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.

- *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-

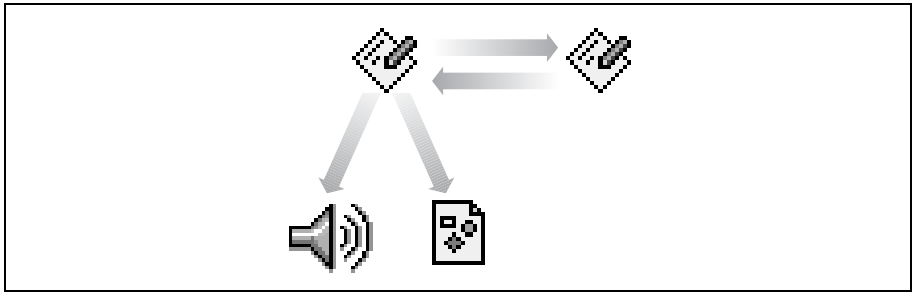

Рис. 3.1. *Группа ресурсов, соединенных ссылками*

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.

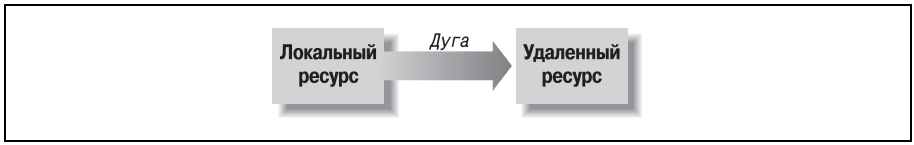

Рис. 3.2. *Простая ссылка*

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

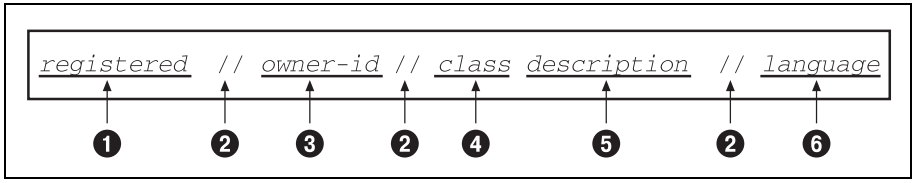

Рис. 3.3. *Синтаксис формального открытого идентификатора*

[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.

