии и служат тому, чтобы в одном документе могли содержаться многие виды носителей. Но эти технологии могут стать источником неприятностей точно так же, как злоупотребление графикой для создания красиво оформленных страниц. Что сделает пользователь, попавший на веб-страницу, созданную с применением интерактивной анимации Flash в меню навигации и не предоставляющую текстовой альтернативы? Он будет вынужден загрузить подключаемый модуль Flash – разумеется, если для его любимого броузера таковой существует.

Применение патентованных технологий для представления информации довольно опасно. Конечный пользователь должен приобрести эту технологию, поддерживая ее даже в том случае, если за нее не нужно платить. Она является зависимой от устройств, требуя броузеров, которые поддерживают эту технологию, и поставщикам слабо удается продвигать свои продукты в небольшие группы, не обладающие достаточной покупательной способностью. Кроме того, это замыкает автора содержимого на инструментарий из одного источника, лишая его преимуществ открытых стандартов. Поэтому, когда это возможно, используйте технологии, которые открыты и доступны для всех, чтобы сделать свою информацию как можно более полезной и универсальной.

Теперь все готово для первого участника истории с таблицами стилей – *каскадных таблиц стилей* (*Cascading Style Sheets, CSS*). CSS отлича-

ются простотой использования, легкостью изучения и достаточной мощью, хотя и несколько недооцененной производителями броузеров. Они описываются здесь для того, чтобы показать, чего можно достичь с помощью таблиц стилей. Едва ли CSS принадлежит будущее, т. к. разрабатываются более мощные языки, такие как XSL, но для наших педагогических целей это хорошее средство обучения.

Каскадные таблицы стилей

Стандарт каскадных таблиц стилей Cascading Style Sheets (CSS) является рекомендацией консорциума World Wide Web Consortium (W3C). Он возник в 1994 г., когда Хэкон Вайум Ли (Hakon Wium Lee), работая в CERN (месте, где родился HTML), опубликовал статью под названием «Cascading HTML Style Sheets». Это был сильный ход, сделанный в нужный момент. К тому времени Сети было четыре года, и она быстро развивалась, однако согласия по поводу стандартного языка описания стиля еще не было. Создатели HTML понимали, что этот язык подвергался опасности стать языком описания стиля, если в скором времени не будет принято что-то вроде CSS.

Целью было создать простой, но выразительный язык, способный объединять описания стиля из разных источников. Другой язык описания стиля, DSSSL, уже применялся для форматирования документов SGML. Будучи очень мощным, DSSSL был слишком большим и сложным для практического использования в Сети. Это полный язык программирования, обеспечивающий большую точность и логическую выразительность, чем CSS, который является простым языком, нацеленным на основные потребности небольших документов.

Хотя в то время, когда были предложены CSS, существовали другие языки таблиц стилей, ни один из них не предоставлял возможности объединения нескольких источников в один набор описаний стилей. Такая возможность и подразумевается термином *каскадирование*, при котором несколько таблиц стилей сливается в одну, как водопады, образующие реку. CSS делает Сеть подлинно доступной и гибкой, позволяя читателю заменять стили, примененные автором, чтобы приспособить документ к конкретным требованиям и приложениям. Например, документ, подготовленный для графического броузера, может быть адаптирован для аудио-носителя.

W3C предложил первую рекомендацию CSS (позднее названную CSS 1) в 1996 году. Вскоре после этого по теме «Каскадные таблицы стилей и свойства форматирования» сформировалась рабочая группа W3C, которая должна была добавить недостающие функции. Их рекомендация CSS 2, вышедшая в 1998 г., увеличила число свойств языка с 50 до более чем 120. В нее также были добавлены такие понятия, как генерируемый текст и отбор по атрибутам, а также новые средства отображения, помимо экрана монитора. Сейчас продолжается работа

над CSS 3, но, с учетом низких темпов реализации производителями броузеров, CSS могут быть заменены более новыми рекомендациями таблиц стилей, например, XSL, прежде чем будет завершена работа над уровнем 3. CSS обсуждаются в приложении B «Таксономия стандартов».

Ожидается поступление: XSL–FO

В данной книге не обсуждается Extensible Stylesheet Language for Formatting Objects (XSL-FO) – расширяемый язык таблиц стилей для форматирования объектов. Этот важный стандарт пока еще является рабочим проектом, поэтому, вместо того чтобы строить предположения, мы оставим его для будущих изданий. Кроме того, нет реализаций этого языка, с помощью которых можно было бы проверить примеры, поэтому данная тема является, в лучшем случае, академической. Те, кто захочет проследить ее развитие, могут воспользоваться материалами приложения A «Ресурсы», а технические подробности рассмотрены в приложении B.

XSL-FO должен стать предпочтительным языком таблиц стилей для сложного форматирования, поскольку он значительно более детализирован, чем CSS. Он имеет также то преимущество, что сам является приложением XML (созданным с помощью правил XML). Его можно редактировать с помощью инструментов XML, и он более тесно связан со структурой вложенных контейнеров XML.

XSL-FO носит следы сходства с XSLT (см. главу 6 «Трансформация: изменение назначения документов»), что не является просто совпадением. Оба языка представляют собой подмножества более крупного языка таблиц стилей, который называется XSL. Они используют одинаковые управляющие структуры и другие базовые элементы синтаксиса, описанные далее в главе 6 «Трансформация: изменение назначения документов». В будущем XSL-FO и XSLT будут совместно использоваться для форматирования документов.

Обзор CSS

В этом разделе бегло рассмотрены главные темы CSS.

Объявление таблицы стилей

Чтобы связать таблицу стилей с документом, необходимо объявить ее в начале, чтобы процессор XML знал, какую таблицу стилей использовать, и где она находится. Обычно это делается с помощью инструкции обработки, синтаксис которой показан на рис. 4.5. Как и все инструкции обработки, она будет проигнорирована всеми процессорами XML,

которым не требуется таблица стилей или которые их не распознают. В данном разделе мы обсуждаем подмножество процессоров, которые действительно преобразуют XML в другой формат с использованием таблиц стилей, например, веб-броузеры, которые могут форматировать XML, превращая его в симпатичную страницу.

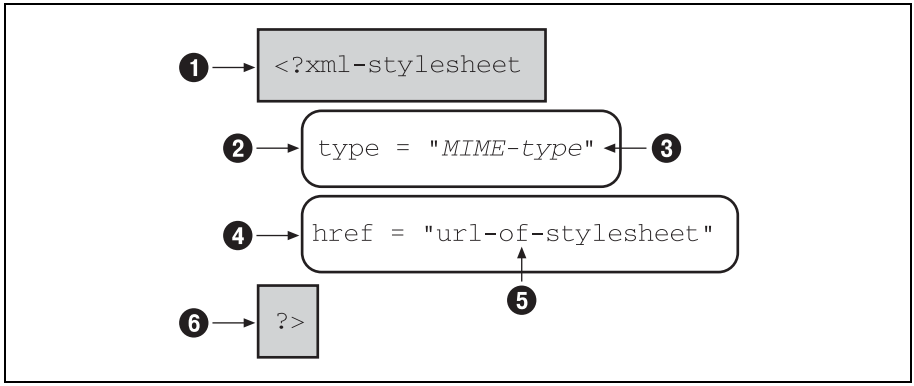


***Рис. 4.5.** Синтаксис объявления таблицы стилей*

Объявление начинается с ограничителя инструкции обработки и указания цели `<?xml-stylesheet>` (1). В инструкции обработки содержатся два назначения свойств, аналогичные атрибутам. Первому свойству, `type` (2), присваивается значение типа MIME (3) таблицы стилей (для CSS это `text/css`). Значением другого свойства, `href` (4), является URL таблицы стилей (5), которая может находиться на той же самой машине или в произвольном месте Интернета. Объявление завершается закрывающим ограничителем (6).

Вот как объявление используется в документе:

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/css" href="bookStyle.css"?>
<book>
  <title>Tom Swift's Aerial Adventures</title>

  <chapter>
    <title>The Dirigible</title>
  ...
```

Объединение нескольких таблиц стилей

Очень важной характеристикой CSS является возможность объединения нескольких таблиц стилей путем импортирования их одна в другую. Это позволяет заимствовать предопределенные стили, чтобы не заниматься постоянно изобретением велосипеда. Любые установки

стиля, которые нужно переопределить или которые вам не требуются, могут быть заменены в локальной таблице стилей.

Одной из причин, по которым объединяются таблицы стилей, является необходимость обеспечения *модульности*. Разбиением одной большой таблицы стилей на несколько маленьких файлов можно добиться лучшей управляемости. Например, можно сохранить все стили, относящиеся к математическим уравнениям, в *math.css*, а все стили для обычного текста в *text.css*. Команда `@import` соединяет текущую таблицу стилей с другой и влечет импортирование настроек стиля из целевой таблицы:

```
@import url(http://www.mycompany.org/mystyles/math.css);
@import url(http://www.mycompany.org/mystyles/text.css);

    Прочее содержание таблицы стилей...
```

Некоторые из импортированных правил стилей могут вам не понравиться или оказаться не вписывающимися в представление. Можно заменить эти правила, переопределив их в собственной таблице стилей. В данном случае мы решили, что правила для элементов `<h1>`, определенные в *text.css*, должны быть изменены:

```
@import url(http://www.mycompany.org/mystyles/text.css);

h1: { font-size: 3em; }    /* переопределение */.
```

Как действуют таблицы стилей

В следующих разделах более подробно рассмотрены каскадные таблицы стилей (CSS).

Соответствие свойств элементам

Таблица стилей CSS является совокупностью *правил* (*rules*). В каждом правиле содержатся настройки стиля, которые нужно использовать с конкретным типом элемента. Процессор CSS (программа, преобразующая XML в форматированный документ с использованием таблицы стилей CSS) проходит поочередно каждый элемент документа XML, пытается найти самое лучшее правило, которое можно к нему применить, и строит изображение.

В качестве аналогии приведем раскраску по номерам. Набор состоит из красок, кисточки и холста, на котором очерчены закрашиваемые области. Каждая область имеет номер, соответствующий цвету закраски. Постарайтесь закрашивать по линиям, и в итоге получится стилизованное изображение пасторали с коровами и старым коровником. Закрашиваемые области похожи на элементы в документе XML. Баночки красок с номерами на них подобны правилам таблиц стилей.

Процесс идет так:

1. Выберите следующую незакрашенную область.
2. Найдите краску, соответствующую номеру области.
3. Залейте область этой краской.
4. Повторяйте, пока не закончатся незакрашенные области.

Процесс для CSS аналогичен:

1. Выберите очередной форматируемый элемент.
2. Найдите правило, наилучшим образом соответствующее элементу.
3. Используйте настройки стилей из правил форматирования элемента.
4. Повторяйте, пока не закончатся неформатированные элементы.

В каждом правиле есть две части: *селектор* (*selector*), который сопоставляет правила элементам, и *объявление свойств* (*properties declaration*), в котором содержатся настройки стиля. Правило CSS выглядит так:

```
sidebar {
  border: thin gray solid;
  padding: 10pt;
}
```

Качественно это все равно, что сказать «для всех элементов `<sidebar>` заключить область в тонкую серую границу и со всех сторон сделать для текста отступ в 10 пунктов». Селектор соответствует любому элементу `<sidebar>`, а объявление содержит два свойства: `border` и `padding`.

Разрешение конфликтов правил

Как говорилось ранее, процессор CSS пытается найти *лучшее* правило (или правила) для каждого элемента. В таблице стилей может быть несколько применимых правил. Например:

```
p.big {
  font-size: 18pt;
}

p {
  font-family: garamond, serif;
  font-size: 12pt;
}

* {
  color: black;
  font-size: 10pt;
}
```

Предположим, что следующим обрабатываемым элементом является <p> с атрибутом class="big". Все три правила соответствуют этому элементу. Как CSS решает, какие свойства применить?

Решение этой дилеммы состоит из двух частей. Первая заключается в использовании всех соответствующих правил (как если бы все применимые правила слились в одно множество). Это значит, что все правила потенциально могут быть применены к элементу:

```
font-size: 18pt;
font-family: garamond, serif;
font-size: 12pt;
color: black;
font-size: 10pt;
```

Вторая часть заключается в том, что избыточные установки свойств разрешаются согласно алгоритму. Как можно видеть, есть три разных установки свойства font-size. Использовать можно только одну из настроек, поэтому процессор CSS должен выбросить худшие две, применив систему разрешения конфликтов свойств. Практическое правило гласит, что побеждает правило с наиболее специфическим селектором. Первое свойство font-size происходит из правила с селектором p.big, который более конкретен, чем p или *, поэтому оно побеждает.

Теперь к текущему элементу применимы следующие свойства:

```
font-size: 18pt;
font-family: garamond, serif;
color: black;
```

Наследование свойств

В документах XML соблюдается иерархия элементов. CSS использует эту иерархию для передачи свойств в процессе, называемом *наследованием* (*inheritance*). Вернувшись в пример DocBook, мы обнаружим, что <sect1> содержит элемент <para>. Рассмотрим такую таблицу стилей:

```
sect1 {
  margin-left: 25pt;
  margin-right: 25pt;
  font-size: 18pt;
}

para {
  margin-top: 10pt;
  margin-bottom: 10pt;
  font-size: 12pt;
}
```

Набор свойств элемента <para> является комбинацией тех, которые явно в нем объявлены, и унаследованных от его элементов-предков (не считая тех, которые были попутно переопределены). В данном случае

он унаследовал свойства `margin-left` и `margin-right` элемента `<sect1>`. Он не наследует `font-size` из `<sect1>`, потому что это свойство объявлено в `<para>` явным образом и закрывает более раннее объявление.

Комментарии

Подобно тому, как XML позволяет вставлять комментарии, которые игнорируются процессором XML, в CSS есть собственный синтаксис для комментариев. Комментарий начинается с разделителя `/*` и заканчивается разделителем `*/`. Он может охватывать несколько строк и содержать правила CSS, которые нужно изъять из рассмотрения:

```
/* эта часть будет проигнорирована
gurble { color: red }
burgle { color: blue; font-size: 12pt; }
*/
```

Ограничения, свойственные CSS

Хотя CSS достаточно для многих задач вывода, свойственная им простота может послужить препятствием более сложному форматированию. Одна из проблем связана с тем, что порядок обработки элементов не может быть изменен. Рассмотрим следующий фрагмент XML:

```
<figure>
<title>The Norwegian Ridgeback Dragon</title>
  <graphic fileref="nr_dragon.png"/>
</figure>
```

В CSS элементы обрабатываются в порядке их появления в документе XML. Поэтому в данном примере название рисунка будет выведено раньше передачи графики. В то время как язык таблиц стилей, основанный на шаблонах, например XSL, может поместить название под рисунком, CSS этого сделать не может.

Поскольку CSS не может создавать структуру, отличающуюся от присутствующей в исходном файле XML, есть много видов представлений, осуществить которые невозможно. Например, нельзя создать оглавление путем поиска в документе всех заголовочных элементов и сборки из них списка. Для этого нужен такой язык, как XSLT, обладающий возможностями XPath для сбора узлов и обработки их в любом порядке.

Синтаксис селектора CSS ограничен именами элементов, атрибутами и контекстом элементов. Для некоторых задач этого недостаточно. Иногда требуется выполнить арифметические действия над позициями элементов и их значениями. Может понадобиться реализация логических рассуждений: «Если у этого элемента нет потомков, то вывести его зеленым цветом, в противном случае – красным». Может потребоваться проследить ссылки и посмотреть, что находится на дру-

гом конце. Для такой усложненной обработки у CSS недостаточно сил, и вам придется перейти на XSLT.

Эти ограничения не помешают нам рассмотреть работу таблиц стилей на начальном уровне. Но в главе 6 мы поднимемся на более высокий уровень выразительности таблиц стилей, и читатель «почувствует разницу».

Правила

Теперь рассмотрим синтаксис и семантику правил CSS более пристально, не акцентируя пока внимания на значениях свойств. Сосредоточимся, чтобы понять, как реализуется соответствие правил и элементов и как процессор CSS выбирает правила.

Синтаксис правила CSS показан на рис. 4.6. Оно состоит из *селектора* (1) для поиска элементов и *объявления* (2) для описания стилей. Объявление является списком, составленным из присвоений значений именам (3), в котором каждому *свойству* стиля (4) присваивается *значение* (5) через разделитель-двоеточие (:).

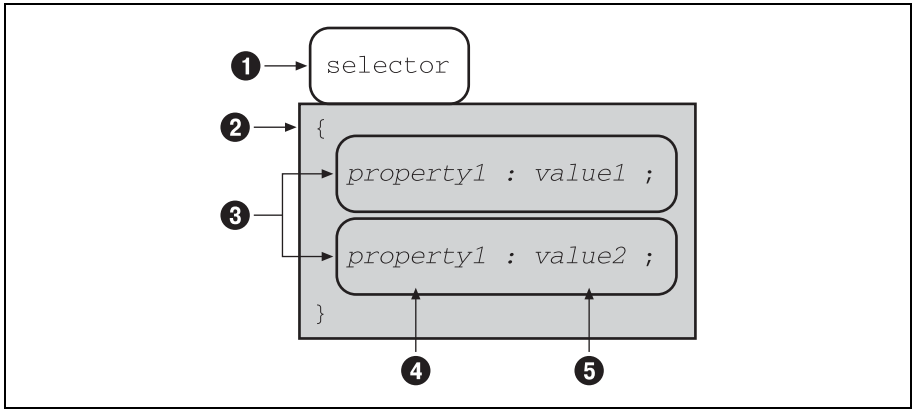


***Рис. 4.6.** Синтаксис правила CSS*

Например:

```
emphasis {
  font-style: italic;
  font-weight: bold; }
```

Для каждого элемента `<emphasis>`, которому соответствует это правило, будут назначены свойства стиля: `font-style` (стиль шрифта) – курсив и `font-weight` (плотность шрифта) – полужирный.

Поиск элемента по имени представляет лишь вершину айсберга. Задать отбор можно многими способами. Можно задавать имена атрибу-

тов, значения атрибутов и элементы, следующие перед ним или после него; можно использовать символы подстановки и специальные параметры, такие как язык или носитель.

На рис. 4.7 показан общий синтаксис селекторов. Обычно они состоят из имени элемента (1), за которым в квадратных скобках следует некоторое число проверок атрибутов (2), в свою очередь содержащих имя атрибута (3) и его значение (4). Заметьте, что каждая из этих частей является необязательной, если задан по крайней мере один элемент или атрибут. Имя элемента может содержать символы подстановки, соответствующие любому элементу, и цепочки элементов, задающие информацию об иерархии. Проверки атрибутов могут включать существование атрибута (с любым значением), существование значения (для любого атрибута) или, в самом строгом случае, конкретной комбинации атрибутов-значений.

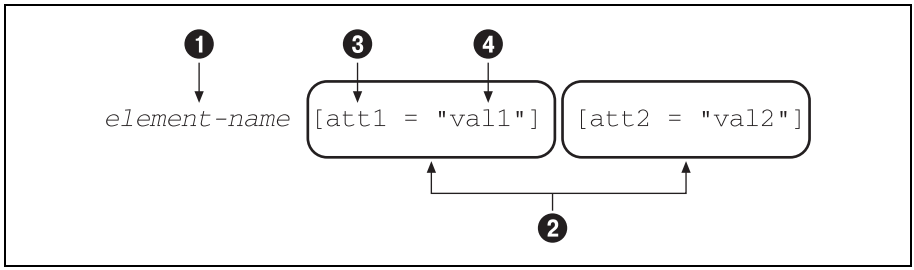


Рис. 4.7. *Синтаксис селектора CSS*

В простейшем случае, как мы только что видели, селектор ищет элементы только по имени. Можно использовать звездочку (*) в качестве символа подстановки, соответствующего любому имени элемента. Например, следующее правило применяется ко всем элементам документа, устанавливая для него голубой цвет:

```
* { color: blue }
```

Поскольку это селектор весьма общего вида, он имеет низкий приоритет в наборе правил. Процессор CSS применяет его в качестве последнего резерва, когда не может найти каких-либо более специфических селекторов правил.

Отбор по атрибутам

Для более тонкого контроля можно задать отбор элементов по их атрибутам. *Селектор атрибутов* (*attribute selector*) состоит из имени типа элемента, сразу за которым следует уточняющий атрибут в квадратных скобках. Есть возможность варьировать уровень точности:

`planet[atmosphere]`

Этот селектор соответствует любому элементу `<planet>`, у которого есть атрибут `atmosphere`. Например, он отбирает `<planet atmosphere="poisonous">` и `<planet atmosphere="breathable">`, но не `<planet>`.

Можно опустить элемент, если надо найти только атрибут. Поэтому `[atmosphere]` соответствует `<planet atmosphere="dense">` и `<moon atmosphere="wispy">`.

`planet[atmosphere="breathable"]`

Добавление значения делает селектор еще более узким. Этот селектор соответствует `<planet atmosphere="breathable">`, но не соответствует `<planet atmosphere="poisonous">`.

`planet[atmosphere~="sweet"]`

Если значением атрибута является разделенный пробелами список строк, можно искать соответствие любой из них с помощью оператора `~=` вместо знака равенства (`=`). Этот селектор соответствует `<planet atmosphere="breathable sweet dense">` или `<planet atmosphere="foggy sweet">`, но не соответствует `<planet atmosphere="breathable stinky">`.

`planet[populace|="barbaric"]`

Аналогично предшествующему варианту, селектор с оператором `|=` соответствует элементу в списке значений, разделенных дефисами, если список начинается со значения, приведенного в селекторе. Этот селектор соответствует `<planet populace="barbaric-hostile">`.

Такой селектор часто служит для различения типа языка. Значением атрибута `xml:lang` является *идентификатор языка*, строка, имеющая вид типа: `en-US`. Двухбуквенный код «en» обозначает «English», а код «US» указывает на версию, используемую в США. Чтобы найти элемент `<planet>` с атрибутом `xml:lang`, указывающим английский язык, используйте селектор `<planet[language|="en"]>`. Он находит как `en-US`, так и `en-UK`.

`planet[atmosphere="breathable"][populace="friendly"]`

Селекторы могут связывать вместе несколько требований к атрибутам. Соответствие обнаруживается, если выполняются все селекторы атрибутов, как если бы они были связаны логическим оператором AND. Приведенный выше селектор соответствует `<planet atmosphere="breathable" populace="friendly">`, но не `<planet populace="friendly">`.

`#mars`

Этот специальный формат применяется для поиска атрибутов `id`. Он соответствует `<planet id="mars">` или `<candy-bar id="mars">`, но не `<planet id="venus">`. Помните, что только один элемент во всем документе может иметь атрибут `id` с заданным значением, поэтому данное правило находит соответствия не слишком часто.

`planet.uninhabited`

Атрибутом, который часто применяется для обозначения особых категорий элементов для таблиц стилей, является `class`. Можно использовать сокращенную запись для поиска элементов с атрибутом класса, добавляя точку (.) и значение атрибута. Приведенный пример находит `<planet class="uninhabited">`, но не `<planet class="colony">`.

`planet:lang(en)`

Такая форма селектора используется, чтобы искать элементы, для которых задан некоторый конкретный язык. В версиях HTML, существовавших до XML, язык задавался в атрибуте `lang`. В XML таким атрибутом является `xml:lang`. Поиск значений атрибута выполняется так же, как в операторе «|=»: список с разделителем-дефисом соответствует значению, заданному в селекторе, если список начинается со строки, совпадающей с заданной в селекторе. Атрибут `xml:lang` представляет исключение из обычных правил XML, касающихся чувствительности к регистру: здесь значения сравниваются без учета регистра. Поэтому в данном примере селектор соответствует `<planet lang="en">`, `<planet xml:lang="EN-us">` или `<planet lang="en-US">`, но не `<planet xml:lang="jp">`.

Отбор по контексту

Для поиска элементов селекторы могут также использовать информацию о контексте, включающую в себя сведения о предках элемента (его родителе, родителе родителя и т. д.) и элементах одного уровня с ним, и полезна в тех случаях, когда элемент должен быть выведен различным образом в зависимости от того, в каком окружении он обнаружен.

Предки

Указать, что элемент является дочерним для другого элемента, можно с помощью символа «больше» (>). Например:

```
book > title { font-size: 24pt; }
chapter > title { font-size: 20pt; }
title { font-size: 18pt; }
```

Здесь элементом, который следует выбрать, является `<title>`. Если `<title>` находится в `<book>`, то применяется первое правило. Если он находится в `<chapter>`, выбирается второе правило. Если заголовок появляется в каком-то еще месте, используется последнее правило.

Оператор > действует, только если два элемента разделяет лишь один уровень. Чтобы добраться до элемента, расположенного на произвольной глубине внутри другого элемента, перечислите их в селекторе, разделив пробелами. Например:

```
table para { color: green }
para { color: black }
```

Первое правило соответствует элементу <para>, находящемуся в любом месте внутри <table>, например:

```
<table>
 <title>A plain ol' table</title>
 <tgroup>
   <tbody>
     <row>
       <entry>
         <para>Hi! I'm a table cell paragraph.</para>
    ...
```

Число элементов, которые можно расположить подряд, не ограничено. Это удобно, когда требуется возвращаться по дереву предков, чтобы собрать информацию. Допустим, что надо разместить один список внутри другого, возможно, для создания структуры. Обычно внутренний (вложенный) список имеет больший отступ, чем внешний. Следующие правила обеспечат создание до трех уровней вложенных списков:

```
list { indent: 3em }
list > list { indent: 6em }
list > list > list { indent: 9em }
```

Звездочка (которую называют также универсальным селектором) служит особым символом подстановки, который может заменять любой элемент. Ее можно использовать в любом месте иерархии. Пусть, например, имеется такое содержимое:

```
<chapter><title>Classification of Bosses</title>
  <sect1><title>Meddling Types</title>
    <sect2><title>Micromanagers</title>
       ...
```

Последние два элемента <title> можно найти с помощью такого селектора:

```
chapter * title
```

Первый элемент <title> не будет отобран, поскольку универсальный селектор требует нахождения хотя бы одного элемента между <chapter> и <title>.

Позиция

Часто требуется знать, в каком месте в последовательности элементов одного уровня находится некоторый элемент. Например, надо обработать первый абзац главы не так, как все остальные, скажем, выведя

его полностью заглавными буквами. Для этого добавьте к селектору элемента специальный суффикс, как показано здесь:

```
para:first-child { text-transform: uppercase; }
```

`para:first-child` соответствует только `<para>`, являющемуся первым дочерним элементом для некоторого элемента. Двоеточие (:), за которым следует ключевое слово типа `first-child`, называется в CSS *псевдоклассом* (*pseudo-class*). Существует несколько других псевдоклассов, применяемых для модификации селекторов, о которых мы вскоре расскажем.

Еще одним способом изучения контекста элемента является рассмотрение элементов одного с ним уровня. *Селектор элементов одного уровня* (*sibling selector*) находит элемент, непосредственно следующий за другим. Например,

```
title + para { text-indent: 0 }
```

находит каждый элемент `<para>`, следующий за `<title>`, и выключает для него начальный отступ. Это относится только к элементам, находящимся рядом друг с другом; между ними может находиться текст, но не другие элементы.

Можно выбирать части содержимого элементов с *помощью селекторов псевдоэлементов* (*pseudo-element selectors*). `:first-line` относится к первой строке элемента в том виде, в каком он появляется в броузере (он может быть различным, т. к. длина строки зависит от непредсказуемых факторов, скажем, размера окна). С помощью этого селектора можно, например, вывести заглавными буквами первую строку абзаца, достигнув этим хорошего стилистического эффекта для начала статьи. Следующее правило преобразует первую строку первого элемента `<para>` из `<chapter>` в заглавные буквы:

```
chapter > para:first-child:first-line {
  text-transform: uppercase }
```

Аналогично, `:first-letter` действует исключительно на первую букву содержимого элемента, а также знаки пунктуации, предшествующие букве в элементе. Это полезно для создания буквиц и украшенных прописных букв:

```
body > p:first-child:first-letter {
  font-size: 300%;
  font-color: red }
```

С помощью псевдоклассов `:before` и `:after` можно выбирать точки непосредственно перед элементом или после него, соответственно. Это особенно ценно при создании генерируемого текста: символьных данных, отсутствующих в документе XML. Следующий пример иллюстрируется на рис. 4.8:

```
warning > *:first-child:before {
  content: "WARNING!";
  font-weight: bold;
  color: red }
```

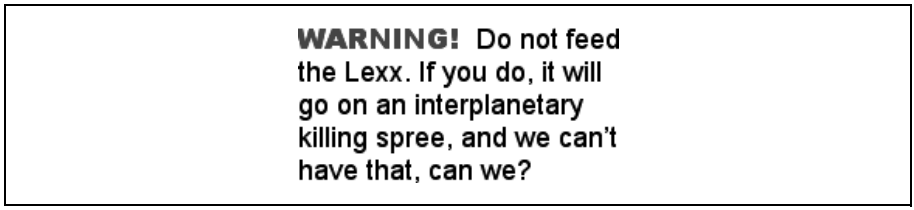


***Рис. 4.8.** Автогенерируемый текст в предостережении*

Разрешение конфликтов между правилами

Как говорилось выше, если одному элементу соответствуют несколько правил, к нему применяются все свойства из этих правил. Если обнаруживается противоречие между установками свойств, оно разрешается путем сравнения селекторов правил и нахождения «наилучшего» соответствия. Рассмотрим такую таблицу стилей:

```
* {font-family: "ITC Garamond"}
h1 { font-size: 24pt }
h2 { font-size: 18pt }
h1, h2 { color: blue }
```

Элемент `<h1>` соответствует трем из этих правил. Окончательный результат состоит в выводе его шрифтом ITC Garamond кеглем 24 пункта голубым цветом.

Что если возникает конфликт между двумя или более значениям одного и того же свойства? Например, в этой таблице стилей могло быть еще одно правило:

```
h1:first-child {
  color: red
}
```

У элемента `<h1>`, являющегося первым дочерним для своего родительского элемента, образовались бы противоречивые значения для свойства `color`.

В CSS есть четкие правила разрешения таких конфликтов. Основной принцип заключается в том, что более общие селекторы переопределяются более конкретными селекторами. В следующем списке очерчен процесс принятия решения:

1. Селекторы с идентификаторами являются самыми конкретными. Если в одном правиле есть селектор с идентификатором, а в другом

нет, то верх одерживает селектор с идентификатором. Большее число идентификаторов сильнее, чем меньшее, хотя, учитывая уникальность идентификатора в документе, больше одного, в сущности, задавать не требуется.

2. Большее число селекторов атрибутов и псевдоклассов преобладает над меньшим. Это означает, что `para:first-child` более конкретен, чем `title+para`, а `*[role="powerful"][class="mighty"]` переопределяет `para:first-child`.
3. Генеалогически более точное описание одерживает верх над менее конкретной цепочкой, поэтому `chapter > para` имеет приоритет над `para`, но уступает `title + para`. Псевдоэлементы при этом не учитываются.
4. Если в результате победитель среди селекторов не выявлен, срабатывает следующее правило разрешения конфликтов: добивается успеха тот, кто появляется в таблице стилей последним.

Конфликты между значениями свойств разрешаются поочередно для каждого свойства. Одно правило может быть более конкретным, чем другое, но устанавливать только одно свойство; остальные свойства могут устанавливаться менее конкретными правилами, даже если одно из свойств правила переопределено. Поэтому в нашем предшествующем примере первый элемент `<h1>` получает красный, а не голубой цвет.

Свойства

Если правило соответствует элементу, процессор применяет объявления свойств к его содержимому. Свойства являются атрибутами форматирования, такими как настройки шрифтов и цвета. Их так много – в CSS2 свыше 120, что обо всех мы здесь рассказать не можем. Коснемся лишь базовых понятий и категорий, а более подробное изложение предоставим специализированным книгам по данному предмету.

Наследование свойств

Некоторые свойства элементы наследуют от своих предков. Это значительно облегчает конструирование таблиц стилей, т. к. можно присвоить свойству значение в одном месте, и таким же свойством будут обладать все потомки соответствующего элемента. Например, если использовать в качестве аналогии документ HTML, то можно сказать, что в элементе `<body>` содержатся все элементы содержимого, а следовательно, логично поместить туда все установки по умолчанию. Всем его потомкам предоставляется возможность при необходимости переопределить значения по умолчанию. К примеру, в документе DocBook можно поместить такие настройки в элемент `<book>`.

Механизм наследования свойств показан на рис. 4.9. Элемент <para> наследует некоторые свойства <section>, который, в свою очередь, наследует свойства <article>. Некоторые свойства, например, font-size в <section> и font-family в <para>, переопределяются в каждом потомке.

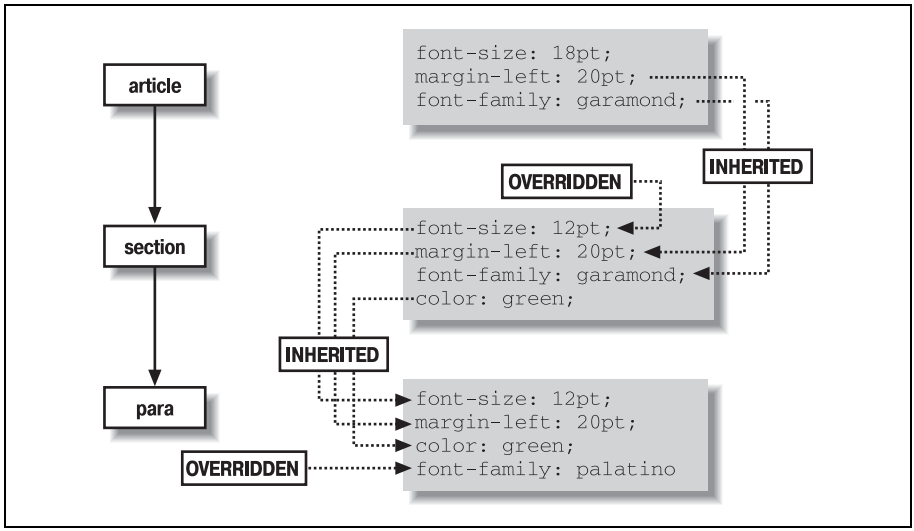


***Рис. 4.9.** Свойства, наследуемые элементами*

Некоторые свойства нельзя наследовать, т. к. это бессмысленно. Например, свойство background-image, управляющее загрузкой графического изображения, которое выступает в качестве фона, не наследуется. Если бы каждый элемент наследовал это свойство, то в результате получилась бы полная неразбериха – ведь каждый абзац и внутритекстовый элемент пытался бы вывести собственный экземпляр графического изображения в своем прямоугольнике вывода. Значительно лучше, когда этим свойством обладает только один элемент, но не его потомки.

Единицы измерения

Во многих свойствах участвуют какие-либо измерения: ширина правила, размер шрифта или величина отступа. Эти размеры могут выражаться в различных единицах измерения. *Абсолютные* измерения используют единицы предопределенной величины, например, дюймы, пункты или цицеро (picas). *Относительные* измерения выражены в процентах или дробных частях некоторых переменных величин, таких как высота текущего шрифта.

Абсолютную величину можно измерять линейкой, потому что единицы измерения остаются неизменными. Миллиметр остается одинаковым для любого шрифта или языка. В число абсолютных единиц, используемых в CSS, входят миллиметры (mm), сантиметры (cm) и дюймы

(in), а также специальные единицы измерения, используемые в полиграфии, такие как пункты (pt) и цицеро (pc).

Относительные единицы не являются фиксированными, масштабируя значения в зависимости от какой-то другой величины. Например, em определяется как размер текущего шрифта. Если этот шрифт имеет размер 12 пунктов, тогда и em составляет 12 пунктов. Если размер текущего шрифта изменяется до 18 пунктов, то же происходит с em. Другой относительной единицей служит ex, определяемая как *x-высота* шрифта (высота строчной буквы «x»). Это дробная часть em, обычно близкая к половине и не имеющая постоянного значения – оно зависит от шрифта. У различных шрифтов x-высота может отличаться, даже если параметр em для них одинаков.

Относительные размеры могут также выражаться в процентах. Такой размер основывается на однотипном свойстве другого элемента. Например,

```
b { font-size: 200% }
```

означает, что размер шрифта для элемента b вдвое больше, чем для его родителя.

В целом, относительные размеры лучше абсолютных. Относительные единицы не приходится изменять после редактирования значений по умолчанию. Значительно легче написать таблицу стилей для нескольких сценариев, определив базовый размер лишь в одном месте, а все остальное задав в относительных единицах.

Свойство display

Большинство элементов попадает в одну из трех категорий форматирования: *блочные* (*block*), *внутритекстовые* (*inline*) или *невидимые* (*invisible*). Перечисленные здесь предназначения управляют отображением содержимого в форматированном документе:

Блочные

Блок представляет собой прямоугольную область текста, отделенную свободным пространством от предшествующего и последующего содержимого. Он начинается с новой строки, часто после каких-то пробельных символов, и имеет границы (называемые полями), которые обеспечивают тексту прямоугольную форму. Текст *переносится* (*wraps*) по полю, что означает прекращение его вывода, и последующее возобновление с новой строки. Блоки могут содержать другие, меньшие, блоки, а также внутритекстовые элементы (inlines). Примерами блоков в традиционных документах служат абзацы, заголовки и разделы.

Внутритекстовые

Внутритекстовым (inline) называется содержимое, которое не прерывает поток текста в блоке. Оно переносится на следующую строку по полю блока, в котором размещается, как обычные символьные данные. Такие элементы могут устанавливать свойства, не влияющие на поток текста, например, `font-family` и `color`, но не могут иметь свойства, относящиеся к блокам, например, поля и заполнение символами. Примером внутритекстовых элементов служат выделение, ключевые слова и гипертекстовые ссылки.

Невидимые

К этой категории относятся элементы, которые не должны включаться в форматированный текст. Процессор CSS пропускает такие элементы. Примером невидимых элементов служат метаданные, элементы предметного указателя и якоря для ссылок.

Другие типы отображаемых объектов, такие как таблицы и некоторые списки, слишком сложны, чтобы обсуждать их здесь.

Каждому элементу назначается свойство `display`, которое сообщает процессору CSS, как форматировать элемент. Если `display` имеет значение `block`, элемент начинается с новой строки. Если установлено значение `inline`, элемент начинается на той строке, где закончились предыдущий элемент или символьные данные. Значение `none` указывает, что элемент невидим.

Это свойство наследуется только в одной ситуации. Если элемент объявлен как `display: none`, все его потомки наследуют это свойство, даже если явно переопределяют эту установку. Благодаря этому можно «выключать» большие участки документа.

В HTML у каждого элемента есть неявное значение свойства `display`. Элемент `<tt>` всегда является внутритекстовым, а `<p>` – блоком. В таблицах стилей для документов HTML редко можно встретить в объявлениях свойство `display`. Для некоторых элементов не нужны правила, т. к. достаточно неявной установки типа отображения.

В отсутствие явной или неявной установки свойства `display` процессору XML общего назначения приходится строить догадки относительно того, что использовать. Он может выбрать `inline`, в результате чего весь текст будет представлять собой один гигантский абзац. Это не слишком красиво, поэтому в приложениях XML может оказаться необходимым объявить свойство `display` для каждого элемента.

Свойства для блочных элементов

У каждого блока есть невидимая прямоугольная рамка, которая служит формой для его содержимого, располагает его на достаточном расстоянии от соседей и определяет фон, на котором производится вывод. На рис. 4.10 показаны все части этой *блочной модели* (*box model*). Содержимое блока непосредственно окружено прямоугольным пространством, которое называют *ограничивающим прямоугольником* (*bounding box*). Расстояние его от содержимого называют *заполнением* (*padding*). Периметр этой области, называемый *границей* (*boundary*), иногда выводится как линия или кайма с одной, двух, трех или всех четырех сторон. Толщина этих линий измеряется в наружную сторону. Снаружи границы находится еще один участок пространства, определяемый четырьмя размерами, называемыми *полями* (*margins*).

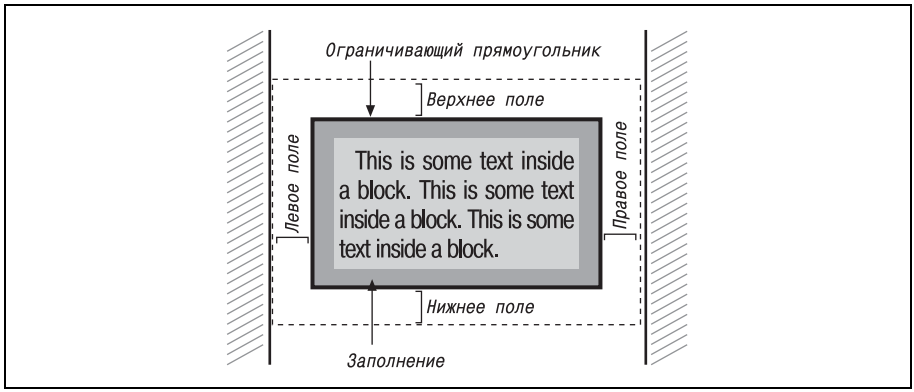


Рис. 4.10. *Блочная модель в CSS*

Границы блока хранят его содержимое, отделяя его от одноуровневых блоков. Изображение или цвет, используемые в качестве фона, заключаются только в область внутри границ, а текст и вложенные блоки заключены в еще меньшую область – прямоугольник, определяемый границей за вычетом заполнения (padding). Если граница не отображается, заполнение часто устанавливается нулевым. Поля разграничивают прямоугольную область вне блока, благодаря которой другие блоки располагаются на известном расстоянии.

Поля

Для организации свободного пространства вокруг элемента часто применяется установка его полей. Поле является расстоянием между ограничивающими прямоугольниками элемента и его соседних (или содержащих) элементов. У каждой из сторон прямоугольника есть собственное свойство поля: `margin-left`, `margin-right`, `margin-top` и `margin-bottom`. Значением может быть абсолютная величина или процент от ширины родительского элемента.

Свойство margin служит сокращением для определения всех четырех значений полей. Его значение является разделенным пробелами списком, содержащим от одного до четырех размеров или процентных отношений. Если задано только одно значение, то оно применяется ко всем четырем сторонам. Если значений два, то первое из них устанавливает верхнее и нижнее поля, а второе – левое и правое поля. Если значений три, то первое устанавливает верхнее поле, второе устанавливает левое и правое поля, а третье – нижнее поле. Наконец, если заданы все четыре значения, то они соответствуют верхнему, правому, нижнему и левому полям, соответственно. Незаданные поля по умолчанию принимаются равными нулю. Следовательно, следующие правила эквивалентны:

```
para { margin-left: 10em; margin-right: 10em; margin-top: 5% }
para { margin: 5% 10em 0 }
```

В обоих случаях элемент <para> определен с левым и правым полями, равными 10 em, и верхним полем, составляющим 5% от размера содержащего его элемента в вертикальном направлении, то есть высоты, и без нижнего поля. Допускаются отрицательные величины. Отрицательное значение поля выводит прямоугольник границы на указанное расстояние за пределы прямоугольника родительского элемента. Для создания эффекта текста, выходящего за пределы окрашенной области, используем с файлом HTML такую таблицу стилей:

```
body { background-color: silver }
p { right-margin: -15% }
```

Часто поля перекрываются. Например, расположенные в колонку абзацы соприкасаются верхними и нижними сторонами. Для того чтобы создать больший промежуток, надо не добавлять поля, а отбросить меньшее поле и на его место вставить большее. Это называется *свертыванием* (*collapsing*) полей. Так, если верхнее поле абзаца установлено равным 24 пунктам, а нижнее – 10 пунктам, то фактическое расстояние между абзацами составит 24 пункта. В действительности, правила свертывания полей несколько сложнее, но полное их изложение не входит в цели этой главы.

Границы

Часто привлекательно выглядят элементы, окруженные прямоугольной рамкой, которая делает их более заметными. Пример предостережения, который приводился выше, сильнее привлекал бы внимание читателей, будучи заключенным в рамку. С помощью свойства border можно создавать различные эффекты – от обрамления блока пунктиром до вывода черты с любой из сторон.

Чтобы определить желаемую границу, нужно задать три параметра. Параметр ширины `width` может выражаться в абсолютном или относительном виде, либо как одно из трех предустановленных значений:

```
thin (тонкая)
medium (средняя, по умолчанию)
thick (толстая)
```

Следующий параметр – `style`. В CSS 2 определено восемь стилей:

```
solid (сплошной)
dashed (штрих-пунктир)
dotted (пунктир)
groove (бороздка)
ridge (гребень)
double (двойная)
inset (врезка)
outset (боковик)
```

Последним параметром является цвет – `color`. Все параметры помещаются в разделяемый пробелами список в любом порядке. Вот несколько примеров:

```
border: thin solid green;
border: red groove thick;
border: inset blue 12pt;
```

Заполнение

Но одной вещи не хватает. Вспомните, что эта граница находится внутри полей, непосредственно рядом с текстом блока. Поэтому нужно некоторое дополнительное пространство внутри границы, чтобы она не сливалась с текстом. Свойство `padding` позволяет очертить границу изнутри свободным пространством, чтобы сжать текст и не допустить его слияния с границей. Значение этого свойства представляет собой разделяемый пробелами список, содержащий от одного до четырех размеров. Размеры применяются к сторонам так же, как в свойстве `margin`.

Расширим наш предыдущий пример `warning`, создав для него рамку, как показано на рис. 4.11:

```
warning {
  display: block;
  border: thick solid gray;
}
warning:before {
  content: "WARNING!";
  font-weight: bold;
  color: red }
warning
```

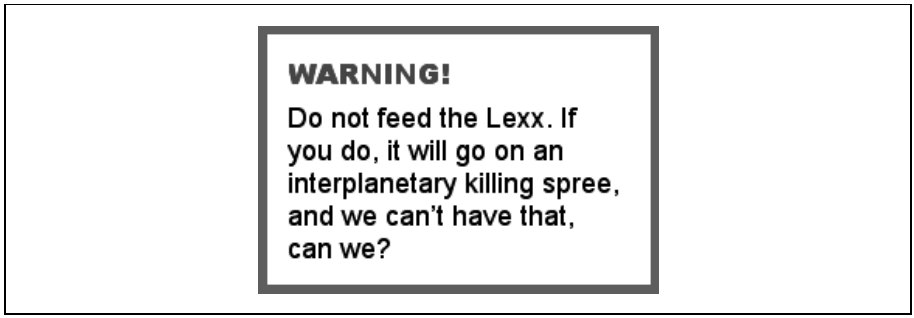


Рис. 4.11. *Предостережение, заключенное в рамочку*

Выравнивание и отступы

Свойство `text-align` определяет выравнивание (alignment) или выключку (justification) строк абзаца. Иногда требуется иметь ясно очерченную границу слева и справа от колонки. В других случаях желателен рваный по левому или правому краю текст, либо выровненный по центру. Поддерживаются следующие значения:

`left`

Выравнивание текста по левой границе (рваный правый край).

`right`

Выравнивание текста по правой границе (рваный левый край).

`center`

Выровнять каждую строку блока по центру (рваный левый и правый края).

`justify`

Расширить каждую строку до левой и правой границ блока.

Большинство процессоров CSS по умолчанию использует выравнивание по левому краю. Обратите внимание, что `left` и `right` являются абсолютными значениями и не зависят от направления вывода текста.

Свойство `text-indent` определяет отступ первой строки блока. Можно задать как положительный, так и отрицательный абсолютный размер либо процентное отношение к ширине блока.

Свойства текста

До сих пор мы говорили о блоках как контейнерах для текста. Теперь займемся самим текстом. В этом разделе перечислены некоторые свойства, управляющие тем, как выглядят и ведут себя символьные данные, например, типами шрифтов, стилями шрифтов и цветом.

Семейство шрифтов

Гарнитура имеет несколько параметров, определяющих ее внешний вид, такие как кегль, вес и стиль. Однако наиболее важным является семейство шрифтов (например, Courier, Helvetica, Times). В каждом семействе могут быть различные стили и веса, например, курсив (italic), полужирный (bold), жирный (heavy).

Свойство `font-family` объявляется как разделяемый запятыми список предпочтений шрифтов, начиная с наиболее конкретного и желательного и завершая самым общим. Этот список задает ряд альтернатив на случай, когда у агента пользователя нет доступа к конкретному требуемому шрифту. В самом конце списка находится общий класс шрифтов, что фактически позволяет программному обеспечению пользователя решать, какой шрифт лучше всего подходит. Вот некоторые общие классы шрифтов:

`serif`

В эту категорию попадают шрифты с декоративными дополнениями, или засечками. Часто используются такие шрифты с засечками, как Palatino, Times и Garamond.

`sans-serif`

Эти шрифты, у которых отсутствуют засечки, относительно просты в сравнении с шрифтами `serif`. В эту группу входят Helvetica и Avant-Garde.

`monospace`

Это шрифты, в которых каждый символ занимает одинаковое пространство, в отличие от большинства шрифтов, в которых составляемые вместе буквы имеют разную ширину. Такой тип шрифта обычно используется для вывода компьютерных программ, моделирования телетайпа и псевдографики.[1] Примерами моноширинных шрифтов служат Courier и Monaco.

`cursive`

В эту группу попадают шрифты, имитирующие каллиграфические или рукописные. Такие гарнитуры часто применяются в свадебных приглашениях и дипломах. Поскольку для большинства машин эта категория шрифтов не является стандартной, следует использовать ее реже, если вообще не избегать.

`fantasy`

Сюда входят оригинальные шрифты типа Comic Strip, Ransom Note и Wild West. Опять-таки, у большинства пользователей таких шрифтов нет, поэтому используйте их с осторожностью.

---

[1] См. *http://www.textfiles.com/art/*.

Примеры этих шрифтов приведены на рис. 4.12.

***Рис. 4.12.** Общие семейства шрифтов*

Допустим, вы хотите выбрать гарнитуру шрифта Palatino. В целом считается, что Zapf Humanist 521 является высококачественной версией Palatino. Клон Monotype в некоторых системах Windows носит название Book Antiqua и не так тщательно проработан, но довольно крепкий. Есть различные дешевые подделки, называемые по-разному – Palisades или Palestine. Если невозможно найти ни Palatino, ни его сородичей, удобной заменой послужит Times New Roman (у него не много общего с Palatino, кроме того, что это тоже шрифт с засечками, но сходство, по крайней мере, большее, чем у Helvetica). Times New Roman иногда встречается под названиями Times Roman, TmsRmn или Times.

Теперь нужно решить, как упорядочить список. Существует компромисс между включением большего числа клонов Palatino с сомнительным качеством и сокращением списка в пользу сильно отличающихся, но более высококачественных альтернативных гарнитур. Можно предложить следующие три подхода:

```
font-family: Palatino, "BT Humanist 521", "Book Antiqua", Palisades,
    "Times New Roman", "Times Roman", Times, serif;
font-family: Palatino, "Times New Roman", serif;
font-family: "BT Humanist 521", Palatino, serif;
```

Заметьте, что названия шрифтов, содержащие пробелы, должны заключаться в кавычки. Использование заглавных букв не обязательно.

В первом варианте предлагается больше всего альтернатив, но есть опасность, что если будет выбран Palisades, то пострадает качество документа. Второй список значительно короче и гласит, что, если нет Palatino, надо использовать Times New Roman. Третий амбициозно требует BT Humanist 521, но удовлетворится и обычным Palatino. Во всех списках в качестве последнего прибежища используется семейство `serif`, позволяя системе выбрать шрифт с засечками, если других не найдется.

Кегль шрифта

Кегль шрифта определяется свойством `font-size`. Значение может быть указано в абсолютных единицах (обычно пунктах) или относительных (процентах или em родительского шрифта). Можно также использовать полуабсолютные ключевые слова:

```
xx-small (очень маленький)
x-small (меньше)
small (маленький)
medium (средний)
large (большой)
x-large (больше)
xx-large (очень большой)
```

В спецификации CSS процессорам каскадных таблиц стилей рекомендовано выводить каждый размер в 1,2 раза крупнее, чем предыдущий (поэтому `xx-large` в 3,6 раза должен превышать кегль `xx-small`), но фактические размеры оставлены на усмотрение пользователя. Это предоставляет удобный способ задавать кегль в соответствии с запросами аудитории, но ценой утраты абсолютной точности. На рис. 4.13 показано, как это может выглядеть.

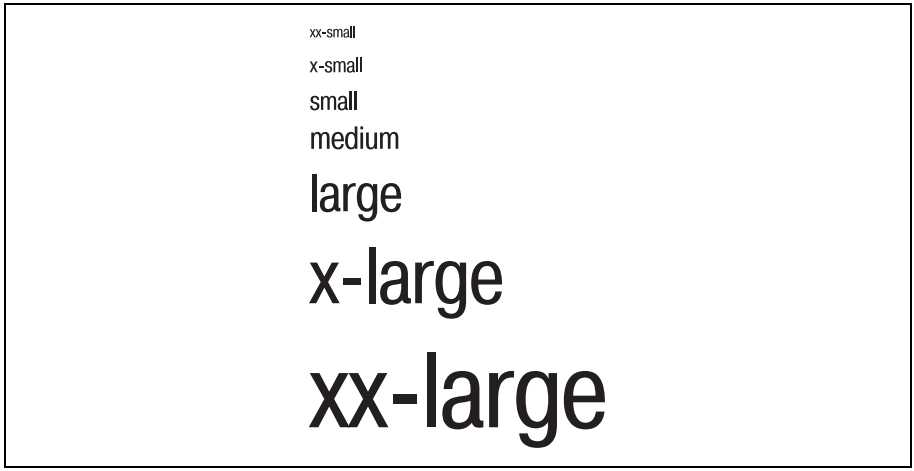


Рис. 4.13. *Кегли шрифтов*

Можно также использовать относительные ключевые слова:

```
Larger (больше)
Smaller (меньше)
```

Они изменяют кегль в большем или меньшем направлении на величину коэффициента.

Настройка высоты строки и кегля шрифта

В то время как `font-size` задает кегль шрифта выводимых символов, свойство `line-height` оказывает воздействие на суммарную высоту шрифта и пустого пространства над ним. Если `line-height` больше, чем `font-size`, то разница между ними создает равное дополнительное свободное пространство вверху и внизу каждой строки. В результате блок выглядит более разреженным или более сжатым.

В следующем примере мы задаем один, два и полтора интервала между строками:

