ORACLE

ОФИЦИАЛЬНОЕ АВТОРИЗОВАННОЕ ИЗДАНИЕ ORACLE PRESS™
ЭКСКЛЮЗИВНЫЕ ПРАВА ПРИНАДЛЕЖАТ ИЗДАТЕЛЬСТВУ OSBORNE

ORACLE 9*i* XML

Разработка приложений электронной коммерции
с использованием технологии XML

Бен Чанг
Марк Скардина
Стефан Киритцов

члены группы
разработки XML-продуктов
компании Oracle

OFFICIAL • AUTHORIZED
Oracle Press
ONIV FROM OSBORNE

ORACLE® *Oracle Press™*

Oracle9*i* XML Handbook

Ben Chang
Mark Scardina
Stefan Kiritzov

Osborne/**McGraw-Hill**

New York Chicago San Francisco
Lisbon London Madrid Mexico City Milan
New Delhi San Juan Seoul Singapore Sydney Toronto

Oracle9*i* XML

Бен Чанг
Марк Скардина
Стефан Киритцов

Издательство "Лори"

Oracle9*i* XML Handbook
Ben Chang, Mark Scardina, Stefan Kiritzov

Oracle9*i* XML
Бен Чанг, Марк Скардина, Стефан Киритцов

Переводчик А. Головко
Корректор И. Гришина
Верстка Т. Кирпичевой

Osborne/**McGraw-Hill**
2600 Tenth Street
Berkeley, California 94710
U.S.A.

ISBN 0-07-213495-X

© Издательство "Лори", 2003

Изд. № : OAI (03)
ЛР № : 070612 30.09.97 г.
ISBN 5-85582-194-3

Подписано в печать 21.04.2003 Формат 70 x 100/16
Бумага офсет № 1 Гарнитура Литературная Печать офсетная
Печ. л. 31 Тираж 3200 Заказ № 237
Цена договорная

Издательство "Лори"
123100, Москва, Шмитовский пр., д. 13/6, стр. 1 (пом. ТАРП ЦАО)
Телефон для оптовых покупателей: (095)256-02-83
Размещение рекламы: (095)259-01-62

WWW.LORY-PRESS.RU

Отпечатано в типографии ООО "Типография ИПО профсоюзов Профиздат"
109044, Москва, ул. Крутицкий вал, д. 18

Нашим семьям за их поддержку

Об авторах

Бен Чанг (Ben Chang) работает в Oracle Corporation 12 лет, сейчас занимает пост директора группы разработки CORE и XML в подразделении серверных технологий. Кроме работы над Oracle6 и Oracle9i он в течение длительного времени был менеджером по разработке Oracle 8.0 и выпустил пять релизов этой СУБД. В течение трех лет он возглавлял комитет по стандартам Oracle C Coding Standards Committee и провел множество презентаций языка XML на конференциях, проходивших на всех континентах. Ныне он является членом комитета рабочей группы по стандарту W3C DOM. Бен Чанг сотрудничал с IBM Corp., Pacific Bell, Bellcore и GE R&D, получил степень магистра по электротехнике (компьютерные системы) в Стэнфордском университете и степень бакалавра по электротехнике и компьютерным наукам в университете штата Калифорния в Беркли.

Марк **Скардина** (Mark Scardina) — это евангелист использования языка XML в серверных продуктах компании Oracle. Он занимает пост менеджера группы продуктов в группе разработки CORE и XML и отвечает за инфраструктурные компоненты XML, используемые в целом ряде продуктов Oracle, в том числе в инструментальных пакетах для XML-разработчиков. Марк Скардина представляет компанию Oracle в рабочей группе W3C XSL. Раньше он работал в компаниях Socket Communications и ACE Technologies. Марк Скардина имеет степень *бакалавра* по менеджменту информационных систем, которую получил в университете Сан-Франциско.

Стефан **Киритцов** (Stefan Kiritzov) — менеджер по разработке в группе CRM-технологий и архитектуры компании Oracle. Он имеет 19-летний опыт разработки системного ПО. До прихода в Oracle он работал в компаниях AT&T, NCR, SHL Systemhouse и ICT. Стефан Киритцов — обладатель степеней бакалавра и магистра математики, которые получил в университете им. Климента Охридского в Болгарии.

Содержание

Благодарности . . . . . *xv*
Введение . . . . . *xvi*

1 Oracle и XML **1**

Основные концепции и терминология XML 2
Пролог 4
Определение типа документа 5
Тело документа 6
API-интерфейсы объектной модели документов DOM 7
Простой API-интерфейс для XML 10
API-интерфейсы пространства имен 14
API-интерфейсы анализатора синтаксиса 17
API-интерфейсы преобразования расширяемого языка стилей 17
API-интерфейсы XML Schema 18
Почему XML? 18
Стратегия Oracle в области XML 20
Деятельность Oracle в XML-индустрии 21
Oracle в комитетах рабочей группы W3C 21
Инструментарий XML-разработчика производства Oracle 22
Технологическая сеть Oracle Technology Network и XML Link 25
Обзор продуктов Oracle, поддерживающих технологию XML 27
Продукты Oracle, предоставляющие XML API-интерфейсы 28
Продукты Oracle, использующие XML для обмена данными 29
Продукты Oracle, использующие технологию XML для конфигурирования . . 30
Продукты Oracle, использующие XML для управления контентом и публикации 31
Обзор использования XML-компонентов Oracle 32
Создание и публикация документов 32
Служба доставки персонализированной информации 32
Легко настраиваемые приложения доставки информации 32
XML-корзина в приложениях электронной коммерции 33
Обмен сообщениями между компаниями через Интернет 33
Интеграция приложений с помощью XML-сообщений 33
Пример и приложение 33

2 Технологии XML CORE компании Oracle 37

Анализатор XML Parser for Java V2 38
Поддержка SAX 39
Поддержка DOM 45
Поддержка XSLT 52
Поддержка XML Schema 59
Генератор Java-классов 63
Входной DTD 64
Обработка DTD для генерирования Java-классов 65
Создание из Java-классов допустимого XML-документа 66
XML-документ, созданный с помощью Java-приложения 67
Входные данные из XML Schema 68
Просмотр и преобразование XML-файлов с помощью Java-программ 71
Модуль DOMBuilder Bean 72
Модуль XSLTransformer Bean 75
Модуль XMLSourceView Bean 76
Модуль XMLTreeView Bean 79
Модуль XMLTransformPanel Bean 79
Модуль DBView Bean 81
Модуль DBAccess Bean 82
Анализатор синтаксиса XML Parser for PL/SQL 82
Примеры 83
Анализатор синтаксиса XML Parser и процессор XSLT Processor for C 86
Автономный анализатор синтаксиса с интегрированным XSLT-процессором . . 86
Библиотека анализатора синтаксиса/XSLT-процессора 88
API-интерфейс Document Object Model (DOM) 94
Simple API for XML (SAX) 94
Поддержка процессора XSLT 100
Поддержка XML Schema 101
Библиотека процессора XML Schema 102
XML Parser, XSLT Processor, XML Schema Processor for C++ 103
Генератор классов C++ 103

3 Разработка приложений для СУБД Oracle9i 107

Oracle9i – СУБД с поддержкой технологии XML 109
JServer и Java XML-компоненты Oracle 110
Основы JServer 111
Java XML-компоненты 113
Публикация и вызов Java XML-компонентов 115
Схема базы данных и XML-документы 118
Отображение XML-документов на схему базы данных 119
Отображение схемы базы данных на виртуальные XML-документы 122
Хранение и извлечение XML-данных 125

XSQL - XSLT/SQL Server страницы . . . 127
Архитектура XSQL . . . 129
Установка XSQL Servlet . . . 130
Динамические XML-документы из SQL-запросов . . . 131
Поддержка условных SQL-команд в XSQL . . . 133
Пример с продажей книг . . . 134
Проектирование схемы базы данных . . . 135
Проектирование Web-сайта с использованием XSQL . . . 136

4 Разработка программ для серверов приложений Oracle Application Server 143

Архитектура Oracle Application Server . . . 146
Программы, прослушивающие соединения по протоколу HTTP . . . 147
Компоненты OAS . . . 147
Картриджи приложений . . . 148
Архитектура Oracle Internet Application Server . . . 150
Коммуникационные службы iAS . . . 150
Презентационные службы iAS Presentation Services . . . 152
Службы бизнес-логики сервера iAS . . . 153
Службы управления данными сервера iAS . . . 155
Системные службы iAS . . . 156
Клиентские компоненты сервера iAS . . . 157
Комплект разработчика Oracle Database Client Developer's Kit . . . 157
Комплекты разработчика Oracle XML Developer's Kits . . . 157
Клиентский инструментарий Oracle LDAP Client Toolkit . . . 157
Программа онлайнового книжного магазина как OAS-сервлет . . . 157
Приложение BookstoreServlet . . . 160
Регистрация приложения BookstoreServlet и картриджа . . . 163
Запуск приложения BookstoreServlet . . . 163
Доступ к базе данных . . . 164
Использование службы транзакций . . . 168
Активизация компонентов сервера OAS . . . 169
Приложение Bookstore как iAS-сервлет . . . 169
Конфигурация Apache . . . 169
Конфигурация JServ . . . 169

5 Файловая система Oracle Internet File System (iFS) 171

Характеристики . . . 172
Хранилище таблиц . . . 173
Анализаторы синтаксиса . . . 173
Программы визуализации . . . 173
Замены . . . 174
Протоколы . . . 174

Преимущества . . . 174
Компоненты . . . 175
XML . . . 175
interMedia Text/Oracle Text . . . 177
Модель документа . . . 177
Свойства документов . . . 178
Свойства, описывающие версии документа . . . 179
Свойства, связанные со стандартностью документа . . . 179
Свойства, описывающие связи документа . . . 181
Пользовательские свойства . . . 182
Обработка документа . . . 182
Определение типов документов . . . 182
Пример определения типа . . . 183
Стандартные свойства типа . . . 184
Свойства настраиваемого типа . . . 186
Атрибуты свойств . . . 187
Расширения файлов . . . 191
Использование системы iFS . . . 193
Пример 1. Создание и сохранение приветствия миру "Hello World" . . . 193
Пример 2. Создание более качественного приветствия миру "Hello World" . . . 194
Пример 3. Работа с файлами . . . 195
Пример 4. Поиск файлов . . . 196
Использование iFS с XML-файлами . . . 196
Хранение синтаксически разобранных XML-файлов . . . 197
Сохранение не анализированных XML-файлов . . . 198
Визуализация XML-файлов . . . 199
Дополнительные важные замечания по поводу XML-файлов . . . 200

6 Поиск XML-документов с помощью Oracle Text 201

Oracle Text как средство текстового поиска нового поколения . . . 203
Архитектура индексирования Oracle Text . . . 204
Информационное хранилище . . . 205
Программа фильтрации . . . 205
Программа разбиения на секции . . . 206
Лексический анализатор . . . 208
Работа с Oracle Text . . . 209
Информационные хранилища . . . 21.1
Секции поля и зоны . . . 213
Секции останова . . . 217
Секции атрибутов . . . 218
Поиск XPATH внутри группы PATH_SECTION_GROUP . . . 219
Секция динамических добавлений . . . 220

7 Службы электронного бизнеса Oracle E-Business XML Services 221

Обзор служб XML Services . . . 222
Компоненты XML Services . . . 223
Интерфейс администратора . . . 223
SOAP Server . . . 223
Клиентские API-интерфейсы . . . 223
Репозиторий служб и событий . . . 223
Терминология . . . 223
XML-служба . . . 224
Web Service . . . 224
Группа служб . . . 224
Основная точка интеграции . . . 224
Точка вызова . . . 224
Запись вызова . . . 224
Событие . . . 225
Подписчик события . . . 225
XML-службы и протокол SOAP . . . 225
Что такое SOAP . . . 226
Как работает протокол SOAP . . . 226
Что делает SOAP-клиент? . . . 228
Что делает SOAP-сервер? . . . 228
Руководство по группам служб . . . 229
Руководство по службам . . . 230
Модель системы безопасности . . . 232
Подробности исполнения службы . . . 232
Руководство по вызову служб . . . 233
Пример вызова службы . . . 234
Что нужно знать о событиях . . . 236
Пример сигнальных событий . . . 236
Пример сигнального события с фильтрацией подписчиков . . . 238
Службы, являющиеся подписчиками событий . . . 240
Разворачивание новой службы . . . 240
Создание профиля аутентификации . . . 253
Создание записи вызова . . . 254
Запуск примера службы . . . 257
API-интерфейсы вызова и события . . . 261
Класс oracle.apps.jtf.services.invocation.Client . . . 261
Класс oracle.apps.jtf.services.invocation.Param . . . 263
Класс oracle.apps.jtf.services.invocation.ServiceResult . . . 268

8 Oracle и XML в действии 271

Oracle XML SQL Utility . . . 272
Извлечение данных в формате XML . . . 273
Сохранение данных в формате XML . . . 275
Выполнение обновлений с помощью XML SQL Utility . . . 276
Удаление документов с помощью XML SQL Utility . . . 278
Установка XML SQL Utility . . . 279
Расширение XML SQL Utility . . . 281
Oracle XSQL Servlet . . . 281
XSQL-страницы . . . 281
Установка XSQL Servlet . . . 283
Составление запросов к XSQL Servlet . . . 284
Преобразование результатов XSQL-запроса с помощью таблиц стилей . . . 286
Вставка XML-документов с помощью XSQL Servlet . . . 289
Обновление данных с помощью XSQL Servlet . . . 291
Web-сайт, усиленный поддержкой технологии XML . . . 293
Решение с поддержкой технологии XML . . . 294
Конструирование требований . . . 294
Архитектура . . . 295
Пример реализации . . . 295
Расширение этого примера . . . 298
Oracle Portal-to-Go . . . 299
Службы обмена XML-сообщениями для электронного бизнеса . . . 299
Решение с поддержкой технологии XML . . . 300
Требования к конструкции Web-сайта . . . 300
Архитектура . . . 301
Пример реализации . . . 301
Расширение этого примера . . . 308
Oracle Integration Server . . . 308