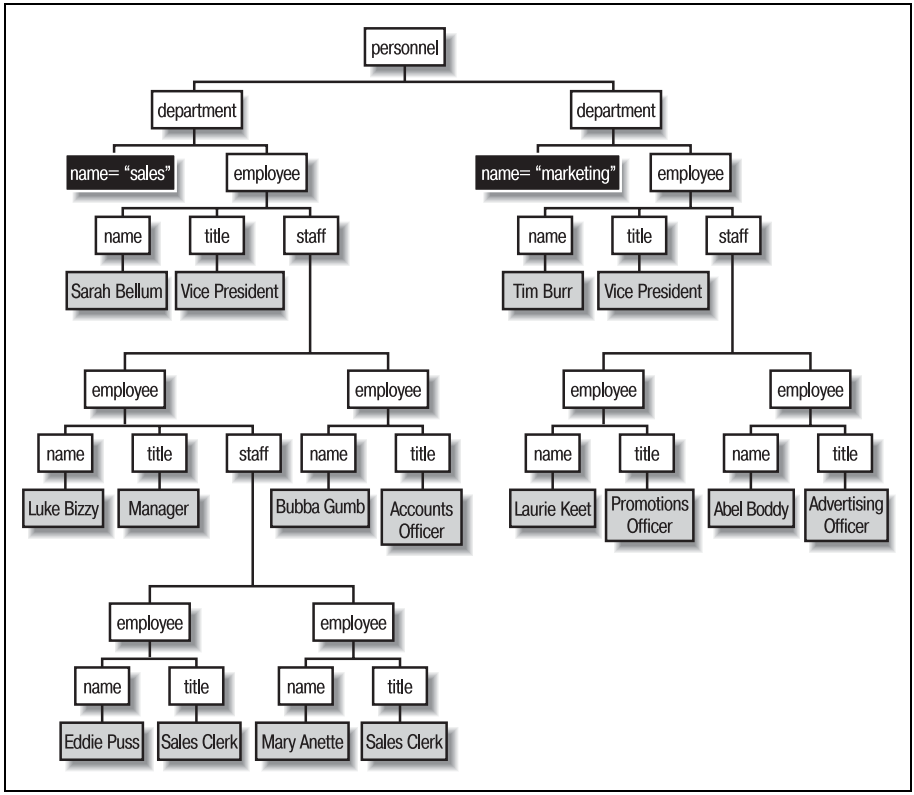

Рис. 3.4. *Представление структуры организации в виде дерева*

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.

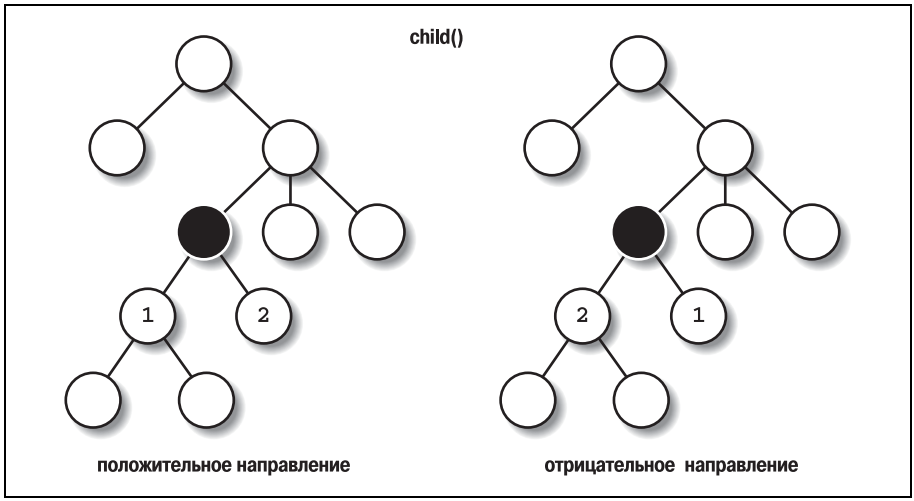

Рис. 3.5. *Путь child( )*

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.

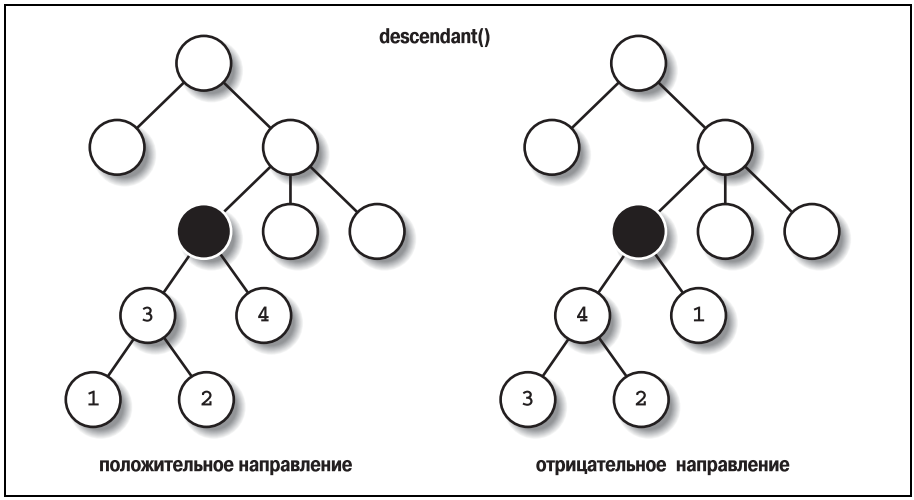

Рис. 3.6. *Путь descendant( )*

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.