```
para1 { line-height: 1 }
para2 { line-height: 2 }
para3 { line-height: 1.5 }
```

Шрифты одинакового кегля могут выглядеть более или менее крупными в зависимости от нескольких характеристик, в совокупности называемых *очком* (*eye*) шрифта. Главной среди них является высота x, или `ex`. В латинских (читай: европейских) шрифтах `ex` является высотой строчной «x» (в других шрифтах используются иные критерии, но это всегда некоторая часть кегля шрифта, или `em`). В результате одни шрифты оказываются более крупными, чем другие.

Для озабоченного этой проблемой разработчика есть выход. Свойство `font-size-adjust` позволяет до некоторой степени регулировать размеры шрифтов в семействе. Величиной *аспекта* (*aspect value*) называют отношение высоты x к кеглю шрифта. Если задать величину аспекта желаемого шрифта, выраженную десятичной дробью, в качестве значения `font-size-adjust`, то броузер может настроить заменяющие шрифты так, чтобы они имели такой же видимый размер, как требуемый шрифт.

Стиль и вес шрифта

Семейства шрифтов содержат варианты, позволяющие автору дополнительно выделять фрагменты текста. Есть два типа версий: стиль шрифта (курсив, наклонный) и вес шрифта – светлый (light), полужирный (bold).

Свойство `font-style` может принимать четыре значения:

`normal`

Обычная вертикальная версия шрифта.

`italic`

Курсивная версия шрифта, если имеется, или наклонная версия.

`oblique`

Наклонная версия, если имеется, или обычный шрифт с небольшим наклоном.

`inherit`

Установка, взятая у родительского элемента.

В спецификации CSS не указано ясно, как должен действовать курсив в языках, в которых не принято его использовать, поэтому для других языков, возможно, лучше задать иную гарнитуру (typeface):

```
em { font-style: italic }
em:lang(ja) { font-family: ...;
              font-style: normal }
```

Свойство `font-weight` управляет степенью жирности шрифта. Определены следующие ключевые слова:

`light`

Легкая, «воздушная» версия шрифта, если она имеется. Если нет, то система пользователя может автоматически генерировать ее.

`normal`

Стандартная версия шрифта.

`bold`

Более темная и тяжелая версия шрифта, если она имеется. Если нет, то система пользователя может автоматически генерировать ее.

Есть также относительные ключевые слова, уменьшающие или увеличивающие вес относительно значения свойства родительского элемента:

`lighter`

Уменьшает вес на одну ступень.

`bolder`

Увеличивает вес на одну ступень.

Есть девять ступеней изменения веса, поэтому `lighter` и `bolder` могут осуществлять тонкую настройку веса по 1/9 диапазона. Альтернативой служит применение численного значения в диапазоне от 100 до 900 (с шагом 100). Не для каждого шрифта имеется девять весов, поэтому изменение числа на 100 может не оказать видимого эффекта. Значение 400 соответствует `normal`, а `bold` установлен равным 700. На рис. 4.14 показаны шрифты различных стилей и весов.

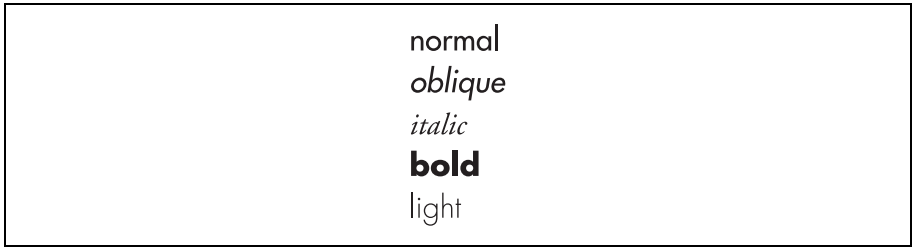


***Рис. 4.14.** Стили шрифтов и веса*

Цвет

Цвет является важной характеристикой текста, особенно при выводе на экран компьютера. Текст имеет два свойства, связанные с цветом: `color` для переднего плана и `background-color` для фона.

Есть много предопределенных цветов, которые можно использовать по имени. С другой стороны, цвет может быть задан как код из трех чисел, в котором числа соответствуют значениям красной, зеленой и голубой составляющих (RGB). Эти числа могут быть заданы процентами, целыми в диапазоне 0–255 или шестнадцатеричными числами от `#000000` до `#ffffff`. Некоторые примеры представлены в табл. 4.1.

Таблица 4.1. Предопределенные цвета

Предустановленное название	В процентах	Целочисленное представление	Шестнадцатеричное представление
aqua	rgb(0%, 65%, 65%)	rgb(0, 160, 160)	#00a0a0
black	rgb(0%, 0%, 0%)	rgb(0, 0, 0)	#000000
blue	rgb(0%, 32%, 100%)	rgb(0, 80, 255)	#0050ff
fuchsia	rgb(100%, 0%, 65%)	rgb(255, 0, 160)	#ff00a0
gray	rgb(65%, 65%, 65%)	rgb(160, 160, 160)	#a0a0a0
green	rgb(0%, 100%, 0%)	rgb(0, 255, 0)	#00ff00
lime	rgb(0%, 65%, 0%)	rgb(0, 160, 0)	#00a000
maroon	rgb(70%, 0%, 32%)	rgb(176, 0, 80)	#b00050
navy	rgb(0%, 0%, 65%)	rgb(0, 0, 160)	#0000a0
olive	rgb(65%, 65%, 0%)	rgb(160, 160, 0)	#a0a000
purple	rgb(65%, 0%, 65%)	rgb(160, 0, 160)	#a000a0
red	rgb(100%, 0%, 32%)	rgb(255, 0, 80)	#ff0050
silver	rgb(90%, 90%, 90%)	rgb(225, 225, 255)	#d0d0d0
teal	rgb(0%, 65%, 100%)	rgb(0, 160, 255)	#00a0ff
white	rgb(100%, 100%, 100%)	rgb(255, 255, 255)	#ffffff
yellow	rgb(100%, 100%, 0%)	rgb(255, 255, 0)	#ffff00

Генерируемый текст

Автоматическая генерация текста является важной возможностью таблиц стилей. Мы уже видели пример, в котором создавалась панель предостережения с автоматически генерируемым заголовком «WARNING!». Общий формат свойства, генерирующего текст, такой:

```
content: string1 string2 ...
```

Каждая строка является значением, заключенным в кавычки (как `"WARNING!"`), или функцией, создающей текст. Вот некоторые функции, создающие текст:

`url(locator)`

Эта функция открывает файл по URL, заданному параметром *locator*, и вставляет содержимое файла в это место текста. Полезно для вставки стандартных текстов.

`attr(attname)`

Эта функция вставляет значение атрибута с именем *attname*.

`counter(name)`

Эта полезная функция читает значение внутреннего счетчика с меткой *name* и превращает его в текст.

Счетчики

Счетчиками в CSS являются переменные, содержащие числовые значения. Они применяются для нумерации глав, упорядоченных списков и вообще всего, что нужно помечать числами. Чтобы воспользоваться счетчиком, нужно дать ему имя и с помощью свойства `counter-increment` сообщить CSS, когда нужно его инкрементировать. Получить значение счетчика можно в любой момент с помощью функции `counter()`. Например:

```
chapter { counter-increment: chapnum }
chapter > title:before { content: "Chapter " counter(chapnum) ". " }
```

Здесь мы создаем счетчик с именем `chapnum` и увеличиваем его значение всякий раз, когда процессор CSS обнаруживает новую главу `<chapter>`. Элемент `<title>` выводится с добавлением перед ним этого числа, например:

```
Chapter 3. Sonic the Hedgehog
```

Другим свойством, воздействующим на счетчики, является `counter-reset`. Оно снова устанавливает значение счетчика в ноль, когда элемент обработан. Это можно использовать, например, в нумерованных списках, когда требуется начинать нумерацию каждого списка с `1`, а не вести ее через весь документ:

```
numberedlist { counter-reset: list_item_number; }
listitem { counter-increment: list_item_number; }
listitem:before { content: counter(list_item_number) ". "; }
```

Теперь для каждого списка нумерация будет начинаться с 1:

```
First list:
  1. Alpha
  2. Bravo
  3. Charlie
  4. Delta
Second list:
  1. Fee
  2. Fi
  3. Fo
  4. Fum
```

Практический пример

Теперь попробуем извлечь пользу из того, что мы изучили к данному моменту. Создадим таблицу стилей для документа XHTML из примера 3.2.

Хотя у элементов XHTML есть стили, заданные неявно, часто требуется расширить их или заменить, чтобы получить более богатый внешний вид. Броузеры, умеющие работать с таблицами стилей, позволяют нам добиться этого с помощью CSS.

XHTML позволяет помещать информацию таблиц стилей внутрь элемента <head> файла XHTML с помощью элемента <style>. Однако включение информации о стилях в файлы документов весьма предосудительно, т. к. при этом возникают многочисленные проблемы, если требуется расширить документ или изменить его назначение. Вместо этого мы объявим таблицу стилей в отдельном файле с помощью элемента XHTML <link>.

Прежде чем создавать таблицу стилей, изменим немного XHTML-файл, чтобы можно было полностью использовать возможности CSS. Во-первых, добавим несколько элементов , которые будут действовать как контейнеры разделов. HTML страдает отсутствием блочных контейнеров, поэтому в спецификацию были добавлены элементы разделов общего назначения . Однако поскольку они очень общие, нам придется указывать в них атрибут class, чтобы дифференцировать их роли. Поэтому добавим атрибуты class и к другим элементам, увеличивая число объектов, которым можно присваивать уникальные стили.

Пример 4.1 содержит новый файл XHTML.

Пример 4.1. Пересмотренный документ XHTML

```
<?xml version="1.0"?>
<!DOCTYPE html
    PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">

  <head>
    <title>Evil Science Institute</title>
    <link rel="stylesheet" href="style1.css"/>
  </head>

  <body>
    <h1>The Evil Science Institute</h1>
    <p class="bigp"><em>Welcome</em> to Dr. Indigo Riceway's <span
      class="inst">Institute for Evil Science!</span></p>

    <div class="toc">
      <h2>Table of Contents</h2>
      <ol>
        <li><a href="#staff">Meet Our Staff</a></li>
        <li><a href="#courses">Exciting Courses</a></li>
        <li><a href="#research">Groundbreaking Research</a></li>
        <li><a href="#contact">Contact Us</a></li>
      </ol>
    </div>

    <a name="staff" />
    <div class="section" id="staff">
      <h2>Meet Our Staff</h2>
      <dl>
        <dt><span class="person"><a href="riceway.html">Dr. Indigo
          Riceway</a></span></dt>
        <dd>
          <img src="images/riceway.gif" width="60" height="80" />
          Founder of the <span class="inst">Institute</span>, inventor
          of the moon magnet and the metal-eating termite, three-time
          winner of Most Evil Genius award. Teaches Death Rays 101,
          Physics, Astronomy, and Criminal Schemes.
        </dd>
        <dt><span class="person"><a href="grzinsky.html">Dr. Ruth
          "Ruthless" Grzinsky</a></span></dt>
        <dd>
          <img src="images/grzinsky.gif" width="60" height="80" />
          Mastermind of the Fort Knox nano-robot heist of 2002.
          Teaches Computer Science, Nanotechnology, and Foiling Security
          Systems.
        </dd>
        <dt><span class="person"><a href="zucav.html">Dr. Sebastian
```

Пример 4.1. Пересмотренный документ XHTML (продолжение)

```
        Zucav</a></span></dt>
      <dd>
        <img src="images/zucav.gif" width="60" height="80" />
        A man of supreme mystery and devastating intellect.
        Teaches Chemistry, Poisons, Explosives, Gambling, and Economics
        of Extortion.
      </dd>
    </dl>
  </div>

  <a name="courses" />
  <div class="section" id="courses">
    <h2>Exciting Courses</h2>
    <p>Choose from such intriguing subjects as...</p>
    <ul>
      <li>Training Cobras to Kill</li>
      <li>Care and Feeding of Mutant Beasts</li>
      <li>Superheros and Their Weaknesses</li>
      <li>The Wonderful World of Money</li>
      <li>Hijacking: From Studebakers to Supertankers</li>
    </ul>
  </div>

  <a name="research" />
  <div class="section" id="research">
    <h2>Groundbreaking Research</h2>
    <p>Indigo's <span class="inst">Evil Institute</span> is a
      world-class research facility. Ongoing projects include:</p>

    <h3 class="projectname">Blot Out The Sky</h3>
    <p class="projdesc">A diabolical scheme to fill the sky with
      garish neon advertisements unless the governments of the world
      agree to pay us one hundred billion dollars. <span
      class="laughter">Mha ha ha ha ha!</span></p>

    <h3 class="projectname">Killer Pigeons</h3>
    <p class="projdesc">A merciless plan to mutate and train pigeons
      to become efficient assassins, whereby we can command <em>huge
      bounties</em> by blackmailing the public not to set them
      loose. <span class="laughter">Mha ha ha ha ha!</span></p>

     <h3 class="projectname">Horror From Below</h3>
     <p class="projdesc">A sinister plot so horrendous and
       terrifying, we dare not reveal it to any but 3rd year
       students and above. We shall only say that it will be the
       most evil of our projects to date!  <span
       class="laughter">Mha ha ha ha ha!</span></p>
   </div>

   <a name="contact" />
```

Пример 4.1. Пересмотренный документ XHTML (продолжение)

```
    <div class="section" id="contact">
      <h2>Contact Us</h2>
      <p>If you think you have what it takes to be an Evil
        Scientist, including unbounded intellect, inhumane cruelty
        and a sincere loathing of your fellow man, contact us for an
        application. Send a self-addressed, stamped envelope to:
      </p>
      <address>The Evil Science Institute,
Office of Admissions,
10 Clover Lane,
Death Island,
Mine Infested Waters off the Coast of Sri Lanka</address>
    </div>

  </body>
</html>
```

В примере 4.2 приведена таблица стилей.

Пример 4.2. Таблица стилей для документа XHTML

```
/*
------------- SECTIONS -------------
*/ (1)

body { (2)
  color: black;
  background-color: silver;
}

.section { (3)
  margin: .5em;
  padding: .5em;
  border: thick solid gray;
  background-color: white;
}

.toc {
  margin: .5em;
  padding: .5em;
  border: thick solid gray;
  background-color: white;
}

/*
------------- HEADS -------------
*/

body > h1 { (4)
```

Пример 4.2. Таблица стилей для документа XHTML (продолжение)

```
  color: black;
  font-size: 3em;
  font-family: sans-serif;
}

.section > h2 {
  color: green;
  font-size: 2em;
}

.toc > h2 {
  color: blue;
  font-size: 2em;
}

.projectname {
  color: navy;
  font-size: 1.5em;
  font-style: italic;
  font-family: sans-serif;
}

/*
------------- BLOCKS -------------
*/

p {
  font-size: 1.2em;
}

p.bigp {
  font-size: 1.5em;
}

p.projdesc {
  margin: .25em;
}

address {
  color: black;
  white-space: normal;
  font-size: 2em;
  font-family: monospace;
}

div li { ❺
  font-size: 1.4em;
  font-family: sans-serif;
}
```

Пример 4.2. Таблица стилей для документа XHTML (продолжение)

```
dd {
  color: black;
  font-size: 1.2em;
  margin: 1em;
}

/*
------------- INLINES -------------
*/

em {
  font-style: italic;
  font-weight: bold;
}

.laughter {
  color: purple;
  font-style: italic;
  font-weight: bold;
}

.person {
  font-weight: bold;
}

.inst {
  font-style: italic;
}
```

1. Комментарии могут облегчить чтение таблиц стилей.
2. Элемент `<body>` служит хорошим местом для размещения объявлений, общих для всего документа.
3. Селектор `.section` соответствует любому элементу с атрибутом `class="section"`. Здесь мы его используем только с элементами `<div>`, поэтому имя элемента можно не указывать.
4. Обратите внимание на иерархический селектор контекста, использующий символ «больше» (>). Названия и заголовки часто форматируются различным образом в зависимости от места, в котором находятся.
5. Это еще один контекстный селектор, в котором два разделенные пробелом элемента должны находиться в одной цепочке наследования.

5

- *Моделирование документов*
- *Синтаксис DTD*
- *Пример: чековая книжка*
- *Советы по проектированию и настройке DTD*
- *Пример: Barebones DocBook*
- *XML Schema: альтернатива использованию DTD*

Модели документов: более высокий уровень контроля

Одним из наиболее сильных свойств XML является возможность создавать собственные языки разметки, в которых определяются элементы и атрибуты, наилучшим образом соответствующие инкапсулируемой информации, и снимаются ограничения, вызываемые малопригодным языком общего назначения. Однако пока нельзя определить язык формальным образом, ограничить словарь элементов и атрибутов поддающимся управлению множеством и управлять грамматикой элементов. Процесс формального определения языка в XML называется *моделированием документов*. В настоящей главе мы обсудим два способа моделирования документов: определения типа документа (DTD), которые описывают структуру документа с помощью декларативных правил, и XML Schema, описывающую структуру документа на примере с помощью шаблонов элементов.

Моделирование документов

Что имеется в виду под «моделью документа»? Модель определяет документы, которые можно создать с помощью языка; или, в рамках терминологии XML, модель документа устанавливает, какие документы *согласуются* (*conform*) с языком. Модель документа отвечает на такие вопросы, как «Может ли быть заголовок у данного элемента?» или «Должна ли быть указана цена для этого элемента?»

Эту мысль иллюстрирует рис. 5.1. В его верхней части находится модель документа (DTD или Schema – в данный момент нам это безразлично). Модель является документом особого рода, написанным по

правилам синтаксиса, предназначенного для описания языков XML, и явно описывает грамматику и словарь отдельного языка разметки. Иногда язык, который она описывает, называют *типом документа* (*document type*) или *приложением XML* (*XML application*). С помощью такой модели можно определить, согласуется ли некоторый документ XML с данным типом документа.

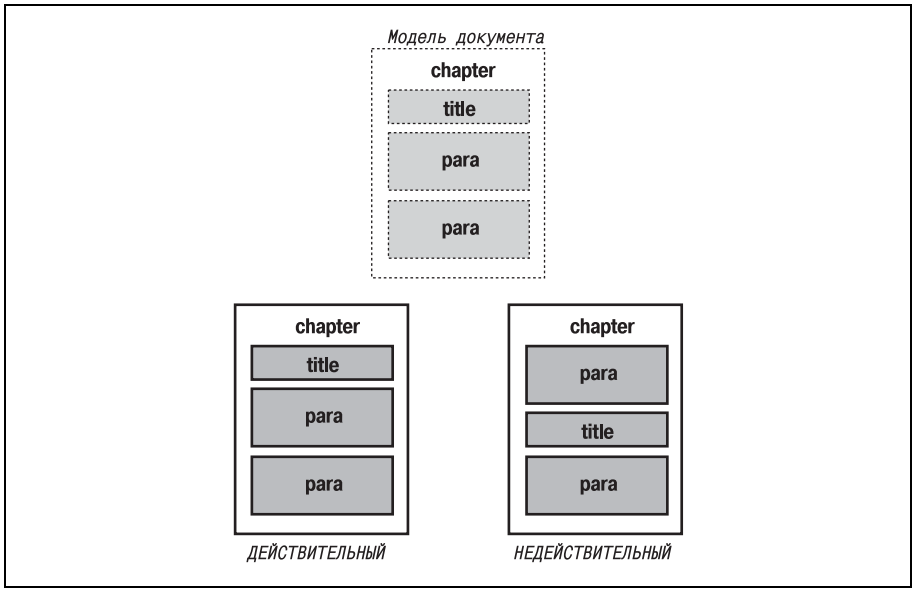


Рис. 5.1. *Сравнение документов с моделью*

Помимо модели документа на рис. 5.1 показаны два документа. Это фактически написанные кем-то документы, которые мы называем *экземплярами документа* (*document instances*), чтобы отличать от абстракции модели документа. Левый документ согласуется с языком, описанным в модели документа, а правый – нет. Согласующиеся документы называют *действительными* (*valid*) в контексте языка; другие документы называют *недействительными* (*invalid*).

А нужна ли вам модель документа?

Теперь, мы знаем, что такое модель документа. Как же решить, нужна ли она нам? Модель документа может быть лишним грузом, если надо сопровождать лишь один-два документа, но если документов много, а требования к качеству высоки, ее создание может окупиться. Вот некоторые ситуации, в которых модель документа в состоянии облегчить жизнь:

- Документы создаются людьми и являются данными для компьютерной программы. Программы особенно привередливы в отноше-

нии форматов данных, потому что трудно создавать программы, способные справляться с отклонениями от формата. Ограничив применяемый шаблон предсказуемым форматом, намного легче писать программы, а вероятность ошибок уменьшается. Сравнение каждого экземпляра документа с моделью гарантирует, что вы не столкнетесь с проблемой несоответствия.

- В документе обязательно должны быть поля. Например, в бланке заказа изделия необходимо указать почтовый адрес, чтобы знать, куда отправлять посылку. Применение модели документа обеспечивает присутствие всех необходимых полей.
- Вы запрашиваете документы у людей, не знакомых с используемым приложением XML. Так как модель сама является документом, она может быть открытым ресурсом, доступным для загрузки, ссылок и передачи. Модель документа может выступать в качестве данных в средах создания структурированных документов, например, в редакторе XML. В такой программе редактор может автоматически вставлять необходимые поля и предлагать разработчику документа списки допустимых групп элементов.
- Разработчику нужна надежная структура для развивающегося языка или семейства языков. Модель документа предоставляет простой способ создания стандарта, такого, например, как HTML Version 4.0. Отслеживание новых версий языка жизненно важно для программ XML, поскольку старые программы могут оказаться несовместимыми с более новыми версиями языка. Модели документов можно объединять для создания составных языков. Например, DocBook использует модель таблиц CALS, а не пытается определить свою.

Конечно, могут быть основания и не использовать модель документов. Сопровождение модели может оказаться неудобным, особенно в начале, когда язык подвергается тестированию и дальнейшей разработке. Она может замедлить обработку, например, если броузеры XML должны загружать модель документа из сети. Наконец, наличие авторитарной модели, указывающей, какие элементы можно использовать, а какие – нет, может просто ломать ваш стиль работы. А, кроме того, нужно потратить силы на то, чтобы разработать модель или найти готовую, отвечающую вашим потребностям.

В конечном счете, автор сам решает, использовать модель документа или нет: XML спроектирован так, что позволяет работать в любом случае. Вероятно, лучше сначала познакомиться поближе с его возможностями и требованиями, а потом решить для себя, стоит ли игра свеч.

Некоторые модели документов (а именно, DTD) не очень хорошо работают с пространствами имен. Вспомните из главы 2 «Разметка и основные понятия», что пространства имен являются способом группировки элементов из различных источников, например, встраивания уравнений MathML внутрь документов HTML. Это создает проблемы, если DTD стремятся ограничить применяемые автором элементы предсказуемым конечным множеством. В настоящий момент исчерпывающего решения этой дилеммы нет. Невозможно предвидеть все виды пространств имен и объявить их элементы и атрибуты внутри своего DTD – их может быть бесконечное число.

Консорциум W3C знает об этой проблеме и работает над ее разрешением. DTD использовались задолго до того, как на сцене появились пространства имен, поэтому маловероятно существенное изменение синтаксиса DTD. А если он все-таки изменится, то в сторону такого расширения, которое будет допускать исключения, не нарушая работы старых систем. Это сложная проблема, но ее решение, будем надеяться, не за горами.

Параметр standalone

В главе 2 кратко упоминалось о параметре `standalone`, но не раскрывались подробности. Сейчас мы разъясним, для чего он используется. Параметр `standalone` указывает процессору XML, что можно без ущерба пропустить загрузку внешнего подмножества DTD. Поэтому, если указать `standalone="yes"`, то приложение может благополучно предположить, что все необходимое для форматирования документа находится во внутреннем подмножестве. Иными словами, вне документа нет ничего, что необходимо для полноценного представления содержимого.

Если вы не знаете точно, какое значение нужно присвоить параметру `standalone`, просто опустите его. По умолчанию ему присваивается значение `"no"`, и процессор XML загрузит все, что нужно. Однако в некоторых случаях установка `standalone="yes"` дает преимущества: она сокращает время обработки документа, броузеру не нужно держать открытым соединение, а система архивации может сэкономить ресурсы, которые иначе были бы потрачены на прослеживание связей документа.

Три вещи влияют на возможность объявить документ автономным:

Внешние сущности

Внешней сущностью (отличной от ссылок на объявления типа DTD) является, по определению, содержимое, находящееся вне докумен-

та. Поэтому при использовании в документе ссылки на внешнюю сущность обрабатывающее приложение должно найти и загрузить замещающий ссылку текст. Очевидно, такой документ не является автономным.

Выход в том, чтобы убрать зависимость от внешнего содержимого, навсегда заменив ссылки на внешние сущности соответствующими им текстами. При этом, фактически, процессор XML освобождается от необходимости заменять ссылки на сущности. С другой стороны, при этом теряется смысл использования сущности, поэтому более удачным может оказаться решение оставить ссылки на внешние сущности.

Исключением в этом правиле являются ссылки на внешние подмножества DTD. Не нужно перемещать все объявления элементов, атрибутов и прочие во внутреннее подмножество документа, т. к. процессору XML не всегда требуется искать DTD, чтобы обработать XML. Если удовлетворяются следующие два условия из этого списка, можно безопасно пометить документ как автономный.

Общие сущности, объявленные во внешнем подмножестве

Если в документе используются общие сущности, то проверьте, объявлены ли они вне внутреннего подмножества. Если да, то придется либо переместить эти объявления во внутреннее подмножество, либо подставить вместо ссылок на сущности заменяющий их текст, как вы это сделали бы с внешними сущностями. Попробуйте вместо односимвольных сущностей использовать ссылки на нумерованные символьные сущности. Если все сущности действительно объявлены во внутреннем подмножестве, то документ можно объявить автономным.

Значения атрибутов по умолчанию

С некоторыми элементами документа могут быть связаны значения атрибутов по умолчанию. Например, у гипертекстовой ссылки может быть фиксированный атрибут `xlink:form`, который автор не должен устанавливать самостоятельно. В отсутствие DTD обрабатывающему приложению не будет известно об атрибуте, устанавливаемом по умолчанию, и действие ссылки может измениться. В этом случае разработчик должен сам задать атрибут. Можно поместить объявление списка атрибутов во внутреннее подмножество или вставить атрибут в каждый элемент. Таким образом, если с некоторыми элементами документа связаны значения атрибутов по умолчанию, и эти атрибуты объявлены во внутреннем подмножестве, то документ может быть объявлен автономным.

Режим работы по умолчанию при отсутствии модели документа

Если принято решение не использовать модель документа, то необходимо знать, как будут себя вести элементы и атрибуты. Процессор XML общего назначения использует допущения о режиме по умолчанию, когда отсутствуют более явные указания. Вот частичный перечень того, что можно ожидать:

Неограниченный словарь

Можно безнаказанно давать элементам любые имена. Однако если они «происходят» из другого пространства имен, проверьте, чтобы оно было объявлено.

Отсутствие грамматических правил

Элемент может содержать все – смешанное содержимое, элементы или текст – или быть пустым.

Отсутствие ограничений на атрибуты

Любой элемент может содержать любой атрибут. Однако атрибуты из пространства имен `xml:` сохраняют свои особые свойства.

Все атрибуты обрабатываются, как имеющие тип `CDATA` (символьные данные). Это означает, что вы не сможете использовать для ссылок атрибуты `ID` и `IDREF`. Если вы используете имена `id` и `idref`, то программное обеспечение может оказаться написанным так, что будет обрабатывать их как типы `ID` и `IDREF`, но оно не обязано это делать.

Атрибуты являются необязательными и не имеют значений по умолчанию.

Однако не все процессоры XML являются универсальными. Можно столкнуться с программой, которая рассчитана на определенный шаблон XML, но не утруждает себя проверкой соответствия файла и DTD.

Синтаксис DTD

Чаще всего в качестве модели документа используются определения типа документа (DTD). DTD фактически предшествовали XML, будучи взятыми в готовом виде из SGML и почти полностью сохранив синтаксис. DTD определяет тип документа следующим образом:

- Оно объявляет множество допустимых элементов. Нельзя использовать другие имена элементов, кроме содержащихся в этом множестве. Можно считать это «словарем» языка.
- Оно определяет *модель содержимого* (*content model*) для каждого элемента. Модель содержимого служит шаблоном, определяющим, какие элементы или данные могут находиться внутри элемента, в

каком порядке, сколько их может быть, а также являются ли они обязательными. Можно считать это «грамматикой» языка.

- Оно объявляет для каждого элемента набор разрешенных для него атрибутов. Каждое объявление атрибута определяет имя, тип данных, значения по умолчанию (если они есть) и режим (обязательность или факультативность) атрибута.
- Оно предоставляет ряд механизмов, облегчающих управление моделью, например, использование параметрических сущностей и возможность импортирования части модели из внешнего файла. Эти механизмы описаны в разделе «Объявления сущностей» далее в этой главе.

Пролог документа

В отличие от документа XML, DTD не обязательно имеет пролог. DTD может иметь необязательное объявление XML, такого же типа, который требуется в документах XML. Если требуется задать набор символов, отличный от используемого по умолчанию UTF-8 (подробнее о наборах символов см. главу 7 «Поддержка многоязычности»), или изменить версию XML на отличающуюся от устанавливаемой по умолчанию версии 1.0, это нужно сделать в данном месте.

Если набор символов задан в DTD явно, то это требование не будет автоматически перенесено в документы XML, использующие данное DTD. Документы XML должны задавать собственную кодировку в своем прологе.

Есть и другие части обычного пролога документа, которые не применимы к DTD. Фактически неуместны здесь объявление свойства `standalone` и объявление типа документа. Первое, скорее всего, будет проигнорировано процессором XML, а второе может быть воспринято как синтаксическая ошибка. В данной книге не используется пролог документа в примерах DTD.

Объявления

Как показано на рис. 5.2, DTD является набором правил, или *объявлений* (*declarations*). Каждое объявление добавляет новый элемент, набор атрибутов, сущность или нотацию в описываемый вами язык. DTD могут объединяться с помощью параметрических сущностей – прием, называемый *разбиением на модули* (*modularization*). Можно также помещать объявления во внутреннее подмножество документа.

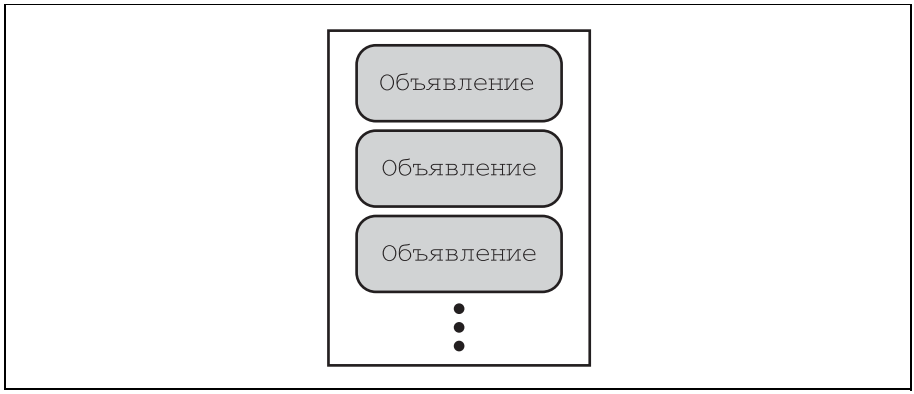


***Рис. 5.2.** Синтаксис DTD*

Порядок объявлений имеет значение в двух ситуациях. Во-первых, если есть избыточные объявления (например, попытка дважды объявить один и тот же элемент), то приоритет отдается первому обнаруженному, а все остальные игнорируются. Об этом важно помнить, если предполагается заменять объявления – во внутреннем подмножестве или путем каскадирования DTD. Во-вторых, если в объявлениях указываются параметрические сущности, они должны быть объявлены раньше, чем будут использованы в качестве ссылок.

Синтаксис объявлений гибок в отношении использования пробельных символов. Можно добавлять пробелы всюду, кроме строки символов в начале, указывающей тип объявления. Например, все нижеследующие объявления являются допустимыми:

```
<!ELEMENT         thingie      ALL>
<!ELEMENT
  thingie
  ALL>
<!ELEMENT thingie (          foo      |
                             bar      |
                             zap      )*>
```

Объявления элементов

Первое и самое важное, чему нужно уделить внимание в языке разметки XML, это множество элементов. Для каждого элемента, который предполагается использовать в документе, должно иметься объявление в DTD. Объявление элемента выполняет две функции: оно добавляет новый элемент в пространство имен языка и устанавливает, что может находиться внутри этого элемента. В совокупности, объявления элементов образуют *грамматику* языка – шаблон, позволяющий определить, какие документы являются действительными.

Как показано на рис. 5.3, объявления элементов состоят из строки <!ELEMENT (1), за которой следуют имя *name* (2), модель содержимого *content-model* (3) и закрывающий ограничитель >. Имя – это имя элемента, т. е. то, что находится внутри тегов разметки в экземпляре документа, например, title или graphic. *Модель содержимого* (*content model*) является формулой, описывающей, какого рода содержимое (данные и элементы) может находиться внутри элемента, в каком количестве и в каком порядке.

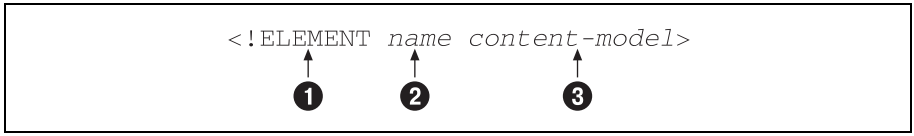


***Рис. 5.3.** Синтаксис объявления элемента*

Имя чувствительно к регистру, поэтому если в объявлении имя записано как ForeignPhrase, то везде нужно писать его в таком виде, но не как FOREIGNPHRASE или foreignphrase. Это важное отличие от SGML. Хотя это может показаться искусственной трудностью, но в результате упрощается написание программ для обработки XML. Принято писать имена элементов строчными буквами, благодаря чему их легче запоминать.

Существует пять различных видов моделей содержимого:

Пустые элементы

Простейшая модель, которую можно объявить, это пустой элемент, и состоит она из ключевого слова EMPTY. Например:

```
<!ELEMENT graphic EMPTY>
```

Элементы без ограничений на содержимое

Эта модель содержимого объявляет элемент, который может содержать любые другие элементы. Она использует ключевое слово ALL:

```
<!ELEMENT contain-anything ALL>
```

Разумеется, элемент, который может содержать что угодно, представляет собой ограниченную ценность в DTD, и такого рода модель содержимого не может управлять структурой так, как это делала бы более строгая модель содержимого. Однако для быстрого создания прототипов документов это может стать удобной временной мерой.

Элементы, содержащие только символьные данные

Для элементов, содержащих символьные данные, но не другие элементы, используйте модель содержимого (#PCDATA):

```
<!ELEMENT emphasis (#PCDATA)>
```

Ключевое слово #PCDATA происходит от «parsed-character data» (анализируемые символьные данные). Это означает, что все символы будут проверены анализатором XML на наличие ссылок на сущности, которые будут заменены на соответствующие значения сущностей. Знак решетки и запись прописными буквами помогут отличить это ключевое слово от каких-либо реально используемых элементов. Обратите внимание, что такие сущности не могут содержать в себе элементы.

Элементы, содержащие только элементы

Содержимое, состоящее только из элементов, описывается с помощью формулы в особых обозначениях, приведенных в табл. 5.1. Такая модель содержимого может выглядеть, например, следующим образом:

```
<!ELEMENT article (para+)>
<!ELEMENT article (title, (para | sect1)+)>
<!ELEMENT article (title, subtitle?, ((para+, sect1*) | sect1+))>
```

Элементы со смешанным содержимым

Смешанное содержимое (*mixed content*) представляет собой смесь элементов и символьных данных. Элемент с содержимым этого типа должен объявляться с помощью модели содержимого, следующей такому шаблону:

```
<!ELEMENT para (#PCDATA | emphasis | xref)*>
```

Первым идет ключевое слово ELEMENT. За ним следуют имя элемента, список допустимых элементов и #PCDATA, разделяемых вертикальной чертой (|) и заключенных в круглые скобки, и, наконец, звездочка. Звездочка, вопросительный знак и другие символы, применяемые в моделях содержимого элементов, описаны в табл. 5.1.

Модели содержимого, содержащие элементы, используют ряд специальных символов, указывающих, какие элементы могут присутствовать, сколь часто и в каком порядке. Синтаксис модели содержимого компактен и поначалу несколько сложен для чтения, но со временем вы научитесь быстро его анализировать.[1] Эти символы тоже перечислены в табл. 5.1.

Проанализируем сложный пример объявления элемента:

```
<!ELEMENT article
  (title, subtitle?, author*, (para | table | list)+, bibliography?)
>
```

[1] Тому, кто знаком с регулярными выражениями, синтаксис модели содержимого может показаться знакомым. В действительности, он очень похож на синтаксис регулярных выражений.

Таблица 5.1. Символы, используемые в моделях содержимого элементов

Символ	Значение	Пример
,	Описывает необходимую последовательность элементов. Действует также в качестве оператора AND.	A, B означает, что B должно следовать за A. B, за которым следует A, неприемлемо.
|	Описывает альтернативный вариант. Иногда называется оператором OR.	red | yellow | green означает, что в данном месте может быть одно из red, yellow или green. Допускается ровно один вариант, не больше и не меньше. Использование одного и того же имени элемента в списке элементов, перечисленных через OR, является ошибкой.
(содержимое)	Объединяет *содержимое* таким образом, что следующий далее оператор применяется ко всему объединению. Скобки могут быть вложенными на любую глубину.	(A | B), C означает, что за A или B должно следовать C, поэтому только A C и B C являются разрешенными последовательностями. Сравните с A | (B, C), которая допускает одно A или B C.
?	Делает предшествующий элемент или группу необязательными.	blug ? означает, что blug может быть использован в этой позиции, а может быть опущен по усмотрению автора документа.
+	Требует по крайней мере одного из предшествующих элементов или групп. Однако верхний предел не устанавливается. (Чтобы потребовать ровно один экземпляр, не используйте никакого оператора.)	(cat | dog) + означает, что значения cat и dog должны быть отличны от нуля. Допустимы, например, cat или dog dog dog cat.
*	Устанавливает, что может присутствовать любое количество предшествующего элемента или группы. Это самый слабый из всех операторов.	(#PCDATA | emphasis) * описывает содержимое как имеющую переменную длину (возможно, нулевую) цепочку символьных данных, перемежающуюся с emphasis в произвольных местах.

Лучше всего читать модель содержимого этого объявления элемента <article>, представив все вместе как группу (вспомните, что скобки

группируют предметы вместе). Каждый член группы может быть элементом или другой группой. Начнем с самой внешней группы и будем продвигаться внутрь:

`title`

Оператора количества (frequency operator) нет, поэтому должен иметься ровно один элемент `<title>` в начале содержимого.

`subtitle?`

Необязательный `<subtitle>` следует за `<title>`. Он не обязателен, потому что оператор «вопросительный знак» означает «ноль или один».

`author*`

Звездочка указывает, что элементов `<author>` может быть ноль или более. Все они должны следовать один за другим.

`(para | table | list)+`

Затем следует группа членов с оператором количества «ноль или более». Это означает, что можно сочетать любое число элементов `<para>`, `<table>` и `<list>`, но должен присутствовать хотя бы один из них.

`bibliography?`

Наконец, в конце содержимого может иметься необязательный элемент `<bibliography>`.

Вот некоторые допустимые варианты этого элемента `<article>`:

```
title, subtitle, author, para, para, para
title, author, author, para, list, para, table, para, bibliography
title, list, list, list, bibliography
title, subtitle, table, para
```

В данном примере показано объявление элемента, содержащего только элементы. Ниже приведен пример смешанного содержания, в которое входят как анализируемые символьные данные, так и элементы:

```
<!ELEMENT para
  (#PCDATA | emphasis | person)*
>
```

Вот пример элемента `<para>`, удовлетворяющего такой модели:

```
<para>And so, <person>Tom Swift</person>
climbed back into his aero-plane to make ready for departure. However,
he noticed a strong smell of gasoline, and realized he was
<emphasis>really</emphasis> in trouble! It was like
&episode-6; all over again!</para>
```

Объявления списков атрибутов

Вслед за элементами объявляются все атрибуты. Обычно атрибуты каждого элемента объявляются в одном месте с помощью *списков объявлений атрибутов* (*attribute declaration lists*). Синтаксис показан на рис. 5.4.

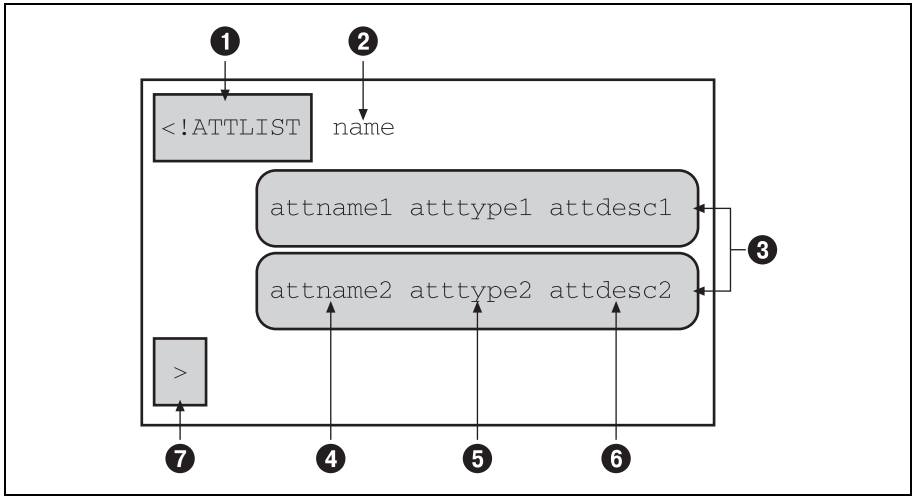


Рис. 5.4. *Синтаксис списка объявлений атрибутов*

Объявление начинается со строки `<!ATTLIST` (1), затем следуют имя элемента, которому принадлежат атрибуты (2), какое-то число объявлений атрибутов (3), а затем – завершающий разделитель `>` (7). Каждое объявление атрибута состоит из имени атрибута (4), его типа данных или перечисления значений (5) и описания режима атрибута (6).

Объявление атрибута выполняет следующие три функции:

- Дает атрибуту имя.
- Указывает тип данных атрибута или список разрешенных значений, которые он может принимать.
- Описывает режим атрибута: есть ли значение по умолчанию или автор документа должен задать его значение.

Ниже приведен пример списка объявлений атрибутов, в котором для элемента `<memo>` определяются три атрибута. Обратите внимание на пробельные символы, добавленные для облегчения чтения примера:

```
<!ATTLIST memo
          id          ID            #REQUIRED     1
          security    (high | low)  "high"        2
          keywords    NMTOKENS      #IMPLIED      3
>
```

1. Первый атрибут, `id`, имеет тип `ID`. Ключевое слово `#REQUIRED` означает, что данный атрибут должен быть задан автором документа.
2. `security`, второй атрибут, может принимать одно из двух значений, `high` или `low`; значением по умолчанию является `high`.
3. Третий атрибут, `keywords`, принимает значение типа `NMTOKENS`, описываемого далее в этом разделе. Ключевое слово `#IMPLIED` означает, что этот атрибут является необязательным и не имеет значения по умолчанию.

Ниже приведен полный список типов данных атрибутов:

`CDATA` *(символьные данные)*

Это самый свободный из всех типов атрибутов. Можно использовать любые символьные данные, в том числе символьные сущности и внутренние общие сущности. Другой разметки, например, элементов или инструкций обработки, в значениях атрибутов этого или любого другого типов быть не должно. Для объявления атрибутов этого типа используйте литерал `CDATA`. Например:

```
<!ATTLIST circle radius CDATA "12 inches">
```

Вот несколько примеров атрибутов со значением типа `CDATA`:

```
dimensions="35x12x9 mm"
company="O'Reilly &amp;amp; Associates"
text=" 5 + 7 = 3 * 4 "
```

Если значение атрибута содержит кавычки, вместо них нужно использовать символьные сущности для кавычек или заключить значение в кавычки другого типа. Например, чтобы задать атрибут `name` со значением `Dave "Brick" Brickner`, можно поместить значение в одинарные кавычки и сказать `name = 'Dave "Brick" Brickner'`. Все пробельные символы в значении превращаются в символ пробела; строки пробелов сохраняются независимо от того, находятся они в начале, в конце или где-нибудь в середине.

`NMTOKEN` *(метка имени)*

Метка имени (*name token*) является строкой символов, начинающейся с буквы, которая может содержать цифры, буквы и некоторые знаки пунктуации. Такой атрибут можно объявить, поместив литерал `NMTOKEN` в поле типа:

```
<!ATTLIST part number NMTOKEN #REQUIRED>
```

Вот некоторые примеры атрибутов со значением типа `NMTOKEN`:

```
skin="reptilian"
file="README.txt"
version="v3.4-b"
```

Все пробельные символы в значении удаляются процессором XML.

NMTOKENS *(список меток имен)*

Список меток имен (*name token list*) является последовательностью из одного или более литералов NMTOKEN, разделенных пробелами. Процессор XML удаляет ведущие и концевые пробельные символы и сокращает другие последовательности пробельных символов до одного пробела. Для объявления атрибута типа списка меток имен укажите литерал NMTOKENS в поле типа. Вот пример:

```
<!ATTLIST article keywords NMTOKENS #IMPLIED>
```

Некоторые примеры атрибутов типа списка меток имен:

```
name="Greg Travis"
format="thin blue border"
```

ID *(уникальный идентификатор)*

Это особый тип атрибута, присваивающий элементу метку, которой гарантируется уникальность в пределах документа. Никаким двум элементам не разрешается иметь одинаковое значение атрибута ID. Атрибут ID имеет одинаковый режим и синтаксис с NMTOKEN и объявляется с помощью литерала ID:

```
<!ATTLIST record ID #REQUIRED>
```

Некоторые примеры атрибутов ID:

```
id="article-2000-03-14"
label="JAVABOOK.CHAPTER.INTRO"
```

IDREF *(ссылка на идентификатор)*

Этот тип атрибута идентичен типу уникального идентификатора, но не помечает элемент, в котором находится, а ссылается на ID другого элемента. Если элемента с заданным ID нет, анализатор сообщает об ошибке. Атрибут применяется для внутренних ссылок, например перекрестных. Используйте литерал IDREF в поле типа следующим образом:

```
<!ATTLIST related-word ref IDREF #REQUIRED>
```

IDREFS *(список ссылок на идентификаторы)*

Разделенный пробелами список значений типа IDREF, составляемый по такой же схеме, как NMTOKENS. Каждый IDREF в списке должен соответствовать значению атрибута ID в документе, иначе анализатор сообщает об ошибке. Пример использования идентификатора типа IDREFS:

```
<!ATTLIST bookset refs IDREFS #REQUIRED>
```

ENTITY *(имя сущности)*

Этот атрибут принимает в качестве значения имя общей сущности. Он должен указываться после объявления сущности в DTD. Вот пример (с ключевым словом ENTITY):

```
<!ATTLIST bulletlist icon ENTITY #IMPLIED>
<!ENTITY bluedot SYSTEM "icons/bluedot.png">
```

А вот как можно его использовать:

```
<bulletlist icon="bluedot">
```

ENTITIES *(список имен сущностей)*

Значением этого атрибута является список имен сущностей, разделенных пробелами. Атрибут типа списка имен сущностей использует имя типа ENTITIES. Например:

```
<!ATTLIST album filelist ENTITIES #REQUIRED>
```

Список перечисляемых значений

Список перечисляемых значений – это список определяемых автором ключевых слов; атрибут со списком перечисляемых значений может быть полезен, если есть небольшая группа возможных значений. Объявление этого типа атрибута производится не с помощью ключевого слова в поле типа, а посредством списка значений в круглых скобках, разделяемых символами вертикальной черты (|). Например, атрибут, принимающий значения «истина» или «ложь», можно объявить так:

```
<!ATTLIST part instock ( true | false ) #IMPLIED>
```

А можно задать в списке значений дни недели:

```
<!ATTLIST schedule
  day ( mon | tue | wed | thu | fri | sat | sun ) #REQUIRED
>
```

Атрибуту может одновременно быть присвоено только одно значение из списка:

```
<schedule day="fri">
```

Если объявляется значение по умолчанию, то им должно быть одно из значений в списке:

```
<!ATTLIST shape
  type (circle | square | triangle) "square">
```

NOTATION *(список нотаций)*

Подобно атрибуту NMTOKENS, значение атрибута NOTATION состоит из последовательности меток имен. Атрибут такого типа соответству-

ет одному или более типам нотаций, являющихся инструкциями, которые указывают, как нужно обрабатывать форматированные или не являющиеся XML данные. Например, можно объявить нотацию, чтобы сохранить ведущие или замыкающие пробельные символы в элементе, которые обычно удаляются анализатором. Подробнее о нотациях мы поговорим далее в этой главе.

Атрибуты могут иметь несколько различных режимов:

Присваивается значение по умолчанию

Если пользователь не указывает значение для этого атрибута, процессор XML предполагает, что в DTD указано значение по умолчанию. Указывать в DTD значения по умолчанию бывает удобно, когда одно из значений встречается чаще всего. Чтобы объявить значение по умолчанию, просто поместите его в кавычках в колонке режима. Например:

```
<!ATTLIST message
  importance (high | medium | low) "medium"
>
```

Здесь мы выбрали `medium` из списка перечисления в качестве значения по умолчанию. Если опустить атрибут `importance` в элементе `<message>`, процессор XML использует значение по умолчанию.

Атрибут является необязательным (`#IMPLIED`)

Когда атрибут объявлен как необязательный, процессор XML не присваивает ему значение по умолчанию: атрибут фактически отсутствует. С помощью ключевого слова `#IMPLIED` мы сообщаем, что значения по умолчанию нет и использование атрибута является факультативным. Например:

```
<!ATTLIST reservation
  frills (aisle-seat | meal-included | pillow) #IMPLIED
>
```

Пользователь должен указать значение (`#REQUIRED`)

Если нет подходящего значения по умолчанию и атрибут нельзя оставить пустым, следует объявить его как `#REQUIRED`. В этом случае пропуск атрибута или присвоение ему пустого значения вызовут сообщение анализатора об ошибке. Вот пример:

```
<!ATTLIST book isbn CDATA #REQUIRED>
```

Значение уже присвоено, пользователю нельзя его изменять (`#FIXED`)

В редких случаях атрибут должен всегда иметь одно и то же значение. Например, мы настраиваем общедоступное DTD и хотим, чтобы оно в некоторых отношениях было более жестким. Помимо ключевого слова `#FIXED` необходимо задать значение. Генри Форд, пер-

вые автомобили которого были только черного цвета, мог бы использовать такое объявление:

```
<!ATTLIST car
  color (beige, white, black, red, blue, silver) #FIXED "black"
>
```

Необязательно объявлять все атрибуты элемента в одном месте. XML позволяет размещать их в разных списках объявлений, которые анализатор XML объединяет (merges) по мере чтения DTD. Это дает нам некоторую степень свободы в переделке DTD. Если нужно добавить новый атрибут элементу, для которого в DTD уже определены некоторые атрибуты, можно поместить объявление в другом месте, например, во внутреннем подмножестве, и оно будет автоматически объединено с существующими.

Нотации и неанализируемые данные

XML разработан, главным образом, так, чтобы служить контейнером для текстовой информации. Он не идеален для хранения двоичных данных, таких как растровые изображения или сжатый текст, но не является совершенно несовместимыми с ними. Пропитанный мультимедиа мир, в котором мы живем, щедро приправлен картинками, звуковыми клипами, анимацией и всем что угодно. Если бы XML мог обрабатывать только текст, то его практическая ценность была бы слишком ограничена. К счастью, XML предоставляет механизм сосуществования с данными, не являющимися XML, который называется спецификацией *нотаций*.

В широком смысле, нотация является особой меткой, сообщающей процессору XML о том, какого типа данные он рассматривает в данный момент. Пометка нетекстовых данных является одним из применений нотаций. Другим из них является пометка текстовых данных, имеющих особый формат, например, дат.

Объявления нотаций

Синтаксис объявления типа нотации следующий:

```
<!NOTATION name identifier>
```

Здесь *name* является именем типа нотации, а *identifier* – внешним идентификатором, имеющим определенное значение для процессора XML. Какое именно значение, зависит от приложения. На самом деле, в сообществе XML идет широкое обсуждение того, что использовать в качестве внешнего идентификатора нотации, который может выражать все что угодно – от URL документа со стандартами до типа MIME и названия программы на локальной машине. В табл. 5.2 приведены примеры таких внешних идентификаторов.

Таблица 5.2. Некоторые внешние идентификаторы нотаций

Пример	Значение
`SYSTEM "application/x-troff"`	Тип MIME для текста в формате troff
`SYSTEM "ISO 8601:1988"`	Номер международного стандарта форматов дат (например, 1994-02-03)
`SYSTEM "http://www.w3.org/TR/NOTE-datetime"`	URL технического документа в Интернете, касающегося форматов дат
`PUBLIC "-//ORA//NON-SGML Preferred Date Format//EN" "http://www.oreilly.com/xml/dates.html"`	Формальный открытый идентификатор ресурса в сети
`SYSTEM "/usr/local/bin/xv"`	Компьютерная программа на локальной машине, которая должна вызываться для обработки неанализируемых данных

Все приведенные варианты имеют право на жизнь. Никто еще не предложил стройной системы идентификаторов нотаций, поэтому сами решайте, какую использовать. Важно, чтобы идентификатор был уникальным и сообщал процессору XML достаточно информации для обработки данных. Строго определенного правила нет, но мы рассмотрим ряд примеров.

Объявление нотации создает метку, которая используется вместе с объявлением атрибута или неанализируемой внешней сущностью. Об этом рассказывается в следующем разделе.

Неанализируемые сущности

Внешние общие сущности импортируют данные XML из других файлов. Есть еще один тип сущности для импорта данных, не являющихся XML, который называется *неанализируемая сущность* (*unparsed entity*). Объявления неанализируемой и общей сущностей аналогичны, за исключением ключевого слова `NDATA` и типа нотации, следующих за системным или открытым идентификатором. Например:

```
<!ENTITY song "jingle_bells.wav" NDATA audio-wav>
```

В следующем примере мы объявили два типа нотаций, `jpeg` и `png`, используя в качестве идентификаторов их типы MIME. Элемент `<graphic>` объявлен пустым с атрибутом `source`, значением которого является имя сущности. Мы также объявляем неанализируемые сущности `bob` и `judy`, ссылающиеся на некоторые графические файлы, которые будут переданы элементу `<graphic>` через его атрибут в экземпляре XML `<doc>`, следующем за объявлениями.