9 Разработка приложения с использованием XML-технологий Oracle 311

Web-сайт с ответами на часто встречающиеся вопросы (FAQ), поддерживающий технологию XML . . . 312
Требования к приложению . . . 313
Проектирование приложения . . . 314
Конструирование схемы базы данных . . . 314
Генерация XML Schema . . . 317
Генерация Java-классов . . . 317
Хранение XML-документов как данных типа XMLType . . . 318
Генерация XML с помощью SYS_XMLGEN и SYS_XMLAGG . . . 319
Извлечение данных из XMLType с помощью функций Extract() и ExistsNode() . . . 319

Использование XMLType для связанных ответов на часто задаваемые вопросы FAQ . . . 320
Создание Web-приложения . . . 323
Составление запросов на получение FAQ . . . 324
Поиск информации в FAQ . . . 332
Использование операторов HASPATH и INPATH для поиска в данных типа XMLType . . . 334
Использование функциональных индексов для повышения производительности поиска . . . 335
Прямая связь с содержимым БД с использованием URI-Refs . . . 335
Построение глоссария . . . 336
Расширение приложения . . . 339

10 XML-приложения, предлагаемые на сайте OTN 341

Доступ к XML-приложениям . . . 343
Что демонстрируют XML-приложения . . . 344
XML-приложения . . . 346
Приложение Hello World . . . 347
Приложение Employee Page . . . 350
Приложение Insurance Claim . . . 353
Приложение Invalid Classes . . . 357
Индекс XSQL Demo Index . . . 359
Сайт Do You XML? . . . 361
Приложение Employee/Department Object View . . . 364
Приложение Airport Code Validation . . . 365
Приложение Airport Code Display . . . 373
Приложение Ad Hoc Query Visualization . . . 373
Приложение XML Document Demo . . . 374
Приложение XML Insert Request Demo . . . 376
Установка и запуск XML-приложений . . . 377

11 Тенденции развития 383

Роль стандартизирующих организаций . . . 385
Роль W3C . . . 386
Роль организации OASIS . . . 393
Схемы внедрения технологии XML в различных отраслях бизнеса . . . 397
Основные игроки на арене XML-схем и DTD . . . 399
Пример попытки создания специализированных DTD и XML-схем . . . 400
Влияние технологии XML на Интернет . . . 401
Основные игроки в XML-бизнесе . . . 402

A Спецификации W3C XML, DOM, SAX и XSLT 405

Спецификация XML . . . 406
Что такое XML? . . . 406
Документы . . . 406
Определение типа документа . . . 407
Спецификация DOM . . . 411
Что такое DOM? . . . 411
DOM Level 2 и Level 3 . . . 412
Ядро DOM . . . 413
Спецификация SAX . . . 420
Что такое SAX? . . . 420
Интерфейсы и классы SAX . . . 421
Спецификация пространства имен XML . . . 423
Что такое пространство имен XML? . . . 423
Терминология пространства имен . . . 425
Атрибуты пространства имен . . . 425
Спецификация XPath . . . 426
Что такое XPath? . . . 426
Выражения языка XPath . . . 427
Функции . . . 428
Узлы XPath . . . 430
Спецификация XSLT . . . 432
Что такое XSLT? . . . 432
Шаблоны . . . 433
Инструкции языка XSLT . . . 434
Функции XSLT . . . 443

B Спецификация W3C XML Schema 445

Что такое XML Schema? . . . 446
Введение . . . 448

C Другие спецификации консорциума W3C 453

Что такое XML Query? . . . 454
Что такое XML Protocol? . . . 460

Глоссарий 463

Благодарности

Авторы выражают благодарность Филипу Гринспуну (Philip Greenspun), вдохновившему их на первое издание этой книги; менеджерам Oracle Роджеру Чоплину (Roger Choplin), Хуану Лоайза (Juan Loaiza), Чаку Розвату (Chuck Rozwat) и конечно же исполнительному директору Oracle Ларри Эллисону (Larry Ellison); редакторской группе Джереми Джадсона (Jeremy Judson) из Oracle Press, Дженни Малник (Jenny Malnick), Джессике Уилсон (Jessica Wilson) и Афине Хонор (Athena Honore), без которых эта книга никогда бы не вышла; Джереми Бартону (Jeremy Burton) за его неиссякаемый энтузиазм; Джин-ю Ванг (Jinyu Wang) и Бернду Ринтелманну (Bernd Rintelmann) за их язвительные комментарии; помогавшим нам членам группы разработки CORE и XML и другим сотрудникам Oracle; и, наконец, нашим семьям за их понимание и поддержку.

Введение

В марте 1999 г. Филип Гринспун в штаб-квартире Oracle Corporation прочитал доклад, который вдохновил нас на написание первой редакции этой книги. Он рассказал об Интернете, языке XML, о том, как создавать Web-сайты с поддержкой баз данных, а также о том, почему СУБД Oracle так широко распространены в индустрии, и почему он думает, что сотрудники групп разработки Oracle должны быть авторами максимального числа книг издательства Oracle Press. По последнему пункту он высказывался неоднократно, причем в свойственной ему шутливой и едкой манере.

Мы восприняли это как вызов и руководство к действию и набросали план книги о работе над технологией XML, которая велась в компании Oracle. Мы связались с Джереми Джадсоном из Oracle Press, подписали необходимые контракты, составили план работ, послали письмо по электронной почте Филипу Гринспуну с просьбой написать для этой книги предисловие, на что он любезно согласился, заявив с энтузиазмом "Запишите меня!", и приступили к написанию глав. Мы также попытались учесть совет Филипа относительно того, что книга должна быть посвящена реальным проблемам, а не технологиям и бюрократическим процедурам.

Быстро приближалась середина 2000 г. Мы изо всех сил пытались выполнять составленный ранее план написания книги, продолжая одновременно работать в Oracle, и, наконец, выдали первую редакцию Oracle XML Handbook. Такой опыт ни мы, ни наши семьи, ни редакторы Oracle Press никогда не забудем! Плодами этого труда были тысячи экземпляров книги, проданных в разных странах мира, перевод на несколько языков, провозглашение ее "Лучшей книгой" 2000 г. по результатам голосования читателей журнала XML-Journal, работающих в XML-индустрии, а их насчитывается в общей сложности около пяти тысяч человек. Кроме того, из средств, полученных от продажи книги, более $2000 было пожертвовано фонду ArsDigita Foundation и организации America's Promise.

Но тут подоспел и конец 2001 г. Работа над второй редакцией книги была менее беспокойной. В нее мы добавили информацию о новых явлениях в XML-индустрии и сведения, касающиеся СУБД Oracle9i. Надеемся, что приведенные в книге простые примеры окажутся полезными нашим читателям, и они кое-что узнают о новых функциях Oracle XML.

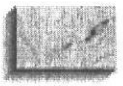
Да здравствует XML!

Внимание

Часть средств, полученных от продажи второй редакции книги, будет вновь направлена в фонд ArsD/gita Foundation и организации America's Promise, чтобы поддержать их в стремлении улучшить систему образования в США, Адреса их Web-сайтов: ***http://arsdigita.org и http://www.americapromise.org*** *соответственно.*

XML в работе

Возможно, вы читаете эту книгу потому, что планируете использовать XML в реальном приложении. Это хорошая идея, так как XML — технология, открывающая новые возможности. Чтобы помочь вам в построении приложений, использующих технологию XML, мы включили в книгу довольно много сценариев приложений и примеров конкретного кода, а также, где это было уместно, примеры XML- и XSL-файлов. Вы можете использовать этот код совершенно свободно, и мы надеемся, что это поможет эффективно использовать технологию XML и благодаря ей существенно улучшить разрабатываемые вами приложения.



Внимание

Комментарии к прочитанному или замечания об ошибках в книге или в материалах, размещенных на сайте издательства "Лори", сообщайте, пожалуйста, по электронной почте по адресу ***www.lory-press.ru.***

На кого мы рассчитываем

Эта книга была написана как руководство для пользователя в мире XML-компонентов Oracle. Это не полный справочник: если бы мы включили в книгу всю документацию по API-интерфейсам компонентов, ее объем увеличился бы вдвое. Со справочной документацией можно ознакомиться на сайте Oracle Technology Network. Книга написана для программистов, которым нужно использовать XML с СУБД Oracle для разработки приложений, и для читателей, интересующихся использованием технологии XML в продуктах Oracle. Хотя в большинстве приведенных примеров используется язык Java, со многими функциональными возможностями технологии XML можно поэкспериментировать с помощью интерфейсов командной строки и простого текстового редактора для XML, XSL и XQSL-файлов.

Как работать с книгой

Это книга вовсе не предназначена для чтения по порядку. В главах 1 и 2 представлено введение в технологии XML и Oracle XML, и если вы уже знакомы с ними, то спокойно можете пропустить их. Тем, кто интересуется построением приложений

СУБД Oracle9i, в которых будет использоваться технология XML, рекомендуем прочитать главы 3, 7, 8 и 9, а также главу 10, где подробно рассмотрены приложения OTN. Главы 4, 5 и 6 нужно прочитать, если вы собираетесь использовать сервер приложений Oracle Application Server, файловую систему Internet File System или поисковое средство Oracle Text. В главе 11 изложено наше видение будущего технологии XML. Наконец, в приложении А представлены основные спецификации консорциума W3W, в приложении В изложена спецификация W3W XML, а в приложении С — дополнительные W3W XML-спецификации. Ниже приведено краткое содержание каждой главы.

Глава 1. Oracle и XML Глава 1 переработана и дополнена по сравнению с первым изданием книги. В ней рассказывается о работе компании Oracle в индустрии XML, объясняются основные концепции и терминология XML, обосновывается право XML на существование и излагается стратегия Oracle в области XML. Здесь также представлен сайт Oracle Technology Network, сделан обзор продуктов Oracle с поддержкой XML и XML-компонентов и приведен пример приложения для продажи книг, в котором используется технология XML.

Глава 2. Технологии XML CORE компании Oracle Эта глава также переработана и дополнена. В ней представлены компоненты инструментария разработчика Oracle XML Developer's Kit (XDK), сделан обзор программ синтаксического анализа Java/C/C++/PLSQL XML и XSLT-процессоров, процессоров Java/C++ XML Schema Processors, генераторов классов Java/C++ и модулей Java Transviewer Beans. Сюда включен пример работающего кода для доступа к XML/XSL файлам и манипуляции ими.

Глава 3. Разработка приложений для СУБД Oracle9i В главе 3, переработанной и дополненной, рассматриваются встроенные в Oracle9i виртуальная Java-машина и Java XML-компоненты Oracle. Здесь также рассматривается вопрос о том, как данные XML можно сохранить в БД и извлекать из нее с помощью сервлета XQSL Servlet, входящего в инструментарий XDK. В конце обсуждается проектирование и реализация приложения по продаже книг с использованием XML.

Глава 4. Разработка программ для серверов приложений Oracle В переработанной и дополненной главе рассматриваются архитектуры сервера приложений Oracle Oracle Application Server и их связь с XML и XDK-компонентами. Обсуждается также создание приложения для книжного магазина в виде сервлета приложения Application Servlet.

Глава 5. Файловая система Oracle Internet File System (iFS) В главе 5, которая также была переработана, представлена файловая система Oracle нового поколения iFS. Рассмотрена ее архитектура, функционирование в качестве файловой системы XML, а также использование системой iFS технологии XML для распределения данных из файла в реляционную схему. В главу включены примеры Java-кода, иллюстрирующие часто встречающиеся операции с файлами.

Глава *6*. Поиск XML-документов с помощью Oracle Text Эта глава, переработанная и дополненная, посвящена Oracle Text — механизму поиска текста из Oracle9i. Здесь сделан обзор его архитектуры и возможностей по поиску информации, хранящейся в XML-документах. Пример с книжным магазином, приведенный ранее, расширен за счет примера кода для создания индексов и поиска информации, в котором также использовалась технология XML.

Глава 7. Службы электронного бизнеса Oracle E-Business XML Services Это новая глава, впервые представленная в этой редакции книги. В ней объясняется структура служб и событий (Services and Events) на базе XML, которая используется для разработки, разворачивания, управления и исполнения Web-служб, предназначенных для интеграции в B2B приложения. Подробно рассматривается технология обмена информацией SOAP и объясняется, как она используется в технологии приложений электронного бизнеса Oracle E-Business Application.

Глава 8. Oracle и XML в действии Глава 8, обновленная и дополненная, рассказывает о том, как XML-компоненты можно собрать в единое комплексное решение. Представлена конструкция приложения базы данных, использующего XML; Web-сайт, модернизированный с помощью этой технологии; и система обмена сообщениями между компанией и ее клиентами. Сюда также включены примеры XML и XSL-файлов и пример кода.

Глава 9. Разработка приложения с использованием XML-технологий Oracle В этой новой главе подробно, с листингами программ, объясняется, как можно с помощью продуктов производства Oracle построить мощное XML-приложение, например справочную систему FAQ. Это приложение иллюстрирует важные возможности процессора XSLT, генератора классов XML Class Generator, сервлета XQSL Servlet, утилиты XML SQL Utility, поискового средства Oracle Text, новых типов данных и операторов XML, представленных в Oracle9i.

Глава 10. XML-приложения, предлагаемые на сайте OTN Глава была переработана и дополнена по сравнению с первой редакцией. В ней представлен Web-сайт Oracle Technology Network (OTN), посвященный технологии XML, и имеющиеся на нем демонстрационные XML-приложения. Здесь рассказывается, как можно установить и запустить эти демонстрационные программы. Кроме того, подробно обсуждаются специальные функции XML с иллюстрациями возможностей каждой из них.

Глава 11. Тенденции развития В переработанной и дополненной главе представлен обзор работ по стандартизации XML, предпринимаемых консорциумами W3W и OASIS, а также будущие направления развития XML, которые должны стать следствием деятельности упомянутых организаций. Рассматривается использование XML и связанных с нею технологий в ряде крупных компаний, работающих в индустрии XML.

Приложение Л. W3W XML, DOM, SAX и XSLT-спецификации
Приложение А переработано и дополнено. В нем представлены основные спецификации XML и обсуждаются два стандарта консорциума W3W для интерфейсов XML-документа DOM и SAX. Изложены связанные с XML-технологией стандарты пространств присваиваемых имен, XPath и XSL Transformation.

Приложение В. Спецификации логической структуры W3W XML Schema
Это новое приложение. В нем представлен общий обзор новых спецификаций W3W XML Schema, а их использование проиллюстрировано на ряде примеров.