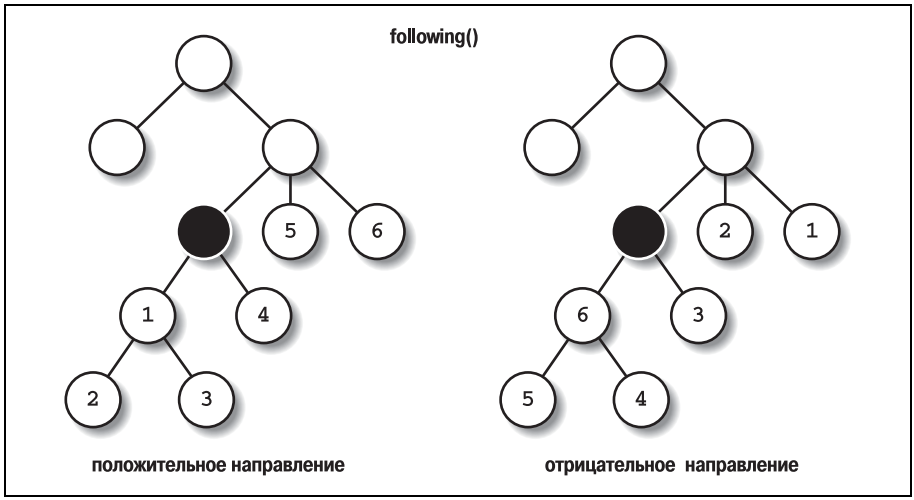

Рис. 3.7. *Путь following( )*

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.

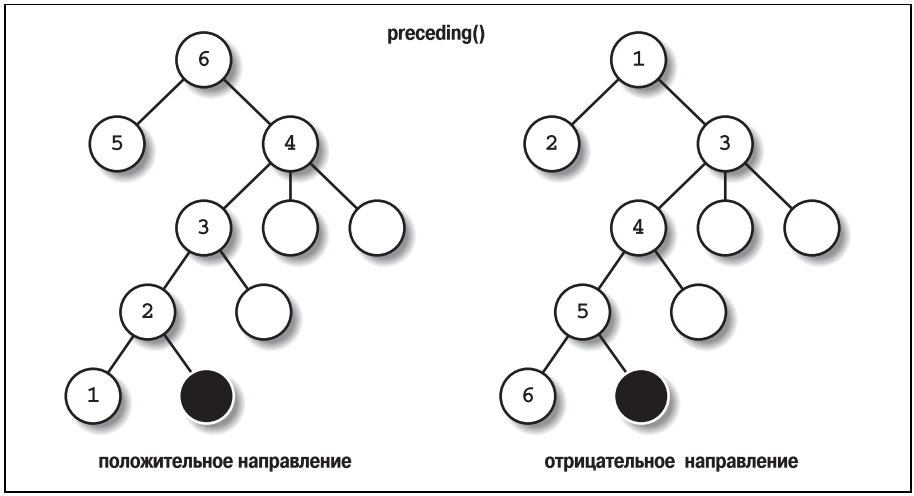

Рис. 3.8. *Путь preceding( )*

`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.