```
<?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.png" 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 (посредством ключевого слова `NDATA`), процессор XML не обрабатывает ее как данные XML, а передает прямо в ту часть программы, которая умеет ее обрабатывать. Например, в веб-броузере есть функция, воспринимающая некоторые типы графических данных и способная выводить их на экран. Иногда программное обеспечение оказывается неспособным что-либо сделать с данными, и тогда оно может вывести сообщение об ошибке, спросить у пользователя, что делать с данными, или просто отбросить их.

Не встраивайте неанализируемую сущность непосредственно в документ XML. Вместо этого передайте ее в элемент с помощью атрибута, как это сделали мы в предыдущем примере. Следующий документ не является корректным (well-formed):

```
<?xml version="1.0"?>
<!DOCTYPE doc [
  <!ELEMENT doc ANY>
  <!NOTATION jpeg SYSTEM "image/jpeg">
  <!ENTITY bob  "pictures/bob.jpeg" NDATA jpeg>
]>
<doc>
  &bob;
</doc>
```

Пометка форматов элементов с помощью нотаций

Посредством нотаций можно указать, как должны интерпретироваться символьные данные. Допустим, например, что есть форма, ко-

торая требует ввода идентификационного номера. Заполняющий эту форму может ввести свой номер социального страхования, водительского удостоверения или какой-то еще. Применение нотации позволяет указать, который из форматов используется.

Рассмотрим следующий пример, в котором нотации служат для того, чтобы различать форматы данных и типы программ:

```
<?xml version="1.0"?>
<!DOCTYPE record [
  <!ELEMENT doc (title, listing+)>
  <!ELEMENT title (#PCDATA)*>
  <!ELEMENT listing (#PCDATA)*>
  <!ATTLIST listing
    format NOTATION (scheme-lisp | ansi-c) #REQUIRED
  >
  <!NOTATION scheme-lisp SYSTEM "IEEE 1178-1990">
  <!NOTATION ansi-c      SYSTEM "ISO/IEC 9899:1999">
]>

<doc>
<title>Factorial Function</title>
  <listing format="scheme-lisp">
    (defun fact (lambda (n) (if (= n 1) 1 (fact (- n 1)))))
  </listing>
  <listing format="ansi-c">
    int fact( int n ) {
      if( n == 1 ) return 1;
      return n * fact( n - 1 );
    }
  </listing>
</doc>
```

В этом примере два листинга. Нотация описывает способ интерпретации текстовых данных. Внешние идентификаторы берутся из международных организаций стандартизации IEEE и ISO. Из документа не ясно, что именно должно быть сделано с данными в элементах `<listing>`, но если процессор XML распознает внешние идентификаторы, можно предполагать, что конечное приложение будет знать, что ему делать.

Предостережение относительно нотаций

В XML не определено ничего конкретного в отношении обработки неанализируемых данных. Мы не можем сказать, как будет действовать приложение, столкнувшись с атрибутом `NDATA` или инструкцией обработки; возможность определить поведение приложения предоставляется его разработчику.

К сожалению, это означает плохую переносимость нотаций. Они зависят от многих предположений, например, сможет ли процессор XML обрабатывать некоторый тип данных или распознать заданный внеш-

ний идентификатор. Если документ XML будет обрабатываться несколькими различными программами, существует большая вероятность того, что он окажется несовместимым по крайней мере с одной из них. Поэтому нотации следует применять экономно и с осторожностью.

Возможно, лучше воспользоваться инструкцией обработки. Поскольку это маркер, относящийся к конкретному приложению, инструкция обработки служит естественным способом передачи конкретных команд для обработки неанализируемых данных. Документ «The XML Recommendation» предлагает использовать имя нотации в качестве первой части инструкции обработки, чтобы оставшаяся часть инструкции была правильно интерпретирована.

Объявления сущностей

Мы вкратце говорили о сущностях и объявлениях сущностей в главе 2; теперь мы обсудим объявления сущностей более подробно. В следующем списке показаны различные типы объявлений сущностей. Объявления общих сущностей уже знакомы читателю, но объявления параметрических сущностей окажутся внове.

Общая сущность

Простая подстановка в анализируемом тексте. Например:

```
<!ENTITY abc "The ABC Group">
```

Общая сущность указывается в виде `&abc;`.

Внешняя общая сущность

Сущность, содержащая текст из внешнего источника. В первом приведенном примере источник задан посредством его формального открытого идентификатора. Во втором примере он указан через местонахождение на машине или в сети:

```
<!ENTITY man PUBLIC "-//Acme Gadgets//TEXT Manual 23//EN"
  "http://www.acme-gadgets.com/manuals/prod23.htm">
<!ENTITY man SYSTEM "/pub/docs/manuals/prod23.htm">
```

Ссылка на эту сущность указывается в виде `&man;`.

Неанализируемая внешняя сущность

Сущность, содержащая данные, не являющиеся XML и находящиеся во внешнем источнике. В первом приведенном примере источник задан посредством его формального открытого идентификатора. Но во втором примере данные импортируются из другого файла:

```
<!ENTITY logo PUBLIC "-//Acme Gadgets//NON-XML Logo//EN"
  "http://www.acme-gadgets.com/images/logo.gif" NDATA gif>
<!ENTITY logo SYSTEM "images/logo.gif" NDATA gif>
```

Ссылка на эту сущность указывается в виде `&logo;`.

Параметрическая сущность

Простая подстановка текста в DTD. Например:

```
<!ENTITY % paratext "(#PCDATA | emph | acronym)*">
```

Ссылка на эту сущность указывается в виде `%paratext;`.

Внешняя параметрическая сущность (External parameter entity)

Сущность, содержащая DTD или часть DTD из внешнего источника. В первом приведенном примере источник задан посредством его формального открытого идентификатора. Во втором примере он указан через местонахождение на машине или в сети.

```
<!ENTITY % tables PUBLIC "-//Acme Gadgets//DTD Tables 2.1//EN"
  "/xmlstuff/dtds/Acme/tables2.1.dtd">
<!ENTITY % tables SYSTEM "http://www.xmljunk.org/dtds/
  tables2.1.dtd">
```

Ссылка на эту сущность указывается в виде `%tables;`.

Параметрические сущности

Ранее уже говорилось о параметрических сущностях, но их описание было отложено до настоящего момента. *Параметрическая сущность* (*parameter entity*) содержит текст из DTD и может использоваться во внутреннем или внешнем подмножестве. Она не может содержать текст XML, и ссылка на параметрическую сущность не может находиться внутри документа XML.

Чтобы отличать параметрические сущности от общих, им дан несколько иной синтаксис. В объявлении перед именем сущности ставится знак процента (`%`), а в ссылке на нее знак процента используется вместо амперсанда (`&`). Вот пример, в котором присутствуют объявления и несколько ссылок для двух параметрических сущностей:

```
<!ENTITY % content "para | note | warning">
<!ENTITY % id.att "id   ID   #REQUIRED">
<!ELEMENT chapter (title, epigraph, (%content;)+)>
<!ATTLIST chapter %id.att;>
<!ELEMENT appendix (title, (%content;)+)>
<!ATTLIST appendix %id.att;>
```

Можно видеть, насколько параметрические сущности упрощают разработку и сопровождение DTD. В моделях содержимого `<chapter>` и `<appendix>` многое совпадает, например, текст `para | note | warning`. Их объявления списка атрибутов одинаковы и требуют атрибута `ID`. Мы упростили DTD, определив параметрические сущности для этих общих частей. После этого можно ссылаться на данные параметрические сущности внутри объявлений элементов и атрибутов, избегая в ре-

зультате лишнего ввода с клавиатуры и загроможденности текста; при этом также появляется возможность изменять модели содержимого нескольких элементов, редактируя код лишь в одном месте.

Будьте осторожны, – применяя параметрические сущности, легко внести труднообнаружимые синтаксические ошибки, скажем, введя лишнюю запятую или пропустив скобки в замещающем тексте. Например, ошибочным будет следующий текст:

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

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

Он транслируется в такой синтаксически неверный текст:

```
<!ELEMENT chapter (title, epigraph, para | note | warning+)>
```

Без круглых скобок, в которые должны быть заключены последние три элемента `para | note | warning`, эта модель содержимого бессмысленна.

Внешние параметрические сущности

Внешние параметрические сущности напоминают внешние общие сущности, т. к. импортируют текст из другого файла. Как и все параметрические сущности, они используются только внутри DTD. Сущности этого типа применяются для импортирования частей DTD, находящихся в других файлах, – прием, называемый *разбиением на модули* (*modularizing*). При осторожном использовании эти сущности дают мощное средство для организации структуры и подгонки больших DTD.

Объявление внешней параметрической сущности аналогично объявлению параметрической сущности, но вместо строки замещающего текста указывается открытый или системный идентификатор, которому предшествует ключевое слово `PUBLIC` или `SYSTEM`. Вот несколько примеров:

```
<!ENTITY % inline-elements SYSTEM "inlines.mod">
<!ENTITY % ISOamsa PUBLIC
  "ISO 8879:1986//ENTITIES Added Math Symbols: Arrow Relations//EN//
    XML"
  "/usr/local/sgml/isoents/isoamsa.ent">

%inline-elements;
%ISOamsa;
```

В первом примере с помощью системного идентификатора объявлена внешняя параметрическая сущность. Системный идентификатор указывает на файл, содержащий объявления элементов и атрибутов, которые можно вставить в DTD с помощью ссылки `%inline-elements;`. Во втором примере используется формальный открытый идентификатор для обращения к группе символьных сущностей, опубликованных ISO

(Added Math Symbols ISO-8879:1986). С помощью внешних параметрических сущностей можно осуществлять комбинирование при создании собственного уникального DTD. О разработке и настройке DTD еще будет говориться на протяжении всей этой главы.

Пример: чековая книжка

Соберемся с силами и применим изученный к данному моменту материал, чтобы разработать DTD для приложения чековой книжки. Пример 5.1 показывает, как может выглядеть такой документ.

Пример 5.1. Образец документа «чековая книжка»

```
<?xml version="1.0"?>
<!DOCTYPE checkbook SYSTEM "checkbook.dtd">

<checkbook>

  <deposit type="direct-deposit">
    <payor>Bob's Bolts</payor>
    <amount>987.32</amount>
    <date>21-6-00</date>
    <description category="income">Paycheck</description>
  </deposit>

  <payment type="check" number="980">
    <payee>Kimora's Sports Equipment</payee>
    <amount>132.77</amount>
    <date>23-6-00</date>
    <description category="entertainment">Kendo equipment</description>
  </payment>

  <payment type="atm">
    <amount>40.00</amount>
    <date>24-6-00</date>
    <description category="cash">Pocket money</description>
  </payment>

  <payment type="debit">
    <payee>Lone Star Cafe</payee>
    <amount>36.86</amount>
    <date>26-6-00</date>
    <description category="food">Lunch with Greg</description>
  </payment>

  <payment type="check" number="981">
    <payee>Wild Oats Market</payee>
    <amount>47.28</amount>
    <date>29-6-00</date>
    <description category="food">Groceries</description>
  </payment>
```

Пример 5.1. Образец документа «чековая книжка» (продолжение)

```
<payment type="debit">
    <payee>Barnes and Noble</payee>
    <amount>58.79</amount>
    <date>30-6-00</date>
    <description category="work">O'Reilly Books</description>
  </payment>

</checkbook>
```

Что можно сказать о типе документа, взглянув на этот пример? Корневым элементом является <checkbook>. Он содержит ряд записей, каждая из которых является элементом <payment> (платеж) или <deposit> (вклад). Вооружившись этой информацией, можно написать первое объявление:

```
<!ELEMENT checkbook (deposit | payment)*>
```

Это было достаточно просто. Похоже, что <deposit> чуточку посложнее. Его дочерние элементы следующие: <payor>, <amount>, <date> и <description> (плательщик, сумма, дата и описание). Попробуем написать для него объявление:

```
<!ELEMENT deposit (payor | amount | date | description)*>
```

С таким объявлением возникают проблемы: оно не препятствует вводу нескольких элементов одного типа, тогда как разумным было бы иметь только по одному элементу каждого типа. Кроме того, некоторые элементы, вероятно, нужно сделать обязательными, а звездочка (*) этого не обеспечивает. Возможно, лучше использовать такое объявление:

```
<!ELEMENT deposit (payor, amount, date, description?)>
```

Теперь обязательными являются все элементы, кроме <description>. Единственная проблема такого варианта в том, что элементы должны присутствовать в указанном порядке, поскольку они разделены запятыми. Однако порядок вряд ли должен иметь значение. К сожалению, в DTD не очень удобно задавать произвольный порядок обязательных элементов. Модель содержимого, которая разрешает появление первых трех обязательных элементов в любом порядке, может выглядеть так:

```
<!ELEMENT deposit (
   ((amount, ((date, payor) | (payor, date))) |
    (date, ((amount, payor) | (payor, amount))) |
    (payor, ((amount, date) | (date, amount)))), description)>
```

Тьфу! Остановимся на втором варианте. Можно потерпеть необходимость соблюдать определенный порядок, если в результате модель содержимого станет проще. В сложных DTD проще ошибиться, а кроме того, ими труднее управлять.

У элемента <deposit> имеется также атрибут type. Он содержит символьные данные, указывающие, например, был ли вклад сделан наличными или чеком. Атрибут можно объявить так:

```
<!ATTLIST deposit type #CDATA #IMPLIED>
```

Улучшить это объявление можно в двух отношениях. Во-первых, этот атрибут, вероятно, должен быть обязательным. Если опустить в операции тип вклада, то это может в будущем создать неудобства. Во-вторых, тип #CDATA является слишком «вольным», позволяя вводить все что угодно, например, «pajamas» или «718». Лучше ограничить возможные значения. Вот наша вторая попытка:

```
<!ATTLIST deposit type (cash | check | direct-deposit | transfer)
  #REQUIRED>
```

Разумеется, если в будущем вы обнаружите, что есть и другие значения, которые требуется использовать в типе вклада, нужно будет сначала добавить их в DTD.

Элемент <payment> аналогичен <deposit>:

```
<!ELEMENT payment (payee?, amount, date, description?)>
<!ATTLIST payment type (atm | check | debit) #REQUIRED>
```

В данном случае мы сделали <payee> (получателя) необязательным. Предполагается, что если он отсутствует, получателем средств является сам автор.

Остальные элементы простые:

```
<!ELEMENT amount (#PCDATA)*>
<!ELEMENT date (#PCDATA)*>
<!ELEMENT description (#PCDATA)*>
<!ELEMENT payee (#PCDATA)*>
<!ELEMENT payor (#PCDATA)*>
```

У <description> тоже есть атрибут:

```
<!ATTLIST description
  category (cash | entertainment | food | income | work) 'food'>
```

Заметьте, что значением по умолчанию для этого элемента мы сделали food . Поэтому если автор не укажет атрибут category, процессор XML вставит это значение. Это хорошая мысль, т. к. еда чаще всего приобреталась автором этой чековой книжки.

Можно сделать еще одну вещь перед тем, как собрать объявления в DTD. У некоторых элементов есть общие характеристики. С помощью параметрических сущностей можно показать эту связь и облегчить чтение и сопровождение данного DTD. Вот сущности, которые можно объявить:

```
<!ENTITY % basic.content '#PCDATA'>
<!ENTITY % entry.content 'amount, date, description?'>
```

Параметрические сущности могут использоваться таким способом только во внешнем подмножестве, но не во внутреннем. Вот слова Рекомендации XML: «Во внутреннем подмножестве DTD ссылки на параметрические сущности могут находиться только там, где могут находиться объявления разметки, но не внутри объявлений разметки. (Это не относится к ссылкам, находящимся во внешних параметрических сущностях, или внешнему подмножеству.)»

Первая параметрическая сущность является моделью содержимого для более мелких элементов, таких как <date>. Во второй хранится общее содержание для элементов <payment> и <deposit>. В примере 5.2 все объединено вместе.

Пример 5.2. DTD чековой книжки

```
<!--

A simple checkbook DTD

-->

<!-- parameter entities -->

<!ENTITY % basic.content '#PCDATA'>
<!ENTITY % entry.content 'amount, date, description?'>

<!-- main elements -->

<!ELEMENT checkbook    (deposit | payment)*>
<!ELEMENT deposit      (payor, %entry.content;)>
<!ATTLIST deposit      type (cash | check | direct-deposit | transfer)
  #REQUIRED
                 number  #CDATA  #IMPLIED>
<!ELEMENT payment      (payee?, %entry.content;)>
<!ATTLIST payment      type (atm | check | debit) #REQUIRED>

<!-- basic elements -->
```

Пример 5.2. DTD чековой книжки (продолжение)

```
<!ELEMENT amount       (%basic.content;)*>
<!ELEMENT date         (%basic.content;)*>
<!ELEMENT payee        (%basic.content;)*>
<!ELEMENT payor        (%basic.content;)*>
<!ELEMENT description  (%basic.content;)*>
<!ATTLIST description
  category (cash | entertainment | food | income | work) 'food'>
```

DTD должно быть таким, чтобы его было просто использовать и расширять. В этом примере все просто и ясно, чему способствуют комментарии и пробельные символы, облегчающие чтение. Объявления `ATTLIST` располагаются рядом с объявлениями `ELEMENT` для тех же элементов, а элементы сгруппированы по назначению. Параметрические сущности делают его более понятным и позволяют одновременно изменять несколько вещей. Например, мы могли бы следующим образом переопределить сущность `%basic.content;`:

```
<!ENTITY % basic.content '#PCDATA | placename'>
```

В результате расширились бы модели содержимого всех основных элементов, в которые помимо символьных данных были бы включены необязательные элементы `<placename>`. Например:

```
<payee><placename>Big Boy's</placename> restaurant at
  <placename>Oneida</placename> rest stop on
  <placename>NYS Thruway</placename></payee>
```

В следующих разделах описываются другие методы, применяемые для организации DTD. Они становятся необходимыми в очень больших DTD, разбиваемых на несколько модулей.

Советы по проектированию и настройке DTD

Проектирование и создание DTD являются частично наукой и частично искусством. Базовые понятия достаточно просты, но управление большим DTD – поддержка сотен определений элементов и атрибутов при сохранении читаемости и отсутствии ошибок – может быть серьезной проблемой. В данном разделе предлагается ряд советов и практических приемов, которые могут оказаться полезными. В следующем разделе показан конкретный пример, использующий эти технологии.

Поддержка структурированности

DTD славятся трудностью чтения, но оно облегчается при хорошей структуризации. Несколько минут, потраченных на наведение порядка и запись комментариев, могут избавить вас от многочасового изучения DTD в будущем. Часто DTD самодокументируются, поэтому если предполагается, что их будут использовать другие люди, опрятность кода важна вдвойне.

Упорядочивайте объявления по назначению

Собирайте объявления в группы соответственно их назначению. В маленьких DTD это облегчает перемещение по файлу. В более крупных DTD может даже потребоваться разбить объявления по отдельным модулям. Группировку можно осуществлять по таким категориям, как блоки, внутритекстовые элементы, элементы иерархии, части таблиц, списки и т. д. В примере, приведенном в следующей главе, вы найдете объявления, разделенные по назначению (блоковые, внутритекстовые, иерархические).

Пробельные символы

Не жалейте пробелов в своих объявлениях. Модели содержимого и списки атрибутов отличаются плотным синтаксисом, поэтому выделение частей пробелами и даже помещение на отдельных строках делает их более понятными. Делайте отступы строк внутри объявлений, чтобы лучше выделялись разделители. Между логическими частями используйте дополнительное свободное пространство и, возможно, комментарий со строкой темных символов, чтобы усилить границу. При быстрой прокрутке файла это значительно облегчает нахождение нужного места.

Вот как может выглядеть DTD, если пренебречь пробелами:

```
<!ATTLIST div id ID #REQUIRED title CDATA #IMPLIED role CDATA
  #IMPLIED>
<!ATTLIST article security (high|low|medium) "high" keywords CDATA
  #IMPLIED author CDATA #IMPLIED id ID #REQUIRED>
<!ATTLIST xref xlink:form CDATA #FIXED "simple"
  xlink:href CDATA #REQUIRED>
```

Добавив некоторое число пробельных символов, можно сделать его значительно привлекательнее:

```
<!ATTLIST div
   id                ID                          #REQUIRED
   title             CDATA                       #IMPLIED
   role              CDATA                       #IMPLIED
   >
<!ATTLIST article
   security          ( high | low | medium )     "high"
   keywords          CDATA                       #IMPLIED
```

```
    author              CDATA                   #IMPLIED
    id                  ID                      #REQUIRED
    >
<!ATTLIST xref
    xlink:form          CDATA                   #FIXED "simple"
    xlink:href          CDATA                   #REQUIRED
    >
```

Комментарии

Не скупитесь на комментарии, они – как указатели в дикой чаще текста. Во-первых, в начале каждого файла помещайте комментарии, разъясняющие назначение этого DTD или модуля, указывающие номер версии и контактную информацию. Если это специализированный интерфейс к общедоступному DTD, не забудьте указать оригинал, на котором он основан, отдайте дань уважения авторам и опишите внесенные вами изменения. Далее, снабдите метками все разделы и подразделы DTD.

Всюду, где комментарий может разъяснять применение DTD или какие-то ваши решения, воспользуйтесь им. По мере модификации своего DTD добавляйте новые комментарии, описывающие вносимые изменения. Комментарии составляют часть документации, а непонятная или устаревшая документация может оказаться хуже, чем просто бесполезной.

Нумерация версий

Так же, как и программное обеспечение, DTD может обновляться по мере изменения требований. Необходимо вести нумерацию версий и, чтобы избежать неприятностей, менять номер версии после внесения изменений в документ. Принято присваивать первой публикуемой версии номер «1.0». После этого небольшим внесенным изменениям соответствует наращивание числа после точки: «1.1», «1.2» и т. д. Серьезные изменения сопровождаются наращиванием целой части: «2.0», «3.0» и т. д. Документируйте изменения, производимые по сравнению с предыдущей версией. Это процесс может быть автоматизирован с помощью систем контроля версий. В системах, основанных на Unix, верными друзьями разработчиков являются пакеты RCS и CVS.

Параметрические сущности

Параметрические сущности могут содержать повторяющиеся части объявлений и позволяют осуществлять редактирование в одном месте. Во внешнем подмножестве они могут быть использованы в объявлениях типов элементов для хранения групп элементов и модели содержимого или в объявлениях списков атрибутов для хранения определений атрибутов. Внутреннее подмножество несколько строже к параметрическим сущностям: они могут содержат только законченные объявления, а не их фрагменты.

Допустим, например, что требуется наличие у каждого элемента необязательного атрибута ID для осуществления ссылок и необязательного атрибута класса для присвоения информации о специфической роли. Можно определить параметрическую сущность, хранящую общие атрибуты, следующим образом:

```
<!ENTITY % common.atts "
  id        ID        #IMPLIED
  class     CDATA     #IMPLIED"
  >
```

После этого можно использовать данную сущность в объявлениях списков атрибутов:

```
<!ATTLIST foo %common.atts;>
<!ATTLIST bar %common.atts;
   extra    CDATA     #FIXED "blah"
   >
```

Выбор атрибутов и элементов

Создавать DTD с чистого листа нелегко. Необходимо разбить информацию на концептуальные атомы и оформить в виде иерархической структуры, но не всегда бывает ясно, как осуществить разделение информации. Модель книги проста, поскольку с легкостью разбивается на иерархию контейнеров, например, глав, разделов и абзацев. Менее очевидны модели уравнений, молекул и баз данных. Для таких применений нужен гибкий ум, чтобы «раздробить» документы в оптимальное сочетание элементов и атрибутов. Приводимые здесь советы являются принципами, которые призваны помочь при проектировании DTD:

- Выбирайте осмысленные имена. Если документ состоит только из таких элементов, как <thing>, <object> и <chunk>, понимать их смысл становится почти невозможно. Имя элемента должно близко соответствовать его логическому назначению. Лучше создавать специальные элементы для различных задач, чем перегружать небольшое число элементов обработкой различных ситуаций. Например, элементы HTML <DIV> и <SPAN> не являются идеальными, поскольку выполняют много различных задач.
- Иерархия несет в себе информацию. Газета содержит статьи, которые содержат абзацы и заголовки. Контейнеры создают границы, облегчающие написание таблиц стилей и обрабатывающих приложений. Они также влекут отношения владения, предоставляющие процессорам удобные указатели и средства навигации. Контейнеры создают глубину – еще одно измерение, увеличивающее степень структурности.

Стремитесь к созданию древовидной структуры, напоминающей обширный разросшийся куст. Если вы уйдете слишком глубоко, то разметка начнет преобладать над содержанием, и редактировать документ станет труднее; слишком мелкая разметка ослабляет информационное содержание. Хорошей аналогией является представление документов и их частей в виде ящиков. С большим ящиком, в котором миллион мелких, значительно труднее работать, чем с ящиком, в котором несколько ящиков среднего размера, внутри которых еще меньшие ящики и т. д.

- Выбирайте, когда лучше использовать элементы, а когда – атрибуты. В элементе хранится содержимое, являющееся частью документа. Атрибут видоизменяет поведение элемента. Сложность в том, чтобы найти равновесие между применением общих элементов с атрибутами, указывающими назначение, и созданием элементов для каждого отдельного случая.

Разбиение на модули

Разбиение цельного DTD на меньшие компоненты, или *модули*, дает определенные преимущества. Первое заключается в том, что DTD, разбитое на модули, проще поддерживать благодаря структуризации, о которой говорилось выше, и потому, что части можно редактировать по отдельности или «выключать» на период отладки. Кроме того, DTD становится конфигурируемым. Модули, находящиеся в отдельных файлах, можно менять один на другой так же легко, как переопределять отдельные параметрические сущности. Даже в пределах одного файла их можно помечать для включения или исключения.

XML предоставляет два способа разделения DTD на модули. Первый состоит в том, чтобы хранить части в отдельных файлах, а затем импортировать их с помощью внешних параметрических сущностей. Второй состоит в применении синтаксического аппарата под названием «условная секция». Оба они представляют собой мощные способы придания DTD большей гибкости.

Импорт модулей из внешних источников

DTD не обязательно должно располагаться в одном файле. В действительности, часто имеет смысл размещать его в различных местах. Можно воспользоваться чьим-то чужим DTD, импортируя его в свое собственное в качестве подмножества. Либо можно сделать свое DTD несколько более четким, разместив его части в нескольких файлах.

Для импорта целых DTD или их частей используйте внешние параметрические сущности. Вот пример законченного DTD, которое импортирует свои части из разных модулей:

```
<!ELEMENT catalog (title, metadata, front, entries+)>
<!ENTITY % basic.stuff   SYSTEM "basics.mod">
<!ENTITY % front.matter  SYSTEM "front.mod">
<!ENTITY % metadata      PUBLIC "-//Standards Stuff//DTD Metadata
  v3.2//EN" "http://www.standards-stuff.org/dtds/metadata.dtd">
```

Это DTD состоит из двух локальных компонент, указанных системными идентификаторами. Каждая компонента хранится в файле с расширением *.mod*, что является обычным способом сообщить о хранении в файле объявлений, которые не должны применяться как самостоятельные DTD. Последней компонентой является DTD, которое может использоваться самостоятельно; в действительности, в данном примере это общедоступный ресурс.

При импортировании текста DTD может возникнуть одна проблема. Внешняя параметрическая сущность импортирует *весь* текст файла, а не просто его часть. Вы получаете все объявления, а не только избранные. Еще хуже, что не существует понятия локальной области видимости, в которой объявления локального DTD автоматически заменяют те, которые находятся в импортированном файле. Объявления собираются в один логический объект, а любая информация о том, что и откуда импортировано, теряется перед тем, как производится анализ DTD.

Есть несколько способов справиться с этой проблемой. Можно подавить объявления путем переобъявления, или, точнее, предварительного объявления. Иными словами:

- Если для одного и того же элемента существует несколько объявлений, используется первое, обнаруженное процессором XML, а остальные игнорируются.
- Если для одного и того же имени элемента существует несколько объявлений списка атрибутов, они объединяются в один список.
- Если в этом списке имя атрибута объявляется несколько раз, то приоритет имеет объявление, обнаруженное первым.

Допустим, что в файле есть ряд объявлений, в том числе следующее объявление элемента:

```
<!ELEMENT polyhedron (side+, angle+)>
```

Если нужно импортировать этот файл в локальное DTD, но подавить при этом объявление данного элемента, решение заключается в том, чтобы поместить нужное объявление *перед* объявлениями, которые нежелательны. Вот как следует это сделать:

```
<!ELEMENT polyhedron (side, side, side+, angle, angle, angle+)>
<!ENTITY % shapes "shapes.mod">
%shapes;
```

При этом предполагается, что элемент <polyhedron> все же будет использован, но как быть, если мы вообще не хотим, чтобы этот элемент

присутствовал в нашем DTD? Как можно блокировать импортированное объявление? Для этого познакомимся еще с одной синтаксической конструкцией, называемой условной секцией.

Условные секции

Условная секция (*conditional section*) является особой формой разметки, предназначенной в DTD для пометки области текста, которая должна включаться в DTD или исключаться из него.[1] Если предполагается, что часть DTD в какой-то момент может стать нежелательной, можно превратить ее в условную секцию и позволить конечному пользователю решать, сохранять ее или нет. Обратите внимание, что условные секции можно использовать только во внешних подмножествах, но не во внутренних.

Условные секции похожи на секции, помеченные как `CDATA`. В них функцию разделителей выполняют квадратные скобки, но вместо ключевого слова `CDATA` используется `INCLUDE` или `IGNORE`. Синтаксис имеет следующий вид:

```
<![switch [DTD text]]>
```

где *switch* похож на выключатель on/off и активирует *DTD text*, если его значение равно `INCLUDE`, или помечает его как недействительный, если он установлен в `IGNORE`. Например:

```
<![INCLUDE[
<!-- these declarations will be included -->
<!ELEMENT foo (bar, caz, bub?)>
<!ATTLIST foo crud CDATA #IMPLIED)>
]]>
<![IGNORE[
<!-- these declarations will be ignored -->
<!ELEMENT blah (#PCDATA)*>
<!ELEMENT glop (flub|zuc) 'zuc')>
]]>
```

Применение жестко кодированных литералов `INCLUDE` и `IGNORE` сопряжено с некоторыми неудобствами, поскольку для изменения состояния переключателя приходится вручную редактировать каждую условную секцию. Обычно переключатель представляет собой параметрическую сущность, которую можно определить в любом месте:

```
<!ENTITY % optional.stuff "INCLUDE">
<![%optional.stuff;[
<!-- these declarations may or may not be included -->
<!ELEMENT foo (bar, caz, bub?)>
```

[1] В SGML условные секции можно использовать как в документах, так и в DTD. XML ограничивает область их применения только DTD.

```
<!ATTLIST foo crud CDATA #IMPLIED)>
]]>
```

Поскольку значением параметрической сущности optional.stuff определено ключевое слово INCLUDE, объявления, входящие в условную секцию, будут использованы. Если бы optional.stuff была определена как IGNORE, то эти объявления игнорировались бы в документе.

Этот прием является особенно мощным, когда сущность объявлена в подмножестве документа. В следующем примере DTD объявляет общую сущность с именем disclaimer. Фактическое значение сущности зависит о того, установлено ли значение use-disclaimer равным INCLUDE:

```
<![%use-disclaimer;[[
  <!ENTITY disclaimer "<p>This is Beta software. We can't promise it
  is free of bugs.</p>">
]]>
<!ENTITY disclaimer "">
```

Для документов, в которые требуется включить отказ от ответственности (disclaimer), нужно просто объявить сущность переключателя во внутреннем подмножестве:

```
<?xml version="1.0"?>
<!DOCTYPE manual SYSTEM "manual.dtd" [
  <!ENTITY % use-disclaimer "IGNORE">
]>
<manual>
  <title>User Guide for Techno-Wuzzy</title>
  &disclaimer;
  ...
```

В этом примере сущность use-disclaimer устанавливается равной IGNORE, поэтому disclaimer объявляется как пустая строка, и текст документа не будет содержать отказ. Это простой пример настройки DTD с помощью условных секций и параметрических сущностей.

Условные секции могут быть вложенными, однако внешние секции подавляют те, которые находятся внутри. Поэтому если наружная секция установлена в IGNORE, то ее содержимое, в том числе любые внутренние секции, полностью отключается независимо от их значений. Например:

```
<![INCLUDE[
<!-- text in here will be included -->
  <![IGNORE[
  <!-- text in here will be ignored -->
  ]]>
]]>
<![IGNORE[
<!-- text in here will be ignored -->
  <![INCLUDE[
```

```
  <!-- Warning: this stuff will be ignored too! -->
  ]]>
]]>
```

Общедоступные DTD часто интенсивно используют условные секции, чтобы обеспечить максимальные возможности настройки. Например, DocBook XML DTD Version 1.0 включает в себя следующее:

```
<!ENTITY % screenshot.content.module "INCLUDE">
<![%screenshot.content.module;[
<!ENTITY % screenshot.module "INCLUDE">
<![%screenshot.module;[
<!ENTITY % local.screenshot.attrib "">
<!ENTITY % screenshot.role.attrib "%role.attrib;">
<!ELEMENT screenshot (screeninfo?, (graphic|graphicco))>
<!ATTLIST screenshot
               %common.attrib;
               %screenshot.role.attrib;
               %local.screenshot.attrib;
>
<!--end of screenshot.module-->]]>
<!ENTITY % screeninfo.module "INCLUDE">
<![%screeninfo.module;[
<!ENTITY % local.screeninfo.attrib "">
<!ENTITY % screeninfo.role.attrib "%role.attrib;">
<!ELEMENT screeninfo (%para.char.mix;)*>
<!ATTLIST screeninfo
               %common.attrib;
               %screeninfo.role.attrib;
               %local.screeninfo.attrib;
>
<!--end of screeninfo.module-->]]>
<!--end of screenshot.content.module-->]]>
```

Самая внешняя условная секция окружает объявления для `<screenshot>`, а также `<screeninfo>`, находящегося внутри нее. Можно полностью отключить `<screenshot>` и `<screeninfo>` путем установки значения `IGNORE` для `screenshot.content.module` в локальном DTD перед загрузкой файла. Другой возможностью является отключение только секции вокруг объявлений `<screeninfo>`, например, чтобы объявить собственный вариант `<screeninfo>`. (Отключение объявлений элемента в импортируемом файле позволяет избежать предупреждений анализатора о наличии избыточных объявлений.) Обратите внимание на параметрические сущности для присвоения различных типов содержимого и объявлений атрибутов, например, `common.attrib`. Есть также ловушки для вставки собственных атрибутов, например, `local.screenshot.attrib`.

Умелое применение условных секций может сделать DTD чрезвычайно гибким, хотя при этом его становится труднее читать. Следует экономно использовать их в своих личных DTD и стараться с самого начала проектировать их так, чтобы они удовлетворяли вашим пот-

ребностям. Позднее, если DTD станет общедоступным ресурсом, будет разумно добавить условные секции, позволяющие конечному пользователю осуществлять собственную настройку.

Использование внутреннего подмножества

Из главы 2 читатель может вспомнить, что внутреннее подмножество представляет собой часть документа XML, в которой могут содержаться объявления сущностей. На самом деле его возможности больше: любые объявления, возможные в DTD, можно поместить во внутреннее подмножество. Единственные ограничения касаются условных секций (их нельзя использовать) и параметрических сущностей (они могут содержать только законченные объявления, но не их фрагменты). Это удобно для переопределения или включения/выключения участков DTD. Вот общий формат:

```
<!DOCTYPE root-element URI [declarations ]>
```

Анализатор, читая DTD, сначала прочитывает внутреннее подмножество, а затем внешнее. (Помните о важности порядка объявлений, т. к. более ранние правила подавляют более поздние.) Здесь применимы те же правила, что и для DTD, импортирующих объявления. Например:

```
<!DOCTYPE inventory SYSTEM "InventoryReport.dtd" [

<!-- override DTD to include a "category" attribute -->
<!ATTLIST item category (screw | bolt | nut) #REQUIRED>

<!-- redefine the <price> element -->
<!ELEMENT price (currency, amount)>
<!ENTITY % price.module "IGNORE">

<!-- use a different module for figures -->
<!ENTITY % figs SYSTEM "myfigs.mod">

]>
```

Объявление списка атрибутов в этом внутреннем подмножестве добавляет атрибут `category` во множество атрибутов `<item>`. Объявление элемента переопределяет объявление DTD для `<price>`. Если анализатор сообщает о многократном объявлении одного и того же элемента, то импортируемое DTD, возможно, содержит условные секции вокруг каждого элемента, и можно отключить их, как в случае с DTD DocBook, о котором говорилось выше. В этом состоит назначение следующего далее объявления сущности `price.module`. Последнее объявление переопределяет внешнюю параметрическую сущность в DTD, которая импортирует модуль, вызывая вместо этого загрузку файла *myfigs.mod*.

Пример: Barebones DocBook

Настало время рассмотреть большое и более сложное приложение. Вдохновленные DocBook, языком разметки технической документации, который курирует OASIS Group (*http://www.oasis-open.org/docbook/index.html*), мы разработали базовую версию для упражнений. Пример документа Barebones DocBook рассмотрен в разделе «Приложение XML: DocBook» главы 2.

DTD в примере 5.3 помечено ссылками на список комментариев, помещенный в конце примера.

Пример 5.3. DTD Barebones DocBook

```
<!-- ================================================================== -->
<!--
     Barebones DocBook DTD Version 0.1
     Offered as a teaching tool, without any warranty whatsoever.
     Module dependencies:
       1. Cals Table (XML) Version 1.0 by Norman Walsh
       2. ISO-8879 character entities
     Contact: Erik Ray <eray@oreilly.com>
-->
<!-- ================================================================== -->
<!--                        ATTRIBUTE GROUPS                            -->
<!--            Common attributes for list declarations.                -->
<!-- ================================================================== -->
(1)
<!ENTITY % common.atts "
                              id        ID        #IMPLIED
                              role      CDATA     #IMPLIED
                              xml:space (default | preserve) 'default'
">
(2)
<!ENTITY % id.required.atts "
                              id        ID        #REQUIRED
                              renderas  NMTOKEN   #IMPLIED
                              role      CDATA     #IMPLIED
                              xml:space (default | preserve) 'default'
">
(3)
<!-- ================================================================== -->
<!--                        ELEMENT GROUPS                              -->
<!--               Sets of elements for content models.                 -->
<!-- ================================================================== -->
<!ENTITY % block.group "
                                blockquote
                              | figure
                              | note
                              | para
                              | programlisting
```

Пример 5.3. DTD Barebones DocBook (продолжение)

```
                                | table
">
<!ENTITY % chaplevel.group "
                                  appendix
                                | chapter
                                | preface
">
<!ENTITY % inline.group "
                                  acronym
                                | application
                                | citetitle
                                | command
                                | date
                                | emphasis
                                | filename
                                | firstterm
                                | quote
                                | sgmltag
                                | symbol
                                | systemitem
                                | xref
">
<!ENTITY % list.group "
                                  itemizedlist
                                | orderedlist
                                | variablelist
">
<!ENTITY % ubiq.group "
                                  indexterm
                                | graphic
                                | comment
">
❹
<!-- ====================================================================== -->
<!--                          CONTENT MODELS                                -->
<!--           Pre-fab content models for element declarations.             -->
<!-- ====================================================================== -->
<!ENTITY % component.title.content "
                                title,
                                subtitle?
">
<!ENTITY % component.content "
                                  %block.group;
                                | %list.group;
                                | %ubiq.group;
">
<!ENTITY % indexterm.content "
                                  #PCDATA
                                | %inline.group;
```

Пример 5.3. DTD Barebones DocBook (продолжение)

```
">
<!ENTITY % para.content "
                                  #PCDATA
                                | footnote
                                | %inline.group;
                                | %ubiq.group;
">
<!ENTITY % title.content "
                                  #PCDATA
                                | %inline.group;
">
❺
<!-- ================================================================== -->
<!--                      HIERARCHICAL ELEMENTS                         -->
<!-- ================================================================== -->
<!ELEMENT appendix (
                                %component.title.content;,
                                (
                                    %component.content;
                                  | sect1
                                )*
)>
<!ATTLIST appendix
                                %id.required.atts;
>
<!ELEMENT book (
                                title?, subtitle*, author?,
                                (
                                    %chaplevel.group;
                                )*
)>
<!ATTLIST book
                                %common.atts;
>
<!ELEMENT chapter (
                                %component.title.content;,
                                (
                                    %component.content;
                                  | sect1
                                )*
)>
<!ATTLIST chapter
                                %id.required.atts;
>
<!ELEMENT preface (
                                %component.title.content;,
                                (
                                    %component.content;
                                  | sect1
                                )*
```

Пример 5.3. DTD Barebones DocBook (продолжение)

```
)>
<!ATTLIST preface
                              %id.required.atts;
>
<!ELEMENT sect1 (
                              %component.title.content;,
                              (
                                  %component.content;
                                | sect2
                              )*
)>
<!ATTLIST sect1
                              %id.required.atts;
>
<!ELEMENT sect2 (
                              %component.title.content;,
                              (
                                  %component.content;
                                | sect3
                              )*
)>
<!ATTLIST sect2
                              %id.required.atts;
>
<!ELEMENT sect3 (
                              %component.title.content;,
                              (
                                  %component.content;
                              )*
)>
<!ATTLIST sect3
                              %id.required.atts;
>
❻
<!-- ================================================================== -->
<!--                        BLOCK ELEMENTS                              -->
<!-- ================================================================== -->
<!ELEMENT author
                              (#PCDATA)*
>
<!ELEMENT blockquote (
                              title?,
                              (
                                  para
                                | %ubiq.group;
                              )+
)>
<!ATTLIST blockquote
                              %common.atts;
```

Пример 5.3. DTD Barebones DocBook (продолжение)

```
>
<!ELEMENT comment
                                (%para.content;)*
>
<!ATTLIST comment
                                %common.atts;
>
<!ELEMENT figure (
                                title, graphic
)>
<!ATTLIST figure
                                %id.required.atts;
>
<!ELEMENT footnote (
                                para+
)>
<!ATTLIST footnote
                                %common.atts;
>
❼
<!NOTATION gif              SYSTEM "image/gif">
<!NOTATION eps              SYSTEM "image/eps">
<!ELEMENT graphic
                                EMPTY
>
<!ATTLIST graphic
                                fileref   CDATA     #REQUIRED
                                %common.atts;
>
<!ELEMENT note (
                                title?,
                                (
                                    %block.group;
                                  | %list.group;
                                  | %ubiq.group;
                                )*
)>
<!ATTLIST note
                                %common.atts;
>
<!ELEMENT para
                                (%para.content;)*
>
<!ATTLIST para
                                %common.atts;
>
❽
<!ELEMENT programlisting
                                (%para.content;)*
>
```

Пример 5.3. DTD Barebones DocBook (продолжение)

```
<!ATTLIST programlisting
                              xml:space (preserve) #FIXED 'preserve'
                              id        ID         #IMPLIED
                              role      CDATA      #IMPLIED
>
<!ELEMENT subtitle
                              (%title.content;)*
>
<!ATTLIST subtitle
                              %common.atts;
>
❾
<!-- Reference CALS table module. -->
<!ENTITY % calstbls PUBLIC
  "-//Norman Walsh//DTD CALS Table Model XML V1.0//EN"
  "calstblx.dtd">
%calstbls;

<!ELEMENT title
                              (%title.content;)*
>
<!ATTLIST title
                              %common.atts;
>
<!-- ================================================================== -->
<!--                          LIST ELEMENTS                             -->
<!-- ================================================================== -->
<!ELEMENT itemizedlist (
                              title?,
                              (
                                  %ubiq.group;
                                | listitem
                              )+
)>
<!ATTLIST itemizedlist
                              %common.atts;
>
<!ELEMENT listitem (
                              (
                                  %block.group;
                                | %list.group;
                                | %ubiq.group;
                              )+
)>
<!ATTLIST listitem
                              %common.atts;
>
<!ELEMENT orderedlist (
                              title?,
```

Пример 5.3. DTD Barebones DocBook (продолжение)

```
                                (
                                    %ubiq.group;
                                  | listitem
                                )+
)>
<!ATTLIST orderedlist
                                numeration (arabic|alpha|roman) 'arabic'
                                %common.atts;
>
<!ELEMENT term
                                (%para.content;)*
>
<!ATTLIST term
                                %common.atts;
>
<!ELEMENT variablelist (
                                title?,
                                (
                                    %ubiq.group;
                                  | varlistentry
                                )+
)>
<!ATTLIST variablelist
                                %common.atts;
>
<!ELEMENT varlistentry (
                                (%ubiq.group;)*,
                                term+,
                                (%ubiq.group;)*,
                                listitem,
                                (%ubiq.group;)*
)>
<!ATTLIST varlistentry
                                %common.atts;
>
⑩
<!-- ================================================================== -->
<!--                        INLINE ELEMENTS                             -->
<!-- ================================================================== -->
<!ELEMENT acronym           (%para.content;)*>
<!ATTLIST acronym
                                %common.atts;
>
<!ELEMENT application       (%para.content;)*>
<!ATTLIST application
                                %common.atts;
>
<!ELEMENT citetitle         (%para.content;)*>
<!ATTLIST citetitle
                                %common.atts;
```

Пример 5.3. DTD Barebones DocBook (продолжение)

```
>
<!ELEMENT command           (%para.content;)*>
<!ATTLIST command
                               %common.atts;
>
<!ELEMENT date              (%para.content;)*>
<!ATTLIST date
                               %common.atts;
>
<!ELEMENT emphasis          (%para.content;)*>
<!ATTLIST emphasis
                               %common.atts;
>
<!ELEMENT filename          (%para.content;)*>
<!ATTLIST filename
                               %common.atts;
>
<!ELEMENT firstterm         (%para.content;)*>
<!ATTLIST firstterm
                               %common.atts;
>
<!ELEMENT function          (%para.content;)*>
<!ATTLIST function
                               %common.atts;
>
<!ELEMENT quote             (%para.content;)*>
<!ATTLIST quote
                               %common.atts;
>
<!ELEMENT sgmltag           (%para.content;)*>
<!ATTLIST sgmltag
                               class     CDATA     #IMPLIED
                               %common.atts;
>
<!ELEMENT symbol            (%para.content;)*>
<!ATTLIST symbol
                               %common.atts;
>
<!ELEMENT systemitem        (%para.content;)*>
<!ATTLIST systemitem
                               role      (computer|url)   #REQUIRED
                               %common.atts;
>
(11)
<!ELEMENT xref              EMPTY>
<!ATTLIST xref
                               linkend   IDREF     #REQUIRED
                               format    CDATA     #IMPLIED
                               %common.atts;
```

Пример 5.3. DTD Barebones DocBook (продолжение)

```
>
⑫
<!-- ================================================================== -->
<!--                        INDEX ELEMENTS                              -->
<!-- ================================================================== -->
<!ELEMENT indexterm (
                              (
                                  primary
                                | secondary
                                | see
                                | seealso
                              )*
)>
<!ATTLIST indexterm
                              class ( singular | startofrange
                                      | endofrange ) "singular"
                              startref  CDATA     #IMPLIED
                              %common.atts;
>
<!ELEMENT primary
                              (%indexterm.content;)*
>
<!ATTLIST primary
                              sortas    CDATA     #IMPLIED
                              %common.atts;
>
<!ELEMENT secondary
                              (%indexterm.content;)*
>
<!ATTLIST secondary
                              sortas    CDATA     #IMPLIED
                              %common.atts;
>
<!ELEMENT see
                              (%indexterm.content;)*
>
<!ATTLIST see
                              sortas    CDATA     #IMPLIED
                              %common.atts;
>
<!ELEMENT seealso
                              (%indexterm.content;)*
>
<!ATTLIST seealso
                              sortas    CDATA     #IMPLIED
                              %common.atts;
>
⑬
<!-- ================================================================== -->
<!--                ISO-8879 Entity modules                             -->
```

Пример 5.3. DTD Barebones DocBook (продолжение)

```
<!-- ================================================================== -->
<!ENTITY % ISOamso
    PUBLIC "ISO 8879:1986//ENTITIES Added Math Symbols: Ordinary//EN//XML"
    "isoamso.ent"
>
<!ENTITY % ISOamsr
    PUBLIC "ISO 8879:1986//ENTITIES Added Math Symbols: Relations//EN//XML"
    "isoamsr.ent"
>
<!ENTITY % ISOdia
    PUBLIC "ISO 8879:1986//ENTITIES Diacritical Marks//EN//XML"
    "isodia.ent"
>
<!ENTITY % ISOgrk3
    PUBLIC "ISO 8879:1986//ENTITIES Greek Symbols//EN//XML"
    "isogrk3.ent"
>
<!ENTITY % ISOlat1
    PUBLIC "ISO 8879:1986//ENTITIES Added Latin 1//EN//XML"
    "isolat1.ent"
>
<!ENTITY % ISOlat2
    PUBLIC "ISO 8879:1986//ENTITIES Added Latin 2//EN//XML"
    "isolat2.ent"
>
<!ENTITY % ISOnum
    PUBLIC "ISO 8879:1986//ENTITIES Numeric and Special Graphic//EN//XML"
    "isonum.ent"
>
<!ENTITY % ISOpub
    PUBLIC "ISO 8879:1986//ENTITIES Publishing//EN//XML"
    "isopub.ent"
>
<!ENTITY % ISOtech
    PUBLIC "ISO 8879:1986//ENTITIES General Technical//EN//XML"
    "isotech.ent"
>
%ISOamso;
%ISOamsr;
%ISOgrk3;
%ISOlat1;
%ISOlat2;
%ISOnum;
%ISOpub;
%ISOtech;
```

Вот примечания к DTD Barebones DocBook:

1. Объявление параметрической сущности для хранения объявлений атрибутов, общих для большинства элементов. Атрибут общего

назначения role предназначен для создания разновидностей элементов. Для большинства элементов xml:space устанавливается в default, чтобы удалить лишние пробельные символы.

2. Еще одна параметрическая сущность с объявлениями часто используемых атрибутов. В данном случае атрибут id является необходимым. Элементы, на которые часто указывают перекрестные ссылки, такие как <chapter>, должны иметь обязательный атрибут ID.

3. Объявления параметрических сущностей для групп элементов. Сгруппированные по назначению, они позднее используются в моделях содержимого элементов.

4. Объявления параметрических сущностей для моделей содержимого элементов. Эти сущности значительно облегчают сопровождение и чтение объявлений элементов.

5. Элементы иерархии – это элементы, представляющие основные структурные составляющие книги, такие как главы, разделы, подразделы и т. д.

6. Блочные элементы содержат текст и разметку в сложных структурах – от абзацев до таблиц.

7. Маркером <graphic> служит для помещения в документ изображения. Атрибут fileref называет файл, из которого должно быть импортировано графическое изображение.

8. В <programlisting> содержится код компьютерной программы, поэтому в нем должны сохраниться все пробельные символы. Для этого объявление списка атрибутов устанавливает xml:space в preserve.

9. Объявления импортируются из другого DTD, модели таблиц CALS. В результате Barebones DocBook получает полную поддержку элементов <table>.

10. Внутритекстовые элементы не прерывают потока текста и наследуют атрибуты форматирования родительских блочных элементов.

11. Атрибут linkend маркера перекрестной ссылки <xref> указывает посредством IDREF на любой элемент с атрибутом ID. Атрибут format может использоваться для задания шаблона, определяющего способ представления перекрестной ссылки. Например, format="title" указывает, что если целевой элемент имеет заглавие, то оно здесь должно быть выведено.

12. Маркер <indexterm> определяет термин, который появится в предметном указателе. Программа построения предметного указателя просмотрит всю книгу и отметит позиции всех элементов <indexterm>. Эта информация о позициях может быть использована для вычисления номеров страниц или помещения якорей для гипертекстовых ссылок от указателя обратно к тексту.

13. Для специальных символов, которые могут потребоваться в документе, импортируются модули с объявлениями сущностей.

XML Schema: альтернатива использованию DTD

Кое-кто жаловался, что синтаксис DTD устарел и недостаточно гибок, из-за чего для некоторых задач DTD недостаточно выразительны. Другие находят странным, что в документах применяется один синтаксис, а в DTD – другой. Модели содержимого и объявления списков атрибутов тяжело читать и понимать, разочаровывает невозможность задания шаблонов данных в элементах и атрибутах.

По этим причинам предложен ряд альтернатив древним DTD. К их числу относится XML Schema (иногда называемая XSchema), и мы сейчас с ней познакомимся. Хотя она все еще имеет статус кандидата в рекомендации, будучи предложенной XML Schema Working Group в W3C, представленные здесь основы не должны серьезно измениться. В отличие от синтаксиса DTD, синтаксис XML Schema является корректным (well-formed) XML, что позволяет использовать при ее редактировании излюбленные инструменты XML. Она также предоставляет значительно больший контроль над типами данных и шаблонами, образуя более привлекательный язык для соблюдения строгих требований ввода данных.

Рассмотрим следующий пример – бланк для переписи населения. Счетчик, переходя от одной двери к другой, вводит данные в маленький электронный блокнот. Схема способствует поддержке организации данных путем контроля типов данных в случае, если что-то будет введено в неверное поле. Вот как может выглядеть в XML экземпляр документа.

```
<census date="1999-04-29">
  <censustaker>738</censustaker>
  <address>
    <number>510</number>
    <street>Yellowbrick Road</street>
    <city>Munchkinville</city>
    <province>Negbo</province>
  </address>
  <occupants>
    <occupant status="adult">
      <firstname>Floyd</firstname>
      <surname>Fleegle</surname>
      <age>61</age>
    </occupant>
    <occupant>
      <firstname>Phylis</firstname>
      <surname>Fleegle</surname>
      <age>52</age>
    </occupant>
    <occupant>
```

```
      <firstname>Filbert</firstname>
      <surname>Fleegle</surname>
      <age>22</age>
    </occupant>
  </occupants>
</census>
```

Теперь напишем для этой схемы такой код:

```
<xsd:schema xmlns:xsd="http://www.w3.org/1999/XMLSchema">

  <xsd:annotation>
    <xsd:documentation>
      Census form for the Republic of Oz
      Department of Paperwork, Emerald City
    </xsd:documentation>
  </xsd:annotation>

  <xsd:element name="census" type="CensusType"/>

  <xsd:complexType name="CensusType">
    <xsd:element name="censustaker" type="xsd:decimal" minoccurs="0"/>
    <xsd:element name="address" type="Address"/>
    <xsd:element name="occupants" type="Occupants"/>
    <xsd:attribute name="date" type="xsd:date"/>
  </xsd: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:attribute name="postalcode" type="PCode"/>
  </xsd:complexType>
  <xsd:simpleType name="PCode" base="xsd:string">
    <xsd:pattern value="[A-Z]-\d{3}"/>
  </xsd:simpleType>

  <xsd:complexType name="Occupants">
    <xsd:element name="occupant" minOccurs="1" maxOccurs="50">
     <xsd:complexType>
      <xsd:element name="firstname" type="xsd:string"/>
      <xsd:element name="surname" type="xsd:string"/>
      <xsd:element name="age">
       <xsd:simpleType base="xsd:positive-integer">
        <xsd:maxExclusive value="200"/>
       </xsd:simpleType>
      </xsd:element>
     </xsd:complexType>
    </xsd:element>
   </xsd:complexType>
</xsd:schema>
```

В первой строке документ идентифицируется как схема и связывается с пространством имен XML Schema. Для удобства в дальнейшем обсуждении мы будем отбрасывать префикс пространства имен `xsd:`. Следующая структура, `<annotation>`, служит местом для документирования назначения схемы и других деталей.