Приложение С. Остальные W3W-спецификации В приложении С представлен обзор новых спецификаций W3W XML Query, XML Protocol и SOAP с примерами их использования.

Глоссарий Глоссарий содержит список используемых в книге общеупотребительных терминов XML и их определения.

Дополнительные материалы

На сайте издательства "Лори" находятся производственные версии компонентов инструментария XDK for Java, C, C++ и PL/SQL, в том числе программа синтаксического анализа XML, XSL-процессор, модули TransViewer Java Beans и XQSL Servlet, а также много демонстрационных программ. Обновленные версии инструментария Oracle XDK можно бесплатно загрузить с сайта OTN.

Исходный код, использованный в этой книге, можно загрузить с Web-сайта Oracle Press по адресу http://www.osborne.com/oracle. Самые новые XML-компоненты Oracle9i находятся на Web-сайте Oracle Technology Network по адресу http://technet.oracle.com/tech/xml.

Глава 1

Oracle и XML

xtensible Markup Language (*XML*)— это расширяемый язык разметки, т. е. язык, как это определено в спецификации XML 1.0 консорциума *World Wide Web Consortium (W3C),* позволяющий пользователям определять свои собственные языки разметки для описания и инкапсуляции данных в XML-файлы. Эти файлы можно затем выводить на экран браузера, например Netscape Navigator и Microsoft Internet Explorer, обмениваться ими через Интернет между приложениями и компаниями, сохранять в базах данных и извлекать оттуда. Мощь языка XML, являющегося частью открытого стандарта,— следствие его простоты и того, что он позволяет включать в создаваемый XML-документ определяемые пользователем теги разметки, при условии, что они соответствуют семантическими правилам этого языка.

XML происходит от языка *Standard Generalized Markup Language (SGML),* одобренного *Международной организацией по стандартизации* (International Standards Organization, *ISO)* в 1986 г., на базе которого построен язык *Hypertext Markup Language (HTML),* созданный в 1990 г. Хотя SGML до сих пор является широко используемым стандартом в мире документов, а на базе HTML созданы миллионы Web-страниц в системе World Wide Web, язык XML быстро набирает популярность благодаря своим преимуществам в реализации обмена данными, в их хранении и описании через существующие языки разметки. Консорциум W3C опубликовал версию 1.0 спецификаций XML в феврале 1998 г., и с тех пор XML широко применяется в качестве языка и средства обмена данными в электронной коммерции.

Основные концепции и терминология XML

При описании языка XML, как это бывает с любым стандартом, нужно сначала объяснить целый ряд концепций и технических терминов. Так как XML был разработан для передачи данных, хорошим примером его использования будет запись данных о каталоге книг из стандартной базы данных. Сложный SQL-запрос обычно возвращает данные в следующем формате:

```
P History of Interviews, Juan, Smith, 99999-99999, Oracle Press, 2000.
```

Однако, если в качестве выходной формы используется язык XML, эта запись будет выглядеть так:

```
<book>
   <title>History of Interviews</title>
   <author>
      <firstname>Juan</firstname>
      <lastname>Smith</lastname>
   </author>
   <ISBN>99999-99999</ISBN>
   <publisher>Oracle Press</publisher>
   <publishyear>2000</publishyear>
   <price type="US">1.00</price>
</book>
```

Некоторые элементы в этом примере будут позднее подробно разобраны. Для начала отметим, что этот файл обладает симметрией, т. е. каждый фрагмент данных имеет свое окончание следующего вида <context> ... </context>. Угловые скобки и текст внутри них называется *тегами,* а каждый набор тегов и заключенные в нем данные называется *элементом.* Здесь можно провести аналогию со столбцом в таблице базы данных, в которой текст тега является заголовком столбца, а текст между тегами — это данные из строки этого столбца. В приведенном примере слово "title" может быть названием столбца, а "History of Interviews" — это данные, содержащимися в строке.

Отметим также, что некоторые теги содержат вместо данных другие теги. Это очень существенная функция языка XML, которая позволяет лучше определять соотношения между данными. Вернемся к метафоре базы данных. Здесь тег <author> можно смоделировать как таблицу, содержащую столбцы с названиями <firstname> и <Iastname>. В терминологии XML эти теги столбцов являются *дочерними* по отношению к *корневому* тегу Author.

Теперь рассмотрим тег <price>. В нем имеется текст вида "имя=значение". Пары "имя-значение" называются *атрибутами,* и в открывающий тег могут быть включены одна или более таких пар. Атрибуты никогда не пишутся в закрывающих тегах (например, </tag name="foo">).

А теперь заключительные слова по поводу терминологии: этот пример применения языка XML заключен между тегами <book> и </book>. Эти теги определяют *корень* документа, и такой корень может быть только один. XML-документы, соответствующие этим правилам, имеют только один корневой тег, а все открытые теги должны быть *надлежащим образом* закрыты.

В любом открытом Интернет-стандарте основные концепции и терминология должны быть формализованными и однозначными. Как говорится в спецификации W3C XML 1.0: "XML-документы состоят из блоков памяти, называемых *"информационными объектами"*, которые содержат синтаксически анализируемые и не

анализируемые данные. *Анализируемые данные* состоят из символов, некоторые из них образуют символьные данные, а некоторые — разметку. В *разметке* кодируется описание хранимой схемы и логической структуры документа". XML-документы имеют и физическую, и логическую структуру. *Физическая структура* XML-документа представляет собой XML-файл и другие файлы, которые этот файл может импортировать, тогда как *логическая структура* XML-документа представляет собой пролог и тело документа.

В примере book тело в XML-документе есть, но в нем нет важной информации, которая помогает идентифицировать его природу. Эта информация находится в прологе, определение которого дается в следующем параграфе.

Пролог

Пролог состоит из XML-описания, т. е. номера версии; необязательной кодировки (в данном случае UTF-8); других атрибутов (пары "имя=значение") и необязательного *определения типа документа* (document type definition, *DTD),* которое может быть внутренним, содержащимся в XML документе, и внешним DTD, находящимся в отдельном файле. Например:

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<!DOCTYPE book SYSTEM "book.dtd">
```

Отметим, что строка, имеющая вид <? ..., ?> является примером XML-команды *обработки информации* (processing instruction, *PI),* которая будет обсуждаться в этой главе ниже. В данном примере xml — это имя XML PI. Представленная здесь кодировка символов UTF-8 является сокращенным вариантом Unicode. Наконец, атрибут standalone определяет, нужен ли процессор для включения или импорта других внешних файлов.

Вторая строка пролога называется *DOCTYPE.* Именно здесь находится описание модели данных этого XML-документа. Почему это так важно? Вспомните, что XML-файл имеет и физическое, и логическое представление. В некоторых приложениях для обработки XML-документа даже не нужно знать, пропущена ли в нем какая-то информация, но в большинстве приложений производится проверка полученных XML-документов на действительность. Чтобы провести ее, приложение должно знать, какие в документе должны быть элементы, какие из этих элементов имеют дочерние элементы, у каких из них могут быть атрибуты и т. д. В XML-терминах эта модель данных называется *определением типа документов* (document type definition, *DTD).* DTD может содержаться внутри самого XML-файла или в этом файле должна быть ссылка на DTD, чтобы процессор мог найти его местоположение.

Определение типа документа

Определения типов документов (Document Type Definition, DTD) попадают *в* XML-файлы в виде элементов языка SGML, так как в синтаксисе XML их нет. Недавно рабочая группа XML Schema W3C Working Group сформулировала новый тип определения модели с использованием XML-синтаксиса и расширила функциональные возможности DTD в деле поддержки разных типов данных. В предыдущем примере DTD содержится во внешнем файле. Ссылка на него дана с помощью *унифицированного указателя ресурса* (uniform resource locator, *URL)* в виде http://www.foobar.com/book.dtd. Кроме того, описание DTD может находиться внутри XML-файла. Это может выглядеть следующим образом:

```
<!-- DTD bookcatalog may have number of book entries -->
<!DOCTYPE bookcatalog [
<!ELEMENT bookcatalog (book)*>
<!-- Each book element has a title, 1 or more authors, etc. -->
<!ELEMENT book (title, author+, ISBN, publisher, publishyear, price)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT author (firstname, lastname)
<!ELEMENT firstname (#PCDATA)>
<!ELEMENT lastname (#PCDATA)>
<!ELEMENT ISBN (#PCDATA)>
<!ELEMENT publisher (#PCDATA)>
<!ELEMENT publishyear (#PCDATA)>
<! ELEMENT price (#PCDATA)>
<!ATTLIST price type (US|CAN|UK I EURO) #REQUIRED>
]>
```

За описанием **DOCTYPE** в определении типа DTD следует описание корневого элемента **<!ELEMENT>** каталога bookcatalog. Элемент состоит из начального тега, например <foo>, соответствующего конечного тега, например </foo>, и текста между ними. Отметим, что в любом XML-документе может быть только один корневой элемент. Корневой элемент отмечает начало документа. Он считается родительским по отношению ко всем другим элементам, расположенным между его начальным и конечным тегами. Для XML-документов, которые являются действительными (valid), т. е. удовлетворяют данному определению типа документа DTD, корневой элемент bookcatalog должен стоять на первом месте, с него начинается тело XML-документа.

Далее идет *описание элемента,* где определяются дочерние элементы, которые должны располагаться внутри корневого элемента bookcatalog, также в описании элемента определяется модель содержимого для корневого элемента. Все дочерние элементы корневого элемента bookcatalog определяются в описании корневого элемента, а элемент author имеет суффикс +. Это пример *расширенного формата*

Бэкуса-Наура (Extended Bacus-Naur Format, *EBNF),* который может использоваться для описания модели содержимого. Допускаются следующие суффиксы:

- ? Если число элементов равно 0 или 1
- * Если число элементов больше или равно О
- + Если число элементов больше или равно 1

Отметим использование параметра **#PCDATA** для описания того факта, что текст данного элемента не должен быть текстом разметки. Кроме того, в данном фрагменте подробно описываются атрибуты цены товара. Разница между параметрами **CDATA и PCDATA** состоит в том, что разделы символьных данных **CDATA** программа синтаксического анализа пропускает и не проверяет на формальную правильность. Следовательно, эти разделы могут рассматриваться как непроверенные символьные данные "non-Parsed Character DATA".

Таким образом, программа синтаксического анализа XML, проверяя XML-Документ на соответствие правилам, определенным в DTD, пытается определить, соответствует ли документ заданному определению типа документа DTD, в котором существуют такие же структурные соотношения и последовательности.

Тело документа

Пролог располагается перед корневым элементом, который содержит остальную часть XML-документа. Оставшаяся часть XML-документа состоит из элементов, инструкций по обработке данных, содержимого, атрибутов, комментариев, ссылок на объекты и т. д. Как уже указывалось ранее, элементы должны иметь открывающие теги и соответствующие им закрывающие теги, расположенные в совершенно определенном порядке. В противном случае XML-документ будет сформирован *неправильно* и программа синтаксического анализа может подать сигнал об ошибках. Элементы могут также иметь атрибуты, или, как их еще называют, пары "имя-значение", например **<author firstname="Juan" Lastname="Smith">**. Существуют также встроенные атрибуты, определяемые спецификацией XML 1.0. Например, **xml:space= "preserve"** указывает, что данные между элементами представляются с сохранением пробелов.

Ссылки на объекты аналогичны макрокомандам. Объекты определяются один раз, а ссылки на них (например, ***&nameofentity***) можно использовать вместо их полных описаний. Например, можно объявить **<!ENTITY Copyright "Copyright 2000 by Smith, Jones, and Doe — All rights reserved">**, после чего использовать указатель **&Copyright** как сокращение на всем протяжении XML-документа. Программа анализа синтаксиса XML должна распознавать объекты, описанные в DTD, даже если проверка на правильность отключена. Повторим еще раз, что встроенные объекты подчиняются спецификациям XML 1.0, поэтому значок "&" будет записываться как **&атр;** апостроф — **'** "меньше, чем" — &lt и т. д. Комментарии заключаются в конструкцию **<!-- -->**.

Дополнительные базовые термины, связанные *с API-интерфейсами программы* анализа синтаксиса XML, можно распределить по следующим категориям:

- Объектная модель документа (Document Object Model, DOM)
- Простой API-интерфейс для XML (SAX)
- Пространства имен
- Программа анализа синтаксиса
- Расширенный язык преобразования таблиц стилей (Extensible Stylesheet Language Transformation, XSLT)
- XML Schema

Большая часть этих API-интерфейсов описывается в спецификациях W3C, поэтому разработчики приложений могут использовать стандартные интерфейсы программирования. Если некоторые функциональные возможности приложения и не определены упомянутыми спецификациями W3C, то поставщики ПО могут внести в эти спецификации свои собственные дополнения и усовершенствования.

API-интерфейсы объектной модели документов DOM

XML-документ всегда структурирован, так как имеющиеся в нем открывающие теги имеют соответствующие закрывающие теги, расположенные определенным образом. Благодаря такой структуре XML-документ можно рассматривать как три документа, в точках пересечений которых находятся теги; информации, расположенной между этими тегами; а также сопутствующих данных. Когда программа анализа синтаксиса XML проверяет XML-документ, в памяти формируется его древовидное представление, которое называется *объектной моделью документа* Document Object Model *(DOM).*

Консорциум W3C создал целый набор API-интерфейсов DOM для доступа и перемещения по этому дереву. В числе компонентов этого дерева будут корневой элемент документа, дочерние узлы, элементы, атрибуты и текстовые узлы, которые представляют текстовое содержимое элемента или атрибута. В состав остальных компонентов входят разделы символьных данных (CDATA), предназначенные для пометки блоков текста, которые в противном случае будут рассматриваться как разметка, комментарии, ссылки на объекты, команды по обработке и т. д. Программы анализа синтаксиса XML, предоставляемые всеми API-интерфейсами DOM, должны соответствовать рекомендациям W3C DOM.

Следующий пример Java-кода демонстрирует, как легко и просто использовать программу анализа синтаксиса и API-интерфейсы DOM. В этом фрагменте анализируется XML-файл приложения и распечатываются узлы элементов и значения

атрибутов документа. Здесь также демонстрируется использование установок опций программы-анализатора.