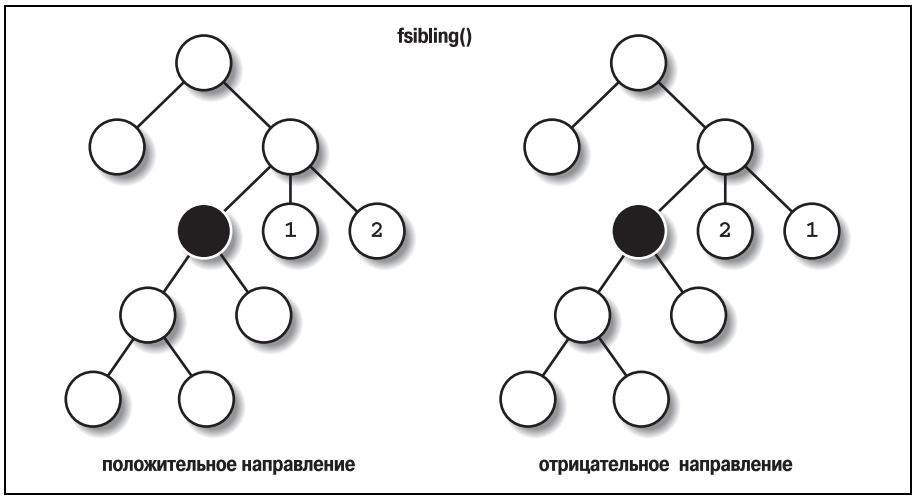

Рис. 3.9. *Путь fsibling( )*

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.

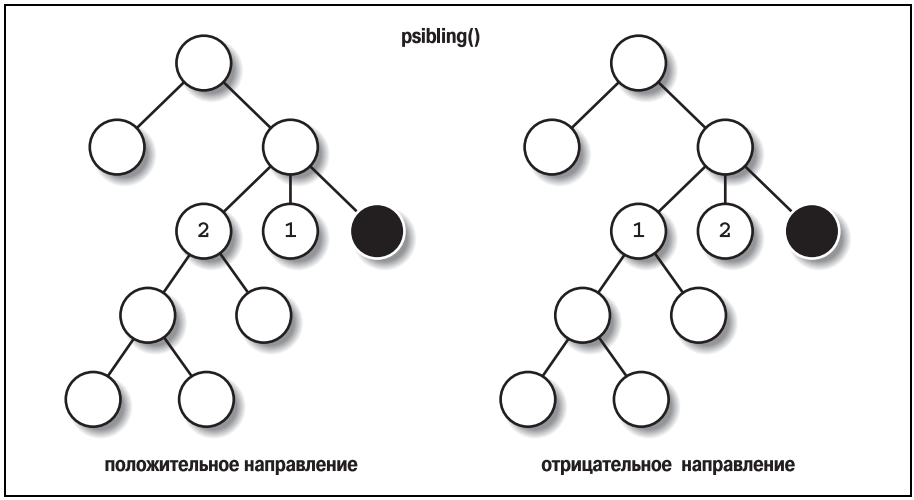

Рис. 3.10. *Путь psibling( )*

`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').
```

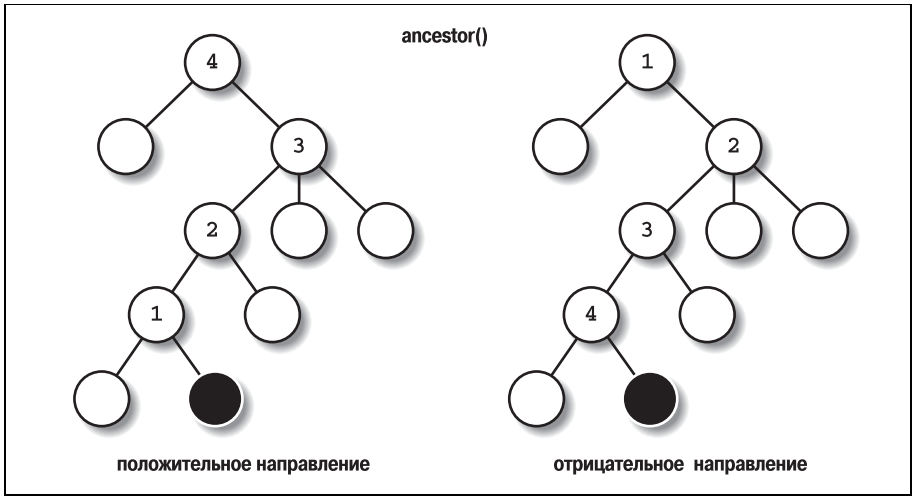

Рис. 3.11. *Путь ancestor( )*

```
    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.