Реальные проблемы и решения

2-е издание

Java™ и XML

O'REILLY®

Бретт Мак-Лахлин

Java & XML

Second Edition

Brett McLaughlin

O'REILLY®

Java и XML

Второе издание

Бретт Мак-Лахлин

Символ

Санкт-Петербург
2002

Бретт Мак-Лахлин

Java и XML

Перевод Т. Морозовой

Главный редактор	*А. Галунов*
Зав. редакцией	*Н. Макарова*
Научный редактор	*М. Зислис*
Редактор	*В. Овчинников*
Корректура	*С. Беляева*
Верстка	*А. Дорошенко*

Мак-Лахлин Б.
Java и XML. – Пер. с англ. – СПб: Символ-Плюс, 2002. – 544 с., ил.
ISBN 5-93286-018-9

Java и XML. Эти две технологии уже давно привлекают внимание разработчиков. И не зря. Они идеально подходят для создания веб-ориентированных корпоративных приложений, обеспечивают независимость от платформы, расширяемость, возможность повторного использования кода, а также поддержку стандарта Unicode. Их соединение позволяет создавать веб-сайты с динамически обновляемыми страницами, разрабатывать корпоративные приложения, снижающие затраты на совместное использование информации, и находить простые и эффективные решения проблемы переносимости данных. Автор описывает применение всего арсенала инструментов и средств XML и Java. Здесь и DTD и пространства имен, XML Schema и XPath, XSL и различные API (SAX, DOM, JDOM). Рассматривается связывание данных, разработка приложений при помощи XML-RPC и SOAP, использование систем веб-публикации (например, Apache Cocoon). Не оставлены вниманием создание веб-служб с применением SOAP, UDDI и WSDL, каналы RSS, динамические данные и XSP.

Второе издание «Java и XML» дополнено главами о расширенных возможностях SAX и DOM, а также о SOAP и связывании данных. Эта книга станет незаменимым спутником для тех, кто пишет программы на Java и собирается применять XML (или планирует заниматься этим), участвует в движении peer-to-peer, разрабатывает программное обеспечение для электронной коммерции либо использует службу сообщений или веб-службы.

ISBN 5–93286–018–9
ISBN 0–596–00197–5 (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.
Налоговая льгота – общероссийский классификатор продукции
ОК 005-93, том 2; 953000 – книги и брошюры.
Подписано в печать 27.05.2002. Формат 70x100 1/16 . Печать офсетная.
Объем 34 печ. л. Тираж 2000 экз. Заказ N
Отпечатано с диапозитивов в Академической типографии «Наука» РАН
199034, Санкт-Петербург, 9 линия, 12.

Оглавление

Предисловие

Когда я чуть больше года назад писал предисловие к первому изданию «Java и XML», я даже и не представлял себе, с чем связываюсь. Я отпускал шуточки о том, что XML скоро появится на шапках и футболках; сейчас, когда я пишу эти строки, на мне футболка с вышитой на ней надписью «XML», а еще у меня есть кепка с надписью XML (на самом деле у меня их даже две!). Так что XML, без сомнения, оправдал все ожидания. И это хорошо.

Но это означает, что каждый день ведутся новые разработки, и ландшафт XML расширяется с такой скоростью, которую я не мог себе представить даже в самых смелых предположениях. И хотя для XML это здорово, при обращении к первому изданию это ввергает в уныние: почему все настолько устарело? Я говорил о SAX 2.0 и DOM Level 2 как о чем-то новом. Теперь это промышленные стандарты. Я только знакомил читателей с JDOM, а теперь он входит в JSR (Sun's Java™ Specification Request, запрос на разработку или изменение спецификации Java-технологии). Я даже не рассматривал SOAP, UDDI, WSDL и связывание данных XML. В этом издании им отведено три главы! Все изменилось, и это не преувеличение.

Даже тем, кто только подозревает, что в ближайшие несколько месяцев, возможно, будет работать с XML, эта книга, весьма вероятно, будет полезна. А если где-то на вашем рабочем столе есть первое издание этой книги, то просмотрите и новое издание; думаю, вы обнаружите, что оно все же актуально. Я избавился от чрезмерных описаний основных концепций, собрал материал об основах XML в одной главе и переписал практически все примеры. Кроме того, я добавил много новых примеров и глав. Другими словами, я попытался создать подробную, многогранную техническую книгу. Начинающим на ее изучение потребуется немного больше времени, поскольку помощи стало меньше, зато знаний будет гораздо больше.

Структура этой книги

Эта книга очень четко структурирована: в ее первой половине – главах с 1 по 9 – основное внимание уделяется изучению основ XML и ключевых интерфейсов прикладного программирования Java для обработки XML. Каждому из трех интерфейсов для работы с XML (SAX, DOM

и JDOM) автор посвящает по главе, в которой рассматривает базовые понятия, и еще по главе, описывающей более сложные концепции. Глава 10 – переходная, здесь начинается подъем по «стеку» XML. В ней рассматривается JAXP, представляющий собой уровень абстракции над SAX и DOM. Остальная часть книги – главы с 11 по 15 – посвящена специфическим вопросам, которые постоянно поднимаются в конференциях и руководствах, с которыми я связан, и предназначена для желающих научиться применять XML в своих приложениях. Среди этих тем новые главы по SOAP, связыванию данных и новый взгляд на приложения business-to-business. Наконец, есть еще два приложения, завершающие книгу. Итак, содержимое книги таково:

Глава 1 «Введение»

Посмотрим, о чем, собственно, шум, изучим начала XML и уделим время обсуждению причин, по которым XML настолько важен для настоящего и будущего корпоративных разработок.

Глава 2 «Основы технологии»

Это ускоренный курс основ XML, от XML 1.0 до DTD и от XML Schema до XSLT и пространств имен. Для тех, кто читал первое издание, это сумма всех глав, посвященных работе с XML, и кое-какой дополнительный материал.

Глава 3 «SAX»

В этой главе вводится и рассматривается простой API для XML (SAX), первый Java API для обработки XML. Подробно рассматривается жизненный цикл процесса анализа, а затем демонстрируются события, которые SAX может перехватывать и которые могут использоваться разработчиками.

Глава 4 «Расширенный SAX»

Здесь мы еще глубже познакомимся с SAX, рассматривая реже используемые, но тем не менее очень мощные элементы API. Узнаем, как применять фильтры XML для объединения в цепи обратных вызовов, как применять классы `XMLWriter` и `DataWriter` для вывода XML при помощи SAX. Кроме того, мы рассмотрим некоторые из менее распространенных обработчиков SAX, такие как `LexicalHandler` и `DeclHandler`.

Глава 5 «DOM»

В этой главе читатель движется дальше по ландшафту XML к следующему API Java и XML – DOM (Document Object Model, объектная модель документа). Вы изучите основы DOM, выясните, что находится в текущей спецификации (DOM Level 2), и узнаете, как читать и создавать деревья DOM.

Глава 6 «Расширенный DOM»

Продолжая изучать DOM, вы узнаете о различных модулях DOM, например Traversal, Range, Events, CSS и HTML. Также мы посмот-

рим, что собой предоставляет новая версия DOM Level 3 и как применять эти новые возможности.

Глава 7 «JDOM»

Эта глава содержит введение в JDOM и описывает его отличия от DOM и SAX, не оставляя без внимания их сходства. Рассматривается чтение и создание XML при помощи этого API.

Глава 8 «Расширенный JDOM»

Углубляя знакомство с JDOM, мы рассмотрим практические приложения API – интеграцию с JAXP и то, как JDOM обеспечивает использование фабрики с вашими собственными подклассами JDOM. Кроме того, вы увидите, как XPath действует в паре с JDOM.

Глава 9 «JAXP»

В настоящее время JAXP – это развившийся API, поддерживающий анализ и преобразования, и он заслуживает отдельной главы. Тут мы рассмотрим обе версии – 1.0 и 1.1, и вы узнаете, как использовать этот API во всей полноте.

Глава 10 «Системы веб-публикации»

В этой главе рассказано, чем является система веб-публикации, почему она имеет к вам отношение и как выбрать хорошую систему. Затем мы остановимся на системе Apache Cocoon, более внимательно рассмотрим множество ее свойств и то, как она может применяться для создания динамического наполнения веб-страниц.

Глава 11 «XML-RPC»

Здесь рассматриваются удаленные вызовы процедур (Remote Procedure Calls, RPC), их уместность в распределенном окружении по сравнению с RMI и то, как XML делает RPC жизнеспособным решением некоторых проблем. Затем мы перейдем к применению Java-библиотек XML-RPC и созданию клиентов и серверов XML-RPC.

Глава 12 «SOAP»

В этой главе мы остановимся на использовании данных конфигурирования в формате XML и увидим, почему этот формат столь важен для кроссплатформенных приложений, особенно по отношению к распределенным системам и веб-службам.

Глава 13 «Веб-службы»

Продолжая говорить о SOAP и веб-службах, в этой главе мы познакомимся с двумя важными технологиями – UDDI и WSDL.

Глава 14 «Объединение содержимого»

Не покидая русла межкорпоративных (business-to-business) приложений, эта глава представляет еще один способ взаимодействия приложений – при помощи соглашений о содержимом. Вы узнаете о Rich Site Summary, создании информационных каналов и даже немного о Perl.

Глава 15 «Связывание данных»

И снова вверх по стеку XML. В этой главе говорится об одном из высокоуровневых интерфейсов для Java и XML, а именно интерфейсе связывания данных XML. Вы узнаете, что такое связывание данных, как оно позволяет существенно упростить работу с XML, а также осознаете существующие возможности. Я рассмотрю три системы: Castor, Zeus и недавнюю версию JAXB (Java Architecture for XML Data Binding, архитектура Java для связывания данных XML) от Sun.

Глава 16 «Взгляд в будущее»

В этой главе я указываю на несколько интересных технологий, появившихся на горизонте, и привожу немного дополнительной информации по каждой из них. Некоторые из моих догадок могут быть совершенно неверными, другие же могут оказаться действительно чем-то интересным.

Приложение A «Справочник по интерфейсам прикладного программирования»

В приложении подробно описаны все классы, интерфейсы и методы, доступные в интерфейсах SAX, DOM, JAXP и JDOM.

Приложение B «Возможности и свойства SAX 2.0»

В этом приложении подробно рассмотрены возможности и свойства, доступные в реализациях синтаксических анализаторов, поддерживающих SAX 2.0.

Для кого предназначена эта книга?

Эта книга основана на предпосылке, что XML очень быстро становится (и в некоторой степени уже стал) важным аспектом программирования на Java. Материал книги учит вас применять XML и Java, и нигде, кроме как в первой главе, не задается вопрос «*следует ли* применять XML». Разработчикам на Java несомненно следует использовать XML. Вот почему эта книга адресована программистам на Java, а также тем, кто собирается программировать на Java, имеет в подчинении программистов на Java либо связан с проектом, в котором используется Java. Если вы хотите совершенствоваться как разработчик, создавать более понятный код или хотите, чтобы проект был завершен вовремя и не вышел за рамки бюджета, если вам необходим доступ к данным устаревших систем, если вам нужно распределение системных компонентов либо если вы просто хотите знать, почему XML уделяется столько внимания, то эта книга для вас.

Я старался как можно меньше представлять себе конкретного читателя; я не верю, что следует усложнять введение в XML настолько, чтобы невозможно было войти в курс дела. Тем не менее, я также пола-

гаю, что если вы потратили деньги на эту книгу, то вам нужны не просто основы. Поэтому я предполагал лишь, что вы знакомы с языком Java и понимаете основные концепции серверного программирования (скажем, из области Java-сервлетов и Enterprise JavaBeans™). Если вы никогда прежде не писали на Java или только начинаете изучать язык, то, возможно, вам стоит прочесть книгу Пэта Нимайера (Pat Niemeyer) и Джонатана Кнудсена (Jonathan Knudsen) «Learning Java», O'Reilly (Изучаем Java), прежде чем приступать к этой книге. Я не предполагаю, что вы знаете что-либо об XML, поэтому начинаю с простого. Но я полагаю, что вы хотите хорошо поработать и быстро научиться; поэтому базовые понятия мы рассмотрим быстро, а основной объем книги будет посвящен работе с более сложными вещами. Материал не повторяется, если нет явной необходимости, так что кому-то, возможно, понадобится перечитать предшествующие разделы или полистать страницы в обоих направлениях, поскольку рассмотренные ранее понятия используются в последующих главах. У тех, кто хочет изучить XML, немного знаком с Java и готов набирать код примеров в своем любимом текстовом редакторе, не должно возникнуть особых проблем при чтении этой книги.

Программное обеспечение и версии

В этой книге рассмотрен стандарт XML версии 1.0 и различные словари XML по состоянию на июль 2001 года. Поскольку многие затронутые спецификации XML еще не приняты в окончательном варианте, могут существовать незначительные несоответствия между печатным изданием данной книги и текущей версией обсуждаемой спецификации.

Весь приводимый код на Java основан на платформе Java 1.2. Если вы пока еще не пользуетесь Java 1.2, постарайтесь это сделать – одни только классы коллекций стоят этого. Синтаксический анализатор Apache Xerces, процессор Apache Xalan, библиотека Apache SOAP и библиотеки Apache FOP применялись в виде самых свежих доступных стабильных версий на июнь 2000 года, а система веб-публикации Apache Cocoon – в версии 1.8.2. Версия использованных Java-библиотек XML-RPC – 1.0 beta 4. Все обсуждаемое программное обеспечение распространяется свободно, и его можно получить в сети по адресам *http://java.sun.com*, *http://xml.apache.org* и *http://www.xml-rpc.com*.

Исходные тексты примеров из этой книги полностью содержатся на ее страницах. Как исходные тексты, так и скомпилированные версии всех примеров (включая объемную документацию Javadoc, не обязательно включенную в текст книги) доступны в сети по адресам *http://www.oreilly.com/catalog/javaxml2* и *http://www.newInstance.com*. Все примеры, которые могут выполняться в качестве сервлетов либо могут быть преобразованы для выполнения в таком качестве, можно просмотреть и использовать, посетив страницу *http://www.newInstance.com*.

Типографские соглашения

В этой книге приняты следующие типографские соглашения:

Курсивом выделяются:

- Имена путей Unix, имена файлов и имена программ
- Адреса Интернета, включая доменные имена и URL
- Определения новых терминов

Полужирным шрифтом выделены:

- Имена элементов GUI: заголовки окон, кнопки, пункты меню и т. д.

Моноширинным шрифтом выделены:

- Командные строки и параметры, которые необходимо ввести дословно
- Имена и ключевые слова в программах Java, включая названия методов, имена переменных и имена классов
- Имена элементов и тегов XML, имена атрибутов и других конструкций XML, приводимых в том виде, в каком они присутствуют в XML-документе

Комментарии и вопросы

Пожалуйста, направляйте замечания и вопросы относительно этой книги издательству:

O'Reilly & Associates, Inc.
101 Morris Street
Sebastopol, CA 95472
(800) 998-9938 (в США или Канаде)
(707) 829-0515 (международный/местный)
(707) 829-0104 (факс)

Также можно отправить сообщение по электронной почте. Чтобы попасть в наш список рассылки или заказать каталог, воспользуйтесь адресом

info@oreilly.com

Чтобы задать технический вопрос или прокомментировать содержание книги, напишите по адресу

bookquestions@oreilly.com

Мы поддерживаем веб-сайт, посвященный этой книге, на котором будут приведены примеры, замечания о найденных ошибках и планы переизданий. Эта страница расположена по адресу

http://www.oreilly.com/catalog/javaxml2/

Более подробную информацию об этой и других книгах можно найти на веб-сайте издательства O'Reilly:

http://www.oreilly.com

Благодарности

Вот и снова я пишу благодарности. Сейчас вспомнить каждого ничуть не проще, чем в первый раз. Мой редактор Майк Лукидес (Mike Loukides) даже по ночам беспокоится о том, как сделать все правильно, и именно таким должен быть хороший редактор! Кайл Харт (Kyle Hart) – суперженщина из отдела маркетинга – ведет всю работу и напоминает мне о том, что в конце туннеля виден свет. Тим О'Рейли (Tim O'Reilly) и Фрэнк Уиллисон (Frank Willison) терпеливы, но настойчивы, и именно такими должны быть хорошие руководители. А Боб Экштейн (Bob Eckstein) и Марк Лой (Marc Loy) помогали мне разрешать досадные проблемы с графическим интерфейсом Swing. (Кроме того, Боб просто забавный парень. Посмотрим правде в глаза.) O'Reilly – настолько хорошее издательство, насколько это возможно, во всех отношениях. Я горжусь этим сотрудничеством.

Также хочу поблагодарить бесподобную команду моих рецензентов. Много раз они просматривали главу меньше чем за 24 часа, при этом выдавая добросовестный отчет по техническим деталям. Именно эти люди внесли весомый вклад в поддержание технического уровня книги. Роберт Сиз (Robert Sese), Филип Нельсон (Philip Nelson) и Виктор Брилон (Victor Brilon) – все они просто изумительны. Конечно же, я всегда готов благодарить своего «соучастника» Джейсона Хантера (Jason Hunter) за то, что он предан JDOM и другим техническим вопросам (старина, устрой себе выходной на одну ночь!). Наконец, компания, в которой я работаю – Lutris Technologies, – это, пожалуй, лучшее место, в котором можно пожелать работать. Они позволили мне много часов работать над книгой и не жаловались ни разу. В особенности я хочу отметить Янси Линд (Yancy Lind), Пола Моргана (Paul Morgan), Дэвида Юнга (David Young) и Кейт Бигелоу (Keith Bigelow) – они просто мастера своего дела. Спасибо!

Снова спасибо моим родителям – Ларри (Larry) и Джуди Мак-Лахлин (Judy McLaughlin). Я люблю вас за то, что вы вырастили в меру честолюбивого и целеустремленного сына (вы, конечно, понимаете, что эти характеристики также относятся и к невыносимому ребенку!). Сара Джейн (Sarah Jane) – моя тетя, и мои бабушка и дедушка Дин (Dean) и Глэдис Мак-Лахлин (Gladys McLaughlin) никогда не допускали даже и мысли о том, что я о них не думаю только из-за того, что вижу их редко. Дедушка, ты даже не догадываешься, насколько я благодарен тебе за то, что ты собираешься посмотреть на второе издание. Я люблю вас всех.

Спасибо моим вторым родителям (родителям жены), Гари (Gary) и Ширли Грейтхаус (Shirley Greathouse) – вы просто лучше всех. Когда-нибудь я соберусь и объясню, что вы значите для меня, но это может потребовать отдельной книги. Я люблю вас обоих за ваш юмор и мудрость. Спасибо Куину (Quinn) и Джони (Joni) за ту легкость, которую они привносят в наши воскресные обеды. Спасибо Лонни и Лауре, не могу дождаться, когда увижу Беби Джей (Baby J). Спасибо Биллу и Терри за то, что они были очень мудрыми друзьями, а также спасибо Биллу за то, что он такой пастор, какого еще нужно поискать.

Источник смеха в моей жизни – это несколько веселых людей, и я просто не могу не упомянуть их тут: Кендра (Kendra), Британи (Brittany), Лизетт (Lisette), Дженей (Janay), Роки (Rocky), Дастин (Dustin), Тони (Tony), Стефани (Stephanie), Робби (Robbie), Эрин (Erin), Анджела (Angela), Майк (Mike), Мэтт (Matt), Карлос (Carlos) и Джон (John). Я увижусь с вами в воскресенье, и, пожалуйста, давайте не пойдем больше в Mazzio's. Также спасибо моим собакам: Сету, Чарли, Джейку, Мозесу, Молли и Дэйзи. Тот не жил по-настоящему, кого не будил по утрам холодный язык бассет-хаунда.

Наконец, спасибо двум людям, которые значат для меня гораздо больше, чем остальные – моему дедушке Роберту Эрлу Бердону, с которым я вновь встречусь однажды. Я думаю о тебе каждый день, и мои дети тоже вскоре о тебе услышат. Но больше всего я хочу поблагодарить мою жену Ли. Словами этого не выразить. Однажды все песни и слезы, связанные с тобой, выплеснутся наружу, и ты наконец-то поймешь, как много для меня значишь.

Наконец, спасибо Господу, который дал мне достичь столь многого.

- *XML имеет значение*
- *Что важно?*
- *Основы*
- *Что дальше?*

1

Введение

Обычно писать вводные главы очень просто. Практически в каждой книге автор приводит обзор рассматриваемой технологии, разъясняет некоторые основы и пытается заинтересовать читателя. Но в случае со вторым изданием «Java и XML» все немного сложнее. Когда шла работа над первым изданием, многие люди только начинали знакомиться с XML, а многие скептики желали проверить, действительно ли этот новый тип разметки так хорош, как обещает рекламная шумиха. Но прошло чуть больше года, и уже все используют XML сотнями различных способов. В известной степени введение, вероятно, вам не нужно. Но я все же вкратце опишу, о чем пойдет речь, почему тема важна и что понадобится, чтобы начать работу.

XML имеет значение

Во-первых, я расскажу о том, что имеет значение. Да, это звучит как вводная фраза на занятии по аутотренингу, но с этого имеет смысл начать. По-прежнему многие разработчики, менеджеры и руководители боятся XML. Их пугает восприятие XML как передовой технологии и скорость, с которой эта технология обновляется. (Это второе издание, вышедшее через год, верно? Многое ли изменилось?) Они боятся затрат, связанных с наймом людей вроде нас с вами, людей, которые будут работать с XML. Но более всего они боятся дальнейшего усложнения своих и без того головоломных приложений.

Для того чтобы устранить эти страхи, кратко перечислим основные причины, по которым следует начать работать с XML уже сегодня. Во-первых, XML переносим. Во-вторых, он предоставляет невероятный потенциал для взаимодействий. И наконец, XML имеет значение... потому что не имеет значения! Если это окончательно вас запутало, читайте дальше, и очень скоро все станет на свои места.

Переносимость

Во-первых, XML переносим. Если вы давно работаете с Java или когда-либо участвовали в конференции JavaOne[1], то слышали заклинание адептов Java: «переносимый код». Скомпилируйте код на Java, перенесите файлы *.class* или *.jar* в любую операционную систему, и код заработает. Все что нужно – окружение Java Runtime Environment (JRE) или виртуальная машина Java (JVM). Переносимость всегда была одной из наиболее привлекательных особенностей Java. Разработчики могут создавать и тестировать код на рабочих станциях под управлением Linux или Windows, и созданный код будет выполняться на Sparc, E4000, HP-UX – в любой операционной среде, которую только можно представить.[2]

Так что технология XML заслуживает больше, чем мимолетного взгляда. XML – это просто текст, и очевидно, что его можно переносить на различные платформы. Что еще более важно, XML должен соответствовать спецификации, определенной консорциумом World Wide Web (W3C), веб-сайт которого расположен по адресу *http://www.w3.org*. Таким образом, XML – это стандарт. Когда вы посылаете XML-текст, он соответствует этому стандарту; когда некоторое приложение его получает, XML по-прежнему соответствует этому стандарту. Приложение, использующее XML-данные, вправе на это рассчитывать. Так работает Java: любая виртуальная машина Java знает, чего ожидать, и коль скоро код соответствует этим ожиданиям, он будет выполняться. Используя XML, вы получаете переносимые данные. В последнее время можно услышать слова «переносимый код», «переносимые данные», относящиеся к комбинации Java и XML. Это хорошие слова, потому что они правдивы (что не всегда справедливо для рекламных лозунгов).

Способность к взаимодействию

Во-вторых, XML позволяет достичь такого уровня взаимодействия, который никогда раньше не был доступен корпоративным приложениям. Некоторые из читателей, вероятно, думают, что речь идет лишь о еще одной форме переносимости. Это не так. Вспомним, что XML – это *расширяемый язык разметки*. Именно расширяемость так важна во взаимодействии. Рассмотрим для примера HTML – язык гипертекстовой разметки. HTML – это стандарт. Он полностью текстовый. В этом отношении он так же переносим, как и XML. И действительно,

---

[1] Конференция JavaOne (в конвент-центре им. Москони, г. Сан-Франциско) ежегодно собирает десятки тысяч разработчиков и ключевых фигур индустрии Java. Более подробную информацию о конференции можно найти по адресу *http://servlet.java.sun.com/javaone/*. – *Примеч. науч. ред.*

[2] Наличие на целевой платформе соответствующей версии JRE или JVM является обязательным условием. – *Примеч. науч. ред.*

клиенты, использующие различные броузеры и различные операционные системы, могут увидеть в той или иной степени одно и то же. Однако HTML специально разработан в целях визуализации информации. HTML нельзя использовать для представления описи имущества или выписанного счета.[1] Дело в том, что стандарт HTML жестко определяет разрешенные теги, формат документа и прочие особенности. Таким образом сохраняется ориентация на визуализацию, или представление данных, что одновременно является и преимуществом и недостатком.

XML, напротив, практически не затрагивает элементы и содержимое документа. Вместо этого речь идет о структуре документа: элементы должны открываться и закрываться, у атрибута должно быть только одно значение и т. д. Содержимое документа, конкретные элементы и атрибуты оставлены на усмотрение разработчика. Существует возможность создать собственный формат документа, определить его содержимое и правила представления данных, повысив таким образом эффективность взаимодействия. Различные мебельные компании (furniture chains) могут согласовать некоторый набор ограничений для XML, а затем обмениваться данными в этих форматах; они получают все преимущества XML (в частности, переносимость), а также возможность применять свои бизнес-знания к данным, чтобы придать им смысл. В биллинговой системе может быть реализован собственный формат, подходящий для генерации счетов, который будет распространяться в качестве формата для обмена данными, для экспорта и импорта счетов из других биллинговых систем. Расширяемость XML делает эту технологию идеальной для реализации совместной работы приложений.

Еще больше интригует большое количество разрабатываемых вертикальных стандартов[2]. Взгляните на проект ebXML по адресу *http://www.ebxml.org*. Компании ведут совместную работу по созданию основанных на XML стандартов, которые сделают возможной глобальную электронную коммерцию. Аналогичное движение имеет место в индустрии телекоммуникаций. В ближайшем будущем вертикальные

---

[1] На самом деле можно. В основе языка HTML – традиционные DTD-определения; в последних версиях существует атрибут `class`, позволяющий классифицировать произвольные элементы с целью применения стилей. Таким образом, теоретически HTML позволяет передавать определенную логическую нагрузку для элементов. Но, само собой, подобный способ менее практичен, чем применение XML. – *Примеч. науч. ред.*

[2] *Вертикальный стандарт*, или *вертикальный рынок*, – это стандарт или рынок, имеющий своей целью определенную отрасль бизнеса. Вместо движения по горизонтали (когда предпочтительна общая функциональность) акцент делается на вертикальном движении, когда функциональность предназначена определенной аудитории, скажем, производителям обуви или гитарным мастерам.

рынки всего мира договорятся о стандартах обмена данными, построенных на XML.

Потеря значимости

Итак, XML имеет значение потому, что он не имеет значения. Я хочу повторить это снова, поскольку речь идет о причинах, делающих XML важной технологией. Фирменные решения, связанные с хранением данных, двоичные форматы, для декодирования которых существуют конкретные алгоритмы, и другие решения этой серии – вот что в конечном счете значимо и учитывается при анализе. Разбор данных подразумевает взаимодействие с другими компаниями, обширную документацию, написание кода и повторное изобретение инструментов для передачи. Привлекательность XML заключается в том, что эта технология не требует никакой специальной подготовки, она позволяет тратить время на другие задачи. В главе 2 «Основы технологии» примерно на 30 страницах описана бо́льшая часть того, что когда-либо может понадобиться авторам XML-данных. Эти данные требуют документирования, поскольку вся необходимая документация уже существует. Они не требуют применения специальных кодировщиков и декодировщиков, поскольку уже существуют интерфейсы прикладного программирования и анализаторы, выполняющие всю обработку. Эта технология не подвергает пользователя риску, т. к. она уже испытана, и многие миллионы разработчиков ежедневно трудятся над ней, улучшая и дополняя новыми возможностями.

XML очень важен, поскольку становится надежной, не привлекающей лишнего внимания частью приложений. Создайте ограничения, закодируйте данные в XML и забудьте об этой задаче. Переходите к важным вещам – сложной бизнес-логике и представлению, которые требуют недель и месяцев размышлений и тяжелого труда. Между тем, XML будет довольно пыхтеть, без плача и жалоб управляясь с вашими данными (да, я несколько драматизирую, но вы поняли, что я хотел сказать).

Так что если вы боялись XML или скептически к нему относились, присоединяйтесь сейчас. Это может быть самым важным решением, которое вы когда-либо принимали, да еще и с минимальным числом побочных эффектов. В оставшейся части книги будут изучены интерфейсы прикладного программирования, транспортные протоколы и прочие вопросы, с которыми вы можете столкнуться.

Что важно?

Если вы согласны, что XML способен вам помочь, встает следующий вопрос – какая его *часть* вам нужна. Как я говорил ранее, существуют сотни приложений XML, и найти нужное – задача довольно сложная.

Я собрал двенадцать или тринадцать ключевых тем из этих сотен и постарался их для вас подготовить – тоже задача не из простых! К счастью, у меня был год на то, чтобы собрать отзывы по первому изданию этой книги, и я работаю с XML над созданием приложений уже два с лишним года. Это означает, что я по крайней мере имею понятие, что интересно и полезно. Если «выпарить» все различные механизмы XML, то все будет сведено лишь к нескольким категориям.

Низкоуровневые API

API – это интерфейс прикладного программирования, а низкоуровневые API – это те интерфейсы, которые позволяют работать непосредственно с содержимым XML-документа. Другими словами, не происходит практически никакой предварительной обработки, и мы получаем «нетронутые» XML-данные, которыми и будем манипулировать. Это самый эффективный, а также и самый мощный способ работы с XML. Однако он требует обширных знаний об XML и обычно связан с колоссальной работой по превращению содержимого документа во что-то полезное.

В настоящее время два наиболее распространенных интерфейса – это SAX (Simple API for XML, простой API для XML) и DOM (Document Object Model, объектная модель документа). Кроме того, в последнее время все большую силу набирает JDOM (это не аббревиатура и не расширение DOM). Все три технологии находятся в той или иной стадии стандартизации (SAX – стандарт де-факто, DOM стандартизирован W3C, а JDOM – Sun) и имеют хорошие шансы на то, чтобы оказаться долгоживущими технологиями. Все три предоставляют доступ к XML-документу в различных формах и позволяют делать с документом практически все что угодно. Довольно большая часть моей книги посвящена этим интерфейсам, поскольку они представляют собой основу всего остального, что вы будете делать в XML. Отдельная глава отведена интерфейсу JAXP (Sun's Java API for XML Processing), который является тонким уровнем абстракции над SAX и DOM.

Высокоуровневые API

Высокоуровневые API – это следующий шаг вверх по лестнице. Они не предоставляют непосредственного доступа к документу, полагаясь в этом вопросе на API низкоуровневые. Кроме того, высокоуровневые интерфейсы представляют документ в иной форме – либо более дружественной к пользователю, либо каким-либо образом моделированной, либо в форме, отличающейся от основной структуры XML-документа. Хотя эти интерфейсы часто проще использовать и с их помощью проще разрабатывать приложения, преобразование данных в иной формат может сказаться на производительности. Кроме того, понадобится потратить некоторое время на изучение интерфейса, а некоторые низкоуровневые интерфейсы придется изучить в любом случае.

В этой книге основной пример высокоуровневого интерфейса – это связывание данных XML. Связывание данных обеспечивает преобразование документа XML в объект Java, который имеет структуру, отличную от древовидной. Если у вас были элементы под названием «person» и «firstName», вы получите объект с методами, подобными `getPerson()` и `setFirstName()`. Очевидно, что это простой способ быстро разобраться с XML; вряд ли для этого требуются какие-либо глубокие знания! Однако нельзя просто изменить структуру документа (например, сделав элемент «person» элементом «employee»), поэтому применение связывания данных ограничено определенным видом приложений. Все о связывании данных можно найти в главе 15 «Связывание данных».

Приложения на основе XML

Помимо интерфейсов прикладного программирования, специально созданных для работы с документом или его содержимым, также существуют приложения, построенные на XML. Эти приложения непосредственно или косвенно используют XML, но фокусируются на решении определенной задачи, например на отображении оформленных веб-данных или на связи между приложениями. В основе работы всех описанных приложений лежит XML, что и определяет базовые особенности их поведения. Некоторые приложения требуют обширных знаний XML, некоторые не требуют ничего, но все они оказываются к месту при обсуждении Java и XML. Из их числа я выделил наиболее популярные и полезные, о которых и буду говорить здесь.

Сначала рассмотрим системы публикации, предназначенные для форматирования XML в виде данных HTML или WML (Wireless Markup Language) либо в таких двоичных форматах, как PDF (Portable Document Format). Такие системы обычно используются для предоставления клиентам сложных, специализированных веб-приложений. Затем перейдем к XML-RPC, являющемуся XML-вариантом удаленных вызовов процедур. Это лишь некоторые из полного комплекта инструментов, обеспечивающих взаимодействие приложений. Основываясь на XML-RPC, я опишу SOAP, простой протокол доступа к объектам (Simple Object Access Protocol), а также его потенциал в сочетании с XML-RPC. Затем, изучая UDDI (Universal Discovery, Description and Integration) и WSDL (Web Services Descriptor Language) в главе, посвященной межкорпоративным (business-to-business) приложениям, вы увидите новых игроков на поле веб-служб. Обогатив свой арсенал этими средствами, вы будете подкованы не только в XML, но и в любой среде корпоративных приложений.

И наконец, в последней главе, всмотревшись в свой хрустальный шар, я попробую предсказать, какие тенденции будут набирать силу в ближайшие месяцы и годы, а также попытаюсь указать те из них, что заслуживают внимания. Это позволит вам вырваться вперед – туда, где и должен находиться любой хороший разработчик.

Основы

Теперь вы готовы к тому, чтобы узнать, как наилучшим образом применить Java и XML. Что вам нужно? Я коснусь этой темы, покажу основные направления, и дальше такими вопросами вы будете заниматься самостоятельно.

Операционная система и Java

В эти слова я вкладываю иронию; если вы считаете, что эту книгу можно прочесть, не имея установленной операционной системы и Java-системы, она может оказаться выше вашего понимания. Поэтому стоит сказать, чего я ожидаю. Я написал первую половину этой книги и примеры для этих глав на машине с Windows 2000, на которой были установлены и JDK 1.2 и JDK 1.3 (а также версия 1.3.1). Большая часть была скомпилирована мной под Cygwin (среда от Cygnus), поэтому обычно я работал в Unix-подобной среде. Вторая половина книги была написана на моем новом Macintosh G4 с операционной системой X. Эта система поставляется с JDK 1.3, и она просто чудо, если хотите знать.

В любом случае все примеры должны работать с Java 1.2 и выше – я не использовал никаких возможностей из JDK 1.3. Правда, я не старался написать код так, чтобы он компилировался в Java 1.1, поскольку мне кажется важным применение классов коллекций Java 2. Кроме того, тем, кто работает с XML и по-прежнему пользуется версией JDK 1.1, потребуется серьезно заняться обновлением JDK (я знаю, не у всех есть выбор). Если вы находитесь в жесткой зависимости от версии 1.1, то можете получить набор классов от Sun (*http://java.sun.com*), внести небольшие изменения, после чего все замечательно заработает.

Анализатор

Вам понадобится XML-анализатор. Одним из наиболее важных компонентов любого XML-ориентированного приложения является анализатор XML. Этот компонент выполняет важную задачу – он получает на входе необработанный XML-документ и придает ему смысл. Анализатор проверяет корректность исходного документа, а если документ ссылается на DTD или схему, может проверить и действительность документа. Результатом анализа XML-документа обычно является структура данных, с которой можно работать при помощи других XML-инструментов или Java API. Подробное обсуждение этих интерфейсов приводится в последующих главах. Пока же достаточно понимать, что анализатор – это один из ключевых элементов в применении XML-данных.

Выбор анализатора – непростая задача. Не существует жестких и простых правил, но обычно учитываются два основных критерия. Первый – быстродействие анализатора. Поскольку XML-документы используются все чаще, а их сложность постоянно растет, быстродействие анализатора становится важным фактором и может сильно повлиять на производительность всего приложения. Второй критерий – соответствие спецификации XML. Поскольку производительность зачастую имеет более высокий приоритет, чем отдельные несущественные возможности XML, некоторые анализаторы могут в чем-то пренебрегать требованиями спецификаций XML в целях увеличения производительности. В зависимости от требований к приложению следует выбирать разумный компромисс между этими двумя критериями. Кроме того, большинство анализаторов (но не все) позволяют проверять действительность документа. Другими словами, они могут проверить действительность документа в соответствии с указанным DTD или схемой. Убедитесь, что анализатор способен проверять действительность документов, если эта возможность необходима вашим приложениям.

Ниже приведен список наиболее распространенных анализаторов XML. В списке не отмечено, позволяет ли анализатор проверять действительность, т. к. в настоящее время ведутся работы по включению этой возможности в некоторые инструменты, в которых она отсутствует. Этот список не является рейтингом перечисленных инструментов, о каждом анализаторе можно найти массу информации на указанных веб-страницах:

- Apache Xerces: *http://xml.apache.org*
- IBM XML4J: *http://alphaworks.ibm.com/tech/xml4j*
- XP Джеймса Кларка: *http://www.jclark.com/xml/xp*
- Oracle XML Parser: *http://technet.oracle.com/tech/xml*
- Sun Microsystems Crimson: *http://xml.apache.org/crimson*
- Анализаторы Lark и Larval Тима Брэя: *http://www.textuality.com/Lark*
- Electric XML от Mind Electric: *http://www.themindelectric.com/products/xml/xml.html*
- Анализатор MSXML от Microsoft: *http://msdn.microsoft.com/xml/default.asp*

Внимание

Я включил в этот список анализатор MSXML от Microsoft из уважения к их попыткам исправить в последних версиях многочисленные проблемы с совместимостью. Однако их анализатор по-прежнему стремится «делать то, что ему вздумается», и поэтому нет гарантий, что он будет работать с примерами из этой книги. Обратитесь к нему при необходимости, но будьте готовы проделать дополнительную работу, если примете это решение.

Работая над этой книгой, я старался использовать свободно доступный анализатор Apache Xerces. Открытость кода для меня является большим плюсом, поэтому если вы еще не выбрали анализатор, рекомендую попробовать Xerces.

Интерфейсы прикладного программирования

После того как проблема выбора анализатора будет решена, вам понадобятся различные интерфейсы прикладного программирования, обсуждаемые ниже (высокоуровневые и низкоуровневые). Некоторые из них войдут в состав дистрибутива загружаемого вами анализатора, другие же придется загрузить самостоятельно. Будем считать, что либо вы ими уже располагаете, либо сможете получить их, для чего необходимо убедиться в наличии доступа к Интернету, прежде чем погружаться в чтение последующих глав.

Сначала перечислим низкоуровневые интерфейсы: SAX, DOM, JDOM и JAXP. Первые два (SAX и DOM) должны входить в состав любого загружаемого анализатора, т. к. они основаны на интерфейсе и будут реализованы в анализаторе. С большинством анализаторов вы также получите JAXP, хотя у вас может оказаться старая версия; можно надеяться, что к моменту выхода этой книги большинство анализаторов будут полностью поддерживать JAXP 1.1 (последняя вышедшая версия). В настоящее время JDOM загружается отдельно, и его можно получить с веб-сайта *http://www.jdom.org*.

Что касается высокоуровневых API, то некоторые альтернативы представлены в главе о связывании данных. Вкратце рассмотрены Castor и Zeus, доступные на сайтах *http://castor.exolab.org* и *http://zeus.enhydra.org* соответственно. Также я уделю некоторое время JAXB, доступному на сайте *http://java.sun.com/xml/jaxb*. Каждый пакет полностью функционален, загрузка дополнительных пакетов не требуется.

Приложения

Последними в списке стоят мириады особых технологий, о которых пойдет речь на страницах книги. К этим технологиям относятся, в том числе, наборы инструментов SOAP, валидаторы[1] WSDL, система веб-публикации Cocoon и т. д. Вместо того чтобы пытаться рассматривать каждую из них здесь, я коснусь конкретных приложений в соответствующих главах и расскажу о том, где можно получить эти пакеты, какие версии нужны, коснусь вопросов установки и всего прочего, что вам понадобится для установки и работы. Я могу избавить вас от всех неприятных подробностей и направить занудство только на тех, кто

[1] Не то же самое, что «validating parsers». На практике вполне может быть, что «validator» проверку действительности производит, а анализировать (parse) не умеет. – *Примеч. науч. ред.*

сделал этот выбор осознанно (шучу! я попытаюсь сделать чтение возможно более увлекательным). В любом случае продолжайте читать и изучайте все, что вам нужно знать.

В некоторых случаях я буду опираться на примеры из предыдущих глав. Например, если вы начнете читать главу 6 до того, как познакомитесь с главой 5, то, вероятно, немного запутаетесь. Если это случится, просто вернитесь обратно к нужной главе, и вы увидите, где берет начало смутивший вас код. Как уже говорилось, можно пропустить главу 2 об основах XML, но я рекомендую последовательно прочесть всю книгу, т. к. материал выстроен по нарастающей.

Что дальше?

Теперь вы, вероятно, готовы продолжить. Следующая глава представляет собой ускоренный курс по XML. Тем, кто не знает XML или нетвердо владеет его основами, эта глава поможет восполнить пробелы. Если же вы имеете опыт работы с XML, я рекомендую пропустить эту главу и перейти к программам из главы 3. В любом случае готовьтесь погрузиться в мир Java и XML; начинаются интересные вещи.

- *Основы*
- *Ограничения*
- *Преобразования*
- *И еще...*
- *Что дальше?*

2

Основы технологии

Введения остались позади, и можно перейти к делу. Но прежде чем коснуться непосредственно Java, необходимо рассказать о некоторых базовых структурах. Это касается фундаментального понимания концепций XML и механизма работы расширяемого языка разметки. Другими словами, нам нужен учебник по XML для начинающих. Что касается экспертов в XML, им стоит просмотреть эту главу, чтобы убедиться, что рассматриваемые темы им знакомы. Те же, кто совершенно не знает XML, могут здесь подготовиться к пониманию остального материала, причем довольно быстро.

По ходу чтения книги эту главу можно использовать в качестве глоссария. Далее я не буду тратить время на разъяснение понятий XML, чтобы сконцентрироваться на Java и перейти к некоторым более продвинутым концепциям. Поэтому если что-то полностью вас запутает, обратитесь за информацией к этой главе. Если же вам по-прежнему будет не все ясно, то очень рекомендую при чтении этой книги держать поблизости открытым справочник Элиотта Харольда (Elliotte Rusty Harold) и Скотта Минса (W. Scott Means) «XML in a Nutshell», O'Reilly.[1] Из него вы почерпнете всю необходимую информацию об основах XML, а я смогу сосредоточиться на Java.

Наконец, я щедр на примеры и собираюсь заполнить ими оставшиеся главы как можно плотнее. Считаю, что лучше предоставить избыточную информацию, чем лишь раздразнить аппетит читателя. Для начала, в соответствии со сказанным, приведу в этой главе несколько документов XML и связанных с ними, чтобы проиллюстрировать концепции начального курса. Можно, затратив время, набрать эти примеры

[1] Гарольд Э., Минс С. «XML. Справочник». – Пер. с англ. – СПб: Символ-Плюс, 2002.

Куда делись все главы?

Читатели первого издания книги «Java & XML» могут немного смутиться. В том издании одному только XML были посвящены целых три главы (пересчитайте!). Когда больше года назад я работал над первым изданием, передо мной стояла цель написать книгу частично об XML, частично о Java, но не рассматривающую целиком ни то ни другое. Тогда не было другого надежного ресурса, к которому можно было адресовать читателей за дополнительной помощью. Сегодня же такие книги, как «Learning XML» (Eric T. Ray, O'Reilly 2001)[1] и «XML. Справочник», решили эту проблему. Именно поэтому здесь достаточно привести лишь краткий курс по XML, а все подробности по «чистому» XML можно найти в этих прекрасных книгах. В результате мне удалось собрать несколько глав в этой одной, вымостив путь для новых глав по Java, которые, я уверен, и есть именно то, что вам надо! Так что подготовьтесь к некоторым радикальным отклонениям от первого издания; по крайней мере теперь вы знаете их причину.

в текстовом редакторе, а можно загрузить их с веб-сайта этой книги (*http://www.newInstance.com*), т. к. они будут использоваться в этой и всех последующих главах. Это позволит вам сэкономить время впоследствии.

Основы

Все начинается с рекомендации консорциума W3C XML 1.0, полный текст которой можно найти по адресу *http://www.w3.org/TR/REC-xml*. В примере 2.1 приведен простой XML-документ, соответствующий этой спецификации и представляющий собой часть оглавления этой книги в формате XML (я включил лишь часть только из-за того, что оно длинное!). Полную версию файла можно найти в примерах из книги по адресу *http://www.oreilly.com/catalog/javaxml2* и *http://www.newInstance.com*. Мы будем обращаться к нему для того, чтобы проиллюстрировать несколько важных концепций.

Пример 2.1. Документ context.xml

```
<?xml version="1.0"?>
<!DOCTYPE book SYSTEM "DTD/JavaXML.dtd">

<!-"Java и XML", Оглавление -->
<book xmlns="http://www.oreilly.com/javaxml2"
      xmlns:ora="http://www.oreilly.com"
>
```

[1] Эрик Т. Рэй «Изучаем XML». – Пер. с англ. – СПб: Символ-Плюс, 2001.

```
  <title ora:series="Java">Java и XML</title>

  <!-- Список глав -->
  <contents>
    <chapter title="Введение" number="1">
      <topic name="XML имеет значение" />
      <topic name="Что важно " />
      <topic name="Основы" />
      <topic name="Что дальше?" />
    </chapter>
    <chapter title="Основы технологии" number="2">
      <topic name="Основы" />
      <topic name="Ограничения" />
      <topic name="Преобразования" />
      <topic name="И далее..." />
      <topic name="Что дальше?" />
    </chapter>
    <chapter title="SAX" number="3">
      <topic name="Подготовка" />
      <topic name="SAX-совместимые анализаторы" />
      <topic name="Обработчики содержания" />
      <topic name="Советы разработчикам" />
      <topic name="Что дальше?" />
    </chapter>
    <chapter title="Расширенный SAX" number="4">
      <topic name="Свойства и возможности" />
      <topic name="Другие обработчики" />
      <topic name="Фильтры и классы Writers" />
      <topic name="И снова обработчики" />
      <topic name="Советы разработчикам" />
      <topic name="Что дальше?" />
    </chapter>
    <chapter title="DOM" number="5">
      <topic name="Объектная модель документа" />
      <topic name="Сериализация" />
      <topic name="Подверженность изменениям (mutability)" />
      <topic name="Советы разработчикам" />
      <topic name="Что дальше?" />
    </chapter>

    <!-- и т.д... -->

  </contents>

  <ora:copyright>&OReillyCopyright;</ora:copyright>
</book>
```

XML 1.0

Значительная часть спецификации описывает то, что, как правило, понятно интуитивно. Тем, кто когда-либо занимался созданием документов HTML или SGML, уже знакомо понятие элементов (таких как

contents и chapter в этом примере) и атрибутов (таких как title и name). В XML определению правил использования этих элементов и структурирования документа уделено не так много внимания. Гораздо больше времени тратится на определение таких сложных моментов, как интерпретация пробельных символов, чем на введение понятий, с которыми вы уже хоть как-то знакомы.

XML-документ можно разбить на две основные части: пролог, предоставляющий XML-анализатору и приложениям XML информацию о том, как обрабатывать документ, и содержание, представляющее собой собственно XML-данные. И хотя такое деление довольно условно, оно помогает различать инструкции для приложений из XML-документа от непосредственно XML-содержимого, и это отличие важно понимать. Пролог – это просто объявление XML в следующем формате:

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

Пролог, кроме того, может включать информацию о кодировке и определять, является ли документ автономным или же его интерпретация требует ссылки на другие документы:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
```

Оставшуюся часть пролога составляют элементы, подобные объявлению DOCTYPE:

```
<!DOCTYPE book SYSTEM "DTD/JavaXML.dtd">
```

В данном случае я ссылаюсь на файл *JavaXML.dtd*, находящийся на моей локальной системе в каталоге *DTD/*. Каждый раз при использовании относительного или абсолютного пути к файлу либо URL следует применять ключевое слово SYSTEM. Другая возможность – использовать ключевое слово PUBLIC, за которым следует публичный[1] идентификатор. Это означает, что W3C или иной консорциум определил стандартный DTD, а также общедоступное имя для DTD. В качестве примера рассмотрим инструкцию DTD для XHTML 1.0:

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

В данном случае за публичным идентификатором (странная короткая строчка, начинающаяся с «-//») следует системный идентификатор (URL). Если публичный идентификатор не может быть преобразован в системный средствами анализатора, вместо него используется явно заданный системный идентификатор.

---

[1] Свойства идентификаторов public/system (публичный/системный) из контекста объявлений XML не следует путать с модификаторами доступа public/private/protected (открытый/закрытый/защищенный) из ООП. – *Примеч. науч. ред.*

Также в начале файла можно увидеть инструкции обработки, которые обычно считаются частью пролога документа, а не его содержимого. Выглядят они примерно так:

```
<?xml-stylesheet href="XSL\JavaXML.html.xsl" type="text/xsl"?>
<?xml-stylesheet href="XSL\JavaXML.wml.xsl" type="text/xsl"
                 media="wap"?>
<?cocoon-process type="xslt"?>
```

Каждая из них имеет *цель* (*target*), определяемую первым словом, в данном примере – `xml-stylesheet` или `cocoon-process`, и *данные* (*data*) – все остальное. Чаще всего данные представлены в виде пар имя–значение, которые заметно улучшают читаемость. Правда, это лишь необязательное правило хорошего тона, поэтому на такой формат не следует рассчитывать.

Вся остальная часть XML-документа представляет его содержимое; другими словами – это элементы, атрибуты и данные, помещаемые в документ.

Корневой элемент

Корневой элемент – это элемент высшего уровня в XML-документе. Он должен быть первым открывающим и последним закрывающим тегом в документе. Он обеспечивает точку отсчета, позволяющую XML-анализатору или XML-ориентированному приложению распознавать начало и конец XML-документа. В нашем примере корневым является элемент `book`:

```
<book xmlns="http://www.oreilly.com/javaxml2"
      xmlns:ora="http://www.oreilly.com"
>

    <!--Содержимое документа -->
</book>
```

Этот тег и парный ему закрывающий тег охватывают все остальные данные в пределах этого XML-документа. Стандарт XML гласит, что в документе может быть только один корневой элемент. Другими словами, корневой элемент должен заключать все прочие элементы документа. Помимо этого ограничения корневой элемент ничем не отличается от любого другого элемента XML. Это очень важно понимать, поскольку XML-документы могут ссылаться на другие XML-документы или включать их. В таких случаях корневой элемент включаемого документа становится вложенным элементом основного документа и должен быть обработан XML-анализатором обычным образом. Определение корневых элементов в качестве стандартных элементов XML, не имеющих особых свойств и не обладающих особым поведением, обеспечивает отсутствие проблем при включении документов.

Элементы

До сих пор я избегал определения собственно элемента. Давайте теперь подробно рассмотрим элементы, которые представляются произвольными именами и должны заключаться в угловые скобки. В приведенном ниже примере показано несколько различных видов элементов:

```
<!-- Стандартный открывающий тег элемента -->
<contents>

 <!-- Стандартный элемент с атрибутом -->
 <chapter title="Основы технологии" number="2">

 <!-- Элемент с текстовыми данными -->
 <title ora:series="Java">Java и XML</title>

 <!-- Пустой элемент -->
 <sectionBreak />

<!-- Стандартный закрывающий тег элемента -->
</contents>
```

Первое правило создания элемента: его имя должно начинаться с буквы или символа подчеркивания и может содержать любое количество букв, цифр, символов подчеркивания, дефисов или точек. Имена элементов не могут содержать пробелы:

```
<!-- Внедренные пробелы недопустимы -->
<my element name>
```

Кроме того, имена элементов XML чувствительны к регистру. Обычно применение тех же правил, которые приняты в Java для именования переменных, приводит к созданию разумных имен элементов XML. Использование элемента под названием `tcbo` для представления бизнес-объекта телекоммуникационной индустрии (Telecommunications Business Object) – не очень хорошая идея, поскольку смысл имени не очевиден, а чрезмерно многословное имя тега, вроде `beginningOfNewChapter`, лишь загромождает документ. Помните, что ваши XML-документы, вероятно, попадут в руки других разработчиков и авторов, так что документирование посредством создания разумных имен весьма важно.

Каждый открытый элемент, в свою очередь, должен быть закрыт. Из этого правила нет исключений, которые встречаются во многих других языках разметки, например в HTML. Закрывающий тег для элемента состоит из прямого слэша и имени элемента: `</content>`. Между открывающим и закрывающим тегами может присутствовать любое количество элементов или произвольный объем текстовых данных. Однако нельзя нарушать порядок следования вложенных тегов: элемент, открытый первым, должен быть закрыт последним. Если в XML-документе нарушается хотя бы одно синтаксическое правило XML, такой документ не является *корректным* (*well-formed*). Кор-

ректный документ – это документ, в котором соблюдаются все синтаксические правила XML, а все элементы и атрибуты расположены допустимым образом. Тем не менее, корректный документ не обязательно является *действительным* (*valid*), что означает соответствие документа ограничениям, устанавливаемым DTD или схемой. Между корректными и действительными документами существует значительная разница. Выполнение правил, о которых идет речь в этом разделе, гарантирует корректность документа, в то время как правила, обсуждаемые в разделе, посвященном ограничениям, определяют действительность документа.

В качестве примера неверно построенного документа рассмотрим следующий фрагмент кода XML:

```
<tag1>
 <tag2>
</tag1>
 </tag2>
```

Порядок вложенности тегов нарушен, поскольку за открывающим тегом `<tag2>` в пределах тега `<tag1>` не следует закрывающий тег `</tag2>`. Однако нет никакой гарантии, что документ станет действительным, если исправить эти синтаксические ошибки.

Хотя этот пример неверно построенного документа может показаться тривиальным, вспомните, что такая разметка была бы допустимой в HTML, и подобное часто встречается в больших таблицах HTML-документов. Другими словами, HTML и многие другие языки разметки не требуют XML-корректности.[1] Строгое соблюдение упорядоченности и правил вложенности в XML позволяет анализировать и обрабатывать данные гораздо быстрее, чем при использовании языков разметки, не имеющих подобных ограничений.

Последнее правило, которое мы рассмотрим, касается случая пустых элементов. Мы уже знаем, что теги XML всегда должны существовать в парах – открывающий тег и закрывающий тег вместе составляют полный XML-элемент. Существуют ситуации, когда элемент используется сам по себе, например в качестве признака того, что глава еще не дописана, или когда элемент имеет атрибуты, но не имеет текстовых данных, подобно ссылке на изображение в HTML.[2] Подобные элементы придется записывать следующим образом:

```
<chapterIncomplete></chapterIncomplete>
<img src="/images/xml.gif"></img>
```

---

[1] Документы XHTML, которые предполагается обрабатывать с помощью XML-анализатора, также должны быть XML-корректными. – *Примеч. науч. ред.*

[2] Речь идет о теге `<IMG>`. – *Примеч. науч. ред.*

Выглядит это несколько глупо, да еще и добавляет путаницу в XML-документ, который и без того может быть довольно объемным. Спецификация XML определяет средство обозначить оба тега – и открывающий и закрывающий – внутри одного элемента:

```
<chapterIncomplete />
<img src="/images/xml.gif" />
```

Бретт, а что это за пробел перед последним слэшем?

Что ж, я расскажу. На мою долю выпало сомнительное удовольствие работать с Java и XML с 1998 года, когда дела обстояли в лучшем случае не очень гладко. В то время (да и сейчас) некоторые веб-броузеры понимали только XHTML (корректный HTML) в специальном формате. Наиболее заметен тот факт, что такие теги, как `<br>`, которые никогда не закрываются в HTML, в XHTML должны быть закрыты, т. е. появляются теги, подобные `<br/>`. Некоторые из упомянутых броузеров игнорируют подобные теги; однако, как это ни странно, они успешно обрабатывают теги `<br  />` (обратите внимание на пробел перед закрывающим слэшем). Я привык делать XML-код не только корректным, но и понятным для таких броузеров. У меня никогда не было веской причины менять эти привычки, так что здесь вы видите их в действии.

Итак, мы видим элегантное решение проблемы замусоривания документов, позволяющее оставаться верным правилу, что каждый элемент в XML должен иметь закрывающий тег. Открывающий и закрывающий теги просто объединяются в один.

Атрибуты

Помимо текста, заключенного в теги элемента, элемент также может иметь атрибуты. Атрибуты вместе со своими значениями включаются в открывающий тег элемента (который по совместительству может быть и закрывающим). Например, в теге `chapter` название главы представляет собой часть атрибута:

```
<chapter title="Расширенный SAX" number="4">
  <topic name="Свойства и возможности" />
  <topic name="Другие обработчики" />
  <topic name="Фильтры и классы Writer" />
  <topic name="И снова обработчики" />
  <topic name="Советы разработчикам" />
  <topic name="Что дальше?" />
</chapter>
```

В данном случае `title` – это имя атрибута, значением которого является название главы, т. е. «Расширенный SAX». Имена атрибутов должны удовлетворять правилам для имен элементов XML, а значения атрибутов необходимо заключать в кавычки. Несмотря на то что допускается применение как одинарных, так и двойных кавычек, использование последних – широко распространенная практика, которая приводит к созданию XML-документов, соответствующих стилю программирования на Java. Кроме того, одинарные и двойные кавычки могут присутствовать в значениях атрибутов. Если заключить значение в двойные кавычки, одинарные кавычки могут выступать как часть значения, и наоборот, если заключить значение в одинарные кавычки, то как часть значения смогут использоваться двойные. Однако это нельзя считать хорошим стилем, поскольку анализаторы и процессоры XML часто преобразуют кавычки, ограничивающие значение атрибута, в двойные (или в одинарные), что может привести к неожиданным результатам.

Помимо вопроса о том, как использовать атрибуты, существует также вопрос о том, когда это уместно. Поскольку XML допускает множество вариантов форматирования данных, редко когда атрибут не может быть представлен элементом и редко когда элемент не может быть легко преобразован в атрибут. И хотя не существует спецификации или широко принятого стандарта, позволяющих определить, когда следует предпочесть ту или иную форму, есть хорошее практическое правило: элементы подходят для разметки сегментов данных, состоящих из нескольких значений, а атрибуты – для разметки сегментов из единственного значения. Если определенный логический сегмент данных состоит из нескольких значений либо имеет большую длину, то вероятнее всего он должен оформляться в виде элемента. Впоследствии эти данные можно будет рассматривать прежде всего как текстовые, которые легко поддаются поиску и обработке. Примеры данных такого рода: описание глав книги или URL-адреса, содержащиеся в соответствующих ссылках на сайте. Если же данные главным образом представимы в виде единственного значения, их лучше всего представить в виде атрибута. Вероятным кандидатом на оформление в виде атрибута является раздел главы; тогда как сам раздел может быть элементом и иметь собственное название, группа глав внутри раздела может быть легко представлена атрибутом `section` элемента `chapter`. Этот атрибут позволит легко группировать и индексировать главы, но он никогда не будет отображаться для пользователя. Еще один хороший пример данных, которые могут быть представлены в XML в виде атрибута, – это указание, зарезервирован ли некий товар покупателем. Такое указание позволит XML-приложению, которое используется для создания брошюры или рекламного листка, определить, какие из имеющихся на складе товаров следует включать в документ. Очевидно, что это будет либо значение «истина», либо «ложь», и единовременно атрибут может принимать только одно из них. Опять же, клиент никогда не

увидит этой информации, но данные будут использоваться при обработке XML-документа. Если подобный анализ все-таки не дал четкой картины, остановитесь на самом надежном варианте – примените для оформления элемент.

Может быть, вам уже пришли в голову альтернативные подходы к представлению рассмотренных примеров. Например, вместо того чтобы использовать атрибут `title`, возможно, имело бы смысл включить элементы `title` в элемент `chapter`. Не исключено, что пустой тег `<layaway />` был бы более подходящим способом отметить, что предмет мебельного гарнитура зарезервирован. В XML очень редко существует только один способ представить данные, и, как правило, есть несколько хороших способов решения одной и той же задачи. Чаще всего решение определяется приложением и перспективой применения данных. Вместо того чтобы рассказывать, как создавать XML-документы, что было бы сложно, я покажу, как применять XML, чтобы вы могли получить представление о том, как можно обрабатывать и использовать различные форматы данных. Это обеспечит вас знаниями, необходимыми для принятия собственных решений о форматировании XML-документов.

Константы и ссылки на сущности

Я не обсудил еще вопрос, касающийся маскировки символов, или ссылки на константы. Например, распространенным способом указания пути к каталогу установки является элемент `<path-to-Cocoon>`. Здесь пользователь должен заменить этот текст подходящим вариантом каталога установки. В данном примере в главе, рассказывающей о веб-приложениях, должны быть приведены некоторые подробности об установке и применении Apache Cocoon, и может потребоваться представить эти данные внутри элемента:

```
<topic>
 <heading>Установка Cocoon</heading>
 <content>
  Найдите файл Cocoon.properties в каталоге <path-to-Cocoon>/bin.
 </content>
</topic>
```

Неприятность заключается в том, что XML-анализаторы пытаются обработать эти данные как тег XML, а затем выдают сообщение об ошибке, поскольку закрывающий тег отсутствует. Это распространенная проблема, поскольку к такому исходу приводит любое использование угловых скобок. Справиться с этим позволяет применение *ссылок на сущности* (*entity references*). Ссылка на сущность – это специальный тип данных XML, используемый для адресации некоторой совокупности данных. Ссылка на сущность состоит из уникального имени, перед которым стоит символ &, а после – точка с запятой: `&[имя сущности];`. Когда анализатор XML встречает ссылку на сущность, ссылка заменяется определенным значением, и это значение не обрабатывается.

В XML определено пять сущностей, которые позволяют решить проблему, представленную в примере: < для знака «меньше», > для знака «больше», & для символа амперсанда (&), " для двойной кавычки, а также ' для одинарной кавычки или апострофа. Посредством этих специальных ссылок можно точно представить ссылку на каталог установки следующим образом:

```
<topic>
 <heading>Установка Cocoon</heading>
 <content>
  Найдите файл Cocoon.properties в каталоге &lt;path-to-Cocoon&gt;/bin.
 </content>
</topic>
```

Когда этот документ обрабатывается анализатором, данные интерпретируются как «<path-to-Cocoon>», и документ считается корректным.

Имейте в виду, что ссылки на сущности могут определяться пользователем. Это делает возможным применение некоего подобия сокращения разметки: в рассмотренном примере XML-кода есть ссылка на внешний, совместно используемый (shared) документами текст с информацией об авторских правах. Поскольку этот текст присутствует во многих книгах издательства O'Reilly, я не хочу включать его в XML-документ. Тем не менее, если текст с информацией о правообладании изменится, изменения должны быть отражены в XML-документе. Обратите внимание, что синтаксис, применяемый в этом случае, похож на синтаксис для ссылок на стандартные сущности XML:

```
<ora:copyright>&OReillyCopyright;</ora:copyright>
```

Мы еще не добрались до раздела о DTD, поэтому пока не знаем, каким образом объяснить XML-анализатору, на что указывает ссылка &OReillyCopyright;, но вам уже должно быть понятно, что применение ссылок на сущности не ограничивается лишь вставкой в текст сложных или необычных символов.

Неанализируемые данные

Последняя конструкция XML, которую мы рассмотрим, – это маркер секции CDATA. Секция CDATA используется в тех случаях, когда вызывающему приложению необходимо передать значительный объем данных, не подлежащих обработке XML-анализатором. Это может пригодиться, если необходимо заменить ссылками на сущности необычайно большое количество символов либо когда важно сохранить расстановку пробелов. Секция CDATA в XML-документе выглядит следующим образом:

```
<unparsed-data>
  <![CDATA[Диаграмма:
       <Step 1>Установите Cocoon в каталог "/usr/lib/cocoon"
```

```
        <Step 2>Найдите верный файл свойств.
        <Step 3>Загрузите Ant с сайта "http://jakarta.apache.org"
              -----> Для этого воспользуйтесь CVS <----
  ]]>
</unparsed-data>
```

В данном примере для информации из секции CDATA нет необходимости прибегать к ссылкам на сущности или другим механизмам, сообщающим анализатору об использовании зарезервированных символов. Вместо этого XML-анализатор передает их в неизменном виде той программе (или приложению), из которой был вызван.

Итак, мы рассмотрели основные составляющие XML-документов. И хотя каждая из них была описана лишь поверхностно, вы получили достаточно информации для того, чтобы опознать теги XML, когда вы их увидите, и понять, для чего они предназначены. При наличии такой книги, как «XML in a Nutshell» (O'Reilly)[1], вы вполне готовы к тому, чтобы взглянуть на некоторые из более «продвинутых» спецификаций XML.

Пространства имен

И хотя мы не будем очень подробно рассматривать здесь пространства имен XML, обратите внимание на использование пространства имен в корневом элементе в примере 2.1. *Пространство имен XML* – это способ связывания одного или более элементов из XML-документа с конкретным URI (Uniform Resource Identifier). На деле это означает, что элемент идентифицируется как своим именем, так и URI пространства имен. В приведенном примере XML-документа позже может понадобиться включить части других книг издательства O'Reilly. Поскольку каждая из этих книг также может иметь элементы Chapter, Heading или Topic, документ следует проектировать и создавать таким образом, чтобы избежать конфликтов пространства имен с другими документами. Спецификация пространств имен XML элегантно решает эту проблему. Поскольку наш XML-документ представляет конкретную книгу и никакой другой XML-документ ту же самую книгу представлять не должен, использование уникального URI, например *http://www.oreilly.com/javaxml2*, позволяет создать уникальное пространство имен. Спецификация пространств имен требует, чтобы с префиксом был связан уникальный URI, который позволит отличать элементы данного пространства имен от элементов других пространств имен. Рекомендуется использовать URL, как мы здесь и поступаем:

```
<book xmlns="http://www.oreilly.com/javaxml2"
      xmlns:ora="http://www.oreilly.com"
>
```

[1] Элиот Расти Гарольд, У. Скотт Минс. «XML. Справочник». – Пер. с англ. – СПб: Символ-Плюс, 2002.

На самом деле здесь определены два пространства имен. Первое считается пространством имен по умолчанию, поскольку для него не задается префикс. Любой элемент без префикса входит в это пространство имен. В результате все элементы из этого XML-документа, за исключением элемента `copyright`, предваряемого префиксом `ora`, принадлежат пространству имен по умолчанию. Второе пространство имен определяет префикс, что позволяет связать тег `<ora:copyright>` со вторым пространством имен.

И наконец, еще один интересный (и несколько сбивающий с толку) момент: спецификация схем XML, о которой подробнее рассказано в следующем разделе, требует указания схемы XML-документа таким образом, который очень похож на набор объявлений пространств имен (см. пример 2.2).

Пример 2.2. Обращение к схеме XML

```
<?xml version="1.0"?>
<addressBook xmlns:xsi="http://www.w3.org/1999/XMLSchema/instance"
             xmlns="http://www.oreilly.com/catalog/javaxml"
             xsi:schemaLocation="http://www.oreilly.com/catalog/javaxml
                                 mySchema.xsd"
>
  <person>
    <name>
      <firstName>Бретт</firstName>
      <lastName>Мак-Лахлин</lastName>
    </name>
    <email>brettmclaughlin@earthlink.net</email>
  </person>
  <person>
    <name>
      <firstName>Эдди</firstName>
      <lastName>Балуччи</lastName>
    </name>
    <email>eddieb@freeworld.net</email>
  </person>
</addressBook>
```

Здесь важно понять несколько моментов. Во-первых, тут определяется и связывается с URL пространство имен экземпляра схемы XML. Это пространство имен, сокращенно `xsi`, служит для указания в XML-документах информации о схеме в точности так, как это сделано в данном случае. Таким образом, в первой строке элементы экземпляра схемы XML становятся доступными в документе. Следующая строка определяет пространство имен для самого XML-документа. Поскольку в документе пространство имен явно не используется, как это было в предыдущих примерах для пространства имен, связанного с *http://www.oreilly.com/javaxml2*, то объявляется пространство имен по умолчанию. Это означает, что все элементы без явного указания простран-

ства имен и соответствующего префикса (все элементы в этом примере) будут принадлежать пространству имен по умолчанию.

Определив таким образом пространство имен как для документа, так и для экземпляра схемы XML, можно далее делать то, что мы хотим, т. е. связать с этим документом какую-либо схему. Для этого воспользуемся атрибутом `schemaLocation`, который принадлежит пространству имен экземпляра схемы XML. Я предваряю этот атрибут идентификатором его пространства имен (`xsi`), которое было только что определено. Аргументом этого атрибута на самом деле являются *два* URI: первый определяет пространство имен, связанное со схемой, а второй – URI схемы, на которую мы ссылаемся. В результате в данном примере первый URI представляет собой только что объявленное пространство имен по умолчанию, а второй – файл *mySchema.xsd* в локальной файловой системе. Как и любой другой XML-атрибут, эта пара целиком заключается в одну пару кавычек. Вот так запросто можно указать схему для XML-документа!

На самом деле это совсем не просто и на сегодняшний день является одним из наиболее сложных для восприятия моментов в использовании пространств имен и схем XML. Чуть позже мы подробнее рассмотрим используемый здесь механизм. А пока запомните, каким образом пространства имен допускают обращение к элементам из различных наборов, сохраняя информацию о принадлежности каждого из элементов к определенной группе.

Ограничения

Следующая задача – это работа с ограничениями XML. Даже если вы ничего не вынесете из этой главы, кроме логического обоснования необходимости ограничения данных XML, то я буду счастливым автором. Поскольку XML – это расширяемый язык, позволяющий представлять данные сотнями и даже тысячами способов, ограничения, налагаемые на документ, придают этим различным форматам смысл. Без ограничений документа невозможно (в большинстве случаев) определить смысл данных в документе. В этом разделе представлены два из существующих механизмов наложения ограничений на XML-данные: DTD (включены в спецификацию XML 1.0) и XML-схемы (недавно опубликованный стандарт от W3C). Выбирайте тот механизм, который больше вам подходит.

Определение типа документа (DTD)

XML-документ не слишком полезен без сопутствующего DTD (или схемы). Точно так же, как XML может эффективно описывать данные, так DTD, определяя структуру данных, обеспечивает возможность использования этих данных самыми разными способами во множестве