```
import java.io.*;
import java.net.*;
import org.w3c.dom.*;
import org.w3c.dom.Node;

import oracle.xml.parser.v2.*;

public class DOMSample
{  static public void main(String[] argv)
   {  try
      {  if (argv.length != 1)
         {  // Must pass in the name of the XML file.
            System.err.println("Usage: java DOMSample filename");
            System.exit(1);
         }

         // Get an instance of the parser
         DOMParser parser = new DOMParser();

         // Generate a URL from the filename.
         URL url = createURL(argv[0]);

         // Set various parser options: validation on,
         // warnings shown, error stream set to stderr.
         parser.setErrorStream(System.err);
         parser.setValidationMode(true);
         parser.showWarnings(true);

         // Parse the document.
         parser.parse(url);

         // Obtain the document.
         XMLDocument doc = parser.getDocument();

         // Print document elements
         System.out.print("The elements are: ");
         printElements(doc);

         // Print document element attributes
         System.out.println("The attributes of each element are: ");
         printElementAttributes(doc);
      }
```

```
    catch (Exception e)
    { System.out.println(e.toString()); }
}

static void printElements (Document doc)
{ NodeList nl = doc.getElementsByTagName ("*");
   Node n;

   for (int i=0; i<nl.getLength ( ) ; i++)
   { n = nl.item(i) ;
      System.out.print (n.getNodeName() + " ");
   }
   System.out.println();
}

static void printElementAttributes (Document doc)
{ NodeList nl = doc.getElementsByTagName ("*");
   Element e;
   Node n;
   NamedNodeMap nnm;

   String attrname;
   String attrval;
   int i, len; .

   len=nl.getLength();

   for (int j=0; j < len; j++)
   { e = (Element)nl.item(j) ;
      System.out.println(e.getTagName() + ":");
      nnm = e.getAttributes();

      if (nnm != null)
      { for (i=0; i<nnm.getLength(); i++)
         { n = nnm.item(i) ;
            attrname=n.getNodeName();
            attrval=n.getNodeValue();
            System.out.print (" " + attrname + " = " + attrval);
         }
      }
      System.out.println();
   }
}
```

```
    static URL createURL(String fileName)
    {  URL url = null;
       try
       {  url = new URL(fileName);  }
       catch (MalformedURLException ex)
       {  try
          {  File f = new File(filename);
             url = f.toURL();
          }
          catch (MalformedURLException e)
          {  System.out.println("Cannot create url for: " + fileName);
             System.exit(0);
          }
       }
       return url;
    }
}
```

Простой API-интерфейс для XML

Простой API-интерфейс для XML (Simple API for XML, *SAX)* — стандартный интерфейс для анализа синтаксиса XML на базе событий. Это означает, что упоминание об определенных событиях и данные, встречающиеся во время анализа XML-документов, могут быть представлены в отчете функциями обратного вызова из этого приложения. И в дальнейшем при уведомлении о таких событиях прикладная программа должна оперировать этими функциями. Например, прикладная программа может иметь структуры данных, которые используют программы *обратного вызова, выполняющие обработку данных на основе событий.* Типы информации и уведомления, направляемые этими функциями обратного вызова, похожи на начало и конец элементов XML-документа и на данные, относящиеся к содержимому элементов (символьные данные CDATA, команды по обработке и/или подэлементы).

Интерфейс SAX, изначально разработанный Дэвидом Меггинсоном (David Megginson), в конце концов стал стандартом W3C XML. Преимущество использования синтаксического анализа SAX по сравнению с использованием технологии DOM состоит прежде всегр в том, что не нужно строить представление документа в виде дерева грамматического разбора в памяти. Это экономит память и повышает скорость выполнения некоторых операций, например процедуры поиска информации. Однако изменение, обновление и выполнение других структурных операций может оказаться более эффективным при использовании синтаксического анализатора DOM Parser.

Следующий пример кода демонстрирует использование этого анализатора и API-интерфейса SAX. XML-файл, выбранный для данного приложения, анализируется синтаксически, после чего печатает некоторую информацию о содержимом файла. Здесь также представлен пример кода различных полезных интерфейсов.

```
import org.xml.sax.*;
import java.io.*;
import java.net.*;
import oracle.xml.parser.v2.*;

public class SAXSample extends HandlerBase
{  // Store the locator
   Locator locator;

   static public void main(String[] argv)
   {  try
      {  if (argv.length != 1)
         {  // Must pass in the name of the XML file.
            System.err.println("Usage: SAXSample filename");
            System.exit(1);
         }
         // Create a new handler for the parser
         SAXSample sample = new SAXSample();

         // Get an instance of the parser
         Parser parser = new SAXParser();

         // Set Handlers in the parser
         parser.setDocumentHandler(sample);
         parser.setEntityResolver(sample);
         parser.setDTDHandler(sample);
         parser.setErrorHandler(sample);

         // Convert file to URL and parse
         try
         {  parser.parse(fileToURL(new File(argv[0])).toString());  }
         catch (SAXParseException e)
         {  System.out.println(e.getMessage());  }
         catch (SAXException e)
         {  System.out.println(e.getMessage());  }
      }
      catch (Exception e)
      {  System.out.println(e.toString());  }
   }
```

```
    static URL fileToURL(File file)
    {  String path = file.getAbsolutePath();
       String fSep = System.getProperty("file.separator");
       if (fSep != null && fSep.length() == 1)
          path = path.replace(fSep.charAt(0), '/');
       if (path.length()>0 && path.charAt(0) != '/')
          path = '/' + path;
       try
       {  return new URL("file", null, path);  }
       catch (java.net.MalformedURLException e)
       {  throw new Error("unexpected MalformedURLException");  }
    }

////////////////////////////////////////////////////////////////////
// Sample implementation of DocumentHandler interface.
////////////////////////////////////////////////////////////////////

    public void setDocumentLocator (Locator locator)
    {System.out.println("SetDocumentLocator: ");
       this.locator = locator;
    }

    public void startDocument()
    { System.out.println("StartDocument");  }

    public void endDocument() throws SAXException
    { System.out.println("EndDocument");  }

    public void startElement (String name, AttributeList atts) throws SAXException
    { System.out.println("StartElement: "+name);
       for (int i=0;i<atts.getLength();i++)
       {  String aname = atts.getName(i);
          String type = atts.getType(i);
          String value = atts.getValue(i);
          System.out.println(" "+aname+"("+type+")"+"="+value);
       }
    }

    public void endElement (String name) throws SAXException
    { System.out.println("EndElement: "+name);  }

    public void characters (char[] cbuf, int start, int len)
    {System.out.print("Characters: ");
       System.out.println(new String(cbuf,start,len));
    }
```

```
    public void ignorableWhitespace(char[] cbuf, int start, int len)
    { System.out.println("IgnorableWhiteSpace"); }

    public void processingInstruction (String target, String data)
                throws SAXException
    { System.out.println("ProcessingInstruction: "+target+" "+data); }

  ///////////I/I/////////////////////I/////////////////////////////
  // Sample implementation of the EntityResolver interface.
  ////////////////////////////////////////////////////I/I//I///III////

    public InputSource resolveEntity (String publicId, String systemId)
                throws SAXException
    { System. out.println("ResolveEntity: "+publicId+" "+systemId);
      System.out.println("Locator: "+locator.getPublicId()+""+
                         locator.getSystemId()+
                         " "+locator.getLineNumber()+" *
                         +locator.getColumnNumber());
      return null;
    }

  ////////////Illllllllllllll/////////////////////////////////////////
  // Sample implementation of the DTDHandler interface.
  ////////////////////////////////////II/////////////////////////////

    public void notationDecl (String name, String publicId, String systemId)
    { System. out.println ("NotationDecl: "+name+" "+publicId+" "+systemId); }

    public void unparsedEntltyDecl (String name, String publicId,
                                    String systemId, String notationName)
    { System.out.println ("UnparsedEntityDecl: "+name + " "+publicId+"
                         "+systemId+" "+notationName);
    }

  ////////////////////////////////////////////////////////////////////
  // Sample implementation of the ErrorHandler interface.
  ////////////////////////////////////I/////////////I//////I////////

    public void warning (SAXParseException e)
                throws SAXException
    { System. out.println ("Warning: "+e.getMessage()); }

    public void error (SAXParseException e)
                throws SAXExcept ion
    { throw new SAXException (e. getMessageO ); }
```

```
    public void fatalError (SAXParseException e)
                throws SAXException
    {  System.out.println("Fatal error");
       throw new SAXException(e.getMessage()) ;
    }
}
```

API-интерфейсы пространства имен

Эти интерфейсы дают информацию о пространствах имен XML-документа, определяемых ссылками *универсального идентификатора ресурса* (Uniform Resource Identifier, *URI),* квалифицирующими элемент, присваивающими атрибуты именам и определяющими адреса ресурсов, которые могут располагаться на разных машинах или в разных XML-документах и т. д. Пространства имен могут разрешать присвоение одинаковых имен элементам и атрибутам, если последние для того, чтобы отличить их, снабжаются URI-идентификаторами. Например, **foo:hello** называется уточненным именем с префиксом пространства имен **foo**, отображенным на URI-идентификатор **www.oracle.com** и локальный элемент **hello.** Отметим, что URI-ссылки могут содержать символы, запрещенные к использованию в именах. Вот почему foo служит заменой для URI-ссылки.

Следующий пример кода демонстрирует простое использование синтаксического анализатора и расширения пространства имен для API-интерфейса DOM. Здесь происходит печать XML-файла вместе со всеми элементами и атрибутами.

```
import java.io.*;
import java.net.*;
import oracle.xml.parser.v2.DOMParser
import org.w3c.dom.*;
import org.w3c.dom.Node;

//Extension to DOM Interfaces for Namespace support.
import oracle.xml.parser.v2XMLElement;
import oracle.xml.parser.v2XMLAttr;

public class DOMNamespace .
{  static public void main(String[] argv)
   {  try
      {  if (argv.length != 1)
         C  // Must pass in the name of the XML file.
            System.err.println("Usage: java DOMSample filename");
            System.exit(1);
         }
```

```
        // Get an instance of the parser
        Class cls = Class.forName("oracle.[ml.parser.v2.DOMParser");
        DOMParser parser = (DOMParser)cls.newInstance

        // Generate a URL from the filename.
        URL url = createURL(argv[0]);

        // Parse the document.
        parser.parse(url);

        // Obtain the document.
        Document doc = parser.getDocument();

        // Print document elements
        printElements(doc);

        // Print document element attributes
        System.out.println("The attributes of each element are: ");
        printElementAttributes(doc);
    }
    catch (Exception e)
    { System.out.println(e.toString()); }
}

static void printElements (Document doc)
{ NodeList nl = doc.getElementsByTagName ("*");
  XMKElement nsElement

  String qName;
  String localName;
  String nsName;
  String expName;

  System.out.println("The element are: ");
  for (int i=0; i < nl/getLength() ; i++)
  { nsElement = XMLElement)nl.item(i);
    // Use the methods getQualifiedName(), getLocalName(), getNamespace(),
    // and getExpandeName() in NSName interfact to get Namespace
    // information.
    qName = nsElement.getQualifiedName();
    System.out.println(" ELEMENT Qualified Name:" + qName);

    LocalName = nsElement.getLocalName();
    System.out.println(" ELEMENT Local Name :" + LocalName);
```

```
            nsName=nsElement.getNamespace();
            System.out.println(" ELEMENT Namespace :" + nsName);

            expName = nsElement.getExpandeName();
            System.out.println(" ELEMENT ExpandeName :" + expName);
        }
        System.out.println();
    }

    static void printElementAttributes (Document doc)
    {  NodeList nl = doc.getElementsByTagName("*");
        Element e;
        XMLAttr nsAttr;

        String attrname;
        String attrval;
        String attrqname;

        NameNodeMap nnm;
        int i, len;

        len = nl.getLength();

        for (int j=0; j < len; j++)
        {  e = (Element)nl.item(j);
            System.out.println(e.getTagName() + ":");

            nnm = e.getAttributesO;

            if (nnm != null)
            {  for (i=0; i<nnm.getLength(); i++)
                {  nsAttr = (XMLAttr) nnm.item(i)
                    // Use the methods getQualifiedName(), getLocalName(),
                    // getNamespace() and getExpandeName() in NSName interfact
                    // to get Namespace information.
                    attrname = nsAttr.getExpandedName();
                    attrqname = nsAttr.getQualifiedName();
                    attrval=nsAttr.getNodeValue();

                    System.out.print(" " + attrname + "(" + attrname +
                                     "(" + " ="+attrval);
                }
            }
            System.out.println();
        }
    }
```