Объявление элементов и атрибутов

Следующим в нашем примере идет первое объявление типа элемента. Атрибут `name` присваивает родовой идентификатор, а атрибут `type` устанавливает *mun* элемента. Есть два типа элементов: простые и сложные. Объявление *простого* (*simple*) элемента не содержит атрибутов или элементов в содержимом. Поскольку данный конкретный элемент является корневым, он должен иметь иной тип – сложный. В данном случае сложному типу в действительности дается имя, `CensusType`, которое нам позднее пригодится для его описания. Хотя имена не являются обязательными, все же лучше использовать их для вашего же собственного блага.

Сложные и простые типы элементов

В следующей части схемы `CensusType` определяется как элемент `<complexType>`. В нем содержатся объявления еще трех элементов и объявление атрибута. Здесь не только объявляются три элемента: `<censustaker>` (счетчик), `<address>` (адрес) и `<occupants>` (жильцы), но и определяется модель содержимого для `CensusType`. Поэтому элемент `<census>` должен содержать все три элемента в указанном порядке и может иметь необязательный атрибут `date`. Это значительно отличается от стиля DTD, в котором модель содержимого состоит из строки внутри объявления элемента, а атрибуты объявляются отдельно в объявлении списка атрибутов.

Ограничения модели содержимого

Если последовательность отдельных элементов не предоставляет достаточной информации, в XML Schema есть другие возможности. Атрибуты `minOccurs` и `maxOccurs` устанавливают (минимальное и максимальное) число появлений чего-либо в документе. `minOccurs="0"` отменяет значение по умолчанию 1 и делает элемент необязательным. `maxOccurs="*"` снимает верхнюю границу, и элемент может появляться любое число раз.

Типы данных

В каждом объявлении элемента и атрибута есть атрибут `type`, как мы видели в объявлении первого элемента. Некоторые типы являются предопределенными в XML Schema, например, `string` и `decimal`. Тип

`string` представляет обычные символьные данные, как тип `CDATA` в языке DTD. Тип `decimal` представляет собой число. Ниже мы объявляем элемент `<age>` как имеющий тип `positive-integer` и ограничиваем сверху его величину значением 200.

Предопределенные типы данных

В DTD мы не могли этого сделать! Нет в DTD способа наложить на символьные данные шаблон ограничений, а в XML Schema таких способов несколько. В следующем списке перечислены еще некоторые предопределенные типы:

`byte, float, long`

Числовые форматы. Формат `byte` является любым 8-разрядным числом со знаком, а `long` - любым 32-разрядным числом со знаком. Тип `float` представляет число с плавающей точкой, например, 5.032E-6. Другие числовые величины представляют скорее абстракции, а не числа, например, INF (бесконечность), –INF (отрицательная бесконечность) и NaN (not a number – не число, категория, определенная IEEE для операций с плавающей точкой).

`time, date, timeinstant, timeduration`

Шаблоны для обозначения времени, даты и длительности.

`boolean`

Значение `true` или `false`. Допускаются также числовые эквиваленты: 0 или 1.

`binary`

Шаблон для двоичных чисел, например, `00101110`.

`language`

Код языка, скажем, `en-US`.

`uri-reference`

Шаблон для любого URI, например, *http://|www.donut.org/cruller.xml#ingredients*.

`ID, IDREF, IDREFS, NMTOKEN, NMTOKENS`

Типы атрибутов, действующие так же, как соответствующие типы в DTD.

Есть еще много других типов, что делает XSchema очень интересной для некоторых документов, особенно тех, которые связаны со специальными приложениями для обработки данных, например, базами данных и бланками ввода заказов. Не приходится писать программу, проверяющую типы данных, поскольку эту работу выполняет за нас анализатор XML.

Грани

Грани (*facets*) представляют собой свойства, используемые для задания типов данных, устанавливая пределы и ограничения на величину данных. Например, элементу `<age>`, тип данных которого `positive-integer`, было разрешено максимальное значение 200, называемое гранью `max-inclusive`. В XSchema есть еще 13 граней, в том числе `precision`, `scale`, `encoding`, `pattern`, `enumeration`, `max-length` (точность, масштаб, кодировка, шаблон, перечисление, максимальная длина) и другие.

Шаблоны

Объявление сложного типа `Address` знакомит нас с другим типом ограничения по шаблону. Он имеет атрибут `postalcode` с типом `PCode`, который определен посредством объявления `<pattern>`. Если среди предопределенных типов нет нужного шаблона, с помощью этого элемента можно создать свой собственный. Мы определили `PCode` строкой шаблона `[A-Z]-\d{3}`, которую нужно читать: «любой символ алфавита, за которым следует дефис и три цифры».

Более развитые возможности

Мы не станем более подробно вникать в то, как писать схемы, потому что стандарт еще не завершен. Однако упомянем о некоторых ожидаемых возможностях. Во-первых, с помощью типов можно делать значительно больше. Элементы, как и атрибуты, могут иметь перечислимые значения. Объявления можно группировать для наследования одинаковых свойств и обеспечения моделирования более сложного содержимого, а, кроме того, они могут наследовать свойства других объявлений (в объектно-ориентированном стиле).

XML Schema предоставляет интересную альтернативу DTD, позволяющую создателям документов проектировать поля, определяя их со значительно большей подробностью. Но это вовсе не замена DTD. Почему должен существовать только один способ описания структуры документа? DTD сохраняют за собой ряд преимуществ: компактный размер, знакомый синтаксис, простоту. Вместе они обеспечивают разные методы достижения сходных целей, и можно рассчитывать на скорое появление и других предложений.

6

- *Основы трансформаций*
- *Отбор узлов*
- *Тонкая настройка шаблонов*
- *Сортировка*
- *Пример: чековая книжка*
- *Более сложные приемы*
- *Пример: Barebones DocBook*

Трансформация: изменение назначения документов

Документ XML представляет собой хорошо структурированный контейнер информации, но все же кажется статичным. Неужели после всех хлопот, связанных с организацией данных и направленных на то, чтобы они составили корректный XML, (возможно) согласующийся с DTD, вы связались теперь с другим тупиковым форматом, который нельзя использовать где-то еще? Конечно, такой формат документа, как DocBook, полезен для структурного представления информации, но не оказывается ли автор документа вынужденным использовать только те программные приложения, которые специально созданы для работы с ним? А если нужно посмотреть документ в броузере HTML, который не поддерживает XML? Или вывести его как текст для печати? Оказываетесь ли вы в зависимости от милости разработчиков программного обеспечения, если требуются такие функции?

Не волнуйтесь, XML вас не подведет. На самом деле, для вас открываются новые двери. Ясное и недвусмысленное представление элементов текста позволяет легко изменить назначение документа. Другие форматы, такие как Microsoft Word, HTML и troff, в столь большой мере используют свойства представления, что в них трудно увидеть какую-либо структуру, и это делает почти невозможной какую-либо автоматическую настройку без нанесения ущерба содержимому. К счастью, XML дает прочную опору, и принесенные в угоду ему жертвы окупятся гибкостью формы и целостью данных.

Это самая захватывающая глава книги: в ней показано, что XML, с его строгими правилами и высокими начальными затратами, дает больше возможностей для использования данных документа в будущем, в том числе приложениями, выполняющими поиск, запросы и другие сложные операции над данными. *Трансформация* (*transformation*), т. е. процесс преобразования документа XML из одного формата в другой, является вершиной полезности XML. Это почти чудо: документ пропускается через трансформирующую программу, и на выходе получается нечто совсем отличное, при этом по пути не теряется ничего важного.

Если описывать в двух словах, то в этом процессе участвуют документ, таблица стилей преобразования и программное обеспечение, которое генерирует новый трансформированный документ. Таблицу стилей пользователь создает самостоятельно с помощью простого языка, который называется Extensible Style Language for Transformation (XSLT) – расширяемый язык стилей для преобразований. Как показано в данной главе, XSLT служит мощным способом записи инструкций по переоформлению информации из одного типа документа XML в другой.

Ниже перечислены некоторые причины, по которым может потребоваться преобразование документа XML в другой формат:

Хранить в одном формате, выводить в другом

Часто встречается такой сценарий: нужно опубликовать документ в Сети, но документ имеет слишком сложный для отображения тип. Каскадные таблицы стилей с задачей не справятся. Проще было бы преобразовать документ в HTML, в котором есть собственное неявное форматирование. Таблица стилей XSLT может превратить любой документ XML в XHTML.

Преобразовать в более удобный формат

Некто передает документ, но получателя не устраивает формат. Например, он пишет на стандартном правительственном бланке заявку на грант, но необходимые ему данные имеют другой формат документа. Таблица стилей XSLT позволяет преобразовать эти данные в тот формат, который ему нужен.

Сделать документ более компактным

Таблица стилей XSLT может взять очень большой файл, в котором масса ненужных данных, и урезать его до небольшого формата, содержащего только требуемые элементы.

Использовать документ как интерфейс для запроса к базам данных

Некоторые веб-серверы выполняют преобразования XSLT «на лету». Это удобный способ создания хорошо оформленных ответов на запросы к базам данных в виде корректных документов. Простой сценарий CGI создает набор расширенных ссылок XPointer, а таблица стилей извлекает данные и строит дерево результата. В итоге она выдает файлы в формате XHTML.

Такие задачи встречаются, но нельзя ли решать их с помощью программирования? Можно, но для многих типов трансформаций писать программы слишком сложно. XSLT специально предназначен для осуществления преобразований и ничего больше, что делает его легким в изучении, простым для чтения и оптимизированным для этой конкретной задачи.

Основы трансформаций

XSLT обозначает Extensible Stylesheet Language for Transformation, подмножество более общего языка таблиц стилей XSL. Сначала может показаться странным, что трансформация рассматривается как форма применения стиля, но после просмотра ее в действии все становится ясным. Как и в других языках таблиц стилей, задание трансформации является набором правил, соответствующих элементам. Каждое правило описывает, что должно быть выведено в зависимости от исходных данных. Разница лишь в том, что в качестве формата выходных данных XSLT использует XML.

Как и в предыдущих главах, документы XML будут представляться графически в виде дерева. Каждая часть структуры XML – элемент, атрибут, кусок текста или даже комментарий – будет представлена на схеме в виде прямоугольника, или *узла* (*node*). Если одна часть содержит другую, то из родительского узла в дочерний опускается линия.

Существует семь различных типов узлов:

Элемент

Узлы элементов и корневой узел (который будет вскоре описан) отличаются от остальных узлов особым свойством: только они могут содержать другие узлы. Узел элемента может содержать другие элементы, а также узлы любого другого типа, кроме корневого. Такой тип узла представляется на дереве как точка ветвления или, если у него нет атрибутов или содержимого, как лист.

Атрибут

Может показаться странным, но атрибут рассматривается как узел, а не часть своего элемента. Поскольку он представляет содержимое элемента, то рассматривается как отдельный узел. Узел атрибута называют *концевой вершиной*, или *листом* (*leaf node*), т. к. он является отдельной ветвью и не имеет потомков.

Текст

Текст также представляет собой концевую вершину. Он всегда является дочерним для некоторого элемента, но может не быть единственным его потомком. Текстовый узел это непрерывная строка символьных данных, не содержащая узлов. В элементе мо-

жет быть несколько текстовых узлов, разделяющихся элементами, инструкциями обработки или комментариями.

Комментарий

Комментарий является узлом, хотя формально ничего не вносит в содержимое документа. Он включается в качестве узла для полноты дерева документа и при необходимости может обрабатываться процессорами XML.

Инструкция обработки

Как и комментарий, инструкция обработки включается для полноты, несмотря на то, что имеет значение только для конкретного процессора XML.

Пространство имен

Объявление пространства имен не рассматривается как атрибут, поскольку оно оказывает особое воздействие на оставшуюся часть документа. Поэтому оно является отдельным типом узла.

Корень

Корневой узел содержит все, что есть в документе. Не путайте его с корневым элементом (элементом документа). Корневой узел – это абстрактная точка над элементом документа. Можно считать его родительским узлом для элемента документа, а также всего, находящегося за пределами элемента документа, кроме объявления типа документа (которое обычно вообще не рассматривают как узел или часть дерева документа).

В следующем документе есть все эти типы узлов, а на рис. 6.1 показано, как они выглядят в виде дерева.

```
<?xml version="1.0"?>
<!-- Dee-licious! -->
<sandwich xmlns="http://www.food.org/ns">
  <ingredient type="grape">jelly</ingredient>
  <ingredient><?knife spread thickly?>
    peanut butter</ingredient>
  <ingredient>bread
    <!-- white bread, preferably --></ingredient>
</sandwich>
```

Метафора дерева является центральной в понимании XSLT. Мы называем входной документ трансформации XSLT *исходным деревом* (*source tree*), а выходной – *результирующим деревом* (*result tree*). Так же, как можно отрезать веточку ивы и посадить новое дерево, можно «оборвать» ветки документа XML и образовать новые деревья, называемые *поддеревьями* (*subtrees*).

Рассмотрим, например, следующий документ XML:

```
<?xml version="1.0"?>
<manual type="assembly" id="model-rocket">
  <parts-list>
    <part label="A" count="1">fuselage, left half</part>
    <part label="B" count="1">fuselage, right half</part>
    <part label="F" count="4">steering fin</part>
    <part label="N" count="3">rocket nozzle</part>
    <part label="C" count="1">crew capsule</part>
  </parts-list>
<instructions>
    <step>
      Glue parts A and B together to form the fuselage.
    </step>
    <step>
      Apply glue to the steering fins (part F) and insert them into
      slots in the fuselage.
    </step>
    <step>
      Affix the rocket nozzles (part N) to the fuselage bottom with a
      small amount of glue.
    </step>
    <step>
      Connect the crew capsule to the top of the fuselage. Do not use
      any glue, as it is spring-loaded to detach from the fuselage.
    </step>
  </instructions>
</manual>
```

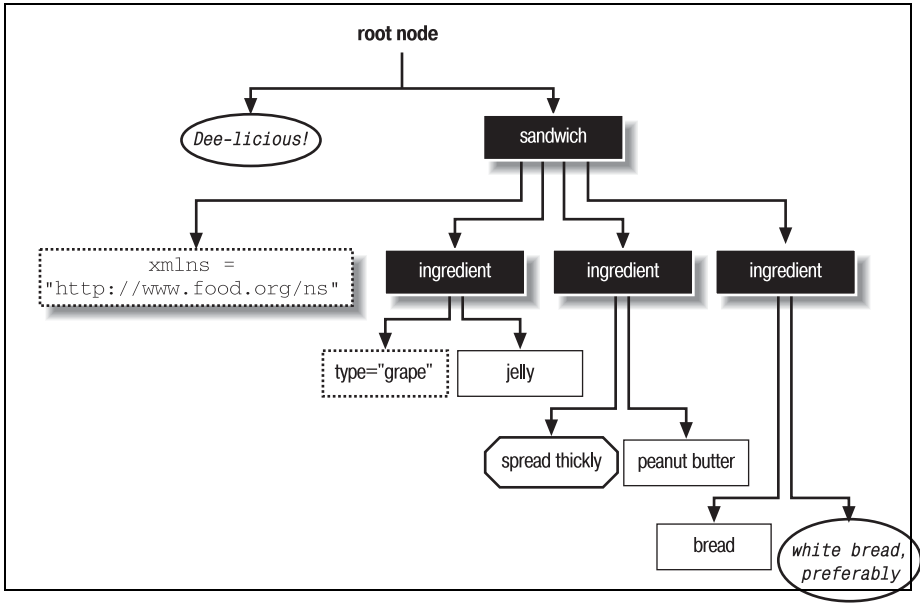


Рис. 6.1. *Представление в виде дерева, показывающее все типы узлов*

Весь документ является деревом, корнем которого служит элемент <manual>; элементы <parts-list> и <instructions> тоже имеют форму дерева, с собственными корнями и ветвями. XSLT использует этот принцип, чтобы разложить трансформацию на меньшие и легче управляемые части. Каждое правило трансформации нацелено на один уровень, не затрагивая остальной части дерева. В нем содержатся ссылки на другие правила, которые продолжают обработку далее вплоть до листьев.

Для этого примера можно написать таблицу стилей со следующими правилами:

- Обработать элемент <manual>. Поскольку это самый внешний элемент, нужно организовать структуры, необходимые для хранения остальной части результирующего дерева, например, оглавление или предметный указатель. Обработать ветви.
- Обработать элемент <parts-list>, создав заголовок и организовав список в первой части результирующего дерева. Перейти к дочерним элементам.
- Обработать элемент <instructions>, у которого другой заголовок и другой список. Обработать имеющиеся дочерние элементы.
- Вывести каждый элемент <part> в виде пункта маркированного списка, содержащего текстовые данные из исходного дерева.
- Вывести каждый элемент <step> в виде пункта нумерованного списка, содержащего текстовые данные из исходного дерева.

Каждое правило выполняет три задачи. Во-первых, оно *находит* (*matches*) узел, в данном случае, тип элемента. Во-вторых, оно определяет структуру результирующего поддерева. В-третьих, оно явно указывает, нужно ли обрабатывать ветви поддерева. С помощью этих компонентов каждое поддерево исходного документа может быть обработано от корня и до листьев, в результате чего создается каскад деревьев, при склеивании которых формируется результирующее дерево.

Что дает такая модель трансформации? Было бы проще использовать метод обработки CSS, в соответствии с которым имена элементов отображаются в стили и не требуется сообщать процессору о необходимости перемещения к потомкам для каждого элемента. Однако этот дополнительный шаг дает возможность лучшего контроля. На входе можно легко отобрать конкретные нужные узлы, а на выходе достигается большая гибкость в осуществлении нисходящей обработки, или *рекурсии* (*recursion*).

Наиболее важным является понятие *контейнеров* (*containments*), которые легко представляются поддеревьями. На практике, это одно из важнейших преимуществ разметки XML. Правило, которое находит элемент-контейнер, можно использовать, чтобы настроить структуру окружения для последующих дочерних элементов. Например, правило для контейнера может создать заголовок и организовать нумеро-

ванный список, поэтому дочерним элементам списка нужно только объявлять, что они являются пунктами списка. В плоском стиле отображения CSS сделать это значительно сложнее: пришлось бы определять, не является ли пункт списка первым, и заставлять его организовывать список для своих собратьев. По этим причинам поддеревья являются лучшим способом осуществления трансформаций.

Описание структуры с помощью шаблонов

В CSS стили присваиваются путем установки параметров в правилах. Это хорошо, если выходные данные соответствуют по структуре входному документу, но для XSLT этого недостаточно, т. к. результирующее дерево может иметь структуру, совершенно отличную от исходной. Легче всего представить структуру поддерева, просто записав последнее в том виде, в котором оно должно быть. Такая буквальная модель называется *шаблоном* (*template*), поэтому мы называем правила для трансформации *правилами шаблонов* (*template rules*).

Вот пример правила шаблонов:

```
<xsl:template match="/">
  <html>
    <head>
      <title>My first template rule</title>
    </head>
    <body>
      <h1>H'lo, world!</h1>
    </body>
  </html>
</xsl:template>
```

Его применение дает такой файл HTML:

```
<html>
<head>
<title>My first template rule</title>
</head>
<body>
<h1>H'lo, world!</h1>
</body>
</html>
```

Это правило является элементом XML с именем `<template>`, содержимое которого представляет собой элементы и данные, образующие результирующее поддерево. В данном случае на выходе получается законченный файл HTML. Этот пример не очень интересен, т. к. в нем не используются никакие данные или структуры исходного дерева. На практике можно применить это правило к любому документу, и результатом всегда будет один и тот же файл HTML. Тем не менее, это совершенно допустимое правило шаблона в XSLT.

Обратите внимание на атрибут `match` в элементе `<template>`, являющийся той частью правила, которая нацеливает на определенный уровень исходного дерева. Этот процесс называется *выбором* (*selection*). В данном случае этот атрибут выбирает корневой узел, абстрактную точку сразу над элементом документа. В этом месте начинается трансформация, и поэтому наше правило станет первым правилом, выполняемым в таблице стилей XSLT. Поскольку правило не разрешает продолжать обработку дальше корневого узла (нет ссылок на дочерние элементы этого узла), оно фактически блокирует все другие правила. Трансформация не только начинается с этого правила, но им и заканчивается.

Более полезное правило могло бы заключать в себе в качестве содержимого один или более специальных элементов `<apply-templates>` или `<value-of>`, которые передают обработку на другой уровень дерева. В таком случае другое правило построило бы еще некоторую часть результирующего дерева, осуществило рекурсию и т. д., пока процессор не достиг бы самого низкого уровня и не вернулся в вершину. Важным в шаблонах является, однако, то, что результирующее дерево придерживается правил самой таблицы стилей.

Таблица стилей как документ XML

Правило шаблона представляет собой элемент XML, и, в действительности, сама таблица стилей в целом это документ XML. Она должна быть, во-первых, корректной (well-formed), а, во-вторых, должна удовлетворять всем правилам XML. Минимальная таблица стилей, содержащая правило из нашего примера, должна выглядеть примерно так:

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

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

<xsl:template match="/">
  <html>
    <head>
      <title>My first template rule</title>
    </head>
    <body>
      <h1>H'lo, world!</h1>
    </body>
  </html>
</xsl:template>

</xsl:stylesheet>
```

Как и в любом документе XML, в начале помещается объявление XML. Затем следует элемент документа типа `<xsl:stylesheet>`, в кото-

ром содержатся все правила шаблонов. Он также указывает в объявлении пространства имен, что это таблица стилей XSLT, и устанавливает версию XSLT посредством `version="1.0"`.

В качестве элемента документа вместо `<xsl:stylesheet>` можно использовать `<xsl:transform>`: оба имени взаимозаменяемы. Ниже следует список атрибутов, допустимых в этом элементе:

`version`

Этот обязательный атрибут устанавливает номер версии XSLT. В настоящее время допускается только значение `1.0`.

`xmlns:xsl`

Здесь устанавливается пространство имен специфических для XSLT элементов. Можно рекомендовать вариант, расположенный по адресу *http://www.w3.org/1999/XSL/Transform/*.

`id`

Этот атрибут используется, когда нужно установить `ID`.

`extension-element-prefixes`

Этот атрибут устанавливает префикс для элементов, которые должны обрабатываться как специальные функции XSLT, даже если они не находятся в пространстве имен XSL. В механизмах XSLT этот атрибут предназначен для объявления собственных специальных функций. Например, *xt* Джеймса Кларка (James Clark) использует префикс `xt`. Применение этого атрибута требует также дополнительного атрибута объявления пространства имен.

`exclude-result-prefixes`

Этот атрибут устанавливает префиксы имен элементов, исключаемых из результирующего дерева, подобно тому, как исключаются из результирующего дерева элементы с префиксом `xsl:`. Применение этого атрибута требует также дополнительного атрибута объявления пространства имен.

При помощи пространства имен `xsl` процессор трансформаций определяет, какие элементы входят в инфраструктуру таблицы стилей, а какие должны быть помещены в результирующее дерево. Если полностью квалифицированное имя элемента начинается с префикса пространства имен `xsl:`, это «печать» трансформации. В противном случае элемент передается в выходной документ, как `<html>` и `<h1>` в предыдущем примере.

Важно заметить, что на элементы вне пространства имен `xsl` распространяются такие же правила XML, как и на специфические для трансформаций. То есть, они должны быть правильно построены, иначе пострадает весь документ. Попробуйте определить, почему недопустимым является следующее правило:

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

```
    <trifle>
      <piffle>
        <flim>
      </piffle>
    </trifle>
  </xsl:template>
```

Ответ: элемент <flim> не имеет закрывающего тега или использует неправильный синтаксис пустого элемента (<flim/>).

С таблицами стилей XSLT связан один интересный нюанс – их нельзя проверять с помощью DTD. Да, словарь XSLT ограничен, но правила шаблонов могут содержать любые элементы, атрибуты или данные. Более того, т. к. на поддеревья могут распространяться несколько правил, то нет возможности проверить содержимое каждого элемента, не произведя полную трансформацию. Поэтому DTD бесполезны для таблиц стилей XSLT. Это может привести к созданию в результате трансформации недействительных результирующих деревьев. Если редактировать таблицу стилей с помощью программы, требующей действительных документов, возникает проблема. Есть несколько редакторов, которым для работы достаточно иметь корректный (well-formed) документ. Однако если писать таблицу стилей с умом, отводить время на отладку и использовать редактор, не жалующийся на отсутствие DTD, все должно быть нормально.

Применение таблиц стилей XSLT

Существует несколько стратегий выполнения трансформации, выбор одной из которых зависит от существующих потребностей. Если трансформированный документ предназначен для личного пользования, можно запустить такую программу, как *xt*, чтобы осуществить трансформацию на локальной машине. Для веб-документов трансформация производится либо на сервере, либо на стороне клиента. Некоторые веб-серверы способны обнаруживать объявление таблицы стилей и преобразовывать документ перед выводом. Другой возможностью является пересылка исходного документа клиенту, который и выполняет преобразование. Internet Explorer 5.0 был первым броузером, реализовавшим XSLT и «открывшим дверь» такого рода процедуре. Выбор того или иного метода зависит от разных факторов, в том числе от того, как часто меняются данные, какую нагрузку может выдержать сервер и есть ли выгода от предоставления пользователям исходных файлов XML.

Если сервер или клиент будут выполнять трансформацию, нужно включить в документ ссылку на таблицу стилей в виде инструкции обработки, подобно тому, как это делается для связывания документов с таблицами стилей CSS (см. главу 4 «Представление: создание конечного продукта»). Она должна иметь следующий вид:

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

Атрибут `type` является типом MIME. Значения `text/xml` должно быть достаточно, хотя в будущем оно может измениться на что-либо другое, например, `text/xslt`. Атрибут `href` указывает на местонахождение таблицы стилей.

XSLT – довольно новая технология, поэтому некоторые реализации трансформирующего программного обеспечения не полны или отстают от официальной рекомендации W3C. Результаты могут различаться в зависимости от используемого инструментария. Например, Internet Explorer 5.0 требует иного пространства имен, нежели рекомендованное в технических условиях XSLT. Положение должно улучшаться по мере того, как большее число производителей будет реализовывать XSLT, а стандарт будет становиться более зрелым. Пока этого не произошло, читайте документацию к применяемому средству, чтобы выяснить, какие в нем есть специфические особенности.

Законченный пример

Теперь соединим все вместе и посмотрим на трансформацию в действии. В примере 6.1 показана законченная таблица стилей, содержащая четыре правила. Первое правило соответствует любому элементу `<quotelist>`. Оно помещает в результирующее дерево самые внешние элементы, контейнеры `<html>` и `<body>`. Обратите внимание на новый элемент `<xsl:apply-templates>` – особую инструкцию XSLT, в соответствии с которой трансформация распространяется на дочерние элементы `<quotelist>`. Это пример ранее упоминавшегося процесса рекурсии. Следующее правило соответствует элементам `<quote>` и заключает их содержимое внутрь тегов `<blockquote>`. Последние два правила создают элементы `<p>` и вставляют в них содержимое соответствующих элементов.

Пример 6.1. Таблица стилей XSLT

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

Пример 6.1. Таблица стилей XSLT (продолжение)

```
                    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="quotelist">
  <html>
    <body>
      <h1>Quotes</h1>
      <xsl:apply-templates/>
    </body>
  </html>
</xsl:template>

<xsl:template match="quote | aphorism">
  <blockquote>
    <xsl:apply-templates/>
  </blockquote>
</xsl:template>

<xsl:template match="body">
  <p><xsl:apply-templates/></p>
</xsl:template>

<xsl:template match="source">
  <p align="right"><xsl:apply-templates/></p>
</xsl:template>

</xsl:stylesheet>
```

В качестве входных данных для трансформации требуется исходное дерево. Пример 6.2 представляет собой файл XML, в котором есть список цитат и содержится более четырех типов элементов. В нашей таблице стилей нет правил для элементов `<speaker>`, `<forum>` и `<date>`. Что произойдет с ними в результате трансформации?

Пример 6.2. Файл XML

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

<quotelist>

  <quote id="1">
    <body>
      Drinking coffee could protect people from
      radioactivity, according to scientists in
      India who have found that mice given caffeine
      survive otherwise lethal doses of radiation.
    </body>
    <source type="publication">
      <forum>The New Scientist</forum>
      <date>6/99</date>
    </source>
```

Пример 6.2. Файл XML (продолжение)

```
</quote>

<category type="humor">
  <category type="twisted">

    <quote type="humor" id="2">
      <comment>
        Find out which episode.
      </comment>
      <body>
        Trying is the first step before failure.
      </body>
      <source type="tv-show">

        <speaker>Homer</speaker>
        <forum>The Simpsons</forum>
      </source>
    </quote>

    <aphorism type="humor" id="3">
      <body>
        Hard work has a future payoff.  Laziness pays off now.
      </body>
    </aphorism>

    <?quote-muncher xyz-987?></category>
  <category type="weird">

    <quote type="humor" id="4" friend="yes">
      <body>
        I keep having these fantasies where the
        Dead Sea Scrolls are full of assembly code.
      </body>
      <source>Greg Travis</source>
    </quote>

  </category>
</category>
<category type="philosophy">

  <aphorism id="5">
    <body>
      The tongue is the only weapon that becomes sharper with
      constant use.
    </body>
  </aphorism>

  <quote id="6">
    <body>
      The superior person understands what is
```

Пример 6.2. Файл XML (продолжение)

```
        right; the inferior person knows what will sell.
      </body>
      <source>Confucius</source>
    </quote>

  </category>

</quotelist>
```

В результате выполнения трансформации будут получены данные, показанные в примере 6.3. К счастью, данные отсутствующих в правилах элементов не были потеряны. На практике, трансформация сохранила все пробельные символы внутри элементов исходного дерева и передала их в результирующее дерево в целости. Данные `Homer` и `The Simpsons` разделяются символом перевода строки так же, как были разделены содержащие их элементы в исходном дереве.

Пример 6.3. Результат трансформации

```
<html>
<body>
<h1>Quotes</h1>

  <blockquote>
    <p>
      Drinking coffee could protect people from
      radioactivity, according to scientists in
      India who have found that mice given caffeine
      survive otherwise lethal doses of radiation.
    </p>
    <p align="right">
      The New Scientist
      6/99
    </p>
  </blockquote>

  <blockquote>
   Find out which episode.
    <p>
      Trying is the first step before failure.
    </p>
    <p align="right">
      Homer
      The Simpsons
    </p>
  </blockquote>

  <blockquote>
    <p>
      Hard work has a future payoff. Laziness pays off now.
```

Пример 6.3. Результат трансформации (продолжение)

```
    </p>
  </blockquote>

  <blockquote>
    <p>
      I keep having these fantasies where the
      Dead Sea Scrolls are full of assembly code.
    </p>
    <p align="right">
      Greg Travis
    </p>
  </blockquote>

  <blockquote>
    <p>
      The tongue is the only weapon that becomes sharper with constant use.
    </p>
  </blockquote>

  <blockquote>
    <p>
      The superior person understands what is
      right; the inferior person knows what will sell.
    </p>
    <p align="right">
      Confucius
    </p>
  </blockquote>

</body>
</html>
```

В этой таблице стилей трансформации содержится большинство важных понятий, представленных в данной главе. Однако это упрощенный пример, и практические требования могут выйти далеко за его пределы. В последующих разделах будет показано, как можно с хирургической точностью находить и связывать вместе части документа, используя язык XPath. Показаны будут и другие «магические трюки», такие как сортировка, управление выдачей, слияние таблиц стилей и прочие.

Отбор узлов

Чтобы делать в XSLT что-либо сложное, нужно перемещаться по документу так же ловко, как прыгает по деревьям мартышка. В любой момент требуется точно знать, где мы находимся и куда пойдем дальше. Должна также иметься возможность с предельной точностью отбирать

группы узлов для обработки. Такие возможности навигации предоставляет XPath[1] – сложный язык для пометки точек и отбора групп узлов в документе.

Пути адресации

Местоположение является важным понятием в навигации XML. В XSLT часто требуется описать местоположение узла или группы узлов где-либо в документе. В XPath это описание называется *путем адресации* (*location path*). Хороший пример его дает атрибут `match` элемента `<xsl:template>`, указывающий путь к группе узлов, которую должно обрабатывать правило. Хотя рассматривавшиеся примеры были простыми, пути местоположения могут оказаться весьма сложными.

Есть два вида путей местоположения: абсолютные и относительные. *Абсолютные пути* (*absolute paths*) начинаются в фиксированной точке отсчета, а именно, в корневом узле. Напротив, *относительные пути* (*relative paths*) начинаются в переменных точках, которые называются *узлами контекста* (context node).

Путь адресации состоит из ряда *шагов* (*steps*), каждый из которых уводит дальше от начальной точки. Сам шаг образуют три части: *ось* (*axis*), которая описывает направление перемещения, *критерий узла* (*node test*), который указывает, какие типы узлов являются пригодными, и набор необязательных *предикатов* (*predicates*), которые используют булевы (истина/ложь) проверки для дополнительного отсеивания кандидатов. В табл. 6.1 перечислены типы осей узлов.

Таблица 6.1. Оси узлов

Тип оси	Чему соответствует
Ancestor	Все узлы, находящиеся выше узла контекста, в том числе родительский, родительский для родителя и т.д. вплоть до корневого узла.
Ancestor-or-self	То же, что выше, но включает узел контекста.
Attribute	Атрибуты узла контекста.
Child	Дочерние элементы узла контекста.
Descendant	Дочерние элементы узла контекста, плюс их дочерние элементы и т. д. вплоть до листьев поддерева.
Descendant-or-self	То же, что выше, но включает узел контекста.

[1] Язык XPath является рекомендацией W3C. Версия 1.0 была ратифицирована в ноябре 1999 года.

Таблица 6.1 (продолжение)

Тип оси	Чему соответствует
Following	Узлы, следующие за узлом контекста на любом уровне документа.
Following-sibling	Узлы, следующие за узлом контекста на том же самом уровне (т. е. имеют того же родителя, что и узел контекста).
Namespace	Все узлы некоторого пространства имен.
Parent	Родительский узел для узла контекста.
Preceding	Узлы, находящиеся перед узлом контекста на любом уровне документа.
Preceding-sibling	Узлы, распложенные перед узлом контекста на том же самом уровне (т. е. имеют того же родителя, что и узел контекста).
Self	Сам узел контекста.

За осью следует параметр критерия, который отделяется от нее двумя двоеточиями (`::`). В некоторых случаях вместо явного типа узла может использоваться имя, и тогда тип узла логически определяется по оси. Для оси атрибутов предполагается, что тип узла – атрибут, а для оси пространств имен предполагается, что тип узла – пространство имен. Для всех остальных осей предполагается, что узел представляет собой элемент. При отсутствии указания типа оси предполагается, что им является `child`, а узел предполагается имеющим тип элемента. В табл. 6.2 перечислены критерии узлов.

Таблица 6.2. Критерии узлов

Термин	Чему соответствует
`/`	Корневой узел: не корневой элемент, но узел, содержащий корневой элемент и предшествующие ему комментарии или инструкции обработки, если они есть.
`node()`	Любой узел, кроме корневого и атрибутов.
`*`	По оси атрибутов – любой атрибут. По оси пространств имен – любое пространство имен. По всем другим осям – любой элемент.

Таблица 6.2 (продолжение)

Термин	Чему соответствует
`rangoon`	По оси атрибутов – атрибут узла контекста, в данном примере `rangoon`. По оси пространств имен – это пространство имен с именем `rangoon`. По всем остальным осям – любой элемент типа `<rangoon>`.
`text()`	Любой текстовый узел.
`processing-instruction()`	Любая инструкция обработки.
`processing-instruction('.Ng 4')`	Инструкция обработки `.Ng 4`.
`comment()`	Любой узел комментария.
`@*`	Любой атрибут. (Термин `@` представляет собой сокращение, замещающее неявно предполагаемый тип узла `element` в отсутствие задания оси.) Это эквивалентно `attribute::*`.
`@role`	Атрибут с именем `role`.
`.`	Узел контекста (иными словами, все что угодно). Это эквивалентно `self::*`.

Соединяются ось и критерий узла просто. Рассмотрим некоторые примеры, основываясь на документе примера 6.2. Предположим, что узлом контекста является элемент `<quote>` с атрибутом `id="2"`. Результат некоторых путей адресации представлен в табл. 6.3.

Таблица 6.3. Примеры путей адресации

Путь	Чему соответствует
`child::node()`	Соответствует трем узлам: комментарию «Find out which episode» и двум элементам, `<body>` и `<source>`. Поскольку осью по умолчанию является `child`, можно опустить спецификатор оси и написать `node()`.
`child::*`	Соответствует только трем узлам: элементам `<comment>`, `<body>` и `<source>`. Опять-таки, можно опустить спецификатор оси и написать `*`.
`parent::*`	Только один узел может быть родительским, поэтому соответствует одному элементу `<category>`.
`parent::quotelist`	Ничему не соответствует, потому что родителем узла контекста не является `<quotelist>`.

Таблица 6.3 (продолжение)

Путь	Чему соответствует
`parent::quotelist`	Ничему не соответствует, потому что родителем узла контекста не является `<quotelist>`.
`ancestor-or-self::/`	Соответствует корневому узлу, в каком бы месте документа мы ни находились, потому что корневой узел (/) является предком всего остального.
`ancestor-or-self::quote`	Это соответствует только узлу контекста, который удовлетворяет как части `self` из `ancestor-or-self`, так и критерию узла `quote`.
`self::quote`	По той же причине это соответствует узлу контекста. Ось `self` полезна для определения типа узла контекста.
`child::comment()`	Это соответствует узлу комментария со значением «Find out which episode».
`preceding-sibling::*`	Это ничему не соответствует, потому что узел контекста является первым дочерним для своего родителя.
`following-sibling::node()`	Соответствует двум узлам: элементу `<aphorism>` и инструкции обработки со значением `xyz-987`.
`following::quote`	Соответствует двум элементам `<quote>` с `id="4"` и `id="6"`.

Если оси и типа узла недостаточно для сужения отбора, можно использовать один или более предикатов. Предикат – это булево выражение, заключенное в квадратные скобки ([ ]). Каждый узел, который проходит эту проверку (в дополнение к критерию узла и указателю оси), включается в окончательное множество узлов. Узлы, не прошедшие проверку (предикат имеет значение `false`), не включаются. В табл. 6.4 приведены некоторые примеры.

Таблица 6.4. Примеры предикатов

Путь адресации	Чему соответствует
`child::product[child::color]`	Соответствует каждому `<product>`, являющемуся дочерним для узла контекста, имеющего дочерний элемент типа `<color>`. `child::color` является путем адресации, который после вычисления становится множеством узлов. Если множество пустое, результатом предиката будет `false`. В противном случае, это будет `true`.

Таблица 6.4 (продолжение)

Путь адресации	Чему соответствует
`product[@price="4.99"]`	Соответствует каждому узлу `<product>`, который является дочерним для узла контекста и атрибут `price` которого равен строке `4.99`. `@price` служит сокращением для `attribute::price`.
`child::*[position()!=last()]`	Соответствует всем дочерним узлам узла контекста, имеющим тип «элемент», кроме последнего. Функция `position()` возвращает число, представляющее местонахождение узла контекста в множестве узлов контекста. Функция `last()` возвращает позицию последнего узла в множестве узлов контекста.
`preceding::node()[1]`	Соответствует узлу непосредственно перед узлом контекста. Число в скобках эквивалентно `[position()=1]`.
`parent::rock[@luster='sparkle' | @luster='gloss']`	Соответствует родителю узла контекста, если это элемент типа `<rock>`, имеющий атрибут `luster`, значением которого является либо `sparkle`, либо `gloss`.

Шаги в пути адресации связываются оператором косой черты (/). Каждый шаг сужает или наращивает множество узлов, как инструкции, которые ведут к месту проведения вечеринки («Пойдите на Дейвис-сквер, затем в сторону Колледж-авеню; у кольцевой дорожной развязки...»). Синтаксис может оказаться многословным; некоторые сокращения приведены в табл. 6.5.

Таблица 6.5. Сокращения для путей адресации

Схема	Чему соответствует
`/*`	Соответствует элементу документа. Любой путь адресации, начинающийся с косой черты (/), является абсолютным путем, и первый шаг представляет корневой узел. Следующий шаг `*`, который соответствует любому элементу.
`parent::*/following-sibling::para`	Соответствует всем элементам `<para>`, которые следуют за родителем узла контекста.
`..`	Соответствует родительскому узлу. Удвоенная точка (`..`) является сокращением для `parent::node()`.

Таблица 6.5 (продолжение)

Схема	Чему соответствует
`.//para`	Соответствует любому элементу типа <para>, который является потомком текущего узла. Удвоенная косая черта (//) является сокращением для `/descendant-or-self::*//`.
`//para`	Соответствует любому элементу <para>, являющемуся потомком корневого узла. Иными словами, он соответствует всем элементам <para> в любом месте документа. Предполагается, что путь адресации, начинающийся с двойной косой черты (//),начинается в корневом узле.
`//chapter[1]/section[1]/para[1]`	Соответствует первому элементу <para> внутри первого элемента <section> внутри первого элемента <chapter> в документе.
`../*`	Соответствует всем узлам того же уровня плюс узлу контекста. Чтобы исключить узел контекста, используйте `preceding-sibling::* | following-sibling::*`.

Шаблоны соответствия

Пути адресации чаще всего используются в атрибутах `match` элементов <xsl:template> (мы будем называть их шаблонами соответствия). Однако они действуют несколько иначе, чем в только что описанном общем случае.

Во-первых, могут использоваться только нисходящие или ссылающиеся на себя оси. Процессор эффективнее всего действует в нисходящем направлении, начиная с корневого узла и заканчивая листьями. Такие оси, как `parent` и `preceding`, слишком осложняют дело и могут приводить к бесконечным циклам.

Второе отличие состоит в том, что шаблоны вычисляются фактически справа налево, а не в обратном направлении, которое подразумевалось ранее. Это более естественное соответствие стилю обработки XSLT. Когда процессор перемещается по исходному дереву, он ведет текущий список узлов, которые должны быть обработаны следующими, называемый *множеством узлов контекста* (*context node set*). Все узлы этого множества обрабатываются по очереди. Процессор заглядывает в список правил в таблице стилей, находит те, которые применимы к обрабатываемому узлу, и из этого множества выбирает правило с

наилучшим соответствием. Критерий выбора правил основан на соответствии шаблону.

Предположим, что есть правило с шаблоном соответствия `chapter/section/para`. Чтобы проверить этот шаблон, процессор сначала устанавливает узел, который нужно обработать как узел контекста. Затем он задает следующие вопросы в указанном порядке:

1. Является ли узел контекста элементом типа `<para>`?
2. Является ли родитель этого узла элементом типа `<section>`?
3. Является ли дедушка этого узла элементом типа `<chapter>`?

Логически это не сильно отличается от путей адресации, рассмотренных выше. Нужно только изменить представление о том, откуда начинается путь. Более разумным может оказаться такое изменение этого шаблона поиска:

```
abstract-node/child::chapter/child::section/child::para
```

где *abstract-node* является некоторым узлом, путь адресации из которого соответствует группе узлов, включающей в себя узел, который нужно обработать.

Теперь рассмотрим некоторые практические примеры, взяв документ из примера 6.2 – набор цитат и афоризмов. Для выбора некоторого типа элемента он просто используется в качестве шаблона поиска. Рассмотрим, например, следующие правила:

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

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

Применяя эти правила, мы получим следующее результирующее дерево:

```
<blockquote>
  <p>
    Hard work has a future payoff.  Laziness pays off now.
  </p>
</blockquote>

<blockquote>
  <p>
    The tongue is the only weapon that becomes sharper
    with constant use.
  </p>
</blockquote>
```

Другая возможность заключается в поиске атрибута. Следующее правило действует с любым атрибутом `friend` и добавляет специальное сообщение:

```
<xsl:template match="@friend">
  <font size="-1">A good friend of mine</font>
</xsl:template>
```

Оператор | может осуществлять поиск нескольких типов элементов:

```
<xsl:template match="quote | aphorism">
  <blockquote>
    <xsl:apply-templates/>
  </blockquote>
</xsl:template>
```

С помощью операторов иерархии / и // можно выполнять более сложный поиск. Следующее правило находит любой `<quote>`, являющийся дочерним для элемента документа и потому не находящийся в `<category>`:

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

Противоположным является поиск только тех элементов `<quote>`, которые расположились внутри `<category>`:

```
<xsl:template match="category/quote">
  <blockquote>
    <xsl:apply-templates/>
  </blockquote>
</xsl:template>
```

Или тех, которые находятся только на первом уровне категории:

```
<xsl:template match="/*/category/quote">
  <blockquote>
    <xsl:apply-templates/>
  </blockquote>
</xsl:template>
```

Еще одним способом сузить область соответствия шаблона служат критерии узла. Оба следующие правила соответствуют элементу `<quote>`, идентификатор `id` которого равен `4`:

```
<xsl:template match="quote[@id='4']">
  <blockquote>
    <xsl:apply-templates/>
  </blockquote>
</xsl:template>
```

```
<xsl:template match="quote[id('4')]">
  <blockquote>
    <xsl:apply-templates/>
  </blockquote>
</xsl:template>
```

Можно также проверять иерархию. Следующее правило соответствует всем элементам <quote>, источником которых является «Greg Travis»:

```
<xsl:template match="quote[source/text('Greg Travis')]">
  <blockquote>
    <xsl:apply-templates/>
  </blockquote>
</xsl:template>
```

Для достижения специфического результата можно объединять правила. Следующие два правила, действующие совместно, ограничивают действие таблицы стилей только теми элементами <category>, которые имеют тип type="humor":

```
<xsl:template match="category[@type='humor']">
  <blockquote>
    <xsl:apply-templates/>
  </blockquote>
</xsl:template>

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

Как будет показано ниже при обсуждении путей разрешения конфликтов, второе правило срабатывает только тогда, когда первое не находит соответствия. Если же оно находит элемент, то предотвращает обработку элемента или его содержимого, потому что это правило является пустым.

Разрешение конфликтов между правилами

Возможны случаи, когда некоторому узлу соответствуют несколько правил. В такой ситуации процессор XSLT должен выбрать из них только одно, и это правило должно оправдать надежды на самое лучшее соответствие. Вот правила старшинства среди совпавших с узлом шаблонов:

1. Если шаблон содержит несколько альтернатив, разделенных вертикальной чертой (|), то все альтернативы считаются имеющими одинаковую важность, как если бы для каждой существовало отдельное правило.
2. Шаблон, содержащий конкретные данные об иерархии, имеет более высокий приоритет, чем шаблон, содержащий только общие

данные. Например, шаблон `chapter/section/para` более конкретен, чем `para`, и имеет над ним приоритет.

3. Шаблон, использующий символы подстановки (wildcards), является более общим и потому имеет более низкий приоритет, чем шаблон с конкретными данными. Шаблон `stuff/cruft` сильнее, чем содержащий символ подстановки `stuff/*`.

4. Шаблон, содержащий в квадратных скобках (`[ ]`) выполненное проверочное выражение, перевешивает шаблон без проверочного выражения, но идентичный в остальном. Поэтому `bobo[@role="clown"]` лучше, чем `bobo`.

5. Если все же остается более одного правила, то для сокращения их числа может использоваться и другая информация, например, положение в таблице стилей.

Основной принцип состоит в том, что более специфические по применению правила берут верх над более общими. В противном случае было бы невозможно писать правила, перехватывающие все случаи, и задавать действия по умолчанию. Положение и порядок правил вступают в игру только тогда, когда все другие средства отбора не действуют. Способ окончательного разрешения конфликта определяет процессор трансформации.

У элемента `<xsl:template>` есть необязательный атрибут `priority`, который можно задавать для установки приоритета над другими правилами и вмешательства в процесс принятия решения. Его значение должно быть действительным числом (т. е. содержать десятичную точку) и находиться в диапазоне между 1,0 и −1,0, со значением по умолчанию равным 0. Большее число имеет приоритет над меньшим.

Правила, действующие по умолчанию

В XSLT определена группа правил, действующих по умолчанию, что облегчает написание таблиц стилей. Если не найдено ни одного правила из таблицы стилей, правила по умолчанию обеспечивают срабатывание аварийной системы. Действие их, в основном, состоит в переносе всех текстовых данных из элементов и атрибутов исходного дерева в результирующее дерево и в допущении, что неявный элемент `<xsl:apply-templates>` разрешает рекурсивную обработку. В следующий список сведены правила по умолчанию для каждого из типов узлов:

Корневой

Обработка начинается с корневого узла. Чтобы вызвать обработку всего дерева, режим по умолчанию применяет шаблоны ко всем дочерним узлам. Правило выглядит так:

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

Элемент

Желательно, чтобы процессор прошелся по всем элементам дерева и не пропустил ветвей, для которых правила определены. Это правило аналогично правилу для корневого узла:

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

Атрибут

В результирующее дерево должно быть включено значение каждого атрибута, поэтому используется следующее правило:

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

Текст

Было бы неудобно включать элемент `<xsl:value-of>` в каждый шаблон, чтобы выводить текст. Поскольку почти всегда нужно, чтобы текстовые данные выводились, это осуществляется по умолчанию:

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

Инструкция обработки

По умолчанию эти узлы пропускаются. Правило следующее:

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

Комментарий

Комментарии тоже по умолчанию опускаются в результирующем дереве:

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

Выражения XPath

Внутри правила может использоваться более сложный механизм XPath, называемый *выражением* (*expression*). Выражения – это команды, способные извлекать из узлов исходного дерева полезную информацию любого вида и использовать ее при выводе. Путь адресации является подмножеством этого богатого языка. Есть пять типов выражений:

Булево

Выражение, которое может принимать одно из двух значений, `true` и `false`.

Множество узлов

Совокупность узлов, удовлетворяющих критерию выражения, обычно получаемых с помощью пути адресации.

Число

Численное значение, используемое для подсчета узлов и выполнения простых арифметических действий.

Строка

Фрагмент текста, который может быть взят из входного дерева, обработан и дополнен сгенерированным текстом.

Фрагмент результирующего дерева

Смесь узлов, принадлежащих структуре дерева, не образующих корректный XML.

В следующих разделах показано, как используется каждый тип выражения.

Булевы выражения

Гибкость использования при различных условиях является мощной особенностью XSLT. Когда необходимо сделать выбор, можно вычислить результат булева выражения и получить один из двух возможных результатов: `true` или `false`.

Чаще всего булевы выражения используются в предикатах шаблонов поиска. Например, в следующем правиле проверка сокращает список допустимых узлов:

```
<xsl:template match="category[type='humor']">
  This one's funny...

  <blockquote>
    <xsl:apply-templates/>
  </blockquote>
</xsl:template>
```

Другой способ применения булевых выражений связан с элементом `<xsl:if>`. Его содержимое может выводиться или нет, в зависимости от проверки условия. Следующее правило показывает это в действии:

```
<xsl:template match="*">
  <xsl:if test="child::*">
    I have children.
  </xsl:if>
</xsl:template>
```

В атрибуте test значением выражения child::* является список узлов, содержащий все дочерние узлы узла контекста. Оно автоматически преобразуется в булево значение true, если это множество не пусто, и false – если оно пусто. Если узел контекста имеет хотя бы один дочерний, условие выполнено, и текст «I have children» включается в результирующее дерево. В противном случае не выводится ничего.

Аналогичной является конструкция <xsl:choose>. Этот элемент предлагает на выбор несколько вариантов, а не просто включает или выключает вывод. Вот пример:

```
<xsl:template match="*">
  <xsl:choose>
    <xsl:when test="self::chapter">
      I am a chapter.
    </xsl:when>
    <xsl:when test="self::appendix">
      I am an appendix.
    </xsl:when>
    <xsl:otherwise>
      I don't know what I am.
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>
```

Все проверки <xsl:when> выполняются последовательно, и используется содержимое того узла, который найден первым. Можно произвести столько проверок, сколько нужно, чтобы создать богатый набор вариантов. Если ни одно из проверяемых условий не выполнено, используется содержимое <xsl:otherwise>.

Как мы только что видели, не булево выражение неявно интерпретируется как булево, если контекст не допускает другого типа. Атрибут test в <xsl:when> должен иметь булев тип, поэтому узел, возвращаемый self::chapter, преобразуется в булево значение true, если узел контекста есть <chapter>, и false – если нет. Явное преобразование выражения в булево можно выполнить с помощью функции boolean(). Правила преобразования приведены в табл. 6.6.

Таблица 6.6. Правила булевых преобразований

Тип выражения	Результат преобразования
Группа узлов	Истина, если в группе есть хоть один узел, ложь – если она пуста.
Строка	Истина, если длина строки ненулевая.
Число	Истина, если значение отлично от нуля и NaN (нечисла).

Таблица 6.6 (продолжение)

Тип выражения	Результат преобразования
Фрагмент результирующего дерева	Истина, если фрагмент результирующего дерева содержит узлы, ложь в противном случае.

Числовые выражения можно сравнивать, образуя при этом булевы выражения. Операторы сравнения подробно описаны в табл. 6.7. Обратите внимание, что оператор «меньше» (<) записан с помощью сущности `<`. Таблица стилей XSLT является документом XML, поэтому необходимо соблюдать правила построения документов.

Таблица 6.7. Операторы сравнения

Оператор	Возвращаемое значение
expr = *expr*	Истина, если оба выражения (строковые или числовые) имеют одинаковое значение, в противном случае ложь.
expr!= *expr*	Истина, если выражения имеют разные значения (строковые или числовые), в противном случае ложь.
expr < *expr*	Истина, если значение первого числового выражения меньше значения второго, в противном случае ложь.
expr > *expr*	Истина, если значение первого числового выражения больше значения второго, в противном случае ложь.
expr <= *expr*	Истина, если значение первого числового выражения меньше или равно значению второго, в противном случае ложь.
expr >= *expr*	Истина, если значение первого числового выражения меньше или равно значению второго, в противном случае ложь.

Булевы выражения можно объединять операторами для булевых арифметических операций. Эти операторы перечислены в табл. 6.8.

Таблица 6.8. Булевы операторы

Оператор	Возвращаемое значение
expr and *expr*	Истина, если оба булевых выражения истинны, в противном случае ложь.
expr or *expr*	Истина, если хотя бы одно булево выражение истинно, в противном случае ложь.
not(*expr*)	Инвертирует значение булева выражения: истина, если выражение ложно, в противном случае ложь.

Таблица 6.8 (продолжение)

Оператор	Возвращаемое значение
`true()`	Истина.
`false()`	Ложь.

Выражения множеств узлов

Выражение множества узлов создает группу узлов, отобранных из документа. В множестве не бывает совпадающих узлов. Выражения множеств узлов имеют такой же вид, как пути адресации; на самом деле, пути адресации являются подмножеством выражений множеств узлов. Однако, в отличие от шаблонов поиска, выражения множеств узлов не ограничивают типы осей, которые можно использовать.

Выражения множеств узлов чаще всего применяются в элементе `<xsl:apply-templates>` и в любых элементах, обладающих атрибутом `select`. Элемент `<xsl:apply-templates>` удобно использовать в правилах шаблонов для определения узлов, которые добавляются к множеству узлов контекста. Если не задан шаблон `select`, применяется шаблон по умолчанию `child::*`, который соответствует всем дочерним элементам узла контекста. Однако этот шаблон можно переназначить для более избирательных действий:

```
<xsl:template match="category">
  <xsl:apply-templates select="aphorism"/>
</xsl:template>
```

В этом правиле шаблона элемент `<xsl:apply-templates>` создает множество узлов контекста, состоящее из всех элементов `<aphorism>`, являющихся дочерними для узла контекста. Функции, которые можно применять к множествам узлов, перечислены в табл. 6.9.

Таблица 6.9. Функции множеств узлов

Функция	Возвращает
`count(`*node set*`)`	Число элементов в множестве узлов *node set*. Например, `count(parent::*)` возвращает значение 1, потому что у узла может быть только один родитель.
`generate-id(`*node set*`)`	Строка, содержащая уникальный идентификатор для первого узла в множестве узлов *node set* или узла контекста, если аргумент опущен. Эта строка генерируется процессором и гарантированно не повторяется для различных узлов.

Таблица 6.9 (продолжение)

Функция	Возвращает
`last()`	Номер последнего узла в множестве узлов контекста. Функция `last()` аналогична `count()`, но действует только над множеством узлов контекста, а не произвольных узлов.
`local-name(`*`node set`*`)`	Имя первого узла в множестве узлов *node set* без префикса пространства имен. Если аргумент не задан, возвращает локальное имя узла контекста.
`name(`*`node set`*`)`	Аналогична `local-name()`, но в результат включается префикс пространства имен.
`namespace-uri(`*`node set`*`)`	URI пространства имен для первого узла в множестве узлов *node set*. Если аргумент не задан, возвращает URI пространства имен для узла контекста.
`position()`	Номер узла контекста в множестве узлов контекста.

Есть также несколько функций, создающих множества узлов и перечисленных в табл. 6.10.

Таблица 6.10. Функции, возвращающие множества узлов

Функция	Возвращает
`id(`*`string`*`)`	Один узел с атрибутом `ID`, равным значению строки *string*, или пустое множество, если таких узлов нет. Возвращается не более одного узла, потому что `ID` должен быть уникальным.
`key(`*`string object`*`)`	Множество всех узлов, имеющих ключ с именем *string* и значением *object*. (О ключах рассказывается далее в этой главе.)
`document(`*`uri, base`*`)`	Множество узлов, заданных URI с необязательным указателем XPointer относительно узла *base*, или, если второй аргумент не задан, узла контекста.

Числовые выражения

Узлы нумеруются, чтобы знать, является ли узел первым, последним или занимает в списке какую-то определенную позицию. Иногда также требуется обрабатывать числовые данные. Все это можно осуществлять с помощью числовых выражений.

В XSLT число определяется как 64-разрядное число с плавающей точкой, независимо от того, есть в нем десятичная точка или нет. Кроме того, число может быть задано как NaN (не-число), если преобразование оказалось неудачным. Правила преобразования любых выражений в численные значения приведены в табл. 6.11.

Таблица 6.11. Правила преобразования выражений в числа

Тип выражения	Правило
Множество узлов	Первый узел преобразуется в строку, затем используется правило преобразования строк.
Булево	Значение `true` преобразуется в число 1, а `false` – в число 0.
Строковое	Если строка является последовательностью знаков, представляющих число (например, `-123.5`), она преобразуется в это число. В остальных случаях используется значение `NaN`.
Фрагмент результирующего дерева	Так же, как множества узлов, фрагмент результирующего дерева преобразуется в строку, которая затем используется с правилом строки.

В XSLT есть ряд функций и операторов для работы с числовыми значениями, которые рассмотрены в табл. 6.12.

Таблица 6.12. Числовые операторы и функции

Функция	Возвращает
expr + *expr*	Сумма двух числовых выражений.
expr - *expr*	Разность между первым и вторым числовым выражениями.
expr * *expr*	Произведение двух числовых выражений.
expr div *expr*	Частное от деления первого числового выражения на второе.
expr mod *expr*	Остаток от деления первого выражения на второе.
round(*expr*)	Значение выражения, округленное до ближайшего целого числа.
sum(*expr*, *expr*, ...)	Сумма выражений.
floor(*expr*)	Наибольшее целое число, меньшее или равное expr.
ceiling(*expr*)	Наименьшее целое число, большее или равное expr.

Строковые выражения

Строка – это участок символьных данных, например, «How are you?», «990» или «z». Любое выражение можно превратить в строку с помощью функции `string()`. Эта функция использует при создании строк правила, описанные в табл. 6.13.

Таблица 6.13. Правила преобразования выражений в строки

Тип выражения	Правило
Множество узлов	В качестве строки используется текстовое значение первого узла.
Булево	Строка имеет значение `true`, если выражение истинно, в противном случае, имеет значение `false`.
Число	Строковое значение является числом в том виде, как оно было бы напечатано. Например, `string(1 + 5 - 9)` имеет значением строку `-3`.
Фрагмент результирующего дерева	Значение строки является конкатенацией текстовых значений всех узлов фрагмента.

В табл. 6.14 перечислены функции, с помощью которых можно создавать строки.

Таблица 6.14. Функции, создающие строки

Функция	Возвращаемое значение
`concat(`*`string`*`, `*`string`*`, ...)`	Строка, являющаяся конкатенацией строк-аргументов.
`format-number(`*`number`*`, `*`pattern`*`, `*`locale`*`)`	Строка, содержащая число *`number`*, форматированное согласно шаблону *`pattern`* и набору национальных правил, задаваемых необязательным аргументом *`locale`*.
`normalize-space(`*`string`*`)`	Строка *`string`* с удаленными ведущими и замыкающими пробелами, а все остальные пробельные символы заменяются одиночными пробелами. Если аргумент отсутствует, используется значение узла контекста.
`substring(`*`string`*`, `*`offset`*`, `*`range`*`)`	Подстрока аргумента *`string`*, начинающаяся через *`offset`* символов после начала и длиной *`range`* символов.
`substring-after(`*`string`*`, `*`to-match`*`)`	Подстрока аргумента *`string`*, начинающаяся в конце первого вхождения строки *`to-match`* и заканчивающаяся в конце *`string`*.

Таблица 6.14 (продолжение)

Функция	Возвращаемое значение
substring-before(*string*, *to-match*)	Подстрока аргумента *string*, начинающаяся с начала *string* и заканчивающаяся первым вхождением строки *to-match*.
translate(*string*, *to-match*, *replace-with*)	Строка *string*, в которой все вхождения подстроки *to-match* заменены строкой *replace-with*.

Для операций со строками могут использоваться функции, перечисленные в табл. 6.15.

Таблица 6.15. Функции, оперирующие строками

Функция	Возвращаемое значение
Contains(*string*, *sub*)	Истина, если в *string* присутствует подстрока *sub*, в противном случае ложь.
starts-with(*string*, *sub*)	Истина, если *string* начинается с подстроки *sub*, в противном случае ложь.
string-length(*string*)	Число символов в *string*.

Тонкая настройка шаблонов

Теперь, закончив изложение основ правил шаблонов и отбора, рассмотрим некоторые способы настройки процесса. В этом разделе рассказывается о создании узлов, генерации чисел и текста, а также сортировке и обходе множеств узлов.

Вывод значений узлов с помощью элемента < xsl:value–of >

По умолчанию действует правило, согласно которому содержимое элементов и атрибутов выводится как текст. Пока у нас нет другого способа вывода значений узлов. Элемент <xsl:value-of> служит средством вычисления и возврата значений. Чтобы понять, как это применяется к узлам, рассмотрим следующее правило:

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

Если применить это правило к примеру <quotelist>, получится следующий результат:

```
publication
The New Scientist
6/99

tv-show
Homer
The Simpsons

Greg Travis

Confucius
```

Значения для каждого типа узла рассчитываются так:

Корневой узел

Корневой узел наследует значение элемента документа.

Элемент

Все анализируемые символьные данные элемента вместе со значениями его потомков.

Атрибут

Значение атрибута, в котором разрешены сущности и удалены ведущие и замыкающие пробельные символы.

Текст

Весь текст узла.

Инструкция обработки

Все, что находится внутри ограничителей инструкции обработки, кроме имени.

Комментарий

Текст внутри разделителей комментария.

Пространство имен

URI пространства имен.

Если элемент `<xsl:value-of>` применяется к множеству узлов, используется значение только первого узла. Может возникнуть соблазн применить следующее правило, которое возвратит только значение первого узла:

```
<xsl:template match="quote">
  <xsl:value-of select="source"/>
</xsl:template>
```

Помимо вывода значений узлов, элемент `<xsl:value-of>` полезен для вывода результатов вычислений; например, следующее правило выводит количество цитат в списке:

```
<xsl:template match="quotelist">
  <p>Total: <xsl:value-of select="count(//quote)"/> quotes.</p>
</xsl:template>
```

Обход узлов с помощью директивы <xsl:for-each>

Директива <xsl:for-each> является конструкцией, обрабатывающей все множество узлов в рамках одного правила. Директива <xsl:for-each> работает, создавая новое множество узлов контекста и узел контекста, что полезно для локализованных эффектов, таких как сборка форматированных списков и вызов специальных функций. Следующее правило генерирует оглавление:

```
<xsl:template match="book">
  <xsl:for-each select="chapter">
    <xsl:value-of select="position()"/>.
    <xsl:value-of select="title"/>.
  </xsl:for-each>
  <xsl:apply-templates/>
</xsl:template>
```

Это правило использует директиву <xsl:for-each> для создания нумерованного списка названий глав, прежде чем обрабатывать каждую главу с помощью директивы <xsl:apply-templates>.

Создание узлов

Обычно элементы и атрибуты создаются просто в результате ввода их в правила шаблонов, что можно было видеть на предыдущих примерах. Хотя этот метод обычно предпочтителен (из-за своей простоты), у него есть ограничения. Например, может потребоваться создать атрибут со значением, которое должно определяться в результате сложного процесса. Следующее правило демонстрирует этот процесс:

```
<xsl:template match="a">
  <p>See the
    <a>
      <xsl:attribute
        name="href">http://www.oreilly.com/catalog/<xsl:call-template
        name="prodname"/></xsl:attribute>
        catalog page
    </a> for more information.)
  </p>
</xsl:template>
```

В этом правиле директива <xsl:attribute> создает новый узел атрибута с именем href. Значением этого атрибута является содержимое элемен-

та, создавшего узел, в данном случае URI с некоторым переменным текстом, создаваемым элементом <xsl:call-template>. Как уже отмечалось, переменный текст невозможно включить внутрь элемента <a>, поэтому создание узла атрибута выделено в самостоятельный шаг.

Создание элементов с помощью директивы <xsl:element>

Помимо директивы <xsl:attribute> существует директива для создания любого типа узла. Директива <xsl:element> создает элементы. Она полезна, когда имя элемента создается «на лету»:

```
<xsl:template match="thing">
  <xsl:element name="{@type}">
    <xsl:value-of select="."/>
  </xsl:element>
</xsl:template>
```

Атрибут name устанавливает тип элемента. Если бы входной элемент содержал:

```
<thing type="circle">radius: 3</thing>
<thing type="square">side: 7</thing>
```

то выходные данные выглядели бы так:

```
<circle>radius: 3</circle>
<square>side: 7</square>
```

Атрибуты и множества атрибутов

Выше было показано, что генерировать атрибуты можно с помощью директивы <xsl:attribute>. Как и при создании элементов, можно образовывать имя атрибута и его значение на лету. Заметьте, однако, что директива <xsl:attribute> должна располагаться перед любым другим содержимым, иначе произойдет ошибка.

Один набор атрибутов можно применять ко многим различным элементам при помощи директивы <xsl:attribute-set>. Во-первых, определим множество атрибутов:

```
<xsl:attribute-set name="common-atts">
  <xsl:attribute name="id"/>
    <xsl:value-of select="generate-id()"/>
  </xsl:attribute>
  <xsl:attribute name="class">
    quote
  </xsl:attribute>
</xsl:attribute-set>
```

При этом создается множество из двух атрибутов, id и class="quote". Доступ к множеству осуществляется из любого элемента с указанием

имени common-atts. Теперь можно при помощи атрибута use-attribute-sets ссылаться на определенное выше множество атрибутов:

```
<xsl:template match="quote">
  <blockquote xsl:use-attribute-sets="common-atts">
    <xsl:value-of select="."/>
  </blockquote>
</xsl:template>
```

Допускается задавать более одного множества атрибутов, перечисляя их имена через пробел.

Вставка текста с помощью директивы <xsl:text>

Обычно текст помещается в шаблон в результате ввода с клавиатуры. Однако при этом часто появляются нежелательные пробелы и переводы строки. Если нужно точно контролировать пробельные символы, следует использовать директиву <xsl:text>. Этот элемент сохраняет пробельные символы в том виде, как они введены.

Эта директива также позволяет вводить запрещенные символы, такие как < и &. Для этого нужно использовать атрибут <disable-output-escaping>:

```
<xsl:template match="codelisting">
  <h3>Code Example</h3>
  <pre>
    <xsl:text disable-output-escaping="yes">
      cout << "How to output strings in C++";
    </xsl:text>
  </pre>
</xsl:template>
```

Создание инструкций обработки и комментариев

Создание инструкций обработки и комментариев является простой задачей. Элемент <xsl:processing-instruction> принимает атрибут name и некоторое текстовое содержимое, из которых он создает инструкцию обработки:

```
<xsl:template match="marker">
  <xsl:processing-instruction name="formatter">
    pagenumber=<xsl:value-of select="{@page}"/>
  </xsl:processing-instruction>
</xsl:template>
```

Это правило создает следующую выдачу:

```
<?formatter pagenumber=1?>
```

Комментарий можно создать с помощью элемента <xsl:comment> без атрибутов:

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

Чтобы создать инструкцию обработки или содержимое комментария, нужно задать либо обычный текст, либо такой элемент, как `<xsl:value-of>`, становящийся текстом. Любое другое содержимое вызовет ошибку.

Нумерация

Можно производить вычисления и выводить числа с помощью директивы `<xsl:value-of>`, но этот элемент ограничивает возможности форматирования только десятичными числами. XSLT предоставляет больше гибкости с помощью директивы `<xsl:number>`. Этот элемент позволяет выводить числа как римские цифры, с незначащими нулями или как буквы. В него также встроена возможность подсчета узлов. Если вернуться к примеру с оглавлением, то с помощью директивы `<xsl:number>` его можно создать так:

```
<xsl:template match="book">
  <xsl:for-each select="chapter">
    <xsl:number value="position()"/>.
    <xsl:value-of select="title"/>.
  </xsl:for-each>
</xsl:template>
```

В результате получается следующее:

```
1. Aquaman Forgets How to Swim
2. Evil King Oystro Sends Assassins
3. Godzilla Saves the Day
4. Aquaman is Poisoned by Pufferfish
5. Aqualad Delivers the Antidote
6. Atlantis Votes Aquaman into Office
```

Режим по умолчанию тот же, что в директиве `<xsl:value-of>`: старые добрые десятичные числа. Атрибут `format` предоставляет дополнительные возможности. Например, можно использовать римские цифры:

```
<xsl:template match="book">
  <xsl:for-each select="chapter">
    <xsl:number value="position()" format="I"/>.
    <xsl:value-of select="title"/>.
  </xsl:for-each>
</xsl:template>
```

что дает:

```
I.   Aquaman Forgets How to Swim
```

```
II.  Evil King Oystro Sends Assassins
III. Godzilla Saves the Day
IV.  Aquaman is Poisoned by Pufferfish
V.   Aqualad Delivers the Antidote
VI.  Atlantis Votes Aquaman into Office
```

В табл. 6.16 показаны некоторые другие способы использования атрибута `format`.

Таблица 6.16. Числовые форматы

Строка формата	Схема нумерации
1	1, 2, 3, 4, …
0	0, 1, 2, 3, …
4	4, 5, 6, 7, …
01	01, 02, 03, …, 09, 10, 11, …
I	I, II, III, IV, …
i	i, ii, iii, iv, …
iii	iii, iv, v, vi, …
A	A, B, C, D, …
a	a, b, c, d, …
G	G, H, I, J, …

Представим себе теперь, что нужно получить алфавитный список, начинающийся с «i». Строка формата `i` по умолчанию дает римские цифры в нижнем регистре, а не алфавит. Нужен дополнительный атрибут, `letter-value`, чтобы явно установить тип формата:

```
<xsl:template match="book">
  <xsl:for-each select="chapter">
    <xsl:number value="position()" format="i"
         letter-value="alphabetic"/>.
    <xsl:value-of select="title"/>.
  </xsl:for-each>
</xsl:template>
```

что дает следующий результат:

```
i. Aquaman Forgets How to Swim
j. Evil King Oystro Sends Assassins
k. Godzilla Saves the Day
l. Aquaman is Poisoned by Pufferfish
m. Aqualad Delivers the Antidote
n. Atlantis Votes Aquaman into Office
```

Можно управлять знаками пунктуации, разделяющими группы цифр в больших числах. В США принято отделять каждые три цифры числа запятой (например 1,000,000), тогда как в других странах используются иные схемы. Есть два атрибута, определяющие соответствующий формат. Первый, `grouping-separator`, устанавливает строку, разделяющую группы цифр. Второй, `grouping-size`, задает число цифр в группе. Чтобы форматировать число в виде 1*0000*0000, следует написать:

```
<xsl:number
  value="100000000"
  grouping-separator="*"
  grouping-size="4"/>
```

Самой полезной характеристикой директивы `<xsl:number>` является возможность подсчета узлов с ее помощью. Атрибут `count` задает тип подсчитываемых узлов. Допустим, например, что требуется выводить перед названием каждой главы ее номер, скажем, так:

```
<h1>Chapter 5. Aqualad Delivers the Antidote</h1>
```

Эту задачу решит следующее правило:

```
<xsl:template match="chapter/title">
  <h1>
    Chapter
    <xsl:number count="chapter"/>
    <xsl:value-of select="."/>
  </h1>
</xsl:template>
```

Атрибут `count` учитывает только узлы, находящиеся на одном уровне. Все элементы `<chapter>` всегда находятся на одном уровне, поскольку являются дочерними для элемента `<book>`, поэтому данное требование удовлетворяется. Но что если не все узлы находятся на одном уровне? Например, сноски появляются в разных абзацах, поэтому если необходимо их пронумеровать, должна быть возможность заглянуть за пределы текущего уровня. Это можно сделать, установив для атрибута `level` значение `any`. Вот пример:

```
<xsl:template match="footnote">
  <xsl:text>[</xsl/text>
  <xsl:number count="footnote" from="chapter" level="any"/>
  <xsl:text>]</xsl/text>
</xsl:template>
```

Это правило вставляет заключенный в квадратные скобки номер туда, где появляется сноска. Атрибут `from="chapter"` заставляет нумерацию начинаться с последнего открывающего тега `<chapter>`, а `level="any"` обеспечивает подсчет всех сносок независимо от уровня, на котором они появляются.

Сортировка

Часто встречается ситуация, когда элементы должны быть отсортированы. Таблицы, каталоги и обозрения являются примерами документов, нуждающихся в сортировке. Представим себе телефонный справочник, отсортированный по трем ключам: фамилии абонента, имени и городу. Документ выглядит так:

```
<telephone-book>
  ...
  <entry id="44456">
    <surname>Mentary</surname>
    <firstname>Rudy</firstname>
    <town>Simpleton</town>
    <street>123 Bushwack Ln</street>
    <phone>555-1234</phone>
  </entry>

  <entry id="44457">
    <surname>Chains</surname>
    <firstname>Allison</firstname>
    <town>Simpleton</town>
    <street>999 Leafy Rd</street>
    <phone>555-4321</phone>
  </entry>
  ...
</telephone-book>
```

По умолчанию процедура трансформации обрабатывает все узлы в том порядке, в котором они появляются в документе. Поэтому запись с `id="44456"` выводится ранее записи с `id="44457"`. Очевидно, при этом не соблюдается алфавитный порядок, поэтому нужно каким-то образом отсортировать результаты. Оказывается, это можно сделать с помощью элемента `<xsl:sort>`. Вот, например, как может выглядеть правило элемента документа:

```
<xsl:template match="telephone-book">
  <xsl:apply-templates>
    <xsl:sort select="town"/>
    <xsl:sort select="surname"/>
    <xsl:sort select="firstname"/>
  </xsl:apply-templates>
</xsl:template>
```

Здесь есть три направления сортировки. Сначала все результаты сортируются по городу (town). Затем записи сортируются по фамилии (surname). Наконец, они сортируются по имени (firstname).

Пример: чековая книжка

Теперь пора привести полезный пример, демонстрирующий обсуждавшиеся понятия. Чековая книжка из примера 5.1 дает хороший материал для проверки, поэтому мы используем ее снова. Пример 6.4 показывает новый вариант документа Checkbook. Это тот же документ, что использовался в главе 5 «Модели документов: более высокий уровень контроля», за исключением того, что в конец добавлен еще один платеж <payment>, что привело к отрицательному остатку на счете:

Пример 6.4. Пример чековой книжки

```
<?xml version="1.0"?>
<!DOCTYPE checkbook SYSTEM "checkbook.dtd">

<checkbook>

  <deposit type="direct-deposit">
    <payor>Bob's Bolts</payor>
    <amount>987.32</amount>
    <date>21-6-00</date>
    <description category="income">Paycheck</description>
  </deposit>

  <payment type="check" number="980">
    <payee>Kimora's Sports Equipment</payee>
    <amount>132.77</amount>
    <date>23-6-00</date>
    <description category="entertainment">kendo equipment</description>
  </payment>

  <payment type="atm">
    <amount>40.00</amount>
    <date>24-6-00</date>
    <description category="cash">pocket money</description>
  </payment>

  <payment type="debit">
    <payee>Lone Star Cafe</payee>
    <amount>36.86</amount>
    <date>26-6-00</date>
    <description category="food">lunch with Greg</description>
  </payment>

  <payment type="check" number="981">
    <payee>Wild Oats Market</payee>
    <amount>47.28</amount>
    <date>29-6-00</date>
    <description category="food">groceries</description>
  </payment>
```

Пример 6.4. Пример чековой книжки (продолжение)

```
  <payment type="debit">
    <payee>Barnes and Noble</payee>
    <amount>58.79</amount>
    <date>30-6-00</date>
    <description category="work">O'Reilly Books</description>
  </payment>

  <payment type="check" number="982">
    <payee>Old Man Ferguson</payee>
    <amount>800.00</amount>
    <date>31-6-00</date>
    <description category="misc">a 3-legged antique credenza that once
    belonged to Alfred Hitchcock</description>
  </payment>

</checkbook>
```

Создадим таблицу стилей трансформации, которая покажет, как можно обработать экземпляр документа XML с помощью XSLT и получить совершенно другой документ. Ее задача в том, чтобы собрать вместе операции и вывести отчет о последних действиях и текущем состоянии чекового счета. Предполагается, что в начале документа текущий баланс равен нулю.

Выходным форматом является HTML, благодаря чему просмотр можно будет осуществлять любым веб-броузером. Первое правило шаблона определяет структуру страницы HTML и необходимые теги:

```
<xsl:template match="checkbook">
  <html>
    <head/>
    <body>
              <!-- page content goes here -->
    </body>
  </html>
</xsl:template>
```

Теперь добавим раздел, подытоживающий приходные операции. Заголовок раздела, помещенный внутрь элемента `<h3>`, генерируется из отсутствующего в документе текста, создаваемого директивой `<xsl:text>`, и дат первой и последней операций (с использованием директивы `<xsl:value-of>`). После заголовка с помощью директивы `<xsl:apply-templates>` перечислены все приходные операции в порядке их появления. Теперь правило выглядит так (вновь добавленное выделено полужирным шрифтом):

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

```
    <body>
      <!-- income information -->
      <h3>
        <xsl:text>Income from </xsl:text>
        <xsl:value-of select="child::*[1]/date"/>
        <xsl:text> until </xsl:text>
        <xsl:value-of select="child::*[last()]/date"/>
        <xsl:text>:</xsl:text>
      </h3>
      <xsl:apply-templates select="deposit"/>
    </body>
  </html>
</xsl:template>
```

После этого добавим раздел, описывающий вычеты с чекового счета. Было бы хорошо отсортировать список операций в порядке убывания, поэтому используем элемент `<xsl:sort>`. Теперь правило такое:

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

      <!-- income information -->
      <h3>
        <xsl:text>Income from </xsl:text>
        <xsl:value-of select="child::*[1]/date"/>
        <xsl:text> until </xsl:text>
        <xsl:value-of select="child::*[last()]/date"/>
        <xsl:text>:</xsl:text>
      </h3>
      <xsl:apply-templates select="deposit"/>
      <!-- payment information -->
      <h3>
        <xsl:text>Expenditures from </xsl:text>
        <xsl:value-of select="child::*[1]/date"/>
        <xsl:text> until </xsl:text>
        <xsl:value-of select="child::*[last()]/date"/>
        <xsl:text>, ranked from highest to lowest:</xsl:text>
      </h3>
      <xsl:apply-templates select="payment">
        <xsl:sort data-type="number" order="descending"
                  select="amount"/>
      </xsl:apply-templates>
    </body>
  </html>
</xsl:template>
```

И, наконец, выведем баланс счета. Для вычисления суммы операций мы используем директиву `<xsl:number>`. Для функции `sum()` нужны два члена: один для суммы расходных операций и один для суммы при-

ходных. Чтобы выделить ситуацию, когда пользователь имеет задолженность, будем отображать вычисленные результаты разными цветами и выводить предупреждение, если результат отрицательный. Вот это правило:

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

      <!-- income information -->
      <h3>
        <xsl:text>Income from </xsl:text>
        <xsl:value-of select="child::*[1]/date"/>
        <xsl:text> until </xsl:text>
        <xsl:value-of select="child::*[last()]/date"/>
        <xsl:text>:</xsl:text>
      </h3>
      <xsl:apply-templates select="deposit"/>

      <!-- payment information -->
      <h3>
        <xsl:text>Expenditures from </xsl:text>
        <xsl:value-of select="child::*[1]/date"/>
        <xsl:text> until </xsl:text>
        <xsl:value-of select="child::*[last()]/date"/>
        <xsl:text>, ranked from highest to lowest:</xsl:text>
      </h3>
      <xsl:apply-templates select="payment">
        <xsl:sort data-type="number" order="descending"
                  select="amount"/>
      </xsl:apply-templates>
      <h3>Balance</h3>
      <p>
        <xsl:text>Your balance as of </xsl:text>
        <xsl:value-of select="child::*[last()]/date"/>
        <xsl:text> is </xsl:text>
        <tt><b>
          <xsl:choose>
            <xsl:when test="sum(payment/amount)
                            > sum(deposit/amount)">
              <font color="red">
                <xsl:text>$</xsl:text>
                <xsl:value-of select="sum(deposit/amount)
                                      - sum(payment/amount)"/>
              </font>
            </xsl:when>
            <xsl:otherwise>
              <font color="blue">
                <xsl:text>$</xsl:text>
                <xsl:value-of select="sum(deposit/amount)
```

```
                                        - sum(payment/amount)"/>
              </font>
            </xsl:otherwise>
          </xsl:choose>
        </b></tt>
      </p>
      <xsl:if test="sum(payment/amount) > sum(deposit/amount)">
        <p>
          <font color="red">
            <xsl:text>DANGER! Deposit money quick!</xsl:text>
          </font>
        </p>
      </xsl:if>
    </body>
  </html>
</xsl:template>
```

Теперь нам нужны некоторые правила для обработки элементов `<payment>` и `<deposit>`. Первое, показанное ниже, нумерует каждый платеж и составляет для него фразу:

```
<xsl:template match="payment">
  <p>
    <xsl:value-of select="position()"/>
    <xsl:text>. On </xsl:text>
    <xsl:value-of select="date"/>
    <xsl:text>, you paid </xsl:text>
    <tt><b>
      <xsl:text>$</xsl:text>
      <xsl:value-of select="amount"/>
    </b></tt>
    <xsl:text> to </xsl:text>
    <i>
      <xsl:value-of select="payee"/>
    </i>
    <xsl:text> for </xsl:text>
    <xsl:value-of select="description"/>
    <xsl:text>.</xsl:text>
  </p>
</xsl:template>
```

Для большинства типов платежей этого достаточно, но не для типа платежа `"atm"`. Из документа видно, что платеж `atm` не содержит описания получателя, т. к. предполагается, что получателем денег является автор чековой книжки. Поэтому составим особое правило именно для данного случая:

```
<xsl:template match="payment[@type='atm']">
  <p>
    <xsl:value-of select="position()"/>
    <xsl:text>. On </xsl:text>
```

```
      <xsl:value-of select="date"/>
      <xsl:text>, you withdrew </xsl:text>
      <tt><b>
        <xsl:text>$</xsl:text>
        <xsl:value-of select="amount"/>
      </b></tt>
      <xsl:text> from an ATM for </xsl:text>
      <xsl:value-of select="description"/>
      <xsl:text>.</xsl:text>
    </p>
  </xsl:template>
```

И, наконец, вот правило для элемента `<deposit>`**:**

```
  <xsl:template match="deposit">
    <p>
      <xsl:value-of select="position()"/>
      <xsl:text>. On </xsl:text>
      <xsl:value-of select="date"/>
      <xsl:text>, </xsl:text>
      <tt><b>
        <xsl:text>$</xsl:text>
        <xsl:value-of select="amount"/>
      </b></tt>
      <xsl:text> was deposited into your account by </xsl:text>
      <i>
        <xsl:value-of select="payor"/>
      </i>
      <xsl:text>.</xsl:text>
    </p>
  </xsl:template>
```

Соединив все в одной таблице стилей, мы получим листинг примера 6.5.

Пример 6.5. Таблица стилей преобразования чековой книжки

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

<!--
=========================================================================
A simple transformation stylesheet to get information out of
a checkbook.
=========================================================================
-->

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

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

Пример 6.5. Таблица стилей преобразования чековой книжки (продолжение)

```
<h3>
  <xsl:text>Income from </xsl:text>
  <xsl:value-of select="child::*[1]/date"/>
  <xsl:text> until </xsl:text>
  <xsl:value-of select="child::*[last()]/date"/>
  <xsl:text>:</xsl:text>
</h3>
<xsl:apply-templates select="deposit"/>
<h3>
  <xsl:text>Expenditures from </xsl:text>
  <xsl:value-of select="child::*[1]/date"/>
  <xsl:text> until </xsl:text>
  <xsl:value-of select="child::*[last()]/date"/>
  <xsl:text>, ranked from highest to lowest:</xsl:text>
</h3>
<xsl:apply-templates select="payment">
  <xsl:sort data-type="number" order="descending" select="amount"/>
</xsl:apply-templates>
<h3>Balance</h3>
<p>
  <xsl:text>Your balance as of </xsl:text>
  <xsl:value-of select="child::*[last()]/date"/>
   <xsl:text> is </xsl:text>
  <tt><b>
    <xsl:choose>
      <xsl:when test="sum(payment/amount) > sum(deposit/amount)">
        <font color="red">
          <xsl:text>$</xsl:text>
          <xsl:value-of select="sum(deposit/amount)
                                - sum(payment/amount)"/>
        </font>
      </xsl:when>
      <xsl:otherwise>
        <font color="blue">
          <xsl:text>$</xsl:text>
          <xsl:value-of select="sum(deposit/amount)
                                - sum(payment/amount)"/>
        </font>
      </xsl:otherwise>
    </xsl:choose>
  </b></tt>
</p>
<xsl:if test="sum(payment/amount) > sum(deposit/amount)">
  <p>
    <font color="red">
      <xsl:text>DANGER! Deposit money quick!</xsl:text>
    </font>
  </p>
```

Пример 6.5. Таблица стилей преобразования чековой книжки (продолжение)

```
      </xsl:if>
    </body>
  </html>
</xsl:template>

<xsl:template match="payment[@type='atm']">
  <p>
    <xsl:value-of select="position()"/>
    <xsl:text>. On </xsl:text>
    <xsl:value-of select="date"/>
    <xsl:text>, you withdrew </xsl:text>
    <tt><b>
      <xsl:text>$</xsl:text>
      <xsl:value-of select="amount"/>
    </b></tt>
    <xsl:text> from an ATM for </xsl:text>
    <xsl:value-of select="description"/>
    <xsl:text>.</xsl:text>
  </p>
</xsl:template>

<xsl:template match="payment">
  <p>
    <xsl:value-of select="position()"/>
    <xsl:text>. On </xsl:text>
    <xsl:value-of select="date"/>
    <xsl:text>, you paid </xsl:text>
    <tt><b>
      <xsl:text>$</xsl:text>
      <xsl:value-of select="amount"/>
    </b></tt>
    <xsl:text> to </xsl:text>
    <i>
      <xsl:value-of select="payee"/>
    </i>
    <xsl:text> for </xsl:text>
    <xsl:value-of select="description"/>
    <xsl:text>.</xsl:text>
  </p>
</xsl:template>

<xsl:template match="deposit">
  <p>
    <xsl:value-of select="position()"/>
    <xsl:text>. On </xsl:text>
    <xsl:value-of select="date"/>
    <xsl:text>, </xsl:text>
    <tt><b>
```

Пример 6.5. Таблица стилей преобразования чековой книжки (продолжение)

```
      <xsl:text>$</xsl:text>
      <xsl:value-of select="amount"/>
    </b></tt>
    <xsl:text> was deposited into your account by </xsl:text>
    <i>
      <xsl:value-of select="payor"/>
    </i>
    <xsl:text>.</xsl:text>
  </p>
</xsl:template>
</xsl:stylesheet>
```

В примере 6.6 показан получающийся в результате файл HTML, а на рис. 6.2 показано его представление броузером.

Пример 6.6. Результирующее дерево

```
<html>
<body>
<h3>Income from 21-6-00 until 31-6-00:</h3>
<p>1. On 21-6-00, <tt><b>$987.32</b></tt> was deposited into your
account by <i>Bob's Bolts</i>.</p>
<h3>Expenditures from 21-6-00 until 31-6-00, ranked from highest to
lowest:</h3>
<p>1. On 31-6-00, you paid <tt><b>$800.00</b></tt> to <i>Old Man
Ferguson</i> for a 3-legged antique credenza that once belonged to
Alfred Hitchcock.</p>
<p>2. On 23-6-00, you paid <tt><b>$132.77</b></tt> to <i>Kimora's
Sports Equipment</i> for kendo equipment.</p>
<p>3. On 30-6-00, you paid <tt><b>$58.79</b></tt> to <i>Barnes and
Noble</i> for O'Reilly Books.</p>
<p>4. On 29-6-00, you paid <tt><b>$47.28</b></tt> to <i>Wild Oats
Market</i> for groceries.</p>
<p>5. On 24-6-00, you withdrew <tt><b>$40.00</b></tt> from an ATM for
pocket money.</p>
<p>6. On 26-6-00, you paid <tt><b>$36.86</b></tt> to <i>Lone Star
Cafe</i> for lunch with Greg.</p>
<h3>Balance</h3>
<p>Your balance as of 31-6-00 is <tt><b><font
color="red">$-128.38</font></b></tt>
</p>
<p>
<font color="red">DANGER! Deposit money quick!</font>
</p>
</body>
</html>
```

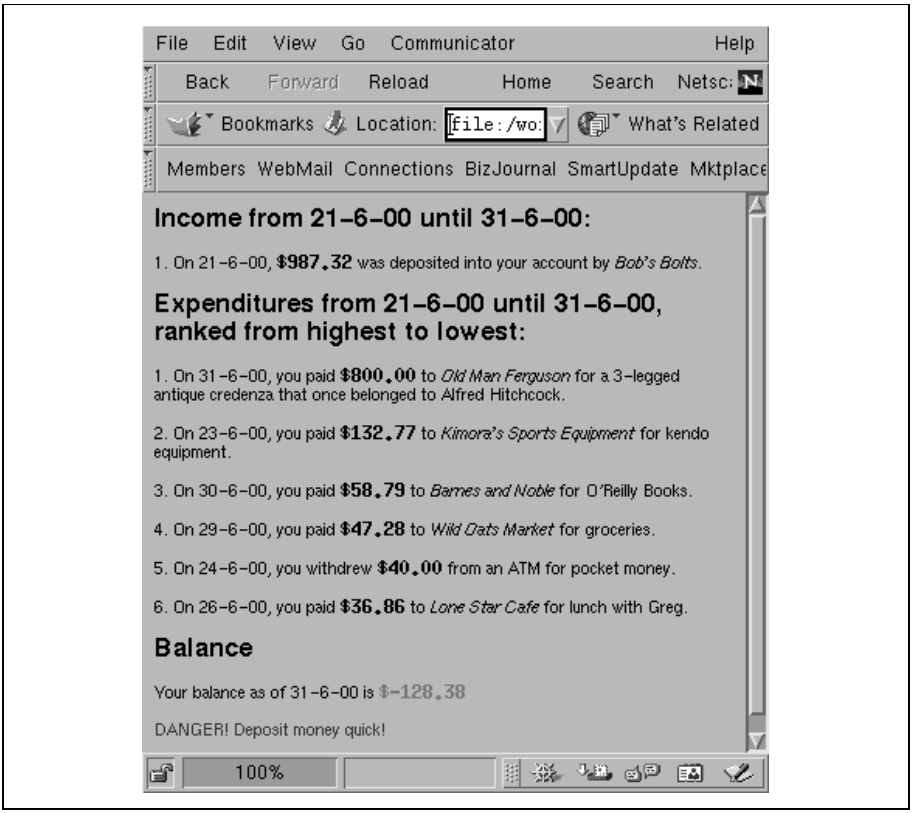


Рис. 6.2. *Чековая книжка в Netscape*

Более сложные приемы

В данный момент вы готовы к тому, чтобы начать использовать мощь XSLT. В этом разделе мы познакомимся с некоторыми первоклассными приемами, такими как именование и передача параметров в правила шаблонов, применение режимов для изменения работы правил и обработка пробельных символов.

Именованные шаблоны

Все правила шаблонов, демонстрировавшиеся до сих пор, задавались с помощью шаблонов поиска. Иногда, однако, требуется вызвать правило шаблона явно; для этой цели XSLT предоставляет *именованные шаблоны* (*named template*). Именованный шаблон удобно использовать в повторяющихся задачах, т. к. он упрощает код и облегчает его чтение. Программисты могут представить себе именованные шаблоны

как подпрограммы, играющие в языках программирования аналогичную роль.

Именованный шаблон – такое же правило, как все другие, но вместо атрибута `match` он имеет атрибут `name`, с помощью которого получает метку. Чтобы перейти к этому правилу, используется элемент с именем `<xsl:call-template>`. В отличие от обычных правил шаблонов, при вызове именованного шаблона не изменяются узел и множество узлов контекста.

Вот пример именованного шаблона, который создает группу ссылок навигации для страницы HTML:

```
<xsl:template name="navbar">
  <div class="navbar">
    <xsl:text>Current document: </xsl:text>
    <xsl:value-of select="title"/>
    <br/>
    <a href="index.htm">Home</a> |
    <a href="help.htm">Help</a> |
    <a href="toc.htm">Contents</a>
  </div>
</xsl:template>
```

Перед ссылками мы поместили две строки, выводящие название текущего документа. Этим демонстрируется, что текущий узел остался тем же, каким он был в правиле, вызвавшем именованный шаблон. Для вызова именованного шаблона `navbar` используется элемент `<xsl:call-template>`. Обратите внимание, что шаблон можно вызывать столько раз, сколько нужно; в данном случае, мы хотим, чтобы панель навигации появлялась и вверху, и внизу страницы, поэтому мы вызываем `navbar` дважды:

```
<xsl:template match="mainblock">
  <body>
    <xsl:call-template name="navbar"/>
    <xsl:apply-templates/>
    <xsl:call-template name="navbar"/>
  </body>
</xsl:template>
```

Параметры и константы

Подобно подпрограммам и функциям языков программирования, именованные шаблоны могут принимать параметры от правил, которые их вызывают. Параметры являются хранимыми выражениями, доступ к которым может осуществляться с помощью символов. Допустим, что нам потребовалось правило шаблона, генерирующее предупредительное или предостерегающее сообщение для читателей. Для ус-

тановки заголовка этого сообщения, например, «Warning» или «Important Tip», может быть использован параметр:

```
<xsl:template name="note">
  <xsl:param name="title">Note</xsl:param>

  <blockquote class="note">
    <h3>
      <xsl:value-of select="$title"/>
    </h3>
    <xsl:apply-templates/>
  </blockquote>
</xsl:template>
```

Элемент `<xsl:param>` выполняет две задачи. Во-первых, он объявляет параметр с именем `title`. Во-вторых, он устанавливает значением этого параметра по умолчанию строку `Note` – на случай, если в вызывающем правиле значение не задано. Элемент `<xsl:value-of>` выводит значение параметра, используя специальное выражение `$title`, которое называется *ссылкой на параметр* (*parameter reference*). Знак доллара (`$`) позволяет отличить ссылку на параметр от простой строки `title`. Если ссылка на параметр используется внутри атрибута, необходимо заключить ее в фигурные скобки ({}):

```
<a href="{$file}">Next Page</a>
```

Вот как можно вызвать именованный шаблон с параметром `title`:

```
<xsl:template name="warning">
  <xsl:call-template name="note">
    <xsl:with-param name="title">
      Look out!
    </xsl:with-param>
  </xsl:call-template>
</xsl:template>
```

Элемент объявления `<xsl:variable>` позволяет определить значение, которое может использоваться в любом месте таблицы стилей. Вопреки его названию, при этом не создается переменной, которую можно изменять во время трансформации. С момента своего объявления значение является постоянным и не может быть изменено в процессе преобразования. Вот несколько примеров объявлений таких постоянных значений:

```
<xsl:variable name="year" select="2001"/>

<xsl:variable name="double-space">
  <xsl:text>

  </xsl:text>
</xsl:variable>
```

```
<xsl:variable name="author-name">
  <xsl:value-of select="/book/bookinfo/authorgroup/author/firstname"/>
  <xsl:text> </xsl:text>
  <xsl:value-of select="/book/bookinfo/authorgroup/author/surname"/>
</xsl:variable>
```

Как и параметры, ссылка на константу имеет обязательный префикс в виде знака доллара ($) и при использовании в значениях атрибутов должна заключаться в фигурные скобки ({}). Константы могут применяться в объявлениях других констант, если при этом не создается рекурсивных определений вроде следующего:

```
<xsl:variable name="PIP">
  $PIP Inline Printing
</xsl:variable>
```

Нельзя также создавать с их помощью определения, ссылающиеся друг на друга:

```
<xsl:variable name="thing1">
  $thing2
</xsl:variable>

<xsl:variable name="thing2">
  $thing1
</xsl:variable>
```

Константы могут применяться внутри правил шаблонов. Это позволяет вычислить значение один раз и затем многократно к нему обращаться. Следующее правило создает номер, заключенный в квадратные скобки, и использует его в качестве ссылки на текст сноски, находящийся в конце страницы. Номер сноски вычисляется один раз, но используется дважды.

```
<xsl:template match="footnote">
  <xsl:variable name="fnum"
      select="count(preceding::footnote[ancestor::chapter//.])+1"/>
  <a>
    <xsl:attribute name="href">
      <xsl:text>#FOOTNOTE-</xsl:text>
      <xsl:number value="$fnum" format="1"/>
    </xsl:attribute>
    <xsl:text>[</xsl:text>
    <xsl:number value="$fnum"/>
    <xsl:text>]</xsl:text>
  </a>
</xsl:template>
```

Вместо выполнения вычислений в содержимом элемента, мы сделали это внутри атрибута `select`. Можно применять оба метода, но метод,

работающий с содержимым элемента, выгоднее в более сложных расчетах, например, если в них участвуют варианты выбора.

Режимы

Иногда желательно по-разному обрабатывать узлы в зависимости от того, в каком месте документа они используются. Например, можно потребовать, чтобы сноски в таблицах обозначались буквами, а не цифрами. Для этого XSLT предоставляет специальные модификаторы правил, называемые *режимами* (*modes*).

Для того чтобы установить режим, нужно просто добавить атрибут `mode`, равный некоторой метке, в соответствующие элементы `<xsl:template>` и элементы вызова шаблонов. Метка режима может иметь любое значение, не совпадающее со значениями других меток режима. В следующем примере показано, как это делается:

```
<xsl:template match="footnote">
  <xsl:variable name="fnum"
      select="count(preceding::footnote[ancestor::chapter//.])+1"/>
  <a>
    <xsl:attribute name="href">
      <xsl:text>#FOOTNOTE-</xsl:text>
      <xsl:number value="$fnum" format="1"/>
    </xsl:attribute>
    <xsl:text>[</xsl:text>
    <xsl:number value="$fnum"/>
    <xsl:text>]</xsl:text>
  </a>
</xsl:template>

<xsl:template match="footnote" mode="tabular">
  <xsl:variable name="fnum"
      select="count(preceding::footnote[ancestor::chapter//.])+1"/>
  <a>
    <xsl:attribute name="href">
      <xsl:text>#FOOTNOTE-</xsl:text>
      <xsl:number value="$fnum" format="1"/>
    </xsl:attribute>
    <xsl:text>[</xsl:text>
    <xsl:number value="$fnum" format="a"/>
    <xsl:text>]</xsl:text>
  </a>
</xsl:template>

<xsl:template match="table-entry">
  <xsl:apply-templates mode="tabular"/>
</xsl:template>
```

Первое правило устанавливает режим действия сноски по умолчанию, а второе определяет особый случай для сносок в режиме `tabular`. Режимы различаются только способом форматирования номера сноски. Третье, и последнее, правило относится к ячейке таблицы и включает режим `tabular`.

Важно помнить, что правила без указателя режима не рассматриваются процессором, когда он находится в некотором конкретном режиме, и вместо них используются правила по умолчанию. Это означает, что для каждого элемента, который может быть выбран, придется написать новое правило.

Текст и пробельные символы

XSLT предоставляет три главные классификации вывода: XML, HTML и текст. Между ними есть тонкие отличия форматирования, например, в обработке пробельных символов и предопределенных сущностей. Элемент `<xsl:output>` относится к верхнему уровню и устанавливает классификацию.

Устанавливаемый по умолчанию тип вывода – XML – прост: пробельные символы и предопределенные сущности выводятся в том же виде, как в исходном дереве, поэтому здесь неожиданностей не будет. Если результирующий документ будет иметь тип XML, поместите в таблицу стилей директиву:

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

HTML представляет собой особый случай. Поскольку не все броузеры понимают новые требования XML, этот режим преобразует результирующее дерево в старый стиль, более похожий на SGML, а именно, в HTML версии 4.0. Такие элементы, как `<br/>`, преобразуются в `<br>`, а инструкции обработки заканчиваются ограничителем `>` вместо `?>`. Для вывода содержимого в стиле HTML поместите в таблицу стилей следующую директиву:

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

Однако, если нужна выдача в XHTML, используйте режим XML, а не HTML.

Текстовый режим полезен для генерации документов с разметкой, в корне отличной от XML или HTML; например, такие форматы, как troff и TEX, не используют в разметке теги, заключаемые в < и >. Если установлен текстовый режим выдачи, то процессор строит результирующее дерево, но выдает только текстовые данные. Кроме того, он выводит сущности `&lt;` и `&amp;` в виде их окончательных текстовых эк-

вивалентов, < и &. Для текстового режима используйте в таблице стилей директиву:

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

Объединение таблиц стилей

Могут быть различные причины, чтобы использовать несколько таблиц стилей для одного и того же документа. Например, может иметься несколько документов, в которых в основном применяется один и тот же стиль, но есть некоторые частные отличия. Либо приходится объединять различные пространства имен, каждому из которых назначен собственный стиль. Либо желательно заимствовать некоторые библиотечные стили и заменить те, которые нужно подредактировать. XSLT предоставляет два способа объединения таблиц стилей: включение и импорт.

Включение (*including*) таблицы стилей означает вставку ее содержимого непосредственно в другую таблицу стилей. Все правила и директивы будут обрабатываться так, как если бы они непосредственно присутствовали в таблице стилей. Элемент `<xsl:include>` имеет атрибут `href`, которому присваивается URI включаемой таблицы. Можно помещать этот элемент в любое место таблицы стилей, только не внутрь правила.

Импортирование (*importing*) таблицы стилей несколько сложнее. Импортированные правила имеют меньший вес, чем правила, физически присутствующие в таблице стилей. Существуют расчеты, с помощью которых определяется, которое из правил будет применено, но, в целом, импортированные правила выбираются процессором только тогда, когда в исходной таблице стилей нет правила с аналогичной спецификацией. Элемент `<xsl:import>` тоже использует атрибут `href` для задания таблицы стилей, но поместить его можно только в самое начало таблицы стилей перед всеми другими правилами и директивами.

В чем преимущества такой более слабой формы включения? Она полезна для замены части более полного набора правил с целью настройки результатов. В то время как `<xsl:include>` «вливает» правила на те же уровни приоритетов, что и исходные, `<xsl:import>` предоставляет возможность лучше контролировать другой набор, позволяя применять правила избирательно.

Иногда желательно в некоторой области отдать предпочтение импортированным, а не собственным правилам. Элемент `<xsl:apply-imports>` аналогичен `<xsl:apply-templates>`, но учитывает только импортированные правила, игнорируя физически присутствующие.

Включать и импортировать можно любое число таблиц стилей, что позволяет смешивать для трансформации различные словари. Можно иметь один набор для общего содержимого документа, другой для об-

работки таблиц, еще один для работы с врезками и т. д. Для разрешения возможных конфликтов между правилами из разных наборов учитывается очередность включения: стиль, импортированный ранее, переопределяет стиль, импортированный позднее. Вот как можно импортировать в свою таблицу стилей несколько других:

```
<xsl:stylesheet version="1.0"
                xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:import href="basic_style.xsl"/>
  <xsl:import href="table_styles.xsl"/>
  <xsl:import href="chem_formulae.xsl"/>
  ...
```

Пример: Barebones DocBook

Обратимся к большому примеру. Впервые мы говорили о DocBook в примере 2.4 как образце языка разметки XML. В примере 5.3 мы привели DTD для упрощенной версии типа документа, которую назвали Barebones DocBook. Пример 6.7 показывает, как написать большую таблицу стилей XSLT для преобразования документа Barebones DocBook в HTML.

В этом примере использована программа трансформации XSLT XT Джеймса Кларка, написанная на Java, которую довольно легко установить и использовать. Есть одна специфическая для XT функция, создающая нескольких выходных файлов: элемент `<xt:document>` указывает процессору на необходимость направить выдачу в файл, заданный в атрибуте `href`, с помощью метода, указанного в атрибуте `method` (например, HTML, XML или text). В XSLT для использования специального управляющего элемента, такого как `<xt:document>`, необходимо сначала задать пространство имен, как мы сделали в элементе `<xsl:stylesheet>`.

Если выполнить трансформацию документа из примера 2.4, получится набор страниц HTML: одна для предисловия, две для глав и еще одна для приложения. На рис. 6.3 показана страница из этого нового документа.

Пример 6.7. Таблица стилей XSLT для преобразования Barebones DocBook в HTML

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

<!--

=============================================================================
XSLT Stylesheet to convert DocBook XML into HTML
Copyright 2000 O'Reilly and Associates

Function: Converts DocBook books into formatted HTML files for easy
          viewing.
```

Пример 6.7. Таблица стилей XSLT для преобразования Barebones DocBook в HTML (продолжение)

```
Input: DocBook Lite XML
Output: HTML files

Style: Each chapter is a single page
========================================================================
-->

<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                xmlns:xt="http://www.jclark.com/xt"
                extension-element-prefixes="xt"
                version="1.0">

<!--
```

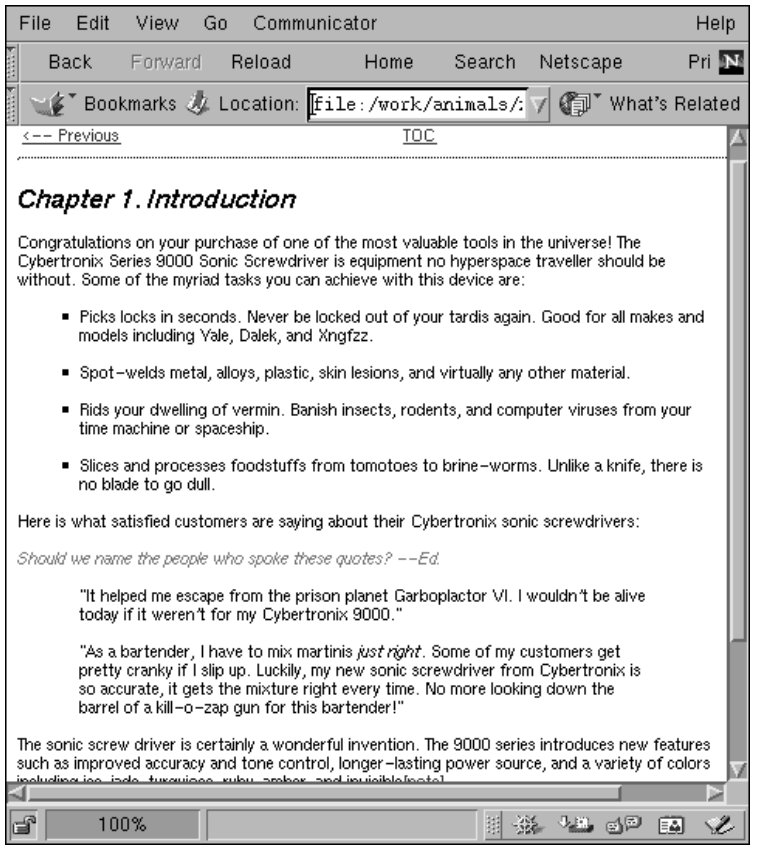


Рис. 6.3. *Страница, полученная при трансформации*

Пример 6.7. Таблица стилей XSLT для преобразования Barebones DocBook в HTML (продолжение)

```
================================================================================
                          HIERARCHICAL ELEMENTS
================================================================================
-->

<!--
match: book
===========
This template is the outermost container for the document.
It sets up a book file (index.html) and front matter files
(copyrght.htm, colophon.htm).

-->
<xsl:template match="book">
  <xt:document href="html/index.html" method="html">
    <xsl:fallback>Error while creating file.</xsl:fallback>
    <html>
      <head>
        <title>
          <xsl:value-of select="title"/>
        </title>
        <link rel="stylesheet" type="text/css" href="../style/style1.css"/>
      </head>
      <body>

        <h1 class="book"><xsl:value-of select="title"/></h1>

        <!-- Book info -->
        <p class="bookinfo">
          <xsl:text>by </xsl:text>
          <xsl:apply-templates select="author"/>
        </p>

        <!-- TOC -->
        <h3 class="tochead">Table of Contents</h3>
        <xsl:call-template name="toc"/>

      </body>
    </html>

    <xsl:apply-templates select="preface"/>
    <xsl:apply-templates select="chapter"/>
    <xsl:apply-templates select="appendix"/>

  </xt:document>
</xsl:template>

<!--
```

Пример 6.7. Таблица стилей XSLT для преобразования Barebones DocBook в HTML (продолжение)

```
toc
===
Build a list of high-level elements

-->
<xsl:template name="toc">

  <xsl:if test="preface">
    <p class="toc">
      <a href="pref.html">Preface</a>
    </p>
  </xsl:if>

  <p class="toc">
    <xsl:for-each select="chapter"/>
      <a>
        <xsl:attribute name="href">
          <xsl:text>ch</xsl:text>
          <xsl:number value="position()" format="01"/>
          <xsl:text>.html</xsl:text>
        </xsl:attribute>
        <xsl:text>Chapter </xsl:text>
        <xsl:value-of select="position()"/>
        <xsl:text>: </xsl:text>
        <i><xsl:value-of select="title"/></i>
      </a>
      <br/>
    </xsl:for-each>
  </p>

  <xsl:if test="count(appendix)>0">
    <p class="toc">
      <xsl:for-each select="appendix"/>
        <a>
          <xsl:attribute name="href">
            <xsl:text>app</xsl:text>
            <xsl:number value="position()" format="a"/>
            <xsl:text>.html</xsl:text>
          </xsl:attribute>
          <xsl:text>Appendix </xsl:text>
          <xsl:number value="position()" format="A"/>
          <xsl:text>: </xsl:text>
          <i><xsl:value-of select="title"/></i>
        </a>
        <br/>
      </xsl:for-each>
    </p>
  </xsl:if>
```

Пример 6.7. Таблица стилей XSLT для преобразования Barebones DocBook в HTML (продолжение)

```
</xsl:template>

<!--
match: appendix
===============
Creates a file for an appendix of the form "appx.html"
where x is the appendix letter.

-->
<xsl:template match="appendix">
  <xsl:variable name="app">
    <xsl:number value="count(preceding::appendix)+1" format="a"/>
  </xsl:variable>
  <xt:document href="app{$app}.html" method="html">
    <xsl:fallback>Error while creating file.</xsl:fallback>
    <xsl:call-template name="file"/>
  </xt:document>
</xsl:template>

<!--
match: chapter
==============
Creates a file for a chapter of the form "chXX.htm"
where XX is the number of the chapter (01, 02, etc.).

-->
<xsl:template match="chapter">
  <xsl:variable name="chap">
    <xsl:number value="count(preceding::chapter)+1" format="01"/>
  </xsl:variable>
  <xt:document href="ch{$chap}.html" method="html">
    <xsl:fallback>Error while creating file.</xsl:fallback>
    <xsl:call-template name="file"/>
  </xt:document>
</xsl:template>

<!--
match: preface
==============
Creates a file for a preface.

-->
<xsl:template match="preface">
  <xt:document href="pref.html" method="html">
    <xsl:fallback>Error while creating file.</xsl:fallback>
    <xsl:call-template name="file"/>
  </xt:document>
</xsl:template>
```

Пример 6.7. Таблица стилей XSLT для преобразования Barebones DocBook в HTML (продолжение)

```
<!--
match: sect1, sect2, sect3
==========================
Create an anchor for linking, then process the contents.

-->
<xsl:template match="sect1|sect2|sect3">
  <xsl:call-template name="drop-anchor"/>
  <xsl:apply-templates/>
</xsl:template>

<!--
drop-anchor
===========
Place an anchor just before the object if it has an ID attribute.

-->
<xsl:template name="drop-anchor">
  <xsl:if test="@id">
    <a>
      <xsl:attribute name="name">
        <xsl:value-of select="@id"/>
      </xsl:attribute>
    </a>
  </xsl:if>
</xsl:template>

<!--
================================================================================
                                HEADS
================================================================================
-->

<xsl:template match="book/title">
  <h1 class="book"><xsl:apply-templates/></h1>
</xsl:template>

<xsl:template match="appendix/title">
  <h1 class="chapter">
    <xsl:text>Appendix </xsl:text>
    <xsl:number value="count(preceding::appendix)+1" format="A"/>
    <xsl:text>.  </xsl:text>
    <xsl:apply-templates/>
  </h1>
</xsl:template>

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

Пример 6.7. Таблица стилей XSLT для преобразования Barebones DocBook в HTML (продолжение)

```
  <h1 class="chapter">
    <xsl:text>Chapter </xsl:text>
    <xsl:value-of select="count(preceding::chapter)+1"/>
    <xsl:text>.  </xsl:text>
    <xsl:apply-templates/>
  </h1>
</xsl:template>

<xsl:template match="part/title">
  <h1 class="chapter">
    <xsl:text>Part </xsl:text>
    <xsl:value-of select="count(preceding::part)+1"/>
    <xsl:text>.  </xsl:text>
    <xsl:apply-templates/>
  </h1>
</xsl:template>

<xsl:template match="preface/title">
  <h1 class="chapter">Preface</h1>
</xsl:template>

<xsl:template match="sect1/title">
  <h2 class="sect1">
    <xsl:choose>
      <xsl:when test="ancestor::appendix">
        <xsl:number value="count(preceding::appendix)+1" format="A"/>
      </xsl:when>
      <xsl:otherwise>
        <xsl:value-of select="count(preceding::chapter)+1"/>
      </xsl:otherwise>
    </xsl:choose>
    <xsl:text>.</xsl:text>
    <xsl:number value="count(../preceding-sibling::sect1)+1"/>
    <xsl:text>. </xsl:text>
    <xsl:apply-templates/>
  </h2>
</xsl:template>

<xsl:template match="sect2/title">
  <h3 class="sect2">
    <xsl:choose>
      <xsl:when test="ancestor::appendix">
        <xsl:number value="count(preceding::appendix)+1" format="A"/>
      </xsl:when>
      <xsl:otherwise>
        <xsl:value-of select="count(preceding::chapter)+1"/>
      </xsl:otherwise>
    </xsl:choose>
```

Пример 6.7. Таблица стилей XSLT для преобразования Barebones DocBook в HTML (продолжение)

```
    <xsl:text>.</xsl:text>
    <xsl:number value="count(../../preceding-sibling::sect1)+1"/>
    <xsl:text>.</xsl:text>
    <xsl:number value="count(../preceding-sibling::sect2)+1"/>
    <xsl:text>. </xsl:text>
    <xsl:apply-templates/>
  </h3>
</xsl:template>

<xsl:template match="sect3/title">
  <h3 class="sect3">
    <xsl:choose>
      <xsl:when test="ancestor::appendix">
        <xsl:number value="count(preceding::appendix)+1" format="A"/>
      </xsl:when>
      <xsl:otherwise>
        <xsl:value-of select="count(preceding::chapter)+1"/>
      </xsl:otherwise>
    </xsl:choose>
    <xsl:text>.</xsl:text>
    <xsl:number value="count(../../../preceding-sibling::sect1)+1"/>
    <xsl:text>.</xsl:text>
    <xsl:number value="count(../../preceding-sibling::sect2)+1"/>
    <xsl:text>.</xsl:text>
    <xsl:number value="count(../preceding-sibling::sect3)+1"/>
    <xsl:text>. </xsl:text>
    <xsl:apply-templates/>
  </h3>
</xsl:template>

<xsl:template match="appendix//figure/title">
  <h4 class="objtitle">
    <xsl:text>Figure </xsl:text>
    <xsl:number value="count(preceding::appendix)+1" format="A"/>
    <xsl:text>-</xsl:text>
    <xsl:number
      value="count(preceding::figure) -
                count(ancestor::appendix/preceding::figure) + 1"/>
    <xsl:text>. </xsl:text>
    <xsl:apply-templates/>
  </h4>
</xsl:template>

<xsl:template match="chapter//figure/title">
  <h4 class="objtitle">
    <xsl:text>Figure </xsl:text>
    <xsl:number value="count(preceding::chapter)+1"/>
    <xsl:text>-</xsl:text>
```

Пример 6.7. Таблица стилей XSLT для преобразования Barebones DocBook в HTML (продолжение)

```
    <xsl:number
      value="count(preceding::figure) -
                count(ancestor::chapter/preceding::figure) + 1"/>
    <xsl:text>. </xsl:text>
    <xsl:apply-templates/>
  </h4>
</xsl:template>

<xsl:template match="appendix//table/title">
  <h4 class="objtitle">
    <xsl:text>Table </xsl:text>
    <xsl:number value="count(preceding::appendix)+1" format="A"/>
    <xsl:text>-</xsl:text>
    <xsl:number
      value="count(preceding::table) -
                count(ancestor::appendix/preceding::table) + 1"/>
    <xsl:text>. </xsl:text>
    <xsl:apply-templates/>
  </h4>
</xsl:template>

<xsl:template match="chapter//table/title">
  <h4 class="objtitle">
    <xsl:text>Table </xsl:text>
    <xsl:number value="count(preceding::chapter)+1"/>
    <xsl:text>-</xsl:text>
    <xsl:number
      value="count(preceding::table) -
                count(ancestor::chapter/preceding::table) + 1"/>
    <xsl:text>. </xsl:text>
    <xsl:apply-templates/>
  </h4>
</xsl:template>

<xsl:template match="title">
  <h4 class="objtitle">
    <xsl:apply-templates/>
  </h4>
</xsl:template>

<!--
========================================================================
                          BLOCK ELEMENTS
========================================================================
-->

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

Пример 6.7. Таблица стилей XSLT для преобразования Barebones DocBook в HTML (продолжение)

```
</xsl:template>

<xsl:template match="blockquote">
  <blockquote class="node">
    <xsl:apply-templates/>
  </blockquote>
</xsl:template>

<xsl:template match="comment">
  <em class="comment"><xsl:apply-templates/></em>
</xsl:template>

<xsl:template match="para">
  <p>
    <xsl:apply-templates/>
  </p>
  <xsl:if test="not(ancestor::table)">
    <xsl:for-each select=".//footnote">
      <blockquote class="footnote">
        <xsl:apply-templates/>
      </blockquote>
    </xsl:for-each>
  </xsl:if>
</xsl:template>

<xsl:template match="programlisting">
  <blockquote>
    <pre class="programlisting">
      <xsl:apply-templates/>
    </pre>
  </blockquote>
</xsl:template>

<xsl:template match="figure">
  <xsl:call-template name="drop-anchor"/>
  <div class="figure">
    <xsl:apply-templates select="graphic"/>
  </div>
  <xsl:apply-templates select="title"/>
</xsl:template>

<xsl:template match="graphic">
  <img alt="figure">
    <xsl:attribute name="src">
      <xsl:value-of select="@fileref"/>
    </xsl:attribute>
  </img>
</xsl:template>
```

Пример 6.7. Таблица стилей XSLT для преобразования Barebones DocBook в HTML (продолжение)

```
<xsl:template match="note">
  <xsl:call-template name="drop-anchor"/>
  <blockquote class="note">
    <xsl:if test="not(title)">
      <h4 class="objtitle">
        <xsl:if test="self::note">
          <xsl:text>NOTE</xsl:text>
        </xsl:if>
      </h4>
    </xsl:if>
    <xsl:apply-templates/>
  </blockquote>
</xsl:template>

<!--
================================================================================
                              FOOTNOTES
================================================================================
-->

<xsl:template match="footnote">
  <a>
    <xsl:attribute name="href">
      <xsl:text>#FOOTNOTE-</xsl:text>
      <xsl:number value="count(preceding::footnote) +1"/>
    </xsl:attribute>
    <xsl:text>[note]</xsl:text>
  </a>
</xsl:template>

<xsl:template match="footnote/para">
  <p>
  <a>
    <xsl:attribute name="name">
      <xsl:text>FOOTNOTE-</xsl:text>
      <xsl:number value="count(preceding::footnote) +1"/>
    </xsl:attribute>
  </a>
    <xsl:if test="count(preceding-sibling::para) =0">
      <xsl:text>[</xsl:text>
      <xsl:number value="count(preceding::footnote) +1"/>
      <xsl:text>]</xsl:text>
    </xsl:if>
    <xsl:apply-templates/>
  </p>
</xsl:template>
```

Пример 6.7. Таблица стилей XSLT для преобразования Barebones DocBook в HTML (продолжение)

```
<!--
============================================================================
                              XREFs
============================================================================
-->

<xsl:template match="xref">
  <xsl:variable name="ident">
    <xsl:value-of select="@linkend"/>
  </xsl:variable>

  <xsl:for-each select="//*[@id=$ident]">
    <a>
      <xsl:attribute name="href">
       <xsl:choose>
          <xsl:when test="ancestor::appendix">
            <xsl:text>app</xsl:text>
            <xsl:number value="count(preceding::appendix)+1"
                        format="a"/>
            <xsl:text>.html</xsl:text>
          </xsl:when>
          <xsl:when test="ancestor::chapter">
            <xsl:text>ch</xsl:text>
            <xsl:number value="count(preceding::chapter)+1"
                        format="01"/>
            <xsl:text>.html</xsl:text>
          </xsl:when>
          <xsl:otherwise>
            <xsl:text>pref.html</xsl:text>
          </xsl:otherwise>
        </xsl:choose>
        <xsl:text>#</xsl:text>
        <xsl:value-of select="$ident"/>
      </xsl:attribute>

      <xsl:choose>
        <xsl:when test="self::appendix">
          <xsl:text>Appendix </xsl:text>
          <xsl:number value="count(preceding::appendix)+1" format="A"/>
          <xsl:text>, "</xsl:text>
          <xsl:value-of select="title"/>
          <xsl:text>"</xsl:text>
        </xsl:when>

        <xsl:when test="self::chapter">
          <xsl:text>Chapter </xsl:text>
          <xsl:value-of select="count(preceding::chapter)+1"/>
          <xsl:text>, "</xsl:text>
```

Пример 6.7. Таблица стилей XSLT для преобразования Barebones DocBook в HTML (продолжение)

```
  <xsl:value-of select="title"/>
  <xsl:text>"</xsl:text>
</xsl:when>

<xsl:when test="self::preface">
  <xsl:text>the Preface</xsl:text>
</xsl:when>

<xsl:when test="self::figure">
  <xsl:text>Figure </xsl:text>
  <xsl:choose>
    <xsl:when test="ancestor::appendix">
      <xsl:number value="count(preceding::appendix) +1"
                  format="A"/>
      <xsl:text>-</xsl:text>
      <xsl:number
        value="count(preceding::figure) -
               count(ancestor::appendix/preceding::figure) +1"/>
    </xsl:when>
    <xsl:when test="ancestor::chapter">
      <xsl:number value="count(preceding::chapter) +1"/>
      <xsl:text>-</xsl:text>
      <xsl:number
        value="count(preceding::figure) -
               count(ancestor::chapter/preceding::figure) +1"/>
    </xsl:when>
    <xsl:otherwise>
      <xsl:text>0</xsl:text>
      <xsl:text>-</xsl:text>
      <xsl:number value="count(preceding::figure) +1"/>
    </xsl:otherwise>
  </xsl:choose>
</xsl:when>

<xsl:when test="self::table">
  <xsl:text>Table </xsl:text>
  <xsl:choose>
    <xsl:when test="ancestor::appendix">
      <xsl:number value="count(preceding::appendix) +1"
                  format="A"/>
      <xsl:text>-</xsl:text>
      <xsl:number
        value="count(preceding::table) -
               count(ancestor::appendix/preceding::table) +1"/>
    </xsl:when>
    <xsl:when test="ancestor::chapter">
      <xsl:number value="count(preceding::chapter) +1"/>
      <xsl:text>-</xsl:text>
```

Пример 6.7. Таблица стилей XSLT для преобразования Barebones DocBook в HTML (продолжение)

```
            <xsl:number
              value="count(preceding::table) -
                     count(ancestor::chapter/preceding::table) +1"/>
          </xsl:when>
          <xsl:otherwise>
            <xsl:text>0</xsl:text>
            <xsl:text>-</xsl:text>
            <xsl:number value="count(preceding::table) +1"/>
          </xsl:otherwise>
        </xsl:choose>
      </xsl:when>

      <xsl:otherwise>
        <xsl:text>{XREF}</xsl:text>
      </xsl:otherwise>

    </xsl:choose>
  </a>
 </xsl:for-each>
</xsl:template>

<!--
filename
========
Return the filename that contains the current node.

-->
<xsl:template name="filename">
 <xsl:choose>
  <xsl:when test="ancestor::appendix">
   <xsl:text>app</xsl:text>
   <xsl:number value="count(preceding::appendix)+1" format="a"/>
   <xsl:text>.htm</xsl:text>
  </xsl:when>
  <xsl:when test="ancestor::chapter">
   <xsl:text>ch</xsl:text>
   <xsl:number value="count(preceding::chapter)+1" format="01"/>
   <xsl:text>.htm</xsl:text>
  </xsl:when>
  <xsl:when test="ancestor::preface">
   <xsl:text>pref.htm</xsl:text>
  </xsl:when>
 </xsl:choose>
</xsl:template>

<!--
```

Пример 6.7. Таблица стилей XSLT для преобразования Barebones DocBook в HTML (продолжение)

```
================================================================================
                                LISTS
================================================================================
-->

<xsl:template match="orderedlist">
  <ol><xsl:apply-templates/></ol>
</xsl:template>

<xsl:template match="orderedlist/listitem">
  <li><xsl:apply-templates/></li>
</xsl:template>

<xsl:template match="itemizedlist">
  <ul><xsl:apply-templates/></ul>
</xsl:template>

<xsl:template match="itemizedlist/listitem">
  <li><xsl:apply-templates/></li>
</xsl:template>

<xsl:template match="variablelist">
  <dl><xsl:apply-templates/></dl>
</xsl:template>

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

<xsl:template match="varlistentry/term">
  <dt><b><xsl:apply-templates/></b></dt>
</xsl:template>

<xsl:template match="varlistentry/listitem">
  <dd><xsl:apply-templates/></dd>
</xsl:template>

<!--
================================================================================
                                TABLES
================================================================================
-->

<!--
match: table, informaltable
===========================
Convert an XML table into an HTML table.
```

Пример 6.7. Таблица стилей XSLT для преобразования Barebones DocBook в HTML (продолжение)

```
-->
<xsl:template match="table|informaltable">
  <xsl:call-template name="drop-anchor"/>
  <xsl:if test="self::table">
    <xsl:apply-templates select="title"/>
  </xsl:if>
  <table border="1">
    <xsl:apply-templates select="tgroup"/>
  </table>
  <xsl:for-each select=".//footnote">
    <blockquote class="footnote">
      <xsl:apply-templates/>
    </blockquote>
  </xsl:for-each>
</xsl:template>

<xsl:template match="tgroup|thead|tbody">
  <xsl:apply-templates/>
</xsl:template>

<xsl:template match="row">
  <tr><xsl:apply-templates/></tr>
</xsl:template>

<xsl:template match="thead//entry">
  <th><xsl:apply-templates/></th>
</xsl:template>

<xsl:template match="entry">
  <td>
    <xsl:if test="@spanname">
      <xsl:attribute name="colspan">
        <xsl:call-template name="getspan">
          <xsl:with-param name="spanname">
            <xsl:value-of select="@spanname"/>
          </xsl:with-param>
        </xsl:call-template>
      </xsl:attribute>
    </xsl:if>
    <xsl:if test="@morerows">
      <xsl:attribute name="rowspan">
        <xsl:value-of select="@morerows"/>
      </xsl:attribute>
    </xsl:if>
    <xsl:apply-templates/>
  </td>
</xsl:template>
```

Пример 6.7. Таблица стилей XSLT для преобразования Barebones DocBook в HTML (продолжение)

```
<xsl:template name="getspan">
  <xsl:param name="spanname" value="span"/>
  <xsl:variable name="colstart">
    <xsl:for-each
      select="ancestor::tgroup/spanspec[@spanname='$spanname']">
      <xsl:value-of select="@namest"/>
    </xsl:for-each>
  </xsl:variable>
  <xsl:variable name="colend">
    <xsl:for-each
      select="ancestor::tgroup/spanspec[@spanname='$spanname']">
      <xsl:value-of select="@nameend"/>
    </xsl:for-each>
  </xsl:variable>
  <xsl:variable name="colstartnum">
    <xsl:for-each select="ancestor::tgroup/colspec[@colname='$colstart']">
      <xsl:value-of select="@colnum"/>
    </xsl:for-each>
  </xsl:variable>
  <xsl:variable name="colendnum">
    <xsl:for-each select="ancestor::tgroup/colspec[@colname='$colend']">
      <xsl:value-of select="@colnum"/>
    </xsl:for-each>
  </xsl:variable>
  <xsl:value-of select="$colendnum - $colstartnum"/>
</xsl:template>

<!--
============================================================================
                          INLINE ELEMENTS
============================================================================
-->

<xsl:template match="acronym">
  <span class="acronym"><xsl:apply-templates/></span>
</xsl:template>

<xsl:template match="application">
  <em class="application"><xsl:apply-templates/></em>
</xsl:template>

<xsl:template match="citetitle">
  <em class="citetitle"><xsl:apply-templates/></em>
</xsl:template>

<xsl:template match="command">
  <tt class="command"><xsl:apply-templates/></tt>
</xsl:template>
```

Пример 6.7. Таблица стилей XSLT для преобразования Barebones DocBook в HTML (продолжение)

```
<xsl:template match="emphasis">
  <em class="emphasis"><xsl:apply-templates/></em>
</xsl:template>

<xsl:template match="emphasis[@role='bold']">
  <b class="emphasis-bold"><xsl:apply-templates/></b>
</xsl:template>

<xsl:template match="filename">
  <em class="filename"><xsl:apply-templates/></em>
</xsl:template>

<xsl:template match="firstterm">
  <em class="firstterm"><xsl:apply-templates/></em>
</xsl:template>

<xsl:template match="function">
  <tt class="function"><xsl:apply-templates/></tt>
</xsl:template>

<xsl:template match="literal">
  <tt class="literal"><xsl:apply-templates/></tt>
</xsl:template>

<xsl:template match="quote">
  <xsl:text>"</xsl:text>
  <xsl:apply-templates/>
  <xsl:text>"</xsl:text>
</xsl:template>

<xsl:template match="sgmltag">
  <tt class="sgmltag-element">
  <xsl:text><</xsl:text>
  <xsl:apply-templates/>
  <xsl:text>></xsl:text>
  </tt>
</xsl:template>

<xsl:template match="symbol">
  <span class="symbol"><xsl:apply-templates/></span>
</xsl:template>

<xsl:template match="systemitem[@class='url']">
  <a>
    <xsl:attribute name="href">
      <xsl:value-of select="."/>
    </xsl:attribute>
```

Пример 6.7. Таблица стилей XSLT для преобразования Barebones DocBook в HTML (продолжение)

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

<!--
============================================================================
                          INDEXTERMS
============================================================================
-->

<xsl:template match="indexterm">
  <a>
    <xsl:attribute name="name">
      <xsl:text>INDEX-</xsl:text>
      <xsl:value-of select="@number"/>
    </xsl:attribute>
  </a>
</xsl:template>

<!--
============================================================================
                          FILE TEMPLATES
============================================================================
-->

<!--
file
====
This template generates a new HTML file to contain a chapter, part,
preface, or appendix.

-->
<xsl:template name="file">
  <html>
    <head>
      <title><xsl:value-of select="title"/></title>
      <link rel="stylesheet" type="text/css" href="../style/style1.css"/>
    </head>
    <body>

      <!-- Top Nav Bar -->

      <xsl:call-template name="nav-bar"/>
      <hr width="515" align="left"/>

      <!-- Body content -->

      <xsl:apply-templates/>
```

Пример 6.7. Таблица стилей XSLT для преобразования Barebones DocBook в HTML (продолжение)

```
      <!-- Bottom Nav Bar -->

      <hr width="515" align="left"/>
      <xsl:call-template name="nav-bar"/>
    </body>
  </html>
</xsl:template>

<!--
nav-bar
=======
Create a set of three navigation buttons: previous, up, and next.

-->
<xsl:template name="nav-bar">
  <div class="navbar">
    <table width="515" border="0">
      <tr>
        <td align="left" valign="top" width="172">
          <a>
            <xsl:attribute name="href">
              <xsl:call-template name="prev-link"/>
            </xsl:attribute>
            <xsl:text><-- Previous</xsl:text>
          </a>
        </td>
        <td align="center" valign="top" width="171">
          <a href="index.html">
            <xsl:text>TOC</xsl:text>
          </a>
        </td>
        <td align="right" valign="top" width="172">
          <a>
            <xsl:attribute name="href">
              <xsl:call-template name="next-link"/>
            </xsl:attribute>
            <xsl:text>Next --></xsl:text>
          </a>
        </td>
      </tr>
    </table>
  </div>
</xsl:template>

<xsl:template name="prev-link">
  <xsl:choose>
    <xsl:when test="preceding-sibling::appendix">
```

Пример 6.7. Таблица стилей XSLT для преобразования Barebones DocBook в HTML (продолжение)

```
      <xsl:text>app</xsl:text>
      <xsl:number value="count(preceding-sibling::appendix)" format="a"/>
      <xsl:text>.html</xsl:text>
    </xsl:when>
    <xsl:when test="preceding-sibling::chapter">
      <xsl:text>ch</xsl:text>
      <xsl:number value="count(preceding-sibling::chapter)" format="01"/>
      <xsl:text>.html</xsl:text>
    </xsl:when>
    <xsl:when test="preceding-sibling::preface">
      <xsl:text>pref.html</xsl:text>
    </xsl:when>
    <xsl:otherwise>
      <xsl:text>index.html</xsl:text>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>

<xsl:template name="next-link">
  <xsl:choose>
    <xsl:when test="self::preface">
      <xsl:text>ch01.html</xsl:text>
    </xsl:when>
    <xsl:when test="self::chapter and following-sibling::chapter">
      <xsl:text>ch</xsl:text>
      <xsl:number value="count(preceding-sibling::chapter)+2" format="01"/>
      <xsl:text>.html</xsl:text>
    </xsl:when>
    <xsl:when test="self::preface and following-sibling::chapter">
      <xsl:text>ch01.html</xsl:text>
    </xsl:when>
    <xsl:when test="self::chapter and following-sibling::appendix">
      <xsl:text>appa.html</xsl:text>
    </xsl:when>
    <xsl:when test="self::appendix and following-sibling::appendix">
      <xsl:text>app</xsl:text>
      <xsl:number value="count(preceding-sibling::appendix)+2"
                  format="a"/>
      <xsl:text>.html</xsl:text>
    </xsl:when>
    <xsl:otherwise>
      <xsl:text>index.html</xsl:text>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>
</xsl:stylesheet>
```

- *Наборы символов и кодировки*
- *Учет языка*

7

Поддержка многоязычности

В этой главе рассказывается о честолюбивых усилиях разработчиков XML, направленных на обеспечение такого уровня разметки, который был бы приемлем в любом месте земного шара. Конечно, проще сказать это, чем сделать. До недавнего времени мысли о поддержке различных языковых стандартов заставляли разработчиков просыпаться в холодном поту. Компьютер нужно было научить распознавать числовые кодировки, выводить правильные глифы и дать пользователям возможность вводить тысячи различных идеограмм с маленькой клавиатуры рабочей станции. В XML, претендующем на статус универсальной системы разметки, обязательно должны быть предусмотрены способы работы с тысячами систем письма, существующими в мире. Как можно решить эту задачу?

В XML начало решения этой проблемы связано с принятием системы кодировки, называемой Unicode, которая может работать с огромным числом различных языков и наборов символов. Во-вторых, XML предоставляет метаданные о языке размечаемого содержимого. Наконец, в сопутствующих технологиях, таких как XSLT и CSS, он позволяет разработчику задавать режим работы в зависимости от языка.

Наборы символов и кодировки

Компьютеры не понимают никаких букв или символов. Все, что они знают, это числа. Любой файл, будь то электронная таблица, письмо или документ XML, внутри компьютера в действительности является длинной строкой двоичных цифр. Данные *кодируются*, что означает представление каждого символа в файле уникальным числом. Программа транслирует вводимые с клавиатуры символы в эти числовые коды, а другая программа транслирует их обратно в текст, понятный человеку.

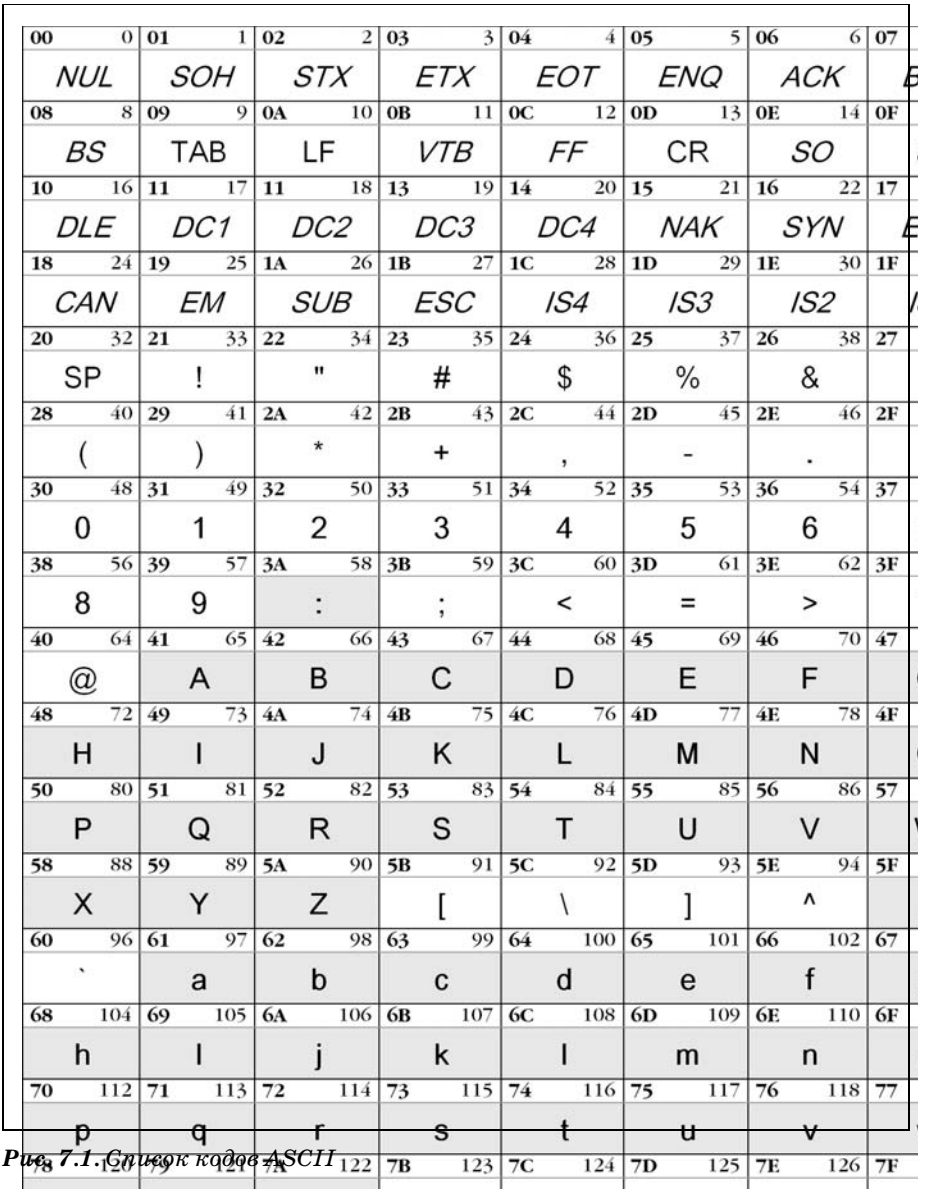
Примером такого процесса является азбука Морзе. Чтобы передать текст по проводам, телеграфист разбивает его на отдельные буквы, числа и символы. Каждый из них он переводит в уникальный эквивалент в азбуке Морзе – последовательность коротких и длинных посылок – и передает в линию. На приемном конце другой телеграфист переводит эти коды снова в текст и записывает сообщение. Отправка электронной почты осуществляется сходным образом: сообщение вводится с клавиатуры, программа транслирует нажатия клавиш в числа, их последовательность отправляется по сети к месту назначения, числа преобразуются обратно в текст и выводятся на экран получателя.

Отображение символов в числовые значения определяет *набор символов* (*character set*). Термин *символ* (*character*) означает любую часть текста или сигнала, которая может быть представлена как отдельная позиция в наборе символов. Например, буква «Q» латинского алфавита является отдельным символом, как и родственная ей строчная «q». То же можно сказать о греческой букве «сигма» (σ) или рунической букве «торн» (þ). Символом может быть произвольный знак, например, ™ или —. В таких языках, как китайский, где целые слова записываются как идеограммы, символ может представлять слово. По существу, все, что представляется как отдельный элемент текста, может быть символом. Компьютер нимало не беспокоится о семантическом смысле: символ является просто числом, которое есть в таблице.

Почтенный и вездесущий ASCII

ASCII (American Standard Code for Information Interchange – Стандартный американский код обмена информацией) – это старый набор символов, все еще популярный сегодня. Коды всех его 128 символов приведены на рис. 7.1. Первоначально разработанный для терминалов телетайпов, ASCII содержит специальные *управляющие символы*, которые не кодируют каких-либо печатаемых символов. Управляющие символы служат сигналами для принимающего устройства, чтобы оно произвело какое-то действие, например, провернуло барабан, возвратило печатающую головку на один символ назад или дало звонок. В настоящее время большинство этих сигналов игнорируется, но некоторые сохранились. Символ возврата каретки по-прежнему используется для завершения строки в файле, а символы табуляции по-прежнему выводятся как серии пробелов. В некоторых программах терминалов символ звонка заставляет машину подать тональный сигнал.

Не удивительно, что, будучи американским изобретением, ASCII содержит лишь чуть больше минимального набора символов, необходимых для кодировки английского текста и, следовательно, не поддерживает другие языки и алфавиты. Тем не менее, для многих современных наборов символов ASCII является основой, на которой строится более обширная совокупность символов.



00 0	01 1	02 2	03 3	04 4	05 5	06 6	07
NUL	SOH	STX	ETX	EOT	ENQ	ACK	
08 8	09 9	0A 10	0B 11	0C 12	0D 13	0E 14	0F
BS	TAB	LF	VTB	FF	CR	SO	
10 16	11 17	11 18	13 19	14 20	15 21	16 22	17
DLE	DC1	DC2	DC3	DC4	NAK	SYN	
18 24	19 25	1A 26	1B 27	1C 28	1D 29	1E 30	1F
CAN	EM	SUB	ESC	IS4	IS3	IS2	
20 32	21 33	22 34	23 35	24 36	25 37	26 38	27
SP	!	"	#	$	%	&	
28 40	29 41	2A 42	2B 43	2C 44	2D 45	2E 46	2F
(	)	*	+	,	-	.	
30 48	31 49	32 50	33 51	34 52	35 53	36 54	37
0	1	2	3	4	5	6	
38 56	39 57	3A 58	3B 59	3C 60	3D 61	3E 62	3F
8	9	:	;	<	=	>	
40 64	41 65	42 66	43 67	44 68	45 69	46 70	47
@	A	B	C	D	E	F	
48 72	49 73	4A 74	4B 75	4C 76	4D 77	4E 78	4F
H	I	J	K	L	M	N	
50 80	51 81	52 82	53 83	54 84	55 85	56 86	57
P	Q	R	S	T	U	V	
58 88	59 89	5A 90	5B 91	5C 92	5D 93	5E 94	5F
X	Y	Z	[	\	]	^	
60 96	61 97	62 98	63 99	64 100	65 101	66 102	67
`	a	b	c	d	e	f	
68 104	69 105	6A 106	6B 107	6C 108	6D 109	6E 110	6F
h	I	j	k	l	m	n	
70 112	71 113	72 114	73 115	74 116	75 117	76 118	77
p	q	r	s	t	u	v	
78 120	79 121	7A 122	7B 123	7C 124	7D 125	7E 126	7F

Рис. 7.1. *Список кодов ASCII*

Размер набора символов представляет собой компромисс между эффективностью расходования памяти (объемом памяти, требуемым для хранения текста) и количеством символов, которые требуется использовать. Из-за того, что компьютер понимает только двоичные числа, размер набора символов обычно является некоторой степенью двух. Символы ASCII представляются двоичными числами, имеющими длину семь цифр (разрядов). Это позволяет иметь в наборе до

2^7=128 символов. Изобретатели ASCII решили, что 128 символов достаточно для их целей, и оставили его на этом уровне.

8-разрядные кодировки

Компьютерные программы быстро переросли рамки 7-разрядного набора символов. Необходимость в европейских символах с акцентами и многих других привела к реализациям 8-разрядных наборов символов. Обычно они совпадают с первоначальными кодами ASCII в младших 128 позициях, но добавляют новые символы в старшую половину. Международная организация стандартизации (International Standards Organization, ISO) определяет много таких наборов символов, представленных в публикациях ISO:

ISO-8859-1 или Latin-1

Набор символов, используемый по умолчанию операционной системой UNIX; ISO-8859-1 содержит большинство западноевропейских букв и знаков.

ISO-8859-2 или Latin-2

Содержит центральноевропейские символы

ISO-8859-4 или Latin-4

Содержит символы ASCII и языков прибалтийских стран.

ISO-8859-5

Содержит символы ASCII и кириллицы.

ISO-8859-6

Содержит символы ASCII и арабские символы.

Это лишь некоторые из имеющихся символов ISO; другие содержат символы греческого, исландского, тайского письма и иврита. Языков и символов столько, что требуется много специализированных 8-разрядных наборов.

Даже в системах, разработанных в одной и той же стране, изобилуют несовместимые наборы. В то время как Unix выбрала стандарт Latin-1, другие платформы используют собственные доморощенные наборы символов. Apple изобрела «MacRoman» для применения в компьютерах Macintosh. Ранние версии операционной системы Microsoft Windows использовали свой набор, названный «Windows ANSI». Это означает, что передача файлов из одной ОС в другую требует некоторой трансляции для сохранения всех символов.

Итак, множества символов объединены в наборы, которые может использовать каждый. Отлично! Проблема в том, что все работают с разными наборами символов. Для тех, кто обычно использует исходные символы ASCII, она может быть незначительна. Лишь несколько символов будут выглядеть «криво», но их можно игнорировать. Но для

тех, кто имеет дело с расширенными наборами символов, возникает серьезная головоломка. Допустим, Дмитрий, используя ISO-8859-5, посылает письмо Марции, почтовая программа которой настроена на раскодирование ISO-8859-1. Даже если Марция понимает русский, у нее все равно проблема: текст выглядит полнейшей тарабарщиной.

По мере того, как Интернет прорывается через межнациональные границы и дает нам возможность нажатием кнопки получать доступ к документам со всего света, необходимость в возможно большей степени устранить такие разногласия становится настоятельной. Очевидно, есть большая потребность в некотором универсальном наборе символов. Эту потребность удовлетворяет Unicode.

Unicode и UCS

Unicode Consortium был основан группой компаний и частных лиц для разработки кодировки символов, включающей все основные виды письма в мире. Отказавшись от ограничений 8-разрядных кодировок, Unicode использует 16 бит, что обеспечивает место для $2^{16} = 65\,536$ символов. В текущей версии стандарта определено почти 50 000 позиций, содержащих алфавитное, слоговое и идеографическое письмо, включая латинское, греческое, кириллическое и тайское, а также 20 000 унифицированных ханьских идеограмм из китайского и японского письма и 11 000 корейских идеограмм хангул.[1] Пустые места зарезервированы для будущих дополнений.

Это было грандиозное предприятие. Тем не менее, Unicode критиковали за то, что сделано было недостаточно. Хотя идеограммы занимают почти треть набора символов, осталось еще 60 000 символов, которые не вошли в Unicode. Для решения этих проблем ISO определила ISO-10646, Universal Character System (UCS), в которой на каждый символ отводится 32 разряда, благодаря чему она может содержать более 2 миллиардов символов. Кажется маловероятным, что кому-либо когда-либо потребуется больше. Разработка UCS еще продолжается, и пока она содержит только копию набора Unicode в своих младших двух байтах.

[1] Не все японцы удовлетворены Unicode по двум причинам. Во-первых, Unicode объединяет похожие иероглифы китайского, японского и корейского языков, но между ними все же есть разница. Создатели Unicode предлагают обойти эту проблему использованием разных шрифтов для каждого из этих языков, но это затрудняет создание многоязычных документов. Во-вторых, сейчас в Unicode определено лишь 20 902 японских иероглифа кандзи из приблизительно 50 000 существующих, а это количество, как считают некоторые японцы, не обеспечивает возможность написания любого японского имени или названия местности. Недостатком иероглифов также обеспокоены и китайцы. – *Примеч. науч. ред.*

XML использует в качестве набора символов Unicode. Этим разрешаются трудности при выборе из несовместимых наборов символов, но возникают другие проблемы. Во-первых, большинство приложений для обработки текста не могут работать с 16-разрядными кодировками, т. к. они создавались в предположении, что размер символа всегда составляет один байт. Во-вторых, если используется только малая часть Unicode, то работа с полными двухбайтовыми символами напрасно расходует память. Если можно обойтись 8-разрядной кодировкой, то размер файла уменьшается вдвое. К счастью, XML предоставляет некоторую гибкость в применении Unicode, позволяющую решить обе эти проблемы.

Подмножества символов

Маловероятно, чтобы для написания какого-то документа кому-то понадобились все 50 000 символов Unicode. По этой причине Unicode может быть разделена на два подмножества, в которых символы для каждого письма сгруппированы вместе. Идея состоит в том, чтобы создать набор символов, основываясь на одном из этих срезов, но преобразовать его кодировку так, чтобы она помещалась в меньшее, 8-разрядное пространство. Символы имеют тот же порядок, но начинаются с меньшего числа. Чтобы различать его от исходного набора, подмножество обычно называется *кодировкой символов* (*character encoding*), а надмножество – *набором символов* (*character set*). Терминология несколько путаная, но ничего не поделаешь.[1]

В заголовках электронной почты и HTTP кодировка символов объявляется в поле с именем `charset`. Этот неудачный выбор имени восходит к временам, когда кодировка символов и набор символов означали одно и то же, поэтому будьте готовы к возможной путанице, если кто-нибудь упоминает наборы символов и кодировки.

В XML вы, вероятно, чаще всего будете встречаться с кодировкой UTF-8 – эффективным способом кодирования документов, состоящих, в основном, из символов ASCII. Часто используемые символы занимают только по одному байту каждый, тогда как более редкие символы Unicode могут выражаться связкой из трех байт. Фактичес-

[1] Если вы действительно хотите помучиться над техническими подробностями, прочтите статью Дэна Конноли (Dan Connolly) «Character Set Considered Harmful», которую можно найти на *http://www.w3.org/MarkUp/html-spec/charset-harmful.html*.

ки, это некоторый тип сжатия, но оказывается возможным его чтение 8-разрядными программами.

Кодировка UTF-8 используется в XML по умолчанию: при отсутствии какой-либо другой информации приложение должно предполагать, что оно будет иметь дело с UTF-8. Это хорошо работает со старыми программами, ничего не знающими о кодировках. Выбирая ту или иную кодировку, следует проявлять осторожность и проверять, могут ли ваши программы работать с вашими документами.

Если бы мир был устроен идеальным образом, ни о чем таком нам знать не потребовалось бы. Текстовый редактор использовал бы кодировку, подходящую для данного документа, и соответствующим образом помечал его, веб-сервер генерировал бы правильный параметр `charset` для этого документа, а приложение-получатель правильно интерпретировало бы его. Но этот мир не совершенен. Приложения не всегда проверяют кодировку и, во всяком случае, реализуют не все кодировки. Неправильное указание кодировки документа приведет к очень странному виду его в редакторе или средстве просмотра XML. По возможности, старайтесь использовать UTF-8; тогда результаты станут более предсказуемы, а ваша жизнь будет значительно легче.

Объявление кодировки

Для указания кодировки своего документа, чтобы другие приложения смогли (будем надеяться) правильно его прочесть, необходимо добавить в объявление XML атрибут `encoding`:

```
<?xml version="1.0" encoding="ISO-8859-5"?>
```

Часто используются следующие кодировки:

`US-ASCII`

Эта почтенная кодировка является надежным выбором, если вы используете только западноевропейский алфавит.

`ISO-8859-1`

Западноевропейская кодировка для UNIX.

`ISO-8859-`*n*

Другие европейские кодировки, которые могут поддерживаться UNIX, а возможно, и Macintosh. Страница, связывающая кодировки и языки, на сайте W3C (*http://www.w3.org/International/O-charset-lang.html*) поможет вам определить, какой язык может использоваться.

`ISO-8859-1-Windows-3.1-Latin-1`

Это кодировка, используемая американской и западноевропейской версиями Microsoft Windows. Она почти совпадает с ISO-8859-1, но

добавляет некоторые полезные знаки пунктуации в области, в которой наборы символов ISO хранят управляющие символы. Данная кодировка известна, кроме того, как кодовая страница 1252, но это не зарегистрированное название кодировки.

`windows-125n`

Другие кодировки Windows. Опять-таки, веб-страница кодировок и языков поможет вам определить, какая кодировка соответствует данному языку.

`UTF-7`

Необычная 7-разрядная кодировка Unicode, которая, вероятно, вам не понадобится. Если при просмотре документа в простом текстовом редакторе на месте не-ASCII-символов стоят плюсы и минусы, то, возможно, используется эта кодировка.

`UTF-8`

8-разрядная кодировка Unicode. Производители операционных систем обещают вскоре обеспечить поддержку Unicode; некоторые операционные системы поддерживают ее в теории, но фактические программные реализации для этих операционных систем являются редкими. Если вам повезло, и у вас есть редактор с поддержкой Unicode, то, возможно, он сохраняет документы в UTF-8. Когда такой документ открывается в текстовом редакторе, не поддерживающем Unicode, латинские символы без акцентов появляются в обычном виде, а другие символы появляются как последовательности из двух и более символов.

`UTF-16`

16-разрядная кодировка Unicode, в которой номера символов точно такие же, как в наборе символов Unicode. В редакторах, поддерживающих Unicode, вероятнее всего, используется эта кодировка.

`ISO-10646-UCS-4`

Unicode фактически представляет собой первую часть ISO/IEC 10646, более крупного набора символов, в котором более 4 миллиардов позиций для символов. Данная кодировка является прямым 32-разрядным представлением этого набора символов.

`ISO-10646-UCS-2`

Это 16-разрядная кодировка более крупного 32-разрядного набора символов ISO/IEC 10646. Это то же самое, что кодировка Unicode UTF-16.

`Shift_JIS`

Это доминирующая кодировка японского языка в Windows. Она также используется некоторыми системами UNIX.

EUC-JP

Кодировка японского языка, используемая многими системами Unix.

Каждый процессор XML обязан понимать UTF-8 и UTF-16, причем первая из них является кодировкой по умолчанию, т. е. если объявление кодировки опущено, то предполагается использование UTF-8.

Это наиболее известные кодировки. Существуют и другие, и в документации к редактору, документации по операционной системе и в ресурсах Интернета вы должны найти, какую метку[1] использовать в своих документах.

Использование символов, не входящих в кодировку

Если у вас достаточно смелости или вы работаете в хорошо управляемой среде, можете попробовать использовать различные кодировки. Первой возникающей при этом проблемой является ввод данных: как ввести нужный символ? Это зависит от применяемого редактора. В Emacs можно нажать клавиши <Ctrl>+<Q>, а затем неинтуитивные клавиши, например, <Meta>+<i> для ввода символа «é». В vi можно нажать клавиши <Ctrl>+<V>, а затем <Meta>+<I>, и получить тот же символ. В Windows надо удерживать клавишу <Alt> и ввести четырехзначное десятичное число, в текущей кодировке соответствующее символу «é» (0233 в кодировке «США/Западная Европа»); на Macintosh надо посмотреть в Key Caps на раскладку для текущего шрифта и определить, с помощью какого сочетания клавиш можно получить желаемый результат (<Option>+<e> и затем <e> для американских систем). Ввод данных в системах, использующих символы, отличные от западноевропейских, оставляется читателю в качестве упражнения.

Следующая задача – определить, какую кодировку фактически использовал редактор при сохранении ваших файлов, чтобы вы могли правильно пометить свой документ. В Unix, вероятно, используется одна из кодировок ISO-8859,[2] что зависит от выбранных вами символов.Аналогично, в Windows, скорее всего, использовалась одна из кодовых страниц Microsoft Windows, но которая именно, зависит от языка. W3C поддерживает частичный список языков и кодировок на

[1] Дело в том, что редактор, с которым работает пользователь, записывает текст в кодировке, принятой в данной системе для его языка, например, для русского – КОИ-8 в Unix или CP1251 в Windows, но пользователю нужно сообщить название кодировки XML-анализатору, который сам не сможет ее определить, если она отлична от кодировок Unicode. – *Примеч. науч. ред.*

[2] Для кириллицы (Россия) в Unix традиционно применяется кодировка KOI8-R. – *Примеч. науч. ред.*

http://www.w3.org/International/O-charset-lang.html. В Macintosh – собственные кодовые страницы; для большинства западноевропейских пользователей (кроме находящихся в Исландии) это просто кодировка `macintosh`. Другие кодировки языков не зарегистрированы в Internet Assigned Numbers Authority (*http://www.iana.org*), поэтому можно надеяться, что ваш Macintosh использует одну из кодировок ISO-8859.

Набором символов для XML по умолчанию является Unicode, который, как упоминалось выше, совпадает с частью International Standard ISO/IEC 10646. Unicode охватывает немалое число символов, большинства из которых нет на американской клавиатуре. Что если нужно вставить «е» с *акутом* (*acute accent*)[1] или малайяламскую «uu»?

Можно каким-то образом ввести символ прямо в файл, но всегда можно использовать числовую ссылку на символ в любом текстовом содержимом или значении атрибута.

Более новым и простым способом сделать это является использование *шестнадцатеричной ссылки на символ*, представляющей собой строку `&#x` с последующими шестнадцатеричным числом и точкой с запятой (`;`). Мы говорим, что это проще, поскольку вам, вероятно, придется искать код для используемой буквы, а справочники Unicode дают шестнадцатеричные значения для кодов символов. Например, малайяламская буква «uu»# имеет шестнадцатеричный номер 0d0a, поэтому шестнадцатеричная ссылка на символ будет иметь вид `&#x0d0a;`. Найти значения символов можно на веб-сайте Unicode, *http://www.unicode.org*, или прямо взять большой и несколько путаный список символов и чисел на *ftp://ftp.unicode.org/Public/UNIDATA/UNIDATA.TXT*.

Более старым и неуклюжим приемом является применение *десятичной ссылки на символ* (*decimal character reference*). Он использует строку `&#` для пометки начала ссылки (обратите внимание на отсутствие `x` из шестнадцатеричной ссылки), за которой идут десятичное число и точка с запятой. Чтобы использовать малайяламскую «uu», нужно перевести шестнадцатеричное 0d0a в десятичное 3338 и сослаться на нее в своем документе так: `&#3338;`. Однако проще придерживаться шестнадцатеричных ссылок.

К несчастью, XML не предоставляет способа включать символы, не входящие в текущую кодировку, в имена элементов и атрибутов. Поэтому, если в именах присутствуют другие символы, помимо неакцентированных латинских букв, цифр и пунктуации (составляющих набор «Основная латиница»), то существует риск разрушения документа при передаче. Конечно, при работе в контролируемой среде, например, в тщательно выстроенном интранете, вы можете делать все, что хотите. Если предполагаемые кодировка и область распространения

[1] Акут, ударение. Диакритический (обособляющий) знак в виде наклонной прямой черты над гласной буквой (Á, á). Имеет разные значения в разных письменных традициях (в русском языке означает ударение). – *Примеч. ред.*

документа таковы, что есть основания рассчитывать на корректную работу с ним каждой обращающейся к нему машины (например, это справедливо для документа ISO-2022-JP, распространяемого преимущественно в Японии), то можете быть спокойны. Но в иных случаях безопаснее использовать только латинские буквы без акцентов в именах и ссылки на символы для всех других символов, имеющихся в ваших данных.

Учет языка

Выбор кодировки символов для документа не определяет, какой язык (или группа языков) может использоваться в документе. Один и тот же набор символов может составлять алфавиты нескольких языков, и если автор документа работает с «чистым» Unicode, то нельзя узнать, говорит он на вьетнамском или итальянском языке. Было бы удобно, если бы документ объявлял веб-серверу или приложению свой язык, чтобы читатель мог сразу узнать, поймет ли он этот документ.

Атрибут xml:lang

XML определяет атрибут `xml:lang` в качестве метки языка для каждого элемента. Никаких официальных действий процессор XML, обнаружив такой атрибут, предпринимать не должен, но можно представить себе некоторые применения его в будущем. Например, поисковые механизмы могут разрабатываться так, чтобы учитывать язык документа и использовать его при распределении результатов своей работы по категориям. Интерфейс поиска сможет тогда иметь меню языков, включаемых или исключаемых при поиске. Другим применением `xml:lang` может быть объединение в одном документе нескольких версий текста, каждая из которых помечена своим языком. Веб-броузер может быть настроен так, чтобы игнорировать все языки, кроме какого-то конкретного, фильтруя документ для вывода только того, что нужно пользователю. Либо, если автор пишет книгу с текстом на разных языках, конфигурировать средство проверки орфографии, чтобы оно использовало для каждого языка свой словарь.

Значением атрибута является строка, содержащая две буквы кода языка, например:

```
xml:lang="en"
```

Код `"en"` обозначает английский язык. Коды языков, стандартизированные в ISO-639, нечувствительны к регистру, поэтому возможных кодов имеется $26^2 = 676$. Трехбуквенные коды тоже есть, но XML распознает лишь двузначные коды; это может породить проблему, поскольку на свете имеются тысячи различных языков, диалектов и поддиалектов.

К счастью, можно также задавать вариант языка с помощью *квалификатора* (*qualifier*), например:

```
xml:lang="en-US"
```

Здесь указан американский вариант английского языка. Обычно принято писать код языка строчными буквами, а квалификатор – прописными. Теперь можно разделять виды английского языка:

```
<para xml:lang="en-US">Please consult the program.</para>
<para xml:lang="en-GB">Please consult the programme.</para>
```

Если по каким-то причинам вам нужно определить собственный язык, можно сделать это с помощью кода языка `x`. Вот некоторые примеры: `x-pascal`, `x-java`, `x-martian`, `x-babytalk`.

Поддержка языков в таблицах стилей

В CSS и XSLT есть проверки, благодаря которым можно задавать разный режим, в зависимости от языка аудитории. Например, в документе может присутствовать элемент, выводимый как замечание с заголовком «CAUTION», генерируемым таблицей стилей. Для перевода на немецкий язык желательно использовать вместо него заголовок «VORSICHT». В следующих разделах описано, как реализовать такое условное действие.

CSS и псевдокласс :lang( )

Каскадные таблицы стилей уровня 2 содержат псевдокласс для добавления в таблицу стилей языковых вариантов. Он определяет язык по атрибуту `xml:lang` или атрибуту `encoding` в объявлении XML. Например, следующее правило изменяет цвет французских элементов `<phrase>` на красный:

```
phrase:lang(fr) { color: 'red'; ; }
```

XSLT и функция lang( )

XSLT также обращает внимание на язык. В Главе 6 «Трансформация: изменение назначения документов» обсуждались булевы функции и их роль в условных правилах шаблонов. Одной из важных функций является `lang()`,возвращающая значение `true`, если язык текущего узла совпадает с языком ее аргумента. Рассмотрим такое правило шаблона:

```
<xsl:template match="para">
  <xsl:choose>
    <xsl:when test="lang('de')">
      <h1>ACHTUNG</h1>
      <xsl:apply-templates/>
```

```
    </if>
    <xsl:otherwise>
      <h1>ATTENTION</h1>
      <xsl:apply-templates/>
    </xsl:otherwise>
</xsl:template>
```

Правило шаблона XSLT выводит слово ACHTUNG, если язык de, или ATTENTION в противном случае. Применим это правило к следующему входному дереву:

```
<warning xml:lang="de">
  <para>Bitte, kein rauchen.</para>
</warning>
```

Элемент <para> наследует свойство языка от содержащего его элемента <warning>, поэтому будет использован первый вариант правила шаблона.

8

- *Обзор XML-программирования*
- *SAX: API, основанный на событиях*
- *Обработка, использующая представление в виде дерева*
- *Заключение*

XML–программирование

Посмотрим правде в глаза. Нельзя вечно сидеть и ждать, пока кто-то создаст совершенную программу, удовлетворяющую всем нашим потребностям: наступит момент, когда придется засучить рукава и сделать ее самому. Но мы не станем учить вас всему, что относится к написанию программ для XML; цель этой главы – дать введение в программные технологии, которые позволят вам работать с XML самым эффективным образом. Изложение будет вестись в кратком и общем духе, чтобы позволить читателю выбрать тот путь, по которому нужно двигаться, а подробности мы оставим другим авторам.

Нет «лучшего» языка или стиля программирования. Есть много способов очистить картофель от кожуры,[1] и это относится к программированию. Некоторые предпочитают делать все на Perl – этом «скотче Интернета», в то время как другие любят программировать на Java, предпочитая его более оформленную и упорядоченную философию. Хотя программисты не могут договориться о единой позиции в отношении кодирования, поддержка XML есть в большинстве языков программирования, используемых в настоящее время. Опять же, выбор средств остается за читателем: в этой главе сосредоточимся на теории.

Сначала мы обсудим синтаксический анализ и обработку XML в общих чертах, обрисовывая доводы «за» и «против» использования XML в качестве средства хранения данных. Затем перейдем к обсуждению технологий обработки XML и рассмотрим пример приложения проверки синтаксиса, написанного на Perl. И, наконец, познакомим вас с рядом готовых компонентов, которые можно использовать в своих прог-

[1] Метафора, дружелюбная к вегетарианцам (и кошкам). ;-)

раммах, и опишем две развивающиеся технологии обработки XML: SAX и DOM.

Обзор XML–программирования

Все чаще и чаще XML применяют для хранения данных. Программные приложения могут использовать XML для хранения настроек и практически любого рода информации – от химических формул до каталогов архивных файлов. Разработчикам следует серьезно рассмотреть преимущества XML, но есть и ограничения, о которых нужно знать.

XML не является идеальным решением для любой задачи хранения данных. Первый недостаток в том, что, по сравнению с другими решениями, время, необходимое для доступа к информации в документе, может быть большим. Реляционные базы данных достигают очень высокого быстродействия благодаря оптимизации с помощью индексов и хеш-таблиц. В XML нет такой оптимизации для быстрого доступа, поэтому для приложений, требующих частого и быстрого поиска информации, реляционная база данных является лучшим выбором, чем XML.

Другая проблема состоит в том, что XML занимает намного больше памяти, чем другие форматы. В нем нет встроенной схемы сжатия. Это значит, что хотя и нет препятствий для сжатия документа XML, но со сжатым документом нельзя будет использовать никакие средства XML. Если объем информации велик, а память (или пропускная способность канала) ограничена, то XML может оказаться не лучшим решением.

Наконец, для некоторых данных просто не требуется структура, устанавливаемая XML. Поэтому его лучше всего применять с текстовыми данными. Он может работать с другими типами данных посредством нотаций и сущностей NDATA, но они недостаточно стандартизованы и могут оказаться неэффективными. Например, растровое изображение обычно хранится как длинная строка двоичных цифр. Она монолитна, не может быть подвергнута синтаксическому анализу и имеет большой размер. Поэтому, если в документе хранятся только двоичные данные, особой надобности в разметке XML не видно.

Несмотря на все это, XML открывает большие возможности для программистов. Он хорошо приспособлен для чтения, записи и изменения программным обеспечением. Его синтаксис прост и может легко анализироваться. В нем определены правильно построенные документы, что сокращает объем программной проверки ошибок и обработки исключительных ситуаций. Он хорошо документирован, и разработчикам доступны многие инструменты и программные библиотеки. А в качестве открытого стандарта, принятого как финансовыми учреждениями, так и хакерами движения открытых исходных текстов (open source), при поддержке практически всеми популярными языками

программирования, XML имеет хорошие шансы стать lingua franca[1] коммуникаций между компьютерами.

Схема процессора XML

В предыдущих главах мы рассматривали процессор XML как черный ящик, в щель которого опускается документ XML, а с другого конца выходит нечто (возможно, напечатанный документ или отображение веб-страницы). Очевидно, это упрощенный взгляд, который не приближает нас к пониманию механизма работы процессоров XML. Итак, вскроем этот черный ящик и ознакомимся с его содержимым.

Типичный процессор XML состоит из компонентов, каждый из которых выполняет важное действие в конвейере обработки. На каждом шаге данные уточняются еще больше по мере их приближения к окончательному формату. Процесс начинается с синтаксического анализа документа, в результате которого исходные данные преобразуются в маленькие пакеты информации, пригодные для обработки следующим компонентом, коммутатором событий (event switcher). Коммутатор событий направляет пакеты процедурам обработки событий, в которых осуществляется большая часть работы. В более мощных программах процедуры обработки строят в памяти древовидную структуру, чтобы процессор дерева (tree processor) мог, обрабатывая его, создать конечную выдачу в требуемом формате.

Теперь опишем компоненты процессора XML более подробно:

Синтаксический анализатор

В каждом процессоре XML есть синтаксический анализатор (parser). Его задача состоит в транслировании разметки XML и данных в поток маленьких крупиц, называемых *лексемами* (*tokens*), которые будут использоваться в обработке. Лексема может быть открывающим тегом элемента, строкой символьного содержания, начальным ограничителем инструкции обработки или какой-то другой частью разметки, указывающей, что область документа сменилась. На этом этапе разрешаются все ссылки на сущности. Этот приведенный в порядок информационный поток управляет следующим компонентом, коммутатором событий.

Коммутатор событий

Коммутатор событий получает от анализатора поток лексем и сортирует их согласно назначению, как в старину телефонистка на коммутаторе. Некоторые лексемы сообщают о том, что необходимо изменить режим. Они называются *событиями* (*events*). Одно событие

[1] Смешанный язык из элементов романских, греческого и восточных языков, служащий для общения в восточном Средиземноморье; жаргон (pidgin) с ограниченной сферой применения. – *Примеч. перев.*

может состоять в том, что обнаружена инструкция обработки с ключевым словом цели, имеющим важность для процессора XML. Другое может состоять в том, что найден элемент <title> и необходимо изменить шрифт. Что является событиями и как они должны обрабатываться, решает каждый конкретный процессор. При получении события он направляет обработку в подпрограмму, которую называют *обработчиком события* (*event handler*) или, иногда, процедурой *обратного вызова* (*call-back*). Часто это все, что должен сделать процессор XML, но иногда требуется более сложная обработка, например, построение внутренней модели дерева и работа с ней.

Представление в виде дерева

Обработчик событий является простым механизмом, который «забывает» о событиях после того, как их увидит. Однако в некоторых задачах требуется, чтобы структура документа сохранялась в памяти в качестве модели для непоследовательных операций, например, для перемещения узлов или разрешения внутренних перекрестных ссылок. Для обработки такого типа программа должна построить внутреннее представление дерева. Процедуры обратного вызова, запускаемые событиями в обработчике событий, просто добавляют к дереву узлы, пока не кончатся все события. После этого программа работает с деревом, а не с потоком событий. Эта стадия обработки осуществляется процессором правил.

Представление в виде дерева может иметь различный формат, но основными являются два типа. Первый является простой структурой, образованной иерархией списков узлов. Такого типа структуру вы обнаружите в подходе, не использующем объекты и показанном в примере 8.1. Другой тип представления называется *объектной моделью* (*object model*); каждый узел в этом случае представляется в виде объекта. На языке программистов *объектом* называют пакет данных и процедур, образующих жесткую и непрозрачную структуру. Такой стиль предпочтительнее в больших программах, поскольку он позволяет минимизировать количество ошибок некоторых типов и обычно облегчает зрительное представление. Деревья объектов дороже обходятся в отношении скорости и памяти, но во многих случаях это является приемлемой платой за удобство разработки.

Процессор дерева

Процессор дерева является частью программы, которая работает с древовидной моделью. Он может быть чем угодно – от средства проверки действительности документа до полновесного механизма трансформаций. Он обходит дерево, обычно методически в порядке «сначала вглубь», при котором проходит до конца ветви, а затем возвращается, до обнаружения первого не посещенного узла. Часто его действиями управляет список правил, где *правило* (*rule*) является некоторым описанием способа обработки участка дерева. Напри-

мер, процессор дерева может руководствоваться правилами, имеющимися в таблице стилей, чтобы преобразовать разметку XML в форматированный текст.

Теперь рассмотрим конкретный пример. В следующем разделе представлен пример простого процессора XML, написанного на языке сценариев Perl.

Пример: средство проверки синтаксиса XML

Согласно нашей схеме компонентов процессора XML, приведенной в предыдущем разделе, простая программа содержит только синтаксический анализатор и коммутатор событий; с помощью двух этих составляющих можно реализовать многое. В примере 8.1 приведено средство проверки синтаксиса XML – того, к чему должен быть доступ у любого пользователя XML. Если документ является некорректным, средство проверки синтаксиса извещает об этом и точно указывает место, в котором находится ошибка.

Программа написана на Perl, хорошем языке обработки текста для небольших приложений. Perl использует операторы анализа строк, называемые *регулярными выражениями* (*regular expressions*). Регулярные выражения выполняют сложные задачи синтаксического анализа с минимальными усилиями, хотя поначалу их синтаксис кажется трудным для чтения.[1] Этот пример не является ни эффективным, ни элегантным по конструкции, но для задач обучения его должно оказаться достаточно. Конечно, в обычных условиях вы не станете писать собственный синтаксический анализатор, а воспользуетесь тем, который уже кем-то создан. Для всех языков, в том числе Perl, есть свободно доступные анализаторы XML, которыми можно пользоваться. Поскольку над ними идет постоянная работа, они, скорее всего, будут действовать быстрее и содержать меньше ошибок, чем то, что вы напишете сами.

Программа основывается на синтаксическом анализаторе XML. В качестве аргумента командной строки программе передается имя файла XML, содержащего элемент документа. Запоминаются объявления внешних сущностей, чтобы можно было разрешать ссылки на внешние сущности. Анализатор помещает содержимое всех файлов в один буфер, затем построчно проходит буфер и проверяет синтаксис. Последовательности команд `if` ищут в буфере известные ограничители. При каждом удачном поиске весь объект разметки удаляется из буфера и цикл повторяется, пока буфер не окажется пустым.

[1] По этой теме есть хорошая книга Джеффри Фридла (Jeffrey Friedl) «Mastering Regular Expressions» (O'Reilly).

Если анализатор не может что-то распознать, это вызывает ошибку синтаксического анализа. Анализатор сообщает, чем, по его мнению, вызвана ошибка, исходя из того, в каком месте последовательности операторов `if` была обнаружена ошибка. Он также выводит имя файла и номер строки, в которой произошла ошибка. Эта информация вставляется в начало каждой строки той частью программы, которая считывает файлы.

Если документ построен правильно (корректен), программа переходит к подсчету узлов и в итоге выводит распределение частот типов узлов и типов элементов. Это демонстрирует способность программы различать типы событий.

В примере 8.1 приведен листинг программы с именем *dbstat*. Для того чтобы проверить ее работу, настройте первую строку так, чтобы она отражала действительное местонахождение интерпретатора Perl на вашей машине.

Пример 8.1. Листинг кода программы dbstat для проверки синтаксиса XML

```
#!/usr/local/bin/perl -w
#

use strict;

# Глобальные переменные
#
my %frefs;              # файловые сущности, объявленные во внутреннем
подмножестве
my %element_frequency;  # список частот элементов
my $lastline = "";      # последняя проанализированная строка
my $allnodecount = 0;   # суммарное количество проанализированных узлов
my %nodecount =         # количество проанализированных узлов
                            (
                             'attribute'   => 0,
                             'CDMS'        => 0,
                             'comment'     => 0,
                             'element'     => 0,
                             'PI'          => 0,
                             'text'        => 0,
                            );

# начало обработки
&main();

# main
# ----
# синтаксический анализ документа XML и вывод статистики
#
sub main {
    # прочесть документ, начиная с файла верхнего уровня
```

Пример 8.1. Листинг кода программы dbstat для проверки синтаксиса XML (продолжение)

```
    my $file = shift @ARGV;
    unless($file && -e $file) {
        print "Файл '$file' не найден.\n";
        exit(1);
    }
    my $text = &process_file($file);

    # анализ сущности документа
    &parse_document($text);

    # вывод статистики узлов
    print "\nNode frequency:\n";
    my $type;
    foreach $type (keys %nodecount) {
        print "  " . $nodecount{ $type } . "\t" . $type . " nodes\n";
    }
    print "\n  " . $allnodecount . "\ttotal nodes\n";

    # вывод статистики элементов
    print "\nElement frequency:\n";
    foreach $type (sort keys %element_frequency) {
        print "  " . $element_frequency{ $type } . "\t<" . $type . ">\n";
    }
}

# process_file
# ------------
# Получить тексты всех файлов XML в документе.
#
sub process_file {
    my($file) = @_;
    unless(open(F, $file)) {
        print STDERR "Невозможно открыть \"$file\" для чтения.\n";
        return "";
    }
    my @lines = <F>;
    close F;
    my $line;
    my $buf = "";
    my $linenumber = 0;
    foreach $line (@lines) {

        # Вставить данные о номере строки и имени файла
        $linenumber ++;
        $buf .= "%$file:$linenumber%";
        # Заменить ссылки на внешние сущности содержимым файлов
        if($line =~ /\&([^;]+);/ && $frefs{$1}) {
            my($pre, $ent, $post) = ($`, $&, $');
            my $newfile = $frefs{$1};
```

Пример 8.1. Листинг кода программы dbstat для проверки синтаксиса XML (продолжение)

```
            $buf .= $pre . $ent . "\n<?xml-file startfile: $newfile ?>" .
                &process_file($frefs{$1}) . "<?xml-file endfile ?>" .
                    $post;
        } else {
            $buf .= $line;
        }

        # Добавить объявленные внешние сущности в список.
        # ЗАМЕЧАНИЕ: идентификаторы типа PUBLIC не обрабатываются!
        $frefs{ $1 } = $2
            if($line =~ /<!ENTITY\s+(\S+)\s+SYSTEM\s+\"([^\"]+)/);
    }
    return $buf;
}

# parse_document
# --------------
# Прочесть узлы верхнего уровня документа.
#
sub parse_document {
    my($text) = @_;
    while($text) {
        $text = &get_node($text);
    }
}

# get_node
# --------
# Получив фрагмент текста XML, возвратить первый найденный узел
# и оставшуюся часть строки текста.
#
sub get_node {
    my($text) = @_;

    # текст
    if($text =~ /^[^<]+/) {
        $text = $';
        $nodecount{ 'text' } ++;

    # императивная разметка: комментарий, помеченная секция, объявление
    } elsif($text =~ /^\s*<\!/) {

        # комментарий
        if($text =~ /^\s*<\!--(.*?)-->/s) {
            $text = $';
            $nodecount{ 'comment' } ++;
            my $data = $1;
            if($data =~ /--/) {
```

Пример 8.1. Листинг кода программы dbstat для проверки синтаксиса XML (продолжение)

```
            &parse_error("в комментарии содержится неполный
            разделитель (--)");
        }

    # секция, помеченная CDATA (обрабатывать как узел)
    } elsif($text =~ /^\s*<\!\[\s*CDATA\s*\[/) {
        $text = $';
        if($text =~ /\]\]>/) {
            $text = $';
        } else {
            &parse_error("синтаксис CDMS ");
        }
        $nodecount{ 'CDMS' } ++;

    # объявление типа документа
    } elsif($text =~ /^\s*<!DOCTYPE.*?\]>\s*/s ||
             $text =~ /^\s*<!DOCTYPE.*?>\s*/s) {
        $text = $';

    # parse error
    } else {
        &parse_error("синтаксис объявления");
    }

# инструкция обработки
} elsif($text =~ /^\s*<\?/) {
    if($text =~ /^\s*<\?\s*[^\s\?]+\s*.*?\s*\?>\s*/s) {
        $text = $';
        $nodecount{ 'PI' } ++;
    } else {
        &parse_error("синтаксис PI ");
    }

# элемент
} elsif($text =~ /\s*</) {

    # пустой элемент с атрибутами
    if($text =~ /^\s*<([^\/\s>]+)\s+([^\s>][^>]+)\/>/) {
        $text = $';
        $element_frequency{ $1 } ++;
        my $atts = $2;
        &parse_atts($atts);

    # пустой элемент без атрибутов
    } elsif($text =~ /^\s*<([^\/\s>]+)\s*\/>/) {
        $text = $';
        $element_frequency{ $1 } ++;

    # элемент-контейнер
```

Пример 8.1. Листинг кода программы dbstat для проверки синтаксиса XML (продолжение)

```
        } elsif($text =~ /^\s*<([^\/\s>]+)[^<>]*>/) {
            my $name = $1;
            $element_frequency{ $name } ++;

            # обработать атрибуты
            my $atts = "";
            $atts = $1 if($text =~ /^\s*<[^\/\s>]+\s+([^\s>][^>]+)>/);
            $text = $';
            &parse_atts($atts) if $atts;
            # обработать дочерние узлы
            while($text !~ /^<\/$name\s*>/) {
                $text = &get_node($text);
            }
            # получить закрывающий тег
            if($text =~ /^<\/$name\s*>/) {
                $text = $';
            } else {
                &parse_error("закрывающий тег элемента <$name>");
            }
            $nodecount{ 'element' } ++;

        # некоторые ошибки, выявленные синтаксическим анализом
        } else {
            if($text =~ /^\s*<\/([^>]+)/) {
                &parse_error("отсутствует открывающий тег элемента <$1>");
            } else {
                &parse_error("зарезервированный символ (<) в тексте");
            }
        }

    } else {
        &parse_error("текст не идентифицирован");
    }

    # обновить текущую информацию
    $allnodecount ++;
    $lastline = $& if($text =~ /%[:]+:[:]+/);
    return $text;
}

# parse_atts
# ----------
# проверить синтаксис атрибутов
#
sub parse_atts {
    my($text) = @_;
    $text =~ s/%.*?%//sg;
    while($text) {
```

Пример 8.1. Листинг кода программы dbstat для проверки синтаксиса XML (продолжение)

```
        if($text =~ /\s*([^\s=]+)\s*=\s*([\"][^\"]*[\"])/ ||
            $text =~ /\s*([^\s=]+)\s*=\s*([\'][^\']*[\'])/) {
            $text = $';
            $nodecount{'attribute'} ++;
            $allnodecount ++;
        } elsif($text =~ /^\s+/) {
            $text = $';
        } else {
            &parse_error("attribute syntax");
        }
    }
}

# parse_error
# -----------
# прекратить синтаксический анализ и вывести сообщение об ошибке с указанием
# номера строки и имени файла, где она встретилась
#
sub parse_error {
    my($reason) = @_;
    my $line = 0;
    my $file = "неизвестный файл";
    if($lastline =~ /%([^:]+):([^%]+)/) {
        $file = $1;
        $line = $2 - 1;
    }
    die("Ошибка синтаксического анализа в строке $line
   файла $file: $reason.\n");
}
```

Эта программа осуществляет два прохода по документу, дважды просматривая текст во время обработки. На первом проходе разрешаются внешние ссылки, чтобы построить документ из всех файлов. На втором проходе осуществляется фактический синтаксический разбор и текст превращается в лексемы. Можно было бы выполнить все за один проход, но это усложнило бы программу, поскольку при обнаружении каждой ссылки на внешнюю сущность пришлось бы приостанавливать анализ и загружать текст. При двух проходах анализатор может считать, что ко второму проходу весь текст загружен.

С этой программой возникают две проблемы. Во-первых, она оставляет неразрешенными общие сущности, что годится для простого текста, но плохо, если сущности содержат разметку. Узлы внутри общих сущностей не будут проверены и подсчитаны, что может привести к пропуску синтаксических ошибок и искажению статистики. Во-вторых, программа не умеет обрабатывать открытые идентификаторы во внешних сущностях и предполагает, что все они являются системными идентификаторами. Это может привести к пропуску разметки.

Подпрограмма process_file начинает с чтения всего документа XML, включая разметку в главном файле и внешних сущностях. При чтении каждая строка добавляется в память буфера. Читая каждую строку, подпрограмма ищет объявления внешних сущностей и добавляет имя каждой сущности и соответствующего файла в хеш-таблицу. Позднее, если она обнаруживает ссылку на внешнюю сущность, то находит соответствующий файл и обрабатывает его таким же образом, добавляя строки в буфер текста.

Когда буфер заполнен, его начинает анализировать подпрограмма parse_document. Она читает поочередно все узлы с помощью подпрограммы get_node. Так как не требуется никакой обработки, кроме подсчета узлов, нет необходимости передавать узлы как лексемы или добавлять их в дерево объектов. При анализе эта подпрограмма вырезает из буфера текст каждого узла и останавливается, когда буфер оказывается пуст.[1]

Затем get_node находит следующий узел в буфере текста. Она использует регулярные выражения для проверки наличия среди нескольких первых символов ограничителей разметки XML. Если первый символ не является угловой скобкой (<), подпрограмма предполагает, что это текстовый узел и ищет дальше очередной ограничитель. Найдя угловую скобку, она смотрит дальше, чтобы сузить возможный тип тега: комментарий, секция CDATA или объявление, если следующий символ является восклицательным знаком; инструкция обработки, если это вопросительный знак; иначе – это элемент. Затем подпрограмма пытается найти конец тега или, если это элемент, ищет закрывающий тег.

Объект разметки, являющийся элементом, представляет особую проблему: концевой тег трудно найти, если элемент имеет смешанное содержимое. Можно представить себе ситуацию, когда элемент оказывается вложенным в элемент того же типа; это ввело бы в заблуждение анализатор, который искал бы впереди только закрывающий тег. Решение состоит в рекурсивном вызове get_node необходимое количество раз, чтобы найти всех потомков элемента. Если она обнаруживает закрывающий тег вместо целого узла, значит найден весь элемент.

Вот выдача, получаемая, когда *dbstat* применяется к файлу *checkbook.xml*, нашему примеру из главы 5 «Модели документов: более высокий уровень управления». Поскольку *dbstat* вывела статистику, мы знаем, что документ правильно построен:

```
> dbstat checkbook.xml

Node frequency:
  17       attribute nodes
```

[1] Передача ссылки на буфер текста вместо самой строки, возможно, сделала бы программу значительно быстрее.

```
  73        text nodes
  0         comment nodes
  1         PI nodes
  35        element nodes
  0         CDMS nodes

  127       total nodes

Element frequency:
  7         <amount>
  1         <checkbook>
  7         <date>
  1         <deposit>
  7         <description>
  5         <payee>
  6         <payment>
  1         <payor>
```

Если бы документ не был правильно построенным (корректным), мы увидели бы вместо списка сообщение об ошибке, например:

```
> dbstat baddoc.xml
Parse error on line 172 in baddoc.xml: missing start tag for
element <entry>.
```

Очевидно, проблема в этом файле была в строке 172:

```
<entry>42</entry><entry>*</entry>            строка 170
<entry>74</entry><entry>J</entry>            строка 171
entry>106</entry><entry>j</entry></row>      строка 172
```

Использование готовых компонентов

К счастью, не обязательно мучиться с написанием собственного анализатора. С каким бы языком вы ни работали, скорее всего, имеется общедоступный анализатор. Некоторые популярные анализаторы перечислены в табл. 8.1.

Таблица 8.1. Анализаторы XML

Язык	Библиотека	Где взять
Perl	XML::Parser	*http://www.cpan.org/modules/by-module/XML / perl-xml-modules.html*
Java	Xerces	*http://xml.apache.org/dist/xerces-j*
	XP Джеймса Кларка	*http://www.jclark.com/xml/xp/index.html*

Таблица 8.1 (продолжение)

Язык	Библиотека	Где взять
	Java API для анализа XML (JAXP)	*http://www.javasoft.com/xml/download.html*
Python	PyXML	*http://www.python.org/doc/howto/xml/*
JavaScript	Xparse	*http://www.jeremie.com/Dev/XML /*
C/C++	IBM Alphaworks XML для C	*http://www.alphaworks.ibm.com/tech/xml4c*
	Microsoft XML Parser для C++	*http://msdn.microsoft.com/xml/IE4/cparser.asp*

SAX: API, основанный на событиях

С тех пор, как на сцене появился XML, были созданы сотни продуктов XML – от средств проверки действительности документов до редакторов и цифровых систем управления активами. Все эти продукты имеют общие черты: они работают с файлами, анализируют XML и обрабатывают разметку XML. Разработчикам известно, что изобретение велосипеда в программном обеспечении обходится дорого, но именно этим они и занимались с продуктами XML. Вскоре стала очевидной необходимость в *интерфейсе прикладного программирования* (API) для работы с XML.

API представляет собой основу для написания программ, которая берет на себя проблемы низкого уровня и позволяет разработчику сосредоточиться на действительной сути программы. API XML занимается такими вещами, как чтение файлов, синтаксический анализ и пересылка данных обработчикам событий, в то время как вы сами пишете программы обработки событий.

Simple API for XML (SAX) является попыткой определить стандартный, основанный на событиях API XML (смотрите приложение B «Таксономия стандартов»). В этом проекте участвовали некоторые пионеры XML. Сотрудничество осуществлялось через список рассылки XML-DEV, и окончательным результатом стал пакет Java с именем *org.xml.sax*. Это хороший пример эффективной совместной работы группы людей по разработке системы – все вместе заняло пять месяцев.

SAX основан на модели, управляемой событиями, использующей функции обратного вызова для осуществления обработки. Представление в виде дерева не создается, и обработка происходит за один проход по документу. Можете представлять себе это как «последовательный доступ» к XML: программа не может перескакивать в произ-

вольное место документа. С одной стороны, утрачивается гибкость работы с хранимым в памяти представлением, что ограничивает круг решаемых задач. С другой стороны, достигается огромная скорость и используется очень мало памяти.

Высокая скорость работы SAX делает его идеальным для обработки XML на сервере, например, для трансляции документа XML в HTML, который можно просматривать с помощью броузера. Управляемая событиями программа также может:

- Искать в документе элемент, в содержимом которого есть ключевое слово.
- Выводить форматированное содержимое в порядке его появления.
- Модифицировать документ XML, осуществляя в нем небольшие изменения, такие как исправление орфографии и переименование элементов.
- Считывать данные для создания внутреннего представления или сложной структуры данных. Иными словами, простой API можно использовать в качестве основы для более сложного API, такого как DOM, о котором будет говориться ниже в разделе «Объектная модель документа».

Однако малый расход памяти влечет и обязательства, и SAX забывает о событиях так же быстро, как и создает их. Вот некоторые вещи, которые не просто осуществлять программе, управляемой событиями:

- Изменять порядок элементов в документе.
- Разрешать перекрестные ссылки между элементами.
- Проверять ссылки `ID-DREF`.
- Проверять состоятельность документа XML.

Несмотря на свои ограничения, основанный на событиях API является мощным инструментом для обработки документов XML. Чтобы лучше объяснить, что такое событие, рассмотрим пример. Допустим, что есть документ:

```
<?xml version="1.0"?>
<record id="44456">
 <name>Bart Albee</name>
 <title>Scrivenger</title>
</record>
```

Управляемый событиями интерфейс разбирает файл один раз и сообщает о следующих событиях в последовательном порядке:

1. `Найден начальный элемент: record`
2. `Найден атрибут: id = "44456"`
3. `Найден начальный элемент: name`
4. `Найден текст`

5. `Найден конечный элемент: name`

6. `Найден начальный элемент: title`

7. `Найден текст`

8. `Найден конечный элемент: title`

9. `Найден конечный элемент: record`

При возникновении каждого события программа вызывает соответствующий ему обработчик. Обработчики событий работают подобно графическому интерфейсу, который тоже управляется событиями, т. к. одна функция обрабатывает щелчок кнопкой мыши, другая обрабатывает нажатие клавиши и т. д. В случае SAX каждый обработчик события обрабатывает такие события, как начало элемента или появление инструкции обработки.

Реализация SAX на Java проиллюстрирована на рис. 8.1.

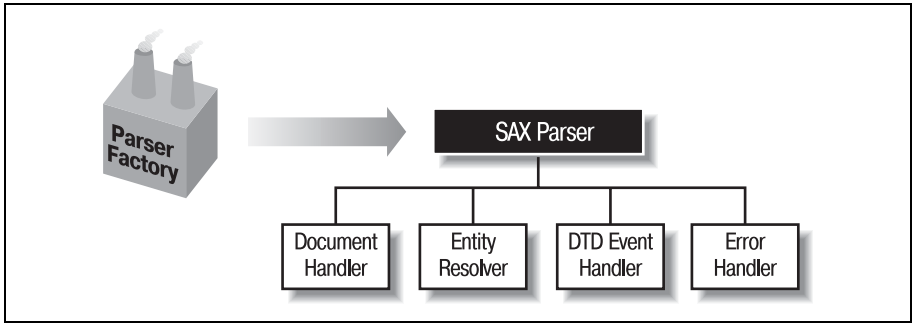


***Рис. 8.1.** API SAX на Java*

Объект *ParserFactory* создает рабочую структуру на основе выбранного анализатора (SAX позволяет использовать излюбленный вами анализатор Java, будь то XP, Xerces или JAXP). Он выполняет синтаксический анализ документа, при необходимости вызывая интерфейсы *Document Handler*, *Entity Resolver*, *DTD Handler* и *Error Handler*. В Java *интерфейс* является совокупностью процедур, или *методов*, в классе. Интерфейс обработчика документа – это место, в которое разработчик помещает код своей программы. Внутри обработчика документа необходимо реализовать методы, обрабатывающие элементы, атрибуты и все прочие события, поступающие при анализе документа XML.

Интерфейс, основанный на событиях, может быть использован при построении API, основанного на дереве, как мы увидим в следующем разделе. Это увеличивает возможности SAX по созданию постоянно находящейся в памяти модели документа, с помощью которой осуществляется более гибкая обработка.

Обработка, использующая представление в виде дерева

Когда одного или двух проходов через документ программе оказывается недостаточно, может потребоваться построить представление документа в виде дерева и хранить его в памяти, пока не будет закончена обработка. Если обработка, основанная на событиях, предоставляет последовательный доступ к XML, то обработка, основанная на дереве, предоставляет произвольный доступ. Программа получает возможность произвольного перемещения по документу, поскольку освобождается от линейного однопроходного пути.

Пример: простой инструмент трансформации

Программа из примера 8.2 расширяет анализатор примера 8.1, создавая древовидную структуру данных. После того как дерево построено, программа обработки один за другим обходит его узлы, применяя правила, для того чтобы модифицировать их. Эта версия называется *dbfix*.

Осуществить обход дерева нетрудно. Для каждого узла, если у него есть дочерние узлы (например, это не пустой элемент), обрабатываются все дочерние, действие повторяется для их дочерних, и так далее, пока не будут достигнуты листья дерева. Этот процесс называется *рекурсией*, и его отличительной чертой является программа, которая вызывает саму себя. Подпрограмма `process_tree()` является рекурсивной, поскольку вызывает себя для каждого из дочерних узлов непустого элемента. Программа, выводящая дерево в файл, `serialize_tree()`, тоже использует рекурсию таким способом.

Эта программа осуществляет ряд трансформаций элементов, запрограммированных в правилах. Хеш-таблица `%unwrap_element` содержит правила для «развертывания» элементов в определенном контексте. *Развернуть* (*unwrap*) элемент означает удалить открывающий и закрывающий теги, оставив на месте содержимое. Правила развертывания задаются показанным ниже присвоением.

```
my %unwrap_element =    # разворачиваемые узлы: контекст => [узел]
                        (
                         'screen' => ['literal'],
                         'programlisting' => ['literal'],
                         );
```

Ключ хеш-таблицы – элемент контекста, являющийся родительским для развертываемого элемента. Значением ключа является список элементов, которые нужно развернуть. Поэтому суть данного правила состоит в том, чтобы развернуть все элементы `<literal>`, находящиеся внутри `<screen>` и `<programlisting>`.

`%raise_and_move_backward` **представляет собой таблицу элементов, которые должны быть перемещены из своих родительских элементов в позицию прямо перед ними. А** `%raise_in_place` **– это таблица элементов, которые должны быть подняты на тот же уровень, что и их родительские элементы, разделив последние на две части вокруг себя.**

Пример 8.2. Листинг кода программы dbfix для проверки синтаксиса XML

```
#!/usr/local/bin/perl -w
#
# Исправляет нарушения структуры:
#   - если список находится внутри параграфа, разбить параграф на две части
#   - развернуть <literal>, если он находится внутри <screen>
#     или <programlisting>
#   - вывести <indexterm> из заголовков.
#
# Анализирует документ XML и строит дерево объектов,
# обрабатывая узлы по порядку,
# затем переводит дерево обратно в XML.
#
# Формат вызова: dbfix <top-xml-file>
#

use strict;

# ----------------------------------------------------------------------------
# ГЛОБАЛЬНЫЕ ДАННЫЕ
# ----------------------------------------------------------------------------

# Структура дерева объектов XML:
#
# узел --> тип
#      --> имя
#      --> данные
#      --> родитель
#      --> [дочерние узлы]
#
# где тип может иметь значение: element, attribute,
# PI, declaration, comment, cdms, text или root;
# имя дополнительно специфицирует разновидность узла;
# данные являются содержимым узла;
# [дочерние узлы] является списком узлов.
#

my %unwrap_element =    # разворачиваемые узлы: контекст => [узел]
                        (
                         'screen' => ['literal'],
                         'programlisting' => ['literal'],
                        );
```

Пример 8.2. Листинг кода программы dbfix для проверки синтаксиса XML (продолжение)

```
my %raise_and_move_backward = # узлы, поднимаемые и выводимые вперед:
                            # контекст => [узел]
                              (
                               'title' => [ 'indexterm', 'icon' ],
                               'refname' => [ 'indexterm', 'icon' ],
                               'refentrytitle' => [ 'indexterm', 'icon' ]
                              );
my %raise_in_place =    # узлы, поднимаемые в том же месте:
                      # контекст => [узел]
                              (
                               'para' => [
                                          'variablelist',
                                          'orderedlist',
                                          'itemizedlist',
                                          'simplelist',
                                          ],
                              );
my %frefs;              # файловые сущности, объявленные во внутреннем
                      # подмножестве
my $rootnode;           # корень дерева объектов XML


# получить для обработки файл верхнего уровня
my $file = shift @ARGV;
if($file && -e $file) {
    &main();
    exit(0);
} else {
    print STDERR "\nUsage: $0 <top-xml-file>\n\n";
    exit(1);
}


# ---------------------------------------------------------------------------
# ПОДПРОГРАММЫ
# ---------------------------------------------------------------------------

# main
# ----
# Подпрограмма верхнего уровня.
#
sub main {
    # 1. Ввод файлов XML, создание текстового буфера для сущности документа
    my $text = &process_file($file);

    # 2. Построение дерева объектов узлов XML
    $rootnode = &make_node('root', undef, undef, &parse_document($text));

    # 3. Обработка узлов, добавление числовых атрибутов и т.д.
```

Пример 8.2. Листинг кода программы dbfix для проверки синтаксиса XML (продолжение)

```
    &process_tree($rootnode);

    # 4. Перевод дерева обратно в текст
    $text = &serialize_tree($rootnode);

    # 5. Вывод текста в отдельные файлы
    &output_text($file, $text);
}


# process_file
# ------------
# Получить текст всех файлов XML в документе.
#
sub process_file {
    my($file) = @_;
    unless(open(F, $file)) {
        print STDERR "Can't open \"$file\" for reading.\n";
        return "";
    }
    my @lines = <F>;
    close F;
    my $line;
    my $buf = "";
    foreach $line (@lines) {

        # Заменить ссылки на внешние сущности содержимым файлов
        if($line =~ /\&([^;]+);/ && $frefs{$1}) {
            my($pre, $ent, $post) = ($`, $&, $');
            my $newfile = $frefs{$1};
            $buf .= $pre . $ent . "\n<?xml-file startfile: $newfile ?>" .
                &process_file($frefs{$1}) . "<?xml-file endfile ?>" .
                    $post;
        } else {
            $buf .= $line;
        }

        # Добавить объявленные внешние сущности в список.
        # ЗАМЕЧАНИЕ: идентификаторы типа PUBLIC не обрабатываются!
        $frefs{ $1 } = $2
            if($line =~ /<!ENTITY\s+(\S+)\s+SYSTEM\s+\"([^\"]+)/);
    }
    return $buf;
}


# parse_document
# --------------
```

Пример 8.2. Листинг кода программы dbfix для проверки синтаксиса XML (продолжение)

```
# Прочесть узлы верхнего уровня документа.
#
sub parse_document {
    my($text) = @_;
    my @nodes = ();
    while($text) {
        my $node;
        ($node, $text) = &get_node($text);
        push(@nodes, $node);
        sleep(1);
    }
    return @nodes;
}


# get_node
# --------
# Получив фрагмент текста XML, возвратить первый найденный узел
# и оставшуюся часть строки текста.
#
sub get_node {
    my($text) = @_;
    my $node;
    my($type, $name, $data) = ('','','');
    my @children = ();

    # текст
    if($text =~ /^[^<]+/) {
        $text = $';
        $type = 'text';
        $data = $&;

    # комментарий, помеченная секция, объявление
    } elsif($text =~ /^\s*<\!/) {
        if($text =~ /^\s*<\!--(.*?)-->/s) {
            $text = $';
            $type = 'comment';
            $data = $1;
            if($data =~ /--/) {
                &parse_error("в комментарии содержится
              неполный разделитель (--)");
            }
        } elsif($text =~ /^\s*<\!\[\s*CDATA\s*\[/) {
            $text = $';
            $type = 'CDMS';
            if($text =~ /\]\]>/) {
                $text = $';
                $data = $`;
            } else {
```

Пример 8.2. Листинг кода программы dbfix для проверки синтаксиса XML (продолжение)

```
            &parse_error("CDMS");
        }
    } elsif($text =~ /^\s*<!DOCTYPE(.*?\])>\s*/s) {
        $text = $';
        $name = 'DOCTYPE';
        $type = 'declaration';
        $data = $1;
    } else {
        &parse_error("синтаксис объявления");
    }

# инструкция обработки
} elsif($text =~ /^\s*<\?/) {
    if($text =~ /^\s*<\?\s*([^\s\?]+)\s*(.*?)\s*\?>\s*/s) {
        $text = $';
        $name = $1;
        $type = 'PI';
        $data = $2;
    } else {
        &parse_error("синтаксис PI ");
    }

# элемент
} elsif($text =~ /\s*</) {

    # пустой элемент с атрибутами
    if($text =~ /^\s*<([^\/\s>]+)\s+([^\s>][^>]+)\/>/) {
        $text = $';
        $name = $1;
        $type = 'empty-element';
        my $atts = $2;
        push(@children, &parse_atts($atts));

    # пустой элемент без атрибутов
    } elsif($text =~ /^\s*<([^\/\s>]+)\s*\/>/) {
        $text = $';
        $name = $1;
        $type = 'empty-element';

    # элемент-контейнер
    } elsif($text =~ /^\s*<([^\/\s>]+)([^<>]*)>/) {
        $text = $';
        $name = $1;
        $type = 'element';
        my $atts = $2;
        $atts =~ s/^\s+//;
        push(@children, &parse_atts($atts)) if $atts;
```

Пример 8.2. Листинг кода программы dbfix для проверки синтаксиса XML (продолжение)

```
            # обработать дочерние узлы
            while($text !~ /^<\/$name\s*>/) {
                my $newnode;
                ($newnode, $text) = &get_node($text);
                push(@children, $newnode);
            }

            # удалить закрывающий тег
            if($text =~ /^<\/$name\s*>/) {
                $text = $';
            } else {
                &parse_error("закрывающий тег элемента <$name>");
            }
        } else {
            if($text =~ /^\s*<\/(\S+)/) {
                &parse_error("отсутствует открывающий тег: $1");
            } else {
                &parse_error("зарезервированный символ (<) в тексте");
            }
        }

    } else {
        &parse_error("неидентифицированный текст");
    }

    # создать узел
    $node = &make_node($type, $name, $data, @children);
    my $n;
    foreach $n (@children) {
        $n->{'parent'} = $node;
    }

    return($node, $text);
}


# parse_atts
# ----------
#
sub parse_atts {
    my($text) = @_;
    my @nodes = ();
    while($text) {
        if($text =~ /\s*([^\s=]+)\s*=\s*([\"][^\"]*[\"])/ ||
            $text =~ /\s*([^\s=]+)\s*=\s*([\'][^\']*[\'])/) {
            $text = $';
            my($name, $data) = ($1, $2);
            push(@nodes, &make_node('attribute', $name, $data));
        } elsif($text =~ /^\s+/) {
```

Пример 8.2. Листинг кода программы dbfix для проверки синтаксиса XML (продолжение)

```
            $text = $';
        } else {
            &parse_error("синтаксис атрибутов");
        }
    }
    return @nodes;
}


# make_node
# ---------
#
sub make_node {
    my($type, $name, $data, @children) = @_;
    my %newnode = (
                    'type' => $type,
                    'name' => $name,
                    'data' => $data,
                    'children' => \@children
                  );
    return \%newnode;
}


# parse_error
# -----------
#
sub parse_error {
    my($reason) = @_;
    die("Ошибка синтаксического анализа: $reason.\n");
}


# process_tree
# ------------
#
sub process_tree {
    my($node) = @_;

    # корень
    if($node->{'type'} eq 'root') {
        # рекурсия по дочерним узлам для обхода дерева
        my $child;
        foreach $child (@{$node->{'children'}}) {
            &process_tree($child);
        }

    # элемент
```

Пример 8.2. Листинг кода программы dbfix для проверки синтаксиса XML (продолжение)

```
    } elsif($node->{'type'} =~ /element$/) {

        # переместить/удалить элементы, если они находятся в неверных местах
        &restructure_elements($node);

        # рекурсия по дочерним узлам для обхода дерева
        my $child;
        foreach $child (@{$node->{'children'}}) {
            &process_tree($child);
        }
    }
}


# get_descendants
# ---------------
# Найти все соответствия имени потомка в поддереве
# на заданную глубину.
#
sub get_descendants {
    my($node, $descendant_name, $descendant_type, $depth) = @_;
    my @results = ();
    # если получен результат, добавить его в список
    if(($descendant_name && $node->{'name'} eq $descendant_name) ||
       ($descendant_type && $node->{'type'} eq $descendant_type)) {
        push(@results, $node);
    }
    # рекурсия, если она возможна
    if($node->{'type'} eq 'element' &&
       ((defined($depth) && $depth > 0) || !defined($depth))) {
        my $child;
        foreach $child (@{$node->{'children'}}) {
            if(defined($depth)) {
                push(@results,
                     &get_descendants($child, $descendant_name,
                                      $descendant_type, $depth - 1));
            } else {
                push(@results,
                     &get_descendants($child, $descendant_name,
                                      $descendant_type));
            }
        }
    }
    return @results;
}


# serialize_tree
# --------------
```

Пример 8.2. Листинг кода программы dbfix для проверки синтаксиса XML (продолжение)

```
# Вывести дерево объектов в виде текста XML
#
sub serialize_tree {
    my($node) = @_;
    my $buf = "";

    # корневой узел
    if($node->{'type'} eq 'root') {
        my $n;
        foreach $n (@{$node->{'children'}}) {
            $buf .= &serialize_tree($n);
        }

    # элемент и пустой элемент
    } elsif($node->{'type'} eq 'element' ||
             $node->{'type'} eq 'empty-element') {
        $buf .= "<" . $node->{'name'};
        my $n;
        foreach $n (@{$node->{'children'}}) {
            if($n->{'type'} eq 'attribute') {
                $buf .= &serialize_tree($n);
            }
        }
        $buf .= ">";
        if($node->{'type'} eq 'element') {
            foreach $n (@{$node->{'children'}}) {
                if($n->{'type'} ne 'attribute') {
                    $buf .= &serialize_tree($n);
                }
            }
            $buf .= "</" . $node->{'name'} . ">";
        } else {
            $buf .= "/>";
        }

    # атрибут
    } elsif($node->{'type'} eq 'attribute') {
        $buf .= " " . $node->{'name'} . "=" . $node->{'data'};

    # комментарий
    } elsif($node->{'type'} eq 'comment') {
        $buf .= "<!--" . $node->{'data'} . "-->";

    # объявление
    } elsif($node->{'type'} eq 'declaration') {
        $buf .= "<!" . $node->{'name'} . $node->{'data'} . ">" .
            &space_after_start("decl:" . $node->{'name'});
```

Пример 8.2. Листинг кода программы dbfix для проверки синтаксиса XML (продолжение)

```
    # CDMS
    } elsif($node->{'type'} eq 'CDMS') {
        $buf .= "<![CDATA[" . $node->{'data'} . "]]>";

    # PI
    } elsif($node->{'type'} eq 'PI') {
        $buf .= "<?" . $node->{'name'} . " " . $node->{'data'} . "?>" .
            &space_after_start("pi:" . $node->{'name'});

    # текст
    } elsif($node->{'type'} eq 'text') {
        $buf .= $node->{'data'};
    }
    return $buf;
}


# output_text
# -----------
# Найти в тексте специальные инструкции обработки, обозначающие
# границы файлов, разрезать файл на соответствующие части и сохранить
# их в отдельных файлах.
#
sub output_text {
    my($file, $text) = @_;
    $text = "<?xml-file startfile: $file ?>" . $text .
        "<?xml-file endfile ?>\n";
    my @filestack = ($file);
    my %data = ();
    while($text) {
        if($text =~ /<\?xml-file\s+([^\s\?]+)\s*([^\?>]*)\?>/){
            $data{ $filestack[ $#filestack ]} .= $`;
            my($mode, $rest) = ($1, $2);
            $text = $';
            if($mode eq 'startfile:' && $rest =~ /\s*(\S+)/) {
                push(@filestack, $1);
            } elsif($mode eq 'endfile') {
                pop(@filestack);
            } else {
                die("Error with xml-file PIs: $mode, $rest");
            }
        } else {
            $data{ $filestack[ $#filestack ]} .= $text;
            $text = "";
        }
    }
    foreach $file (sort keys %data) {
        print "updated file: $file\n";
        if(open(F, ">$file")) {
```

Пример 8.2. Листинг кода программы dbfix для проверки синтаксиса XML (продолжение)

```
            print F $data{ $file };
            close F;
        } else {
            print STDOUT "Warning: can't write to \"$file\"\n";
        }
    }
}

# restructure_elements
# --------------------
#
sub restructure_elements {
    my($node) = @_;
    # развернуть элементы
    if(defined($node->{'name'}) &&
        defined($unwrap_element{ $node->{'name'} })) {
        my $elem;
        foreach $elem (@{$unwrap_element{ $node->{'name'} }}) {
            my $n;
            foreach $n (&get_descendants($node, $elem)) {
                &unwrap_element($n) if($n->{'name'} eq $elem);
            }
        }
    }
    # поднять элементы
    if(defined($node->{'name'}) &&
        defined($raise_in_place{ $node->{'name'} })) {
        my $elem;
        foreach $elem (@{$raise_in_place{ $node->{'name'} }}) {
            my $n;
            foreach $n (&get_descendants($node, $elem, undef, 1)) {
                &raise_in_place($n) if($n->{'name'} eq $elem);
            }
        }
    }
    # поднять элементы и переместить назад
    if(defined($node->{'name'}) &&
        defined($raise_and_move_backward{ $node->{'name'} })) {
        my $elem;
        foreach $elem (@{$raise_and_move_backward{ $node->{'name'} }}) {
            my $n;
            foreach $n (&get_descendants($node, $elem, undef, 1)) {
                &raise_and_move_backward($n) if($n->{'name'} eq $elem);
            }
        }
    }
}
```

Пример 8.2. Листинг кода программы dbfix для проверки синтаксиса XML (продолжение)

```
# unwrap_element
# --------------
# удалить элемент, поставив на его место (без атрибутов) его дочерние
#
sub unwrap_element {
    my($node) = @_;
    # создать список сохраняемых дочерних элементов
    my @children_to_save = ();
    my $child;
    foreach $child (@{$node->{'children'}}) {
        push(@children_to_save, $child)
            if($child->{'type'} ne 'attribute');
    }
    # удалить узел из списка родителя
    my $count = &count_older_siblings($node);
    # вставить на его место дочерние
    my $parent = $node->{'parent'};
    while($child = pop @children_to_save) {
        &insert_node($child, $parent, $count);
    }
    # удалить узел
    &delete_node($node);
}

# count_older_siblings
# --------------------
# подсчитать количество узлов, которые предшествуют выбранному узлу
# в атрибуте {'children'}
#
sub count_older_siblings {
    my($node) = @_;
    my $n;
    my $count = 0;
    foreach $n (@{$node->{'parent'}->{'children'}}) {
        last if($n eq $node);
        $count ++;
    }
    return $count;
}

# delete_node
# -----------
# удалить узел из иерархии, уничтожив ссылку на него в родительском узле
# и его ссылку на родителя
#
sub delete_node {
    my($node) = @_;
    # если у узла есть родитель...
    if(defined($node->{'parent'})) {
```

Пример 8.2. Листинг кода программы dbfix для проверки синтаксиса XML (продолжение)

```
        my $parent = $node->{'parent'};
        my @siblings = ();
        my $sib;
        # получить список братьев (без удаляемого узла)
        foreach $sib (@{$parent->{'children'}}) {
            push(@siblings, $sib) unless($node eq $sib);
        }
        # назначить родителю новый список дочерних элементов
        $parent->{'children'} = \@siblings;
        # разорвать семейные связи
        undef($node->{'parent'});
    }
}

# insert_node
# -----------
# поместить узел в выбранное место, задаваемое его новым
# родительским узлом и числом старших братьев
# (то есть индексом в массиве $parent->{'children'})
#
sub insert_node {
    my ($new_node, $parent, $pos) = @_; # $n = число старших братьев
    # при необходимости разорвать прежние связи
    if(defined($new_node->{'parent'})) {
        &delete_node($new_node);
    }
    # создать новый список дочерних узлов
    my @siblings = ();
    my $n;
    my $count = 0;
    foreach $n (@{$parent->{'children'}}) {
        if($count == $pos) {
            push(@siblings, $new_node);
        }
        push(@siblings, $n);
        $count++;
    }
    # назначить дочерние узлы родителю
    $parent->{'children'} = \@siblings;
    # назначить родителя новому узлу
    $new_node->{'parent'} = $parent;
}

# raise_and_move_backward
# -----------------------
# поднять узел на более высокий уровень, поместив его прямо перед родителем
#
sub raise_and_move_backward {
```

Пример 8.2. Листинг кода программы dbfix для проверки синтаксиса XML (продолжение)

```
    my($node) = @_;
    if(defined($node->{'parent'})) {
        my $parent = $node->{'parent'};
        my $count = &count_older_siblings($parent);
        &insert_node($node, $parent->{'parent'}, $count);
    }
}

# поднять_и переместить вперед
# ----------------------
# поднять узел на более высокий уровень, поместив его сразу после родителя
#
sub raise_and_move_forward {
    my($node) = @_;
    if(defined($node->{'parent'})) {
        my $count = &count_older_siblings($node->{'parent'});
        &insert_node($node, $node->{'parent'}->{'parent'}, $count +1);
    }
}

# raise_in_place
# --------------
# разбить элемент на части вокруг дочернего и поднять его
# на такой же уровень
#
# было: <a>xxx<b>yyy</b>zzz</a>
# стало:  <a>xxx</a><b>yyy</b><a>zzz</a>
#
sub raise_in_place {
    my($node) = @_;
    if(defined($node->{'parent'}) &&
        defined($node->{'parent'}->{'parent'})) {
        # получить списки старших и младших братьев
        my $parent = $node->{'parent'};
        my $n;
        my @older_sibs = ();
        my @younger_sibs = ();
        my $nodeseen = 0;
        foreach $n (@{$parent->{'children'}}) {
            if($node eq $n) {
                $nodeseen = 1;
            } else {
                push(@older_sibs, $n) unless($nodeseen);
                push(@younger_sibs, $n) if($nodeseen);
            }
        }
        # назначить прежнему родителю новые дочерние узлы
        $parent->{'children'} = \@older_sibs;
        # удалить и вставить узел сразу после родителя
```

Пример 8.2. Листинг кода программы dbfix для проверки синтаксиса XML (продолжение)

```
        &delete_node($node);
        my $count = &count_older_siblings($parent);
        &insert_node($node, $parent->{'parent'}, $count+1,);
        # создать новый узел, содержащий младших братьев
        my $new_node =
            &make_node('element', $parent->{'name'}, undef,
                       @younger_sibs);
        &insert_node($new_node, $parent->{'parent'}, $count+2,);
        foreach $n (@{$new_node->{'children'}}) {
            $n->{'parent'} = $new_node;
        }
    }
}
```

Объектная модель документа

Объектная модель документа (Document Object Model, DOM) является рекомендацией W3C для стандартного основанного на представлении в виде дерева API для документов XML. Первоначально задуманная как способ единообразной реализации программ Java и JavaScript для различных веб-броузеров, она превратилась в API XML для любых приложений – от редакторов до систем управления файлами.

Как и SAX, DOM является набором интерфейсов Java (и JavaScript), объявляющих методы, которые должен создать разработчик. В отличие от SAX, однако, интерфейсы определяют не подпрограммы обратного вызова для событий, а методы, позволяющие создавать и модифицировать объекты. Это связано с тем, что дерево, служащее представлением документа в памяти, является деревом *программных объектов* (*programming objects*), т. е. пакетов данных и методов, разрешающих лишь некоторое число способов модификации данных; поэтому мы говорим, что данные «скрыты» из виду. Это значительно более аккуратный и упорядоченный способ хранения данных, чем тот, который мы видели в примере 8.2.

По существу, DOM делает для программ то, что XML делает для документов. Она высоко организована, структурирована, защищена от ошибок и настраиваема. В базовом модуле DOM описываются контейнеры для элементов, атрибутов и других основных типов узлов. DOM содержит также множество других модулей с дополнительными функциями, от особой обработки HTML до пользовательских событий и таблиц стилей. Некоторые основные модули DOM перечислены ниже:

Базовый модуль

Определяет базовый тип объекта для элементов, атрибутов и т. д.

XML-модуль

Содержит более изощренные компоненты XML, такие как секции CDATA, которые не нужны для HTML и простых документов XML.

HTML-модуль

Специализированный интерфейс для документов HTML, который знает о таких элементах, как <p> и <body>.

Модуль представлений

Документ может иметь одно или несколько *представлений* (*views*), форматированных после применения правил CSS отображений. Этот интерфейс описывает взаимодействие между документом и его представлениями.

Модуль таблиц стилей

Это базовый интерфейс, на основе которого можно получать объекты таблиц стилей.

Модуль CSS

Интерфейс CSS, производный от интерфейса таблиц стилей, предназначенных для документов, форматирование которых при выводе производится с помощью правил CSS.

Модуль событий

Это база событийной модели, которая обрабатывает события пользователя, такие как щелчок по ссылке, или события трансформации, например, изменение свойств элементов.

Базовый интерфейсный модуль описывает, как каждый узел дерева документа XML может быть представлен объектом в дереве DOM. Дочерние узлы могут быть как у XML-узлов, так и у DOM-узлов. Структура должна быть близкой к генеалогической структуре дерева XML, хотя в дереве DOM несколько больше типов объектов, чем типов узлов. Например, ссылки на сущности не считаются в XML самостоятельными узлами, но в DOM они рассматриваются как отдельные типы объектов. Типы узлов перечислены в табл. 8.2.

Таблица 8.2. Типы узлов DOM

Название	Дочерние узлы
Document	Element **(только один)**, ProcessingInstruction, Comment, DocumentType **(только один)**
DocumentFragment	Element, ProcessingInstruction, Comment, Text, CDATASection, EntityReference
DocumentType	**Нет**
EntityReference	Element, ProcessingInstruction, Comment, Text, CDATASection, EntityReference

Таблица 8.2 (продолжение)

Название	Дочерние узлы
Element	Element, Text, Comment, ProcessingInstruction, CDATASection, EntityReference
Attr	Text, EntityReference
ProcessingInstruction	**Нет**
Comment	**Нет**
Text	**Нет**
CDATASection	**Нет**
Entity	Element, ProcessingInstruction, Comment, Text, CDATASection, EntityReference
Notation	**Нет**

Методы `set()` и `get()` взаимодействуют с данными узла. Остальные методы определяются в зависимости от типа узла, например, сравнение имен, анализ содержимого, создание атрибутов и т. д.

Поскольку в DOM определяются интерфейсы, а не фактические классы, реализация готовых к использованию пакетов DOM предоставляется разработчикам. Вы можете написать классы сами или найти уже готовую реализацию. DOM требует также, чтобы в качестве ее базы использовался основанный на событиях процессор. Хорошим выбором является SAX, поверх которого построено несколько реализаций DOM.

Заключение

На этом завершается наш обзор разработки XML. Он по необходимости был сделан расплывчатым, иначе пришлось бы писать по данной теме целую книгу, а это могут сделать и уже сделали другие. Теперь у вас есть знание основных концепций программирования XML, и это хорошая отправная точка при выборе направления дальнейшего движения. Приложение A «Ресурсы» и Приложение B содержат источники по программированию XML, которыми вы можете руководствоваться на выбранном пути.

A

Ресурсы

Ресурсы, перечисленные в этом приложении, оказались бесценными при создании этой книги, и могут помочь вам узнать больше о XML.

Сетевые ресурсы

XML.com

Веб-сайт *http://www.xml.com* является одним из наиболее полных и своевременно обновляемых источников информации по XML и сопутствующим новостям. Его следует включить в список ежедневного чтения, если вы изучаете или используете XML.

XML.org

Спонсируемый OASIS *http://www.xml.org* содержит новости и ресурсы XML, в том числе XML Catalog, проводник по продуктам и службам XML.

XMLHack

Для программистов, жаждущих поработать с XML, *http://www.xmlhack.com* – то место, где надо побывать.

The XML Cover Pages

http://www.oasis-open.org/cover/, издатель которого Robin Cover, – один из самых крупных и свежих ресурсов XML.

DocBook

OASIS, которые поддерживают DocBook, поддерживают web-страницу, посвященную этому приложению XML на *http://www.oasis-open.org/docbook/*. Там можно найти последнюю версию и множество документации.

Учебное пособие по вопросам кодирования символов

Юкка Корпела (Jukka Korpela) собрал огромное количество информации о наборах символов на *http://www.hut.fi/~jkorpela/chars.html*. Этот учебник хорошо написан, и его очень интересно читать.

Почтовый список рассылки XSL

Подписка на почтовый список рассылки XSL служит отличным способом слежения за последними достижениями в инструментах и технологиях XSL и XSLT. Это также форум, на котором можно задать вопросы и получить советы. Интенсивность сообщений довольно высока, поэтому следует взвесить свои потребности и большой объем сообщений, который будет поступать в ваш почтовый ящик. Чтобы подписаться, зайдите на *http://www.mulberrytech.com/xsl/* и следуйте инструкциям по подписке.

Apache XML Project

Эта часть проекта Apache сосредоточена на технологиях XML и находится на *http://xml.apache.org*. Там разрабатываются инструментарии и технологии для использования XML с Apache и предоставляются отзывы на реализации XML организациям стандартизации.

Руководство разработчика XML

Сетевой семинар Microsoft Developers Network по XML и информацию об использовании XML в приложениях Microsoft можно найти на *http://msdn.microsoft.com/xml/XMLGuide/*.

Dr. Dobb's Journal

Этот журнал содержит статьи, ресурсы, мнения, новости и обзоры, охватывающие все аспекты программирования. Электронный журнал можно найти на *http://www.ddj.com*, и там же осуществляется подписка на журнал.

Perl.com

Perl является интерпретируемым языком программирования для любых видов обработки текста, в том числе XML. Лучшим местом в сети для получения информации или загрузки кода и модулей является *http://www.perl.com*.

Javasoft

Лучшим источником новостей и информации по Java является сайт *http://www.javasoft.com*. Java представляет собой язык программирования, доступный на большинстве компьютеров, и поддерживает XML, в том числе реализации SAX и DOM, а также несколько замечательных анализаторов.

Книги

«DocBook, the Definitive Guide» (DocBook, полное руководство), Норман Уолш (Norman Walsh) и Леонард Мюлнер (Leonard Muellner) (O'Reilly & Associates)

DocBook является популярным и гибким языком разметки для технической документации с версиями для SGML и XML. Эта книга имеет исчерпывающий, подобный глоссарию формат и подробно описывает каждый элемент. В ней также содержится масса практической информации о том, как начать использовать XML и таблицы стилей.

«The XML Bible» (Полное руководство по XML), Эллиотт Расти Гарольд (Elliotte Rusty Harold), (Hungry Minds)

Серьезное введение в XML, содержащее полный обзор его возможностей.

«XML in a Nutshell», Elliotte Rusty Harold and W. Scott Means (O'Reilly & Associates)[1]

Обширный справочник по всем вопросам, касающимся XML.

«HTML and XHTML, the Definitive Guide», Chuck Musciano and Bill Kennedy (O'Reilly & Associates)[2]

Актуальный и обширный источник для изучения HTML.

«Developing SGML DTDs: From Text to Model to Markup» (Разработка DTD для SGML: от текста к модели и разметке), Ив Малер (Eve Maler) и Джин Эл Андалусси (Jeanne El Andaloussi) (Prentice Hall)

Учебное пособие, шаг за шагом разъясняющее разработку и использование DTD.

«The SGML Handbook» (Руководство по SGML), Чарльз Гольдфарб (Charles F. Goldfarb) (Oxford University Press)

Полный справочник по SGML, включающий снабженную комментариями спецификацию. Как и ее предмет, книга сложная и тяжелая, поэтому для новичков она может оказаться не самым лучшим выбором.

«Java and XML», Brett McLaughlin (O'Reilly & Associates)[3]

Руководство по соединению XML и Java для построения практических приложений.

---

[1] Э. Гарольд, С. Минс «XML. Справочник», издательство «Символ-Плюс», IV кв. 2001 г.

[2] Ч. Муссиано, Б. Кеннеди «HTML и XHTML», издательство «Символ-Плюс», IV кв. 2001 г.

[3] Б. Маклахлин «Java и XML», издательство «Символ-Плюс», IV кв. 2001 г.

«Building Oracle XML Applications» (Средства Oracle для разработки приложений XML), Стив Мунч (Steve Muench) (O'Reilly & Associates)

Детальное рассмотрение средств Oracle для разработки XML и рассказ о том, как сочетать мощь XML и XSLT с функциональностью базы данных Oracle.

Организации стандартизации

ISO

Посетите сайт Международной организации стандартизации, всемирной федерации национальных организаций стандартизации, на *http://www.iso.ch*.

W3C

Консорциум Всемирной паутины World Wide Web Consortium на *http://www.w3.org* наблюдает за спецификациями и руководствами по технологиям World Wide Web. Здесь можно получить информацию по CSS, DOM, (X)HTML, MathML, XLink, XML, XPath, XPointer, XSL и другим веб-технологиям.

Unicode Consortium

Организацию, ответственную за определение набора символов Unicode, можно найти на сайте *http://www.unicode.org*.

OASIS

Organization for the Advancement of Structured Information Standards – Организация по распространению стандартов структурирования информации – является международным консорциумом, разрабатывающим взаимодействующие промышленные спецификации, основанные на открытых стандартах, таких как XML и SGML. Посетите веб-сайт *http://www.oasis-open.org*.

Инструментарий

GNU Emacs

Чрезвычайно мощный текстовый редактор, и даже больше. Узнать о нем можно на странице *http://www.gnu.org/software/emacs/emacs.html*.

psgml

Основной режим Emacs для редактирования документов XML и SGML, который можно найти на странице *http://www.lysator.liu.se/~lenst/*.

SAX

Megginson Technologies опубликовали страницу, посвященную SAX, простому API для XML, по адресу *http://www.megginson.com/SAX/*. Здесь вы найдете исходный код Java и некоторую полезную документацию.

Xalan

Высокопроизводительный процессор таблиц стилей XSLT, полностью реализующий XSLT и XLinks. Подробнее узнать о нем можно на веб-сайте Apache XML Project по адресу *http://xml.apache.org*.

Xerces

Полностью подтверждающий действительность анализатор (fully validating parser), реализующий XML, DOM уровней 1 и 2 и SAX2. Подробности можно узнать в Apache XML Project по адресу *http://xml.apache.org*.

XT

Реализация XSLT на Java, находящаяся на странице *http://www.jclark.com/xml/xt.html*.

Разное

User Friendly, Illiad

Этот комикс, в котором действуют страшно остроумный «dust puppy» и шумная компания трутней из компьютерной индустрии, впрыснет вам в жилы столь необходимую после долгого дня работы с XML веселость. Архив целиком имеется на *http://www.userfriendly.org* и в двух книгах, изданных O'Reilly, «User Friendly, the Comic Strip» и «Evil Geniuses in a Nutshell».

The Cathedral and the Bazaar, Eric S. Raymond (O'Reilly & Associates)

Это философский анализ и евангелическая проповедь народного движения программирования открытых исходных текстов, в котором Реймонд превозносит добродетели общинности, бескорыстия и то особенное теплое чувство, которое мы испытываем при работе на общее благо.

B

Таксономия стандартов

Расширяемость XML хорошо подтверждается числом стандартов и спецификаций, расцветших на основе базовой идеи XML. Это приложение служит удобным справочником по различным видам деятельности, относящимся к XML.

Разметка и структура

XML – расширяемый язык разметки состоит из базовых правил разметки.

Статус

XML 1.0 (второе издание) стал рекомендацией в октябре 2000 г. Прочесть спецификацию можно на странице *http://www.w3.org/TR/REC-xml*.

Описание

XML является подмножеством SGML, предназначенным для использования, получения и обработки в Web так, как это сейчас делается с помощью HTML. Преимуществами XML являются простота реализации и совместимость с SGML и HTML.

Namespaces in XML – Пространства имен используются для разделения элементов и атрибутов на разные группы.

Статус

Пространства имен стали рекомендацией в январе 1999 г. Спецификация опубликована на странице *http://www.w3.org/TR/REC-xml-names/*.

Описание

Пространства имен XML предоставляют простой способ уточнения имен элементов и атрибутов, используемых в документах XML, путем связывания их с пространствами имен, идентифицируемыми ссылками URI.

XML Schema – Язык XML Schema навязывает документу структуру.

Статус

XML Schema стал кандидатом в рекомендации W3C в октябре 2000 г. Рекомендация состоит из трех частей:

XML Schema Part 0: Primer

http://www.w3.org/TR/xmlschema-0/

XML Schema Part 1: Structures

http://www.w3.org/TR/xmlschema-1/

XML Schema Part 2: Datatypes

http://www.w3.org/TR/xmlschema-2/

Описание

Язык XML Schema используется для определения документов таким образом, который находится за пределами возможностей DTD. Schema использует действительный XML, чтобы объявлять элементы и атрибуты для структурирования документа, а также предоставляет расширяемые средства для определения типов данных элементов и атрибутов.

Создание ссылок

XLink – Язык XML Linking Language создает ссылки между ресурсами.

Статус

XLink стал кандидатом в рекомендации W3C в июле 2000 г. Спецификация опубликована на странице *http://www.w3.org/TR/xlink /*.

Описание

XLink позволяет вставлять в документы XML элементы, которые создают и описывают ссылки между ресурсами. Он использует синтаксис

XML для создания структур, описывающих ссылки – от простых однонаправленных ссылок сегодняшнего HTML до более сложных ссылок.

XBase – XML Base предоставляет средства для определения базовых URI отдельных частей документов XML.

Статус

XBase стал кандидатом в рекомендации W3C в сентябре 2000 г. Спецификация опубликована на странице *http://www.w3.org/TR/xmlbase/*.

Описание

XBase описывает механизм предоставления сервисов базовых URI для XLink. Спецификация является модульной, поэтому она может использоваться другими приложениями XML.

XInclude – XML Inclusions является стандартом для встраивания документов XML.

Статус

Последний рабочий проект XInclude датируется октябрем 2000 г. и опубликован на странице *http://www.w3.org/TR/xinclude/*.

Описание

XInclude определяет модель обработки и синтаксис для универсальных включений. Включение осуществляется слиянием нескольких информационных наборов XML (XML Infoset) в один составной информационный набор. Спецификация документов XML (информационных наборов), которые должны быть слиты вместе, и управление процессом слияния описывается с помощью дружественного XML синтаксиса (элементов, атрибутов, ссылок на URI).

Поиск

XPath – XML Path Language используется для определения местоположения объектов XML.

Статус

XPath стал рекомендацией W3C в ноябре 1999 г. Спецификация опубликована на странице *http://www.w3.org/TR/xpath/*.

Описание

XPath является языком для обращения к частям документов XML, разработанным для использования XSLT и XPointer.

XPointer – XML Pointer Language является стандартом задания путей в URI.

Статус

XPointer стал кандидатом в рекомендации W3C в июне 2000 г. Спецификация опубликована на странице *http://www.w3.org/TR/xptr/*.

Описание

XPointer используется как основа идентификатора фрагмента в любой ссылке URI, определяющей в Интернете ресурс с типом text/xml или application/xml. Основанный на XML Path Language (XPath), XPointer поддерживает адресацию внутренних структур документов XML. Он позволяет изучать иерархическую структуру документа и выбирать его внутренние части, исходя из таких свойств, как тип элемента, значение атрибутов, символьное содержание и относительное положение.

XQL – XML Query Language предоставляет средства доступа к документам web, схожие с запросами в базах данных.

Статус

XQL в настоящее время поддерживается Джонатаном Роби (Jonathan Robie); полную информацию можно получить на странице *http://www.ibiblio.org/xql/*. Самая последняя спецификация была представлена в рабочую группу W3C XSL в 1998 г. как *http://www.w3.org/TandS/QL/QL98/pp/xql.html*. Эта спецификация все еще используется сегодня с незначительными изменениями.

Описание

XQL – это язык запросов, который использует XML в качестве модели данных и аналогичен XSL Patterns. Выражения XQL легко анализировать и задавать. Их можно использовать в различных программных средах: как часть URL, в атрибутах XML или HTML, в строках языков программирования и т. д.

Стиль и трансформация

CSS – Спецификация каскадных таблиц стилей служит языком форматирования элементов документов.

Статус

CSS Level 2 стала рекомендацией W3C в мае 1998 г. и опубликована на странице *http://www.w3.org/TR/REC-CSS2/*. CSS Level 1 стала рекомендацией W3C в 1996 г., пересмотрена в январе 1999 г. Она опубликована на странице *http://www.w3.org/TR/REC-CSS1*.

Описание

CSS2 является языком таблиц стилей, позволяющим авторам и пользователям присоединять стили (т. е. шрифты, разрядку и звуки) к структурированным документам, например, к документам HTML и приложениям XML. Отделяя стиль представления от содержимого документа, CSS2 упрощает создание веб-документов и поддержку сайтов.

CSS2 основана на CSS1, и, за немногими исключениями, таблицы стилей, соответствующие CSS1, соответствуют также CSS2. CSS2 поддерживает таблицы стилей, специфичные для носителя, поэтому авторы могут приспосабливать представление своих документов к зрительным броузерам, звуковым устройствам, принтерам, устройствам азбуки Брайля, карманным устройствам и т. д. Кроме того, эта спецификация поддерживает задание местоположения содержимого, загружаемые шрифты, структуры таблиц, многоязычность, автоматические счетчики и нумерацию, а также некоторые свойства интерфейса пользователя.

XSL – Extensible Stylesheet Language является языком таблиц стилей для XML.

Статус

XSL стал кандидатом в рекомендации W3C в ноябре 2000 г. Спецификация опубликована на странице *http://www.w3.org/TR/xsl/*.

Описание

XSL служит языком трансформирования документов XML и состоит из XML-словаря семантики форматирования. Таблица стилей XSL задает представление для класса документов XML, описывая способ, которым экземпляр класса преобразуется в документ XML, использующий словарь форматирования.

XSLT – XSL Transformations является языком для трансформирования документов XML.

Статус

XSLT стал рекомендацией W3C в ноябре 1999 г. Спецификация опубликована на странице *http://www.w3.org/TR/xslt/*.

Описание

XSLT является языком для трансформирования документов XML в другие документы XML. Он был разработан для использования как часть XSL, являющегося языком таблиц стилей для XML. XSL включает в себя также словарь XML для задания форматирования и использует XSLT, чтобы описать, как документ трансформируется в другой документ XML, использующий словарь форматирования.

Программирование

DOM – Document Object Model (объектная модель документа) является общим API для анализа XML.

Статус

DOM Level 2 стала рекомендацией W3C в ноябре 2000 г. и состоит из пяти спецификаций:

Ядро спецификации DOM2

http://www.w3.org/TR/DOM-Level-2-Core/

Спецификация представлений в DOM2

http://www.w3.org/TR/DOM-Level-2-Views/

Спецификация событий DOM2

http://www.w3.org/TR/DOM-Level-2-Events/

Спецификация стилей в DOM2

http://www.w3.org/TR/DOM-Level-2-Style/

Спецификация обхода и задания диапазонов документа в DOM2

http://www.w3.org/TR/DOM-Level-2-Traversal-Range/

Описание

DOM Level 2 является нейтральным относительно платформы и языка интерфейсом, позволяющим программам и сценариям динамически получать и обновлять содержимое и структуру документов. Ядро спецификации DOM Level 2e основано на ядре спецификации DOM Level 1 и состоит из набора базовых интерфейсов, которые создают и обрабатывают структуру и содержимое документа. Ядро спецификации содержит также специализированные интерфейсы, предназначенные для XML.

SAX – Простой API для XML является свободно распространяемым API для синтаксического анализа XML, основанным на событиях.

Статус

SAX был коллективно разработан в почтовом списке рассылки XML-DEV (поддерживаемым OASIS). Текущей версией является SAX 2.0 от мая 2000 г. SAX сопровождает David Megginson на странице *http://www.megginson.com/SAX/*.

Описание

SAX2 является новой основанной на Java версией SAX. В SAX2 вводятся конфигурируемые функции и свойства и добавлена поддержка пространств имен XML. Он также содержит адаптеры, позволяющие взаимодействовать с анализаторами и приложениями SAX1.

Издательское дело

DocBook – DocBook представляет собой DTD для технических публикаций и программной документации.

Статус

Последняя SGML-версия DocBook имеет номер 4.1 и датируется июлем 2000 г., а последняя XML-версия DocBook имеет номер 4.1.2 и датируется августом 2000 г. DocBook официально сопровождается комитетом DocBook Technical Committee of OASIS, и официальная страница его размещена на *http://www.oasis-open.org/docbook/index.html*.

Описание

DocBook является большим и устойчивым DTD, предназначенным для технических изданий, например, документов, относящихся к аппаратному и программному обеспечению компьютеров.

Гипертекст

XHTML – Расширяемый язык разметки гипертекста представляет собой переформулировку HTML 4 на XML 1.

Статус

XHTML 1.0 стал рекомендацией W3C в январе 2000 г. Спецификация опубликована на странице *http://www.w3.org/TR/xhtml1/*.

Описание

XHTML 1.0 является переформулировкой HTML 4 как приложения XML 1.0. В спецификации определены три DTD, соответствующие тем,

которые определены в HTML 4. Семантика элементов и их атрибуты определены в рекомендации W3C для HTML 4 и закладывают основы возможного расширения XHTML в будущем. Совместимость с имеющимися пользовательскими агентами HTML возможна при соблюдении небольшого набора правил.

HTML – Язык разметки гипертекста используется в веб-документах.

Статус

HTML 4.01 – это последняя версия рекомендации W3C, датированная декабрем 1999 г. Спецификация опубликована на странице *http://www.w3.org/TR/html401/*.

Описание

В дополнение к функциям прежних версий, касающимся текста, мультимедиа и гиперссылок, HTML 4 поддерживает новые возможности мультимедиа, языки сценариев и таблицы стилей, а также улучшенные средства печати и документы, более доступные пользователям с физическими недостатками. В HTML 4 также сделаны большие изменения в направлении поддержки многоязычности документов.

Описательные/процедурные

SOAP – Simple Object Access Protocol является протоколом обмена информацией.

Статус

Спецификация SOAP представлена в W3C в качестве рабочего доклада и предлагает сформировать новую рабочую группу. Доклад опубликован на странице *http://www.w3.org/TR/SOAP/*.

SOAP разрабатывается группой компаний компьютерной индустрии, возглавляемой Microsoft. Страница Microsoft Developer's Network SOAP находится по адресу *http://msdn.microsoft.com/xml/general/soapspec.asp*.

Описание

SOAP является облегченным основывающимся на XML протоколе для обмена информацией в децентрализованной распределенной среде. Он состоит из трех частей: конверта, определяющего структуру описания того, что находится в сообщении и как его обрабатывать, набора правил кодировки для описания экземпляров определяемых приложениями типов данных и соглашения по представлению вызовов удаленных процедур и ответов.

RDF – Resource Description Framework (структура описания ресурсов) стандартизирует представление метаданных.

Статус

RDF Model and Syntax Specification стала рекомендацией W3C в феврале 1999 г. и опубликована на странице *http://www.w3.org/TR/REC-rdf-syntax/*. RDF Schema Specification стала кандидатом в рекомендации W3C в марте 2000 года и опубликована на странице *http://www.w3.org/TR/rdf-schema/*.

Описание

RDF служит основой использования XML для обработки метаданных. Она предоставляет возможность взаимодействия между приложениями, обменивающимися информацией в Сети, понимаемой машинами. RDF придает особое значение средствам, позволяющим осуществлять автоматическую обработку ресурсов Интернета.

Мультимедиа

SVG – Спецификация Scalable Vector Graphics является языком описания двумерной векторной графики.

Статус

Спецификация SVG 1.0 стала кандидатом в рекомендации W3C в ноябре 2000 г. Спецификация опубликована на странице *http://www.w3.org/TR/SVG/*.

Описание

SVG является языком описания двумерной векторной и смешанной векторной/растровой графики в XML.

SMIL – Synchronized Multimedia Integration Language является HTML-подобным языком создания мультимедийных презентаций.

Статус

Спецификация SMIL 1.0 стала рекомендацией W3C в июне 1998 г. Она опубликована на странице *http://www.w3.org/TR/REC-smil/*.

Описание

SMIL позволяет объединить группу независимых мультимедийных объектов в синхронизированной мультимедийной презентации.

Наука

MathML – Язык математической разметки для XML, описывает математические обозначения.

Статус

MathML 1.01 стал рекомендацией W3C в июле 1999 г. Спецификация опубликована на странице *http://www.w3.org/TR/REC-MathML/*.

Описание

MathML является приложением XML для описания математических обозначений и ввода их структуры и содержания. Задачей MathML является дать возможность передавать, получать и обрабатывать математическую нотацию в Web, подобно тому, как HTML сделал это для текста.

CML – Язык химической разметки для XML, описывает химическую информацию.

Статус

CML 1.0 определяет DTD и первые версии CML Schema. Информация о разработке CML поддерживается на странице *http://www.xml-cml.org/*.

Описание

CML является набором средств XML, которые облегчают обмен химической информацией через World Wide Web и другие сетевые ресурсы.

Глоссарий

absolute location term — терм абсолютной адресации

Терм, который полностью задает местонахождение ресурса через указатель XPointer. В качестве терма абсолютной адресации может использоваться уникальный атрибут ID, присвоенный элементу. *См. также* **relative location term, XPath, XPointer**.

activity — активность

Деятельность по созданию стандарта в какой-то области интересов. *См. также* **standards body**.

actuation — приведение ссылки в действие

Способ срабатывания ссылки. Например, ссылка на импортируемое графическое изображение автоматически включает его в документ, а ссылка на ресурс URL требует действия пользователя.

application — приложение

Это слово имеет различные значения в разных контекстах (к сожалению, это частое явление в компьютерном жаргоне). В контексте XML приложение обычно означает конкретный язык разметки, основанный на правилах XML. DocBook служит одним из примеров приложений XML.

В более общем контексте программного обеспечения компьютеров приложением называется программа высокого уровня для пользователей, например, веб-броузер, текстовый процессор или программа электронных таблиц. Иными словами, это приложение компьютерной системы.

arc — дуга

Абстрактный термин для связи между ссылкой в документе и ее целевым объектом. *См. также* **resource, simple link**.

ASCII (American standard code for information interchange)

Произносится «АСКИ». Этот почтенный стандарт описывает набор из 128 символов, предназначенных для отображения текста. Раньше, когда развитие компьютеров происходило, в основном, в США, этого было достаточно для всех операций с текстом. Однако сейчас он заменен более крупными наборами символов, такими как Unicode, которые содержат буквы, символы и идеограммы практически для всех мировых языков. Несмотря на это, ASCII еще долго будет использоваться. UTF-8 является новым набором символов, основанным на ASCII с возможностью обращения к любым символам Unicode. *См. также* **character encoding, character set**.

attribute — атрибут

Переменная или терм, которые определяют конкретную настройку элемента или передают ему дополнительную информацию. Атрибуты имеют вид пар имя-значение в на-

чальном теге элемента. *См. также* **element, markup**.

block element – блочный элемент

Блок текста или содержимого, например, абзац, заглавие или раздел, который отделен от остального текста пробельными символами. *См. также* **inline element**.

box model – блочная модель

Понятие CSS (таблицы стилей), используемое для форматирования блочных элементов. Блочная модель формирует вокруг элемента границу, имеющую свойства, которые можно задавать, например, заполнение и ширина полей. *См. также* **presentation, stylesheet**.

candidate recommendation – кандидат в рекомендации

В процессе стандартизации кандидат в рекомендации является спецификацией, относительно которой в ее рабочей группе достигнуто согласие, достаточное для представления ее на рассмотрение общественности. *См. также* **recommendation, standards body**.

catalog – каталог

Особым образом форматированный текстовый файл (обычно локальный), информация в котором используется для разрешения открытых идентификаторов в системные идентификаторы. Формат файла каталога был формально определен в OASIS Technica Resolution 9401:1997. *См. также* **formal public identifier, URI**.

CDATA (character data) – символьные данные

CDATA является типом данных сущностей, состоящих из неанализируемых символов. Ссылки на сущности, содержащиеся в таких данных, не будут разрешены. Раздел, помеченный CDATA, выглядит так:

```
<![CDATA не анализируемое содержимое]]>
```

См. также **PCDATA**.

character encoding – кодировка символов

Представление символов как уникальных чисел в наборе символов. *См. также* **character set**.

character entity – символьная сущность

Обозначение для любого символа или знака, использующее его номер в наборе символов или аббревиатуру. Обычным синтаксисом для кодирования символа являются амперсанд (&), за которым следуют имя или знак # и число, заканчивающиеся точкой с запятой. Например, символ © (Copyright) может быть выведен в документе с помощью `&#169;` или `&copy;`.

character set – набор символов

Совокупность букв, чисел и символов, представляющих язык или группу языков, отображаемая в заданные числа, которую могут понимать компьютерные программы. *См. также* **character encoding**.

comment – комментарий

Специальным образом помеченный текст в исходном документе, который не интерпретируется анализатором. В XML комментарии заключены в разделители `<!--` и `-->`. Все, что находится внутри комментария, игнорируется анализатором, включая элементы в тегах. В комментариях может содержаться дополнительная информация о разметке и содержимом документа. Они также предоставляют удобный способ изъять содержимое из выходных данных, не удаляя их полностью. *См. также* **markup**.

container element — элемент-контейнер

Элемент, содержащий символьные данные или другие элементы, называется контейнером. Он имеет свойство быть корнем своего собственного поддерева в документе. *См. также* **content**, **content model**, **element**.

content — содержимое, «контент»

Все, что находится в документе и не является разметкой. Уберите теги, комментарии и инструкции обработки, и все, что останется, будет содержимым или символьными данными. Разметка позволяет изменять назначение содержимого различными способами. *См. также* **container element**, **content model**.

content model — модель содержимого

Техническая спецификация DTD, описывающая содержимое элемента. Модель содержимого определяет, какого рода элементы и данные могут находиться в элементе, сколько их может быть и в каком порядке. *См. также* **element**, **content**.

CSS (Cascading Style Sheets) — каскадные таблицы стилей

Эта спецификация определяет стандартный способ задания представления документа путем применения к элементам правил форматирования. Каскадирование обозначает способ форматирования элемента, когда есть несколько перекрывающихся правил, например, локально используемое и глобальное правила. *См. также* **presentation**, **stylesheet**.

current node — текущий узел

Узел, в котором вычисляется выражение. *См. также* **current node set**, **node**, **XSLT**.

current node set — текущее множество узлов

Группа выбранных узлов, создающих непосредственный контекст для выражения. *См. также* **current node**, **node**, **XSLT**.

declaration — объявление

Особый объект, конфигурирующий окружение документа. Объявление может вводить новый элемент, создавать сущность или давать имя типу документа. В объявлениях используется особый ограничитель, позволяющий отличать их от элементов, имеющий восклицательный знак после открывающей угловой скобки:

```
<!name statement>
```

где *name* служит типом объявления, а *statement* содержит остальную часть информации, необходимой для создания объявления. *См. также* **markup**.

delimiter — ограничитель

Любой символ или группа символов, отделяющие данные от разметки. Ограничителями являются угловые скобки (< >) в тегах XML. Разделы CDATA имеют три ограничителя: <![, [и]]>. *См. также* **markup**.

document — документ

В XML документом является весь корневой элемент после того, как разрешены все ссылки на внешние сущности. Он может содержать не более одного необязательного объявления XML и одного определения типа документа. Документ может быть распределен по нескольким файлам, которые могут располагаться на разных машинах. *См. также* **document element**, **root element**.

document element — элемент документа

Называемый также корневым элементом, элемент документа является самым внешним элементом в доку-

менте. Он содержит все, кроме пролога документа и находящихся вне его комментариев или инструкций обработки. *См. также* **document, root element**.

document instance – экземпляр документа

Реальный документ, согласующийся с общей моделью документа. *См. также* **document, DTD**.

document model – модель документа

Шаблон документа, определяющий элементы и модели их содержимого. DTD или схема являются моделями документа. *См. также* **DTD, schema**.

DOM (document object model) – объектная модель документа

Спецификация, определяющая структуру документа как совокупность объектов и способ доступа к документу и его обработки. Объектная модель документа является API (интерфейсом прикладного программирования), который описывает, как в действительности программы интерпретируют структуру и содержимое документа.

document prolog – пролог документа

Раздел в начале документа, в котором объявляется принадлежность документа XML и указывается версия XML, которой он удовлетворяет (например `<?xml version="1.0"?>`). Могут быть объявлены дополнительные сведения о документе – обычно определение типа документа. Пролог документа предшествует элементу документа, или корневому элементу, в котором находится все содержимое документа. *См. также* **document, document type declaration, XML declaration**.

document tree – дерево документа

Каждый документ XML может быть представлен в особой структурной форме, которая называется деревом. Она древообразна, поскольку исходит из одной точки (корня) и ветвится вплоть до листьев. Каждая точка дерева, где происходит ветвление, называется узлом. Дерево состоит из корневого узла, большого числа узлов ветвления и небольшого числа оконечных вершин (листьев). Странно, но деревья документов обычно рисуют «вверх тормашками», т. е. корень находится наверху.

Деревья документов могут быть разделены на несколько меньших деревьев, поскольку каждый узел можно считать корнем собственного поддерева. Это обстоятельство важно для понимания трансформаций XML с использованием XSLT, в ходе которых деревья фактически разрубаются на все меньшие и меньшие поддеревья, а затем результирующее дерево собирается в обратном порядке. *См. также* **document, node tree**.

document type declaration – объявление типа документа

Раздел `DOCTYPE` в прологе документа. В этом разделе объявляется структура, которой должен удовлетворять документ. Может использоваться для указания DTD, предоставляя его открытый идентификатор и местонахождение через URI, и для перечисления внутреннего подмножества объявлений сущностей, используемых в документе. *См. также* **document, DTD**.

DTD (document type definition) – определение типа документа

Группа объявлений, в которых определяются имена элементов и их атрибуты и задаются правила для их сочетания и последовательности. *См. также* **document, document type declaration**.

editor – редактор

Для того чтобы создать текст на компьютере, нужно воспользоваться ре-

дактором – компьютерной программой, которая оформляет нажатия клавиш в виде файлов. Одни редакторы обладают бо́льшими возможностями, чем другие. Простейшим из редакторов является *vi* – старый, но все еще эффективный редактор Unix. Emacs является прекрасным редактором с массой возможностей конфигурирования, но с несколько крутой кривой обучения и отсутствием представления стилей. Оба эти редактора одновременно показывают и разметку, и текст, что может затруднить редактирование для тех, кто не привык видеть в содержимом все теги. *См. также* **document**, **markup**.

element – элемент

Определенная (defined) часть документа XML. Элементы XML обозначаются начальным и конечным тегами и могут содержать данные и другие элементы в определенной иерархии. *См. также* **attribute**, **content**, **markup**.

empty element – пустой элемент

Элемент, образованный из единственного тега и не содержащий данных содержимого. Пустой элемент начинается с открывающей угловой скобки (<), за которой следует имя элемента, возможно, атрибуты со своими значениями и завершается косой чертой и закрывающей угловой скобкой (/>). *См. также* **attribute**, **content**, **element**.

entity – сущность

Имя, посредством объявления присваиваемое какому-либо участку данных. Некоторые сущности предопределены для специальных символов, таких как <, > и &, которые не могут непосредственно использоваться в содержимом документа XML, т. к. они вступили бы в конфликт с разметкой. *См. также* **entity reference**, **markup**.

entity reference – ссылка на сущность

Специальная строка, которая ссылается на сущность, обозначаемая & в начале и точкой с запятой в конце. Ссылки на сущности появляются в тексте и подвергаются анализу процессором XML. *См. также* **entity**.

external entity – внешняя сущность

Сущность, ссылающаяся на другой документ. *См. также* **entity**, **file**, **URL**.

external subset – внешнее подмножество

Группа объявлений, составляющих все или часть определений типов в документе, хранимая во внешней сущности, ссылка на которую производится из DTD с помощью открытого (public) или системного (system) идентификаторов. *См. также* **document**, **DTD**.

FPI (formal public identifier) – формальный открытый идентификатор

Открытый идентификатор, удовлетворяющий спецификации открытых идентификаторов ISO 8879. FPI содержит такую информацию, как класс ресурса (то есть DTD), его автор и язык. *См. также* **catalog**, **external entity**.

fragment identifier – идентификатор фрагмента

Расширение URL, указывающее на местоположение внутри документа HTML с помощью именованного элемента. *См. также* URL.

HTML (hypertext markup language) – язык разметки гипертекста

Язык разметки, используемый при создании документов World Wide Web. *См. также* **hypertext**, **markup**, **XHTML**.

hypertext — гипертекст

Способ связывания текста или объектов в документе, который позволяет осуществлять нелинейный доступ к содержимому. *См. также* **HTML**, **markup**, **XHTML**.

inheritance — наследование

Метод, с помощью которого объект сохраняет параметры, приписанные его родительскому объекту. *См. также* **CSS**, **rule**.

inline element — внутритекстовый элемент

Элемент, который находится внутри текстового содержимого другого элемента, например, подчеркнутый член внутри абзаца. *См. также* **block element**.

internal subset — внутреннее подмножество

Элементы, атрибуты и прочие объявления, которые составляют по крайней мере часть DTD и содержатся внутри объявления типа документа. *См. также* **document type declaration**, **external subset**.

ISO (International Standards Organization) — Международная организация стандартизации

Эта организация основана в 1947 году для создания открытых технических спецификаций. *См. также* **open standard**, **recommendation**, **standards body**.

local resource — локальный ресурс

Ресурс, который содержит ссылку. *См. также* **remote resource**, **resource**.

logical structure — логическая структура

Узловая, или иерархическая, структура элементов и содержимого в документе, в противоположность физическому расположению элементов и данных. *См. также* **document**, **physical structure**.

markup — разметка

Совокупность символов, которые группируют, организуют и помечают участки содержимого в документе. Теги разметки перемежаются с содержимым как инструкции для анализатора, который удаляет их по мере построения структуры документа в памяти. Разметка содержит начальные и конечные теги элементов, ссылки на сущности, объявления, комментарии, инструкции обработки и маркированные разделы. *См. также* **element**, **tag**.

markup language — язык разметки

Набор формальных правил для представления данных и кодирующих структур, окружающих данные. Документ, который подчиняется правилам языка разметки, называется корректным (well-formed). Сам XML является не языком разметки, а набором правил для построения языков разметки. (Может быть поэтому он является языком мета-разметки?) Язык разметки предоставляет способы, позволяющие помечать части документа с помощью элементов, навязывать структуру с помощью DTD и импортировать данные с помощью ссылок на сущности. *См. также* **application**, **DTD**, **markup**.

metadata — метаданные

Описательные данные, не включаемые непосредственно в содержимое документа. Например, дата создания документа или его автор являются метаданными. *См. также* **document**, **markup**.

mixed content — смешанное содержимое

Смесь элементов и символьных данных, которые могут быть заданы как допустимое содержимое элемента в его модели содержимого. *См. также*

CDATA, **content**, **content model**, **element**.

modular DTD — модульное DTD

DTD, разделенное на части, что позволяет легко его сопровождать и отбирать только те модули, которые нужны в документе. Модули в таком DTD часто хранятся во внешних файлах и объявляются как внешние сущности. *См. также* **DTD**.

name — имя

Любой объект в XML имеет спецификатор типа, называемый именем. Например, элемент, который описывает заглавие книги, может получить имя «booktitle». Представление элемента должно содержать имя в начальном и конечном тегах, например:

```
<booktitle>Wart Removal</booktitle>
```

Имена в XML должны подчиняться правилам сочетания символов. Например, имена элементов должны начинаться с символа подчеркивания или буквы. *См. также* **attribute**, **element**, **markup**.

namespace — пространство имен

Любой из группы определенных элементов и атрибутов, которые могут быть использованы в документе путем указания идентификатора пространства имен как префикса имени элемента, например, `<namespace:element>`. Пространства имен должны быть объявлены с помощью объявления `xmlns`. В объявлении пространства имен применяется следующий синтаксис:

```
<xmlns:name=uri>
```

Имя пространства имен задается в *name*, а адрес того, кто сопровождает пространство имен, или номер его версии задается *uri* (хотя анализаторы обычно никак не используют эту информацию). Пространства имен позволяют смешивать в одном документе различные наборы определений элементов, например, при использовании математических уравнений в документе HTML. *См. также* **element**, **qualified element name**.

namespace prefix — префикс пространства имен

Идентификатор, предшествующий имени элемента, указывающий на пространство имен, которому оно принадлежит; например, `<namespace:element>`. *См. также* **element**, **qualified element name**, **namespace**.

node — узел

Этот термин происходит из информатики, где он означает точку в абстрактной сети соединенных между собой объектов. В XML он относится к точке ветвления в дереве документа или конечной вершине дерева. Узлы, которые распознает XPath, включают в свой состав элементы, атрибуты, инструкции обработки, комментарии, непрерывные участки символьных данных (текст), объявления пространств имен и корневой узел. *См. также* **node tree**.

node tree — дерево узлов

Иерархическое представление узлов в документе. Начинаясь с корневого узла, дерево узлов показывает, какие узлы содержат другие узлы. *См. также* **document tree**, **node**.

notation — нотация

Данные, которые не должны анализироваться или требующие специальной обработки, помечаются как тип нотации с помощью атрибута или внешней сущности. Объявление нотаций в DTD определяет типы нотаций, используемые процессором XML, чтобы направить данные специальному обработчику.

open standard — открытый стандарт

Техническая спецификация, открытая обществу для неограниченного использования.

PCDATA

PCDATA, или анализируемые символьные данные, является типом содержимого элемента, состоящим из анализируемых символов (то есть ссылок на сущности), но не элементов. Ссылки на сущности, входящие в эти данные, будут разрешены. *См. также* **CDATA**, **parsed-character data**.

parsed-character data — анализируемые символьные данные

Любые символьные данные, в которых процессор XML должен искать ссылки на сущности. Эти ссылки разрешаются и заменяющий их текст анализируется рекурсивно, для того чтобы заменить все ссылки на сущности. *См. также* PCDATA.

parser — синтаксический анализатор

Программа, которая читает XML, проверяет его действительность и передает для дальнейшей обработки. Если документ построен неправильно (т. е. в разметке есть ошибка), анализатор перехватывает ее и сообщает о проблеме. *См. также* **markup**, **XML processor**.

physical structure — физическая структура

Физическая организация данных в файле, в противоположность структуре документа, или логической структуре. Объектно-ориентированная база данных, например, является физической структурой, которая отличается от ее логической структуры. *См. также* **document**, **logical structure**.

presentation — представление

Внешний вид документа, отформатированного для использования человеком. *См. также* **stylesheet**.

processing instruction — инструкция обработки

Объект разметки, передающий информацию определенному процессору XML. Он имеет вид:

```
<?name data?>
```

где *name* является ключевым словом, которое должно быть замечено конкретным процессором, а *data* является строкой, содержащей специальные данные. Всякий процессор, который не распознает инструкцию обработки, игнорирует ее.

properties declaration — объявление свойств

Часть правила таблицы стилей, которая устанавливает форматирование для выбранного элемента. *См. также* **CSS**, **rule**, **stylesheet**.

proposed recommendation — предлагаемая рекомендация

Спецификация, которая прошла оценку общественности и рассматривается организацией стандартизации, чтобы получить полную рекомендацию.

pseudo-class — псевдокласс

Селектор CSS, задающий конкретный экземпляр элемента вместо всех вхождений элемента. Например, первые абзацы всех разделов могут быть объединены в псевдокласс. *См. также* **CSS**, **rule**, **selector**.

PUBLIC identifier — открытый идентификатор

Идентификатор в прологе документа, устанавливающий имя DTD или внешней сущности.

qualified element name – уточненное имя элемента

Идентификация элемента в конкретном пространстве имен. Уточненное имя элемента использует префикс пространства имен в имени тега. *См. также* **element**, **namespace**, **namespace prefix**.

recommendation – рекомендация

В процессе стандартизации рекомендация является спецификацией, получившей одобрение большинства членов своей рабочей группы и опубликованной для сторонней оценки.

reference – ссылка

Ссылаться на что-либо означает указывать на связь между текущим контекстом и внешним ресурсом. Обычно целью является импортирование значения в совокупность данных. Например, ссылка на сущность является участком текста, указывающим анализатору XML, что нужно ввести значение сущности, определенной в DTD и хранящейся в памяти. Ссылка на ID представляет собой обозначение, указывающее на связь с элементом, обладающим уникальным ID и находящимся где-то в другом месте документа. Термин *ссылка* используется для проведения различия между объектом (или значением) и тем, что пытается его использовать. *См. также* **entity**, **entity reference**.

relative location term – терм относительной адресации

Адрес XPointer, который определен на основе ссылки на другой адрес, например, терм абсолютной адресации или текущий узел. *См. также* **absolute location term**, XPath, XPointer.

relative URL – относительный URL

URL, описывающий частичный адрес. Этот адрес понимается как относительный к текущему адресу ссылки, который называется базовым URL.

remote resource – удаленный ресурс

Ресурс, на который указывает простая ссылка. *См. также* **local resource**, **resource**.

resource – ресурс

Источник информации. В XML ресурсом является нечто, к чему можно присоединиться, например, документ XML, графический файл или программа.

root element – корневой элемент

Элемент документа, находящийся на базовом уровне и содержащий все остальные элементы документа. (То же, что **элемент документа**.) *См. также* **document**, **document element**, **element**.

root node – корневой узел

Узел документа, находящийся на базовом уровне и содержащий все узлы, образующие документ. *См. также* **node**.

rule – правило

Основной конструктивный элемент таблицы стилей, в котором указано, для какого элемента (или группы элементов) установить стиль (селектор) и какой стиль применить (объявление свойств). *См. также* **stylesheet**.

SAX (Simple API for XML) – простой API для XML

Управляемый событиями интерфейс прикладного программирования для обработки документов XML с использованием Java.[1] Этот API описывает плоскую модель документа

[1] Известны реализации SAX для C, Python и других языков. – *Примеч. науч. ред.*

(без иерархии объектов или наследования), которая позволяет быстро обработать документ.

scheme — схема

Префикс URL, устанавливающий тип адреса и применяемый протокол. Например, префикс `http` указывает, что нужно использовать протокол передачи гипертекста, как в следующем URL:

```
http://www.oreilly.com/
```

См. также **URL**.

selector — селектор

Часть правила таблицы стилей CSS, определяющая элементы, к которым применяется стиль. *См. также* **CSS**, **rule**, **stylesheet**.

SGML (standard generalized markup language)

Международный стандарт (ISO-8879), определяющий правила создания независимых от платформ языков разметки электронных текстов.

simple link — простая ссылка

Простейшая форма ссылки, состоящая из элемента в документе (локальный ресурс), в котором задан целевой объект или адрес удаленного ресурса. *См. также* **local resource**, **remote resource**.

standards body — организация стандартизации

Организация, занимающаяся созданием технических стандартов для всей отрасли.

stylesheet — таблица стилей

Набор инструкций форматирования, находящихся в отдельном файле или сгруппированных внутри документа, которые определяют внешний вид документа. Существует несколько стандартов таблиц стилей, в том числе CSS, XSLT и XSL-FO.

SYSTEM identifier — системный идентификатор

Локальный, зависящий от системы идентификатор документа, DTD или внешней сущности. В XML системный идентификатор должен быть URI.

tag — тег

Имя элемента, заключенное в угловые скобки, используемое для разметки семантики или структуры документа.

Unicode

Стандарт набора символов с 16-разрядной кодировкой, пытающийся охватить символы из всех главных существующих в мире видов письма.

URI (uniform resource identifier) — универсальный идентификатор ресурса

Установленная W3C кодификация синтаксиса имени и адреса для настоящих и будущих объектов в Интернет. В базовом формате URI состоит из имени схемы (например http, ftp, mailto и т. п.), далее двоеточия и затем пути, определяемого предшествующей схемой. Термин URI охватывает URL и все другие универсальные идентификаторы ресурсов.

URL (uniform resource locator) — универсальный локатор ресурса

Имя и адрес существующего объекта, доступного через Интернет.

UTF-8

UCS Transformation Format for 8-bit platforms (формат преобразования UCS для 8-разрядных платформ). UTF-8 является преобразованием кодировки символов Unicode, в результате которого становится возможным их использование на 8-разрядных системах.

well-formed — правильно построенный

Термин, описывающий документ, согласующийся с синтаксическими правилами XML. Например, в правильно построенном документе теги имеют надлежащие ограничители, концевой тег следует за начальным тегом, элементы не перехлестываются.

working draft — рабочий проект

Находящаяся в развитии версия спецификации, выработанная рабочей группой организации стандартизации. Рабочий проект часто существенно изменяется перед тем, как стать рекомендацией.

XHTML

XHTML является переформулировкой HTML 4 как приложения XML. DTD XHTML определяет те же элементы и атрибуты, что и HTML 4.01. *См. также* **HTML**, **hypertext**, **markup**.

XML declaration — объявление XML

Первый элемент в прологе документа, объявляющий, что документ является документом XML, и указывающий версию XML, которой он удовлетворяет. В большинстве документов первой строкой является такое объявление XML:

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

XLink (XML Linking Language)

Этот язык задает элементы, которые можно использовать в документах XML для создания и описания ссылок между ресурсами. XLink обеспечивает более надежные отношения ссылки, чем это делают простые гиперссылки в HTML.

XML processor — процессор XML

Общее название всех программ, принимающих на входе документ XML и что-то с ним делающих. Программа, которая читает документ XML, производит его синтаксический анализ и осуществляет форматированный вывод, является процессором XML. *См. также* **parser**.

XML Schema

Альтернатива DTD для моделирования документов. Схемы пишутся как XML и, подобно DTD, определяют элементы, сущности и модель содержимого документов. Схемы имеют многие дополнительные возможности, такие как контроль типов данных и ограничения на содержимое.

XPath (XML path language)

Язык, предназначенный для обращения к частям документа XML. Синтаксис локатора XPath использует базовые функции, основанные на иерархии узлов документа, и вычисляет выражения для определения местонахождения объекта. Адреса XPath применяются в XSLT и XPointer.

XPointer (XML Pointer Language)

Специальная схема, основанная на XPath, которая задает местоположение с использованием особых расширений URL. Адреса XPointer используют атрибуты ID для абсолютных адресных ссылок в документе XML и перемещение по иерархии узлов при поиске конкретных элементов.

XSL (Extensible Stylesheet Language)

Язык, который позволяет прикреплять к документам XML таблицы стилей. Таблица стилей XSL сама является документом XML.

XSLT (XSL Transformations)

Преобразование, действующее аналогично таблице стилей, но вместо простого применения правил форматирования к элементам может изменять структуру документа для создания документа с новой структурой.

Алфавитный указатель

Специальные символы

Числа

A

B

C

D

E

F

O

P

R

S

T

U

V

W

X

A

Б

В

Г

K

Л

М

Н

П

P

C

T

У

Э

Я