```
    static URL createURL(String fileName)
    {  URL url = null;
       try
       {  url = new URL(fileName);  }
       catch (MalformedURLException ex)
       {  try
          {  File f = new File(fileName);
             url = f.toURL();
          }
          catch (MalformedURLException e)
          {  System.out.println("Cannot create url for: " + fileName);
             System.exit(0);
          }
       }
       return url;
    }
}
```

API-интерфейсы анализатора синтаксиса

Прикладные программы вызывают API-интерфейс синтаксического анализа для чтения XML-документа и предоставляют доступ к его содержимому и структуре с помощью API-интерфейсов DOM или SAX. Обычно вместе с функцией синтаксического анализа активизируются функции начального запуска и завершения работы. Отметим, что перед тем как прикладной программой будет запущена функция синтаксического анализа, с помощью некоторых функций инициализации можно установить различные признаки, например для удаления пробелов или для включения функции проверки. Некоторые синтаксические анализаторы XML вообще не проверяют достоверность XML-документов на соответствие DTD, а в других анализаторах (например, разработанных компанией Oracle) существует опция такой проверки, и пользователи могут включить ее перед запуском функции анализа. Кроме того, программа синтаксического анализа, как это определено в XML-документе, должна воспринимать различные кодировки языка.

API-интерфейсы преобразования расширяемого языка стилей

Для преобразования или форматирования XML-документа консорциум W3C выпустил рекомендацию *преобразования расширяемого языка стилей* (Extensible Stylesheet Language Transformation, *XSLT*), которую разрешается применять для таблиц стилей XML-документов, написанных в определяемых пользователем XML- и XSLT-тегах и/или функциях с помощью XSLT-процессора. Подчеркнем, что этот

XSLT-процессор коррелирует картины в таблице стилей внутри XML-документа, а затем выполняет команды (например, устанавливает HTML-теги для окружения найденных данных) и выдает результаты в виде отдельного XML-документа. Вышеприведенный пример можно дополнить использованием XSLT-процессора. Тег заголовка книги **<title>** можно обработать функцией таблицы стилей и снабдить его, например, HTML-тегами полужирного шрифта <Ь> и ****, в результате чего получится другой XML-документ, который будет иметь HTML-теги полужирного шрифта, окружающие данные с названием книги. Эти API-интерфейсы для применения таблиц стилей к XML-файлам довольно просты, и некоторые XML-анализаторы синтаксиса, например анализатор от Oracle, уже включают в себя XSLT-процессор.

API-интерфейсы XML Schema

Консорциум W3C уже разработал рекомендацию для XML Schema, позволяющую определять простые и сложные типы данных для построения элементов документа и узлов. Раньше, когда существовали только DTD, для описания элементов можно было использовать лишь строковые типы данных, и значения этих элементов могли быть только действительными или целыми числами. Сейчас в модель содержимого для документа можно встраивать не только традиционные типы данных *long, short, int* и другие, но и *facets* для обозначения максимальных и минимальных допустимых значений данных. Для проверки документа на соответствие XML Schema в прикладную программу нужно добавить один или два дополнительных API-интерфейса, а также провести необходимые изменения в XML-документе, которые будут соответствовать XML Schema.

Почему XML?

Если XML-документ составлен корректно, он легко читается и в нем легко разобраться. Теги, характеризующие данные, например **<author first name>Juan </author first name>,** добавляют этим данным смысла, так что понять их совсем нетрудно. Первое следствие этого состоит в том, что поиск в Интернете можно вести более эффективно, если Web-страницы написаны в XML-формате. Причем таким способом можно искать не только данные, но и связанное с ними содержание. То есть механизм поиска становится более точным, если в запрос кроме самих данных можно добавить и связанное с ними содержимое. Кроме того, если теги, связанные с данными, достаточно точно их описывают, такой документ смог бы без проблем понять и пользователь, живший 10 лет назад, чего не скажешь о некоторых других форматах документов.

Другое преимущество XML основано на открытых стандартах, разработанных консорциумом W3C. Поскольку они являются открытыми, ни одна компания не может захватить их в свою личную собственность и помешать внесению в них

изменений со стороны других компаний. Если же какие-то компании хотят дополнить стандарт определенными функциональными возможностями, то комитеты рабочих групп консорциума W3C, состоящие из представителей компаний, работающих в этой отрасли индустрии, встречаются и обсуждают приемлемость вносимых изменений и при необходимости добавляют в стандарт предлагаемые функции. Если же компании-новаторы не хотят ждать, пока консорциум W3C внесет изменения в стандарт, они могут попробовать внедрить функции, которые в данный момент еще официально не определены. Конечно, остается риск, что конечные пользователи могут и не принять предлагаемые ими расширения, не представленные консорциумом W3C. Но даже если нововведения восприняты сообществом пользователей, их все равно нужно официально представить в консорциум W3C, чтобы преимущества новых функций могли получить все разработчики.

Создавать XML-документы довольно легко, кроме того, они просты для понимания и основаны на открытых стандартах, поэтому можно представить себе огромное многообразие приложений, основанных на технологии XML, которые разрабатываются в настоящее время в индустрии программного обеспечения. Эти приложения имеют дело с построенными на базе технологии XML электронными книгами, журналами или любыми другими документами, которые нужно хранить и которыми нужно обмениваться с другими приложениями. С такими приложениями работают онлайновые компании, извлекающие данные и направляющие их в другие онлайновые компании в каком-то общем формате. Так, например, происходит в приложениях *электронной коммерции* (business-to-business, *B2B),* которые преобразуют данные, хранящиеся в традиционных реляционных базах данных, в XML-формат. Преимущества от использования технологии XML получают как системы, централизованные по документам, так и системы, централизованные по данным. И современные приложения наглядно отражают эту ситуацию.

Разработчикам приложений также на руку желание компаний разрешить использование общих функциональных возможностей XML и бесплатную загрузку готовых XML-анализаторов синтаксиса и XSLT-процессоров. Такие функции можно бесплатно скопировать из Интернета и использовать для быстрой разработки приложений, предназначенных для внутренних корпоративных компьютерных систем компаний. Кроме того, эти XML-компоненты созданы на базе открытых стандартов, и разработчики смогут переносить свое ПО с одной платформы на другую, добавляя и отключая функции. Естественно, что перед использованием бесплатного ПО разработчикам приложений необходимо рассмотреть проблемы производительности, использования памяти, реализации полного набора функциональных возможностей, технической поддержки и т. д., но при этом следует помнить, что XML-продукты нужно разрабатывать быстро, иначе теряется смысл внедрения технологии XML.

Наконец, при условии всех ранее перечисленных преимуществ, для управления форматом XML-файлов, обмениваемых между компаниями через Интернет, были созданы специфические для конкретных отраслей определения типов данных DTD (здравоохранение, финансы и др.). Явное преимущество такого подхода состоит в том, что работу онлайновых приложений электронной коммерции B2B можно

существенно упростить благодаря использованию XML как формата данных для их обмена и хранения. Компании в ряде отраслей индустрии уже не беспокоятся о преобразовании файлов из одного формата в другой, так как могут договориться о специальном XML-формате, например для автомобильных запчастей. Выгоды от преобразования данных и документов в XML-формат получат потребители и компании, которые совершают транзакции через Интернет.

Стратегия Oracle в области XML

В 1997 и 1998 гг., когда велась работа по созданию рекомендации XML 1.0, компания Oracle еще не участвовала в этом процессе. В те времена поиск в Интернете информации о глобальных интранет-сетях Oracle не давал никаких ссылок на "XML". Но не прошло и двух лет, как в ответ на тот же самый поисковый запрос стали выдаваться тысячи ссылок. Быстрое внедрение технологии XML началось с того, что Совет по архитектуре (Architectural Review Board) компании Oracle представил в группу по разработке библиотеки функций CORE (CORE Development Group) документ для разработки инфраструктурных компонентов XML, которыми могли бы пользоваться группы разработчиков внутри самой компании Oracle.

Вместе с этим документом Совет по архитектуре сформулировал стратегию Oracle в области XML одним коротким предложением: "Создать наилучшую платформу для разработчиков, чтобы продуктивно строить и недорого разворачивать надежные и масштабируемые Интернет-приложения, использующие технологию XML". Важно отметить, что в этом предложении делается акцент не на самом продукте, а на его разработке. Это очень последовательный шаг, так как XML в отличие от других языков разметки не является приложением. XML — это технология, позволяющая создавать приложения, и таким образом подчеркивается ее важность в контексте разработанных решений.

Как только стратегия XML-разработки начала реализовываться, возник всеобщий интерес к использованию этой технологии, что нашло выражение практически в каждом программном продукте Oracle. По последним подсчетам, в 2001 г. более 60 групп разработчиков так или иначе использовали XML в своих продуктах.

Интерес этот не ограничился внутренними группами разработчиков Oracle. Слияния и приобретения компаний среди клиентов Oracle поставили их перед необходимостью интегрировать самые разные по своей природе информационные системы, многие из которых созданы по патентованным технологиям. XML и сопутствующая ей технология XSL хорошо подходят для построения связующего фундамента таких систем.

Сеть Интернет с ее требованием распределенного доступа к самым разным системам через сетевые экраны и множество проводных и беспроводных сред является идеальной средой для транспортировки XML-документов и данных. Такая ситуация существует благодаря несложному технологическому соглашению, которое требуется

установить между приложениями и серверами, поддерживающими XML. Технология XML быстро стала объединяющим элементом для всех данных, независимо от того, извлекаются ли они из информационных хранилищ, из приложений самообслуживания клиентов, из приложений электронной B2B коммерции или управления контентом. Поэтому вполне естественно, что объектно-реляционные базы данных Oracle стали поддерживать технологию XML.

Деятельность Oracle в XML-индустрии

Oracle активно занимается XML, причем не только как основной технологией, которая должна быть доступна для разработки приложений, но и как двигателем стандартов и связанных с нею технологий. Ниже представлен обзор работ Oracle по разработке и стандартизации XML, которые проводились совместно с такими организациями, как World Wide Web Consortium, Sun Java Community Process и Open Applications Group.

Oracle в комитетах рабочей группы W3C

Несмотря на то, что Oracle не принимала участие в рабочей группе W3C XML с самого начала ее существования, огромная корпоративная поддержка этой технологии заставила Oracle не только войти во все группы разработки XML-стандартов, но и помочь в управлении ими. Oracle является членом следующих рабочих групп W3C XML в области технологии XML:

- **Рабочая группа XML CORE** — отвечает за определения технологии, которые используют все остальные рабочие группы. Например, XBase, определяющая формат и поведение построенных на базе XML универсальных индикаторов ресурсов URI, и XInclude, определяющая синтаксис и способ включения других XML-файлов в XML-документ.
- **Рабочая группа XSL** — отвечает за определение языка Extensible Stylesheet Language (XSL) в двух его реализациях: XSL:T определяет синтаксис таблиц стилей и функциональные возможности, используемые для преобразования одного XML-документа в другой; XSL:FO определяет синтаксис таблицы стилей для перевода XML-документа в различные форматы электронных публикаций.
- **Рабочая группа XML Schema** — отвечает за определение синтаксиса и использование простых типов данных и сложных структур внутри XML-документа. Она также дает схематическое определение формата XML-файла для замены определений типов данных DTD и добавления поддержки данных и подтверждения структуры типов.

- **Рабочая группа XML Linking** — отвечает за определение того, как производятся обращения к XML-ресурсам из XML-документов. Обобщив функциональные возможности HTML-ссылок, рабочая группа XML Linking разрешила ссылаться на несколько документов, которые находятся внутри и вне XML-документов.
- **Рабочая группа DOM** — отвечает за API-интерфейсы DOM для доступа к XML-документам. Эти API-интерфейсы обеспечивают программируемый доступ и манипуляции со структурой элементов и узлов XML-документа, находящихся в памяти.
- **Рабочая группа XML Query** — отвечает за спецификацию грамматики и синтаксиса составления запросов к XML-документам. Благодаря иерархической древовидной структуре XML-документа по нему можно легко перемещаться и получать результаты запросов.

Инструментарий XML-разработчика производства Oracle

Как упоминалось ранее, группа разработки CORE и XML в подразделении баз данных Oracle разработала все основные компоненты инфраструктуры XML, которые используют группы разработчиков Oracle. Любой продукт, который имеет дело с XML, должен уметь определять, является ли данный XML действительным (valid), т. е. соответствующим заданным определениям типа документа DTD или XML Schema, и правильным (well formed), т. е. его открывающие теги имеют соответствующие им закрывающие теги, расположенные без нарушения порядка вложенности. Это проверяется с помощью анализатора XML-синтаксиса. Так как группы разработчиков Oracle имеют различные требования и разных клиентов, анализатор XML-синтаксиса и встроенный XSLT-процессор были написаны на четырех разных языках: C, C++, Java и PL/SQL. Кроме того, существуют разные реализации программы анализа синтаксиса XML, включающие исполняемые программы, вводимые с командной строки, библиотеки и **.jar** файлы. Чтобы упростить визуальную разработку приложения, существует несколько Java-модулей для просмотра, редактирования и преобразования XML. В инструментарий Oracle XDK включена также поддержка W3C XML Schema, DOM Level 2, SAX Level 2 и т. д.

Программа анализа синтаксиса Oracle XML Parser for Java написана на языке Java, и ее можно запускать в виртуальной Java-машине в СУБД Oracle8i и Oracle9i. В ней есть опциональная проверка на действительность XML-документов с захватом DTD, полная поддержка наборов символов разных языков, полная поддержка технологий DOM и SAX и интегрированный XSLT-процессор. По сравнению с другими Java XML-анализаторами она работает довольно быстро. Программы XML Parser for C и C++ поставляются для ОС Solaris, Linux, HP-UX и Windows NT.

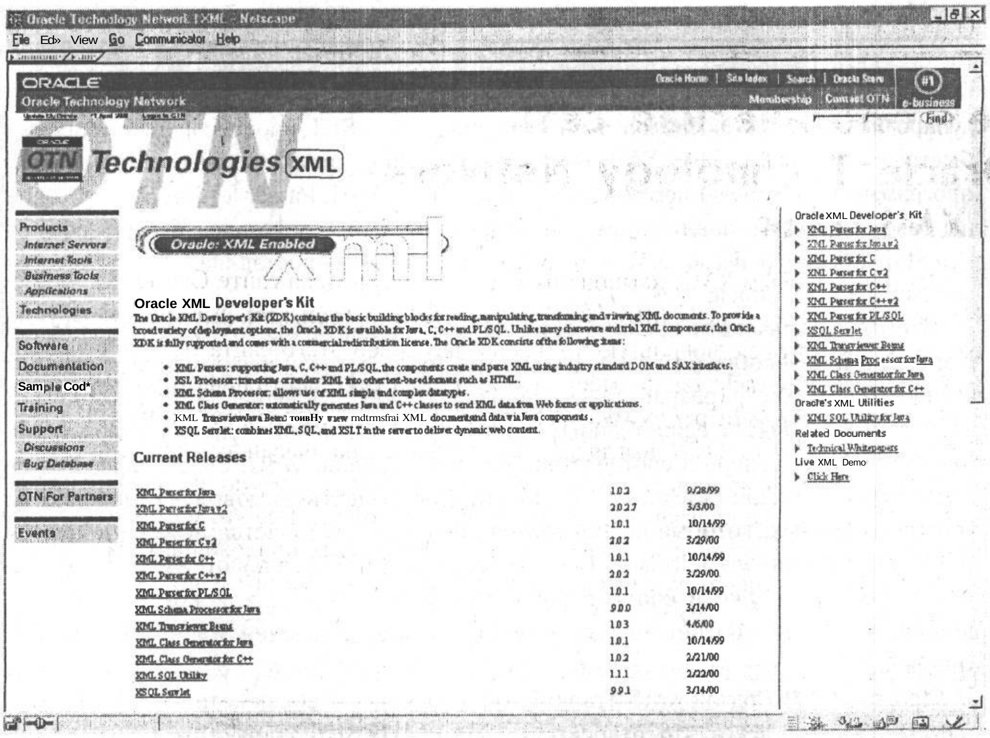
Они имеют полную поддержку интерфейсов DOM и SAX, полную поддержку международного набора символов и интегрированный XSLT-процессор. Они работают быстрее других C и C++ XML-анализаторов и поставляются вместе с серверным программным обеспечением от Oracle. Анализатор XML Parser for PL/SQL построен на базе Java XML-анализатора, он разрешает взаимодействие API-интерфейсов PL/SQL с интерфейсами XML-анализатора. Он также поставляется вместе с серверным ПО от Oracle.

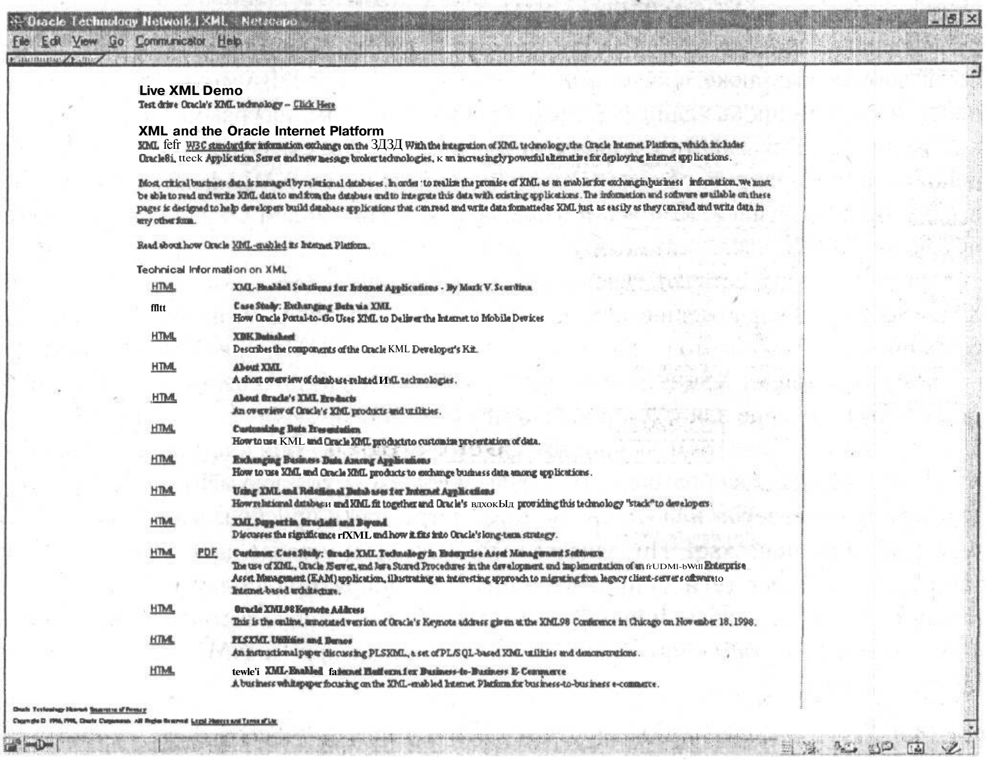
Чтобы использовать преимущества рекомендаций W3C XML Schema Recommendation и разнообразить XML-документы более богатой информацией из данных разных типов, были разработаны процессоры XML Schema на языках Java, C и C++. Вместо DTD с единственным типом строковых данных для описания элементов и атрибутов теперь в соответствии с рекомендацией W3C, описывающей части XML-документа, для описания этих документов существует довольно широкий набор типов данных, которые используются наряду с пространствами имен и другими конструктивными элементами. Все эти процессоры объявляют XML-документы действительными уже в соответствии с новой формой модели контента, а если документы ей не соответствуют, выдаются предупреждения или сообщения об ошибках.

Программные модули XML TransViewer включают в себя некоторые XML/XSLT-классы. Прежде всего это относится к модулям XSLT Transformer, XSLT DOM Parser, XML Source Viewer/Editor и XML Tree Viewer, которые установлены в инструментальной линейке XML Palette, начиная с версии JDeveloper 3.1. В этот набор были добавлены также программные модули DBView и DBAccess, что позволяет использовать мощь баз данных Oracle. Однако при этом программистам приложений не приходится писать слишком много кода, чтобы связаться с базой данных, исполнить SQL-запрос, сформатировать результат в виде XML-файла и применить к нему таблицу стилей, хранить и извлекать XML из таблиц больших символьных объектов CLOB в базе данных и в файловой системе.

Кроме того, существуют генераторы классов Java и C++, на основе DTD или XML Schema генерирующие код, к которому можно программно обращаться для создания XML-документов, соответствующих данным DTD или XML Schema.

Наконец, сервлет XSQL Servlet представляет собой Java-сервлет, который работает на Web-сервере для создания динамических XML-документов, построенных на базе одного и более SQL-запросов. Он может также опционально с помощью XSLT-процессора преобразовывать результирующий XML-документ в формат, поддерживаемый сервером или клиентом. Для инструкций и преобразований он использует файл описаний **.xsql**. По существу, XSQL Servlet устанавливает связь с базой данных, снимая все теги до прохождения SQL-оператора в базу данных. Он извлекает строчные данные и внедряет их в XML с помощью тегов, соответствующих колонкам и таблицам базы данных, и применяет к XML опциональную таблицу стилей.


Oracle Technology Network | XML - Netscape
File Edit View Go Communicator Help
ORACLE
Oracle Technology Network
Oracle Home | Site Index | Search | Oracle Store
Membership | Contact OTN
#1 e-business
Find
OTN Technologies XML
Products
Internet Servers
Internet Tools
Business Tools
Applications
Technologies
Software
Documentation
Sample Code
Training
Support
Discussions
Bug Database
OTN For Partners
Events
Oracle: XML Enabled
Oracle XML Developer's Kit
The Oracle XML Developer's Kit (XDK) contains the basic building blocks for reading, manipulating, transforming and viewing XML documents. To provide a broad variety of deployment options, the Oracle XDK is available for Java, C, C++ and PL/SQL. Unlike many shareware and trial XML components, the Oracle XDK is fully supported and comes with a commercial redistribution license. The Oracle XDK consists of the following items:
XML Parsers: supporting Java, C, C++ and PL/SQL, the components create and parse XML using industry standard DOM and SAX interfaces.
XSL Processor: transforms or renders XML into other text-based formats such as HTML.
XML Schema Processor: allows use of XML simple and complex datatypes.
XML Class Generator: automatically generates Java and C++ classes to send XML data from Web forms or applications.
XSQL Servlet: combines XML, SQL, and XSLT in the server to deliver dynamic web content.
Current Releases
XML Parser for Java 1.0.2 9/28/99
XML Parser for Java v2 2.0.2.7 3/3/00
XML Parser for C 1.0.1 10/14/99
XML Parser for C v2 2.0.2 3/29/00
XML Parser for C++ 1.0.1 10/14/99
XML Parser for C++ v2 2.0.2 3/29/00
XML Parser for PL/SQL 1.0.1 10/14/99
XML Schema Processor for Java 9.0.0 3/14/00
XML Transviewer Beans 1.0.3 4/6/00
XML Class Generator for Java 1.0.1 10/14/99
XML Class Generator for C++ 1.0.2 2/21/00
XML SQL Utility 1.1.1 2/22/00
XSQL Servlet 9.9.1 3/14/00
Oracle XML Developer's Kit
XML Parser for Java
XML Parser for Java v2
XML Parser for C
XML Parser for C v2
XML Parser for C++
XML Parser for C++ v2
XML Parser for PL/SQL
XSQL Servlet
XML Transviewer Beans
XML Schema Processor for Java
XML Class Generator for Java
XML Class Generator for C++
Oracle's XML Utilities
XML SQL Utility for Java
Related Documents
Technical Whitepapers
Live XML Demo
Click Here


Oracle Technology Network | XML - Netscape
File Edit View Go Communicator Help
Live XML Demo
Test drive Oracle's XML technology -- Click Here
XML and the Oracle Internet Platform
With the integration of XML technology, the Oracle Internet Platform, which includes Oracle8i, Application Server and new message broker technologies, an increasingly powerful alternative for deploying Internet applications.
Most critical business data is managed by relational databases. In order to realize the promise of XML as an enabler for exchanging business information, we must be able to read and write XML data to and from the database and to integrate this data with existing applications. The information and software available on these pages is designed to help developers build database applications that can read and write data formatted as XML just as easily as they can read and write data in any other form.
Read about how Oracle XML-enabled its Internet Platform.
Technical Information on XML
HTML XML-Enabled Solutions for Internet Applications - By Mark V. Scardina
Case Study: Exchanging Data via XML
How Oracle Portal-to-Go Uses XML to Deliver the Internet to Mobile Devices
HTML XDK Datasheet
Describes the components of the Oracle XML Developer's Kit.
HTML About XML
A short overview of database-related XML technologies.
HTML About Oracle's XML Products
An overview of Oracle's XML products and utilities.
HTML Customizing Data Presentation
How to use XML and Oracle XML products to customize presentation of data.
HTML Exchanging Business Data Among Applications
How to use XML and Oracle XML products to exchange business data among applications.
HTML Using XML and Relational Databases for Internet Applications
HTML XML Support in Oracle8i and Beyond
Discusses the significance of XML and how it fits into Oracle's long-term strategy.
HTML PDF Customer Case Study: Oracle XML Technology in Enterprise Asset Management Software
HTML Oracle XML98 Keynote Address
This is the online, annotated version of Oracle's Keynote address given at the XML98 Conference in Chicago on November 18, 1998.
HTML PLSXML Utilities and Demos
An instructional paper discussing PLSXML, a set of PL/SQL-based XML utilities and demonstrations.
HTML XML-Enabled Internet Platform for Business-to-Business E-Commerce
A business whitepaper focusing on the XML-enabled Internet Platform for business-to-business e-commerce.

Технологическая сеть Oracle Technology Network и XML Link

Доступ к релизам XML-компонентов можно получить на сайте Oracle Technology Network (OTN) по адресу http://technet.oracle.com/tech/xml. Внутренние группы разработчиков Oracle используют зеркальный сервер этого сайта без регистрации, а подразделение Oracle Japan имеет свой японский сайт, зайти на который можно по следующему адресу: **http://www.oracle.co.jp/download/rdbms/index.html**. Главная Web-страница сайта OTN выглядит, как показано на рисунках слева.

Релизы каждого компонента XML имеют свою зону, в которую входят библиотеки и DLL, исполняемые программы, документация и примеры. Все это можно совершенно бесплатно загружать и использовать в любых других приложениях! Например, Web-страница анализатора синтаксиса Java XML Parser V2 выглядит, как показано на рис. 1.1.

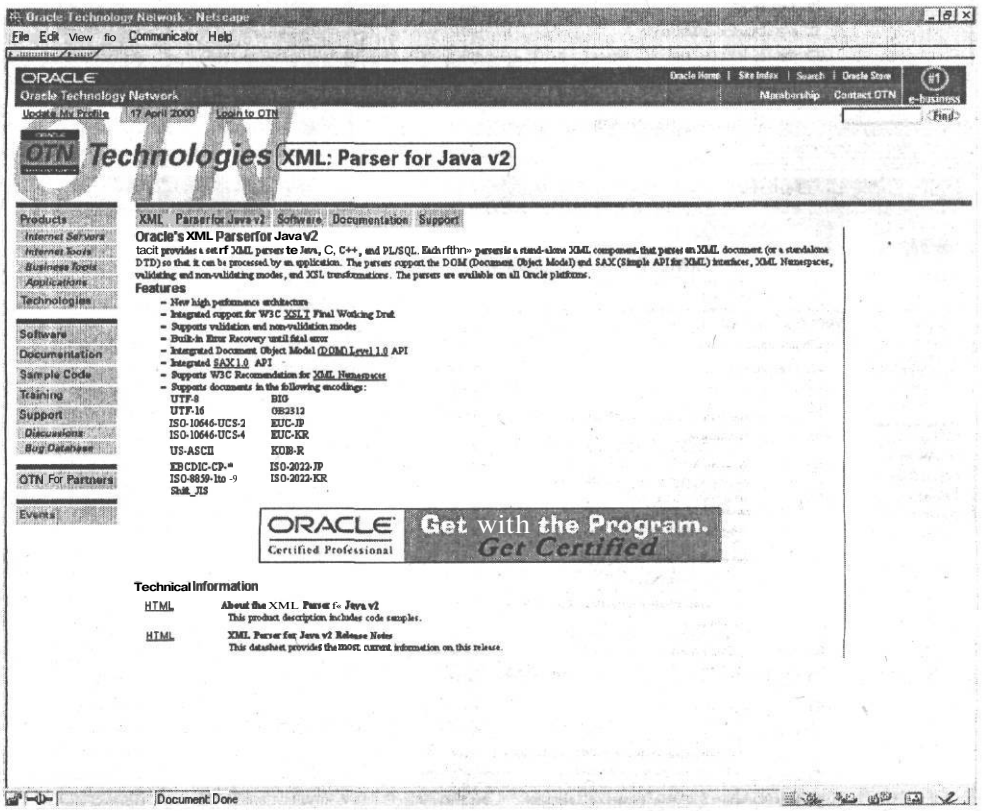
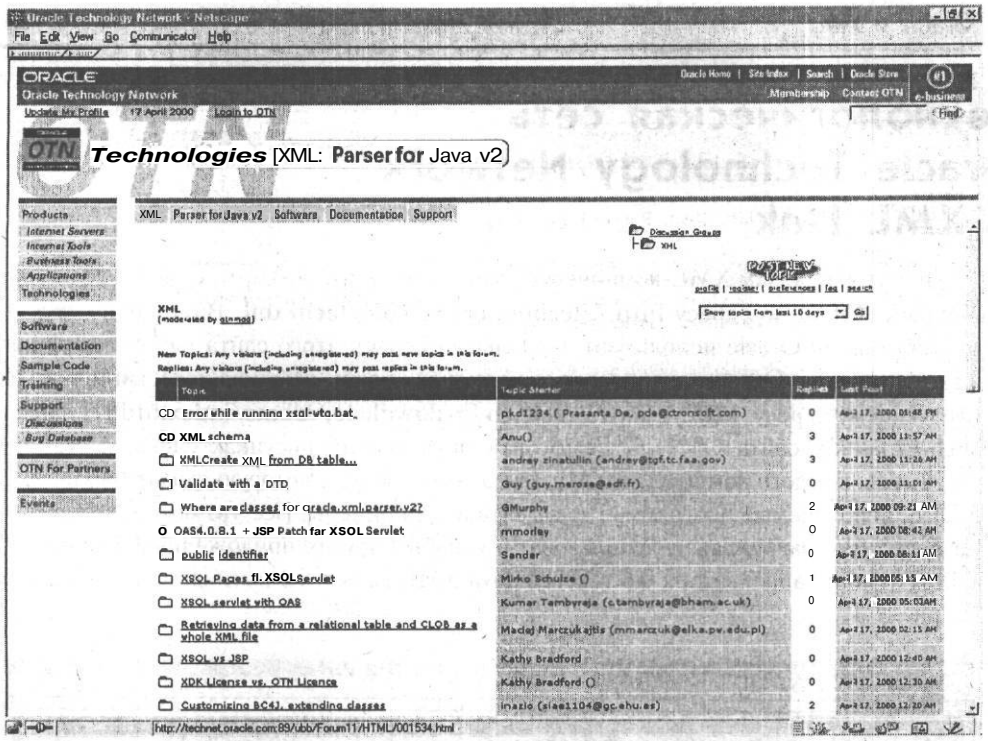
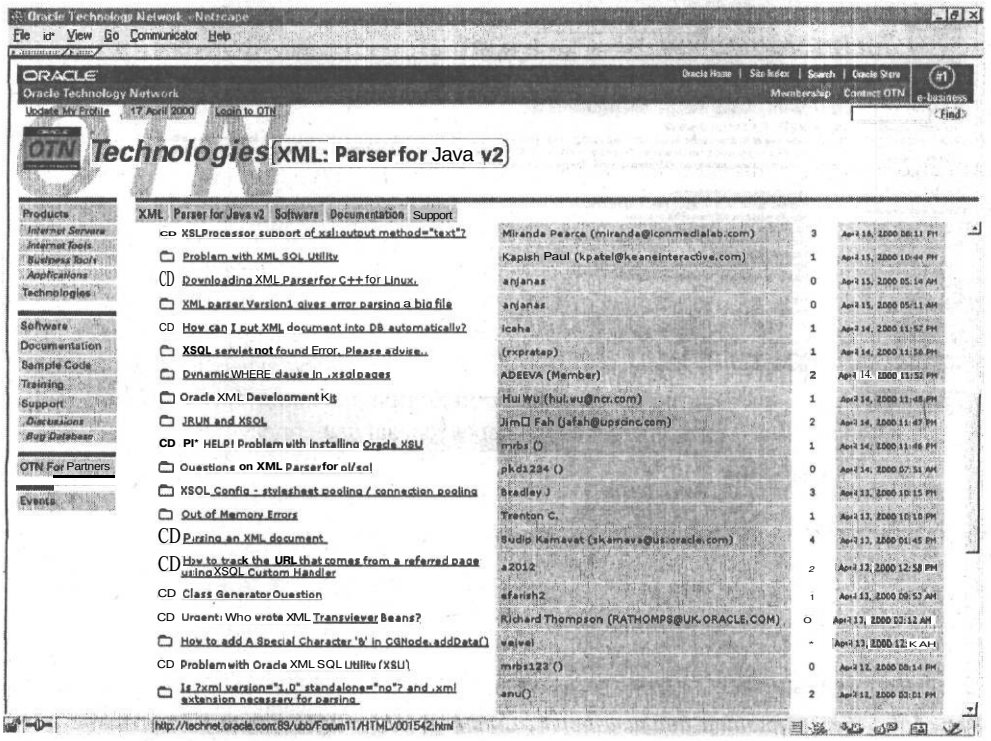
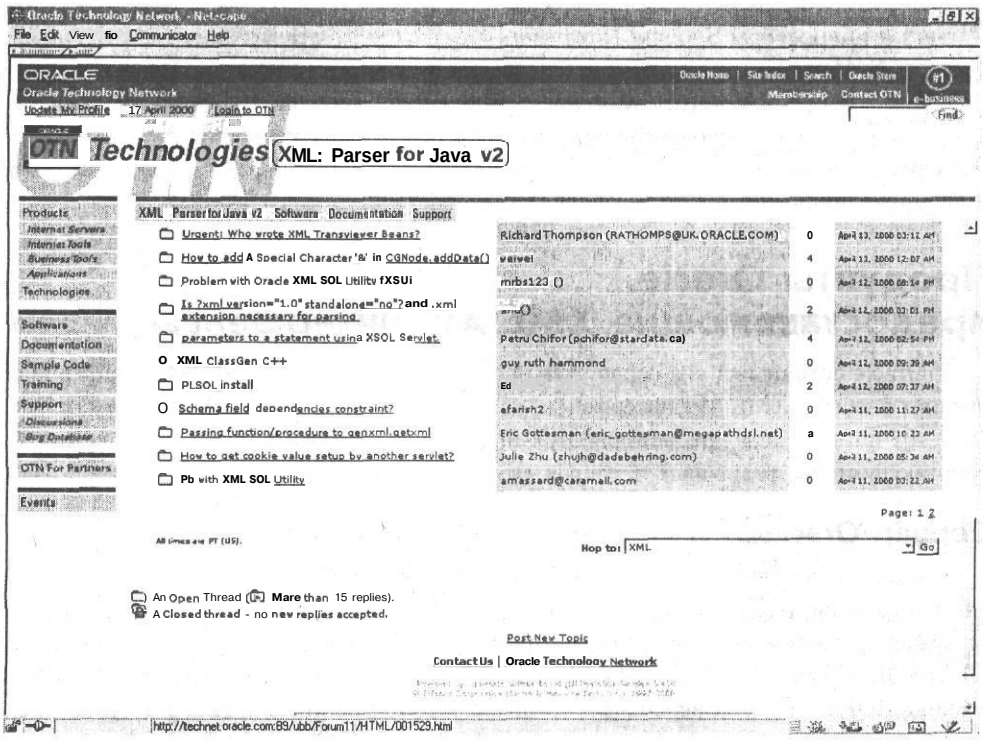


Рис. 1.1. *Web-страница анализатора синтаксиса Java XML Parser V2 на сайте OTN*


Oracle Technology Network - Netscape
File Edit View Go Communicator Help
ORACLE
Oracle Technology Network
Oracle Home | Site Index | Search | Oracle Store
Membership | Contact OTN
Update My Profile | 17 April 2000 | Login to OTN
OTN Technologies [XML: Parser for Java v2]
Products
Internet Servers
Internet Tools
Business Tools
Applications
Technologies
Software
Documentation
Sample Code
Training
Support
Discussions
Bug Database
OTN For Partners
Events
XML | Parser for Java v2 | Software | Documentation | Support
Discussion Groups
XML
POST NEW TOPIC
profile | register | preferences | faq | search
XML
(moderated by simmod)
Show topics from last 10 days | Go
New Topics: Any visitors (including unregistered) may post new topics in this forum.
Replies: Any visitors (including unregistered) may post replies in this forum.
Topic | Topic Starter | Replies | Last Post
CD Error while running xsql-vto.bat | pkd1234 (Prasanta De, pde@ctronsoft.com) | 0 | Apr 17, 2000 01:48 PM
CD XML schema | Anu() | 3 | Apr 17, 2000 11:57 AM
XMLCreate XML from DB table... | andrey zinatullin (andrey@tpf.tc.faa.gov) | 3 | Apr 17, 2000 11:26 AM
Validate with a DTD | Guy (guy.marose@edf.fr) | 0 | Apr 17, 2000 11:01 AM
Where are classes for oracle.xml.parser.v2? | GMurphy | 2 | Apr 17, 2000 09:21 AM
O OAS4.0.8.1 + JSP Patch far XSOL Servlet | mmorley | 0 | Apr 17, 2000 08:42 AM
public identifier | Sander | 0 | Apr 17, 2000 08:11 AM
XSQL Pages fl. XSQLServlet | Mirko Schulze () | 1 | Apr 17, 2000 05:15 AM
XSQL servlet with OAS | Kumar Tambyraja (c.tambyraja@bham.ac.uk) | 0 | Apr 17, 2000 05:03AM
Retrieving data from a relational table and CLOB as a whole XML file | Maciej Marczukajtis (mmarczuk@elka.pw.edu.pl) | 0 | Apr 17, 2000 02:15 AM
XSQL vs JSP | Kathy Bradford | 0 | Apr 17, 2000 12:40 AM
XDK License vs. OTN Licence | Kathy Bradford () | 0 | Apr 17, 2000 12:30 AM
Customizing BC4J, extending classes | inaxio (ziaes1104@gc.ehu.es) | 2 | Apr 17, 2000 12:20 AM
http://technet.oracle.com:89/ubb/Forum11/HTML/001534.html


Oracle Technology Network - Netscape
File Edit View Go Communicator Help
ORACLE
Oracle Technology Network
Oracle Home | Site Index | Search | Oracle Store
Membership | Contact OTN
Update My Profile | 17 April 2000 | Login to OTN
OTN Technologies [XML: Parser for Java v2]
Products
Internet Servers
Internet Tools
Business Tools
Applications
Technologies
Software
Documentation
Sample Code
Training
Support
Discussions
Bug Database
OTN For Partners
Events
XML | Parser for Java v2 | Software | Documentation | Support
CD XSLProcessor support of xsl:output method="text"? | Miranda Pearce (miranda@iconmedialab.com) | 3 | Apr 18, 2000 06:11 PM
Problem with XML SQL Utility | Kapish Paul (kpatel@keaneinteractive.com) | 1 | Apr 15, 2000 10:44 PM
CD Downloading XML Parser for C++ for Linux. | anjanas | 0 | Apr 15, 2000 05:14 AM
XML parser Version1 gives error parsing a big file | anjanas | 0 | Apr 15, 2000 05:11 AM
CD How can I put XML document into DB automatically? | icaha | 1 | Apr 14, 2000 11:57 PM
XSQL servlet not found Error. Please advise.. | (rvpratap) | 1 | Apr 14, 2000 11:36 PM
Dynamic WHERE clause in .xsql pages | ADEEVA (Member) | 2 | Apr 14, 2000 11:52 PM
Oracle XML Development Kit | Hui Wu (hui.wu@ncr.com) | 1 | Apr 14, 2000 11:48 PM
JRUN and XSQL | Jim Fah (jafah@upsonc.com) | 2 | Apr 14, 2000 11:47 PM
CD PI* HELP! Problem with installing Oracle XSU | mrbs () | 1 | Apr 14, 2000 11:46 PM
Questions on XML Parser for pl/sql | pkd1234 () | 0 | Apr 14, 2000 07:51 AM
XSQL Config - stylesheet pooling / connection pooling | Bradley J | 3 | Apr 13, 2000 10:15 PM
Out of Memory Errors | Trenton C. | 1 | Apr 13, 2000 10:10 PM
CD Parsing an XML document | Sudip Kamavat (skamavs@us.oracle.com) | 4 | Apr 12, 2000 01:45 PM
CD How to track the URL that comes from a referred page using XSQL Custom Handler | a2012 | 2 | Apr 13, 2000 12:58 PM
CD Class Generator Question | sfarish2 | 1 | Apr 13, 2000 09:53 AM
CD Urgent: Who wrote XML Transviewer Beans? | Richard Thompson (RATHOMPS@UK.ORACLE.COM) | 0 | Apr 13, 2000 03:12 AM
How to add A Special Character '&' in CGNode.addData() | vaivai | - | Apr 13, 2000 12:KAH
CD Problem with Oracle XML SQL Utility (XSU) | mrbs123 () | 0 | Apr 12, 2000 06:14 PM
Is ?xml version="1.0" standalone="no"? and .xml extension necessary for parsing | anu() | 2 | Apr 12, 2000 03:01 PM
http://technet.oracle.com:89/ubb/Forum11/HTML/001542.html



Наконец, кроме белых страниц и других вспомогательных материалов на сайте есть дискуссионный форум по проблемам XML, где сообщество пользователей может публиковать самую разную информацию, в том числе сообщения о замеченных ошибках в ПО. Web-страница дискуссионного форума по проблемам XML показана на трех представленных слева и вверху рисунках.

Анализаторы синтаксиса входят в комплект других Интернет-продуктов, включенных в состав инструментария Internet Platform ISV. Такие продукты Oracle, как Oracle8i и Oracle9i, JDeveloper 3.1, Applications 1/1i и Oracle9iAS (новый сервер приложений Oracle), имеют в составе библиотек или виртуальных Java-машин анализатор XML-синтаксиса, поэтому разработчики приложений могут получить доступ к API-интерфейсам этого анализатора через указанные продукты.

Обзор продуктов Oracle, поддерживающих технологию XML

Более 60 групп разработчиков Oracle в подразделениях разработки серверов, сервисных программ и приложений используют в создаваемых ими продуктах XML-компоненты. Они используют анализаторы C, C++, Java, PL/SQL XML/XSLT

Parsers и процессоры Schema Processors для синтаксического анализа и преобразования XML-документов, генераторы классов для Java и C++ в разработке приложений и сервлет XSQL Servlet для служб публикации документов и рендеринга. Разработчики Oracle-продуктов создают продукты, обеспечивающие API-интерфейсы XML, а также использующие XML для обмена данными, конфигурации приложений, управления контентом и публикации информации.

Продукты Oracle, предоставляющие XML API-интерфейсы

У Oracle есть целый ряд серверных продуктов и сервисных программ, предоставляющих внешние XML API-интерфейсы, которые могут использоваться приложениями. Эти API-интерфейсы предоставляются вышеперечисленными XML-компонентами, работающими на различных серверах от Oracle.

Сервер Oracle8i/9i JServer

С момента появления Oracle8i разработчики получили возможность использовать внутреннюю виртуальную Java-машину, работающую внутри базы данных. JServer загружает Java-классы как хранимые процедуры точно так же, как это происходит в PL/SQL. Самый первый релиз JServer был совместим с инструментарием разработчика JDK 1.1, а его версия для Oracle9i уже совместима с JDK 1.2. В JServer можно загрузить Java-компоненты инструментария XDK, и тогда приложение получит быстрый доступ к базе данных.

JDeveloper

JDeveloper — это основная интегрированная среда разработки (IDE) компании Oracle. С версии 3.0 в ней поддерживается технология XML и для внутреннего использования, и для разработки приложений. Все Java-компоненты инструментария XDK можно использовать вместе с JDeveloper, а модули XML TransViewer можно установить на инструментальной панели. Начиная с версии 3.1, в JDeveloper были проделаны существенные усовершенствования в деле поддержки технологии XML. Кроме уже упоминавшейся панели модулей XML TransViewer Bean данные в XML-формате могут генерировать и другие модули. Имеющийся в нынешней версии JDeveloper редактор понимает файлы типов **.xml** и **.xsl** и помимо проверки синтаксиса обеспечивает еще и цветную подсветку синтаксических элементов. Наконец, компоненты Business Components for Java из JDeveloper используют XML-формат для своих бизнес-правил и данных.

Сервер приложений Oracle9i Application Server

Oracle9i Application Server (Oracle9iAS) — это серверный продукт промежуточного слоя компании Oracle. Oracle9iAS выполняет функции и Web-сервера, и сервера приложений, кроме того, имеет встроенные компоненты инструментария XDK.

Java-компоненты XDK можно запускать на виртуальной Java-машине Oracle9iAS с целью создания XML-интерфейсов для Web- и Java-приложений. Компоненты PL/SQL можно запускать также в среде реального времени PL/SQL из Oracle9iAS. Способность работать на промежуточном слое особенно важна для приложений, которые выполняют большие объемы XSL-обработки или должны поддерживать работу тысяч пользователей.

XML SQL Utility

Написанная на языке Java в составе поставляемого пакета PL/SQL утилита XML SQL Utility (XSU) позволяет разработчикам отправлять SQL-запросы через JDBC и получать результаты в виде схемы XML-формата. Кроме обеспечения выхода текстового потока XML SQL Utility может выдавать ответ на запрос в виде объекта DOM, а также в виде определений DTD или Schema, соответствующих схеме запрошенной базы данных. Подобно Java-компонентам XDK, утилиту XML SQL Utility можно загрузить как хранимую процедуру в Oracle8i/9i и использовать с любой реляционной базой данных, поддерживающей JDBC.

Продукты Oracle, использующие XML для обмена данными

Многие приложения Oracle используют таблицы базы данных для обмена информацией. С появлением стандарта для XML они стали использовать эту технологию в форматах информационных сообщений и для составления отчетов.

Сервер Oracle Integration Server

Oracle Integration Server использует XML в качестве формата сообщений, которыми обмениваются приложения. Например, брокер сообщений Oracle Message Broker упаковывает XML-сообщения в конверт службы обмена сообщениями Java Messaging Service для доставки подключенным приложениям.

Oracle Report 6i

Oracle Report используется для обмена данными в приложениях B2B. Он позволяет реализовать поддержку отчетов и публикацию каталогов в XML-формате. Это ПО также поддерживает в отчетах таблицы стилей XSL, поэтому их можно выводить на экране Интернет-браузеров.

Oracle Application Release Hi

В пакет бизнес-приложений Oracle для предприятий входят средства управления отношений с клиентами (Customer Relationship Management), организации производства и поставок (Manufacturing/Supply Chain), управления финансами (Financials),

организации поставок через Интернет (Internet Procurement), управления персоналом (Human Resources) и проектами (Projects). Эти приложения используют XML в качестве формата для обмена данными и сообщениями с целью достижения интеграции.

Self-Service Purchasing

Программа Self-Service Purchasing, которая обеспечивает самообслуживание клиента при оформлении покупок, использует XML-документы в качестве корзины при обмене данными между клиентами и поставщиками. Она также использует XML для форматирования каталожной информации и ее доставки клиенту через Интернет.

Продукты Oracle, использующие технологию XML для конфигурирования

Файлы конфигурации приложений исторически имели формат плоских файлов и создавались с помощью настраиваемых анализаторов синтаксиса. Теперь при наличии XML-формата эти приложения могут использовать легко расширяемую структуру XML, а для получения данных использовать анализаторы синтаксиса, построенные на базе открытых стандартов.

Oracle Designer 6i

Это средство быстрой разработки приложений использует репозиторий компонентов. Последний, в свою очередь, для описания атрибутов и характеристик компонентов использует формат *Extensible Metadata Interchange* (*XMI*).XMI — это стандарт, являющийся подмножеством XML, он позволяет производить открытый обмен компонентами через Web.

Internal Release Application

Internal Release Application — внутреннее приложение Oracle, написанное на языке Java, которое использует файлы формата XML для конфигурации и выдачи информации о состоянии приложения.

Oracle Configuration Developer

Oracle Configuration Developer представляет собой среду моделирования конфигурации для продуктов Oracle Applications Release 11i, которые предоставляют интерфейс типа "схватить-и-перетащить" (drag-and-drop) для разработки моделей продаж. XML здесь используется для форматирования правил конфигурации, которые ограничивают модель в области состояния объекта, определяют количественные параметры моделирования, совместимость продукта, зависимость его работы от разных параметров и пр.

Продукты Oracle, использующие XML для управления контентом и публикации

Управление контентом и публикация информации — основная область применения, в которой технология XML максимально эффективна. В ассортименте Oracle целый ряд продуктов, которые могут читать и записывать документы в XML-формате, вести в них поиск и визуализировать их с помощью языка XSL.

Файловая система Oracle Internet File System

Oracle Internet File System (iFS) — это набор Java API-интерфейсов, предоставляющих разработчикам, которые хотят хранить документы в Oracle8i/9i универсальным способом, интерфейсы на уровне файлов. Система iFS используется не только для поддержки самых разных типов файлов, в том числе XML, но и обеспечивает их целостное хранение в LOB-объектах, при этом данные можно синтаксически анализировать и разбирать на части для хранения в распределенной среде по одной из схем. Распределение данных в такой ситуации определяется в файле типа .typ.

Oracle Text

Oracle Text — это средство поиска текста, входящее в состав СУБД Oracle8i. Его поддержка технологии XML включает возможность индексирования XML-документов, хранимых в больших символьных *объектах CLOB,* и возможность поиска местонахождения элементов и атрибутов через SQL-запросы.

Сервер приложений Oracle9i Application Server Wireless Edition

Oracle9iAS WE — это платформа, созданная для предоставления информационных служб самым разным проводным и беспроводным устройствам, в -том числе сотовым телефонам, карманным компьютерам и пейджерам. Используя XML для ввода и вывода информации, программа Portal-to-Go включает специфичные для каждого устройства сетевые экраны, которые передают XML-файлы, преобразованные с помощью языка XSL в формат, понятный каждому конкретному устройству.

Oracle Discoverer 3i

Продукт, полностью переделанный с целью использования технологии XML не только для передачи данных, но и для инкапсуляции поведения компонентов и их атрибутов. Вместе с языком XSL программа Discoverer предоставляет легко настраиваемые Web-отчеты и пользовательские интерфейсы.

Обзор использования XML-компонентов Oracle

XML-компоненты Oracle используются не только внутри компании, как это указывалось ранее, но и во многих приложениях, требующих высокой степени мобильности данных между приложениями и предприятиями. Ниже приводится список и краткое описание нескольких таких приложений. Не забывайте о них, читая эту книгу.

Создание и публикация документов

Допустим, что некая компания имеет несколько репозиториев документов с размеченными текстовыми фрагментами, написанными на языках XML и SGML, и заинтересована в динамичной публикации сложных документов. Она может осуществить это с помощью таблиц стилей XML для сборки фрагментов и электронной доставки составных документов пользователям.

Служба доставки персонализированной информации

К примеру, компания получает информацию из нескольких источников и использует технологию XSL для ее нормализации и хранения в базе данных. Это хранилище информации затем используется для наполнения одного или нескольких Web-сайтов или порталов, которые получают запросы по протоколу http от самых разных клиентов, имеющих устройства с проводными или беспроводными средствами связи. Таблицы стилей XSL вместе с сервлетом XSQL Servlet будут обеспечивать динамичную визуализацию информации для каждого конкретного устройства, с которого пришел запрос.

Легко настраиваемые приложения доставки информации

Допустим, создано приложение, предназначенное для доставки информации и интерактивного общения с тонким клиентом. Информация запрашивается из базы данных вместе с компонентами пользовательского интерфейса, и все это динамически визуализируется с помощью одной или нескольких таблиц стилей XSL для конкретного клиентского приложения. Хранилище в данном случае представляет собой реляционные данные, представленные в XML-формате, а также XML-файл, хранимый в большом LOB-объекте.

XML-корзина в приложениях электронной коммерции

Web-сайт электронной коммерции собирает заказы от клиентов в XML-файлы и доставляет эту информацию для обработки в одну или несколько баз данных поставщиков товаров. Здесь XSL используется для преобразования и распределения содержимого корзины, где находятся заказанные клиентом товары, для отправки этой информации соответствующим поставщикам. В данном случае информация хранится в реляционной базе данных и материализуется в формате XML.

Обмен сообщениями между компаниями через Интернет

Существует целый ряд приложений типа клиент-сервер или сервер-сервер, где данные о ресурсах или товарах хранятся в репозитории, доступ к которым имеют сразу несколько предприятий. Выпуск какого-то товара открывает доступ к XML-сообщению, которое затем преобразуются с помощью языка XSL в форматы, понятные сразу нескольким клиентам. И наоборот, приобретение какого-то товара одним из клиентов генерирует рассылку XML-сообщения другим клиентам с информацией об удалении этого товара из базы данных. Такие сообщения хранятся в LOB-объектах. Сама информация содержится в реляционной базе данных и материализуется в формате XML.

Интеграция приложений с помощью XML-сообщений

Для взаимодействия и совместного использования данных с целью интеграции бизнесов или бизнес-процессов нужно сразу несколько приложений. Здесь технология XML используется в качестве полезной нагрузки сообщения, преобразованного с помощью команд языка XSL, помещенного в конверт и отправленного. Эти XML-сообщения хранятся в LOB-объектах.

Пример и приложение

Прикладные программы, которые используют преимущества XML для упрощения процесса ведения электронной коммерции, можно написать намного быстрее, если они используют инструментарий Oracle *XML Developer's Kit* (*XDK*). Вместо того чтобы заниматься кодированием для синтаксического анализа XML-файлов, преобразования их в соответствии с таблицами стилей, извлечения информации из таблиц баз данных и преобразования ее в XML-формат, разработчики приложения с помощью открытых API-интерфейсов, поставляемых в инструментарии XDK, могут

использовать уже имеющиеся функциональные возможности и быстро создавать прикладные приложения, которые делают то, что нужно заказчикам. Таким образом, если приходящие к клиенту с Web-сайтов данные должны инкапсулироваться в форму XML и передаваться на какой-либо другой Web-сайт, разработчики с помощью инструментария XDK могут быстро создать полезную прикладную программу, осуществляющую все вышеописанное. Примером такой программы может быть уже упоминавшееся в этой главе приложение, которое позволяет покупать книги через Интернет.

Например, владельцу Web-сайта Amazingly Enjoyable Books ("Потрясающе приятные книги") нужна прикладная программа, которая захватывает данные, вводимые покупателем на этом сайте, сохраняет их в XML-формате, а сами XML-файлы — в БД, а потом выдает эти данные в надлежащей форме. Программисту нужно написать приложение за один день, поскольку владелец Web-сайта пообещал заплатить втрое больше, если приложение будет запущено уже сегодня. Помня о том, что на Web-сайте Oracle Technology Network XML есть целый ряд полезных XML-компонентов и утилит, можно отправиться по адресу http://technet.oracle.com/tech/xml и загрузить такие компоненты, как Java XML/XSLT Parser и XSQL Servlet. Познакомившись с имеющимися там примерами программ, нетрудно обнаружить, что заказанная прикладная программа будет написана очень быстро, так как там есть все нужные блоки.

Если объем данных, вводимых клиентами на Web-сайте, невелик, работа еще проще, и можно немедленно выдать DTD, управляющий структурой XML-документа, который хранит данные клиента:

```
<!-- DTD book customer list may have number of book customers -->
<!ELEMENT bookcustomerlist (bookcustomer)*>
<!-- Each book customer has a name, address,
     1 or more book orders, etc. -->
<!ELEMENT bookcustomer (customer*, book+, totalsales)>
<!ELEMENT customer (firstname, lastname, address, city, state, country, zip)>
<!ELEMENT firstname (#PCDATA)>
<!ELEMENT lastname (#PCDATA)>
<!ELEMENT address (#PCDATA)>
<!ELEMENT city (#PCDATA)>
<!ELEMENT State (#PCDATA)>
<!ELEMENT country (#PCDATA)>
<!ELEMENT zip (#PCDATA)>
<!ELEMENT book (title, author+, ISBN, publisher, publishyear, price,
               purchase date)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT author (firstname, lastname)>
<!ELEMENT firstname (#PCDATA)>
<!ELEMENT lastname (#PCDATA)>
```

```
<! ELEMENT ISBN (#PCDATA)>
<!ELEMENT publisher (#PCDATA)>
<!ELEMENT publishyear (#PCDATA)>
<!ELEMENT price (#PCDATA)>
<!ELEMENT purchasedate (#PCDATA)>
<!ELEMENT totalsales (#PCDATA)>
```

Прикладная программа начинается с захвата данных, вводимых клиентами, затем эти данные в XML-формате отправляются в память (или во внешний файл), которая соответствует заданному типу данных DTD. Например, пользовательский интерфейс может иметь вид формы, показанной на рис. 1.2.

Для хранения и поиска информации в базе данных на сервере можно использовать SQL-оператор. Тогда сервлет Oracle XSQL Servlet сможет обработать XML-файл, а также, возможно, XSL-файл и составленный SQL-оператор. Потом сервлет XSQL Servlet, работающий на Web-сервере, устанавливает соединение с базой данных и выполняет SQL-оператор. Затем из таблиц базы данных возвращаются строковые данные, они преобразуются в XML-формат, к ним применяется XSL-таблица стилей, и они отправляются с Web-сервера на браузер клиента, если ему нужна эта информация. Если же для ввода данных используется только DML-операция, не нужно никакой операции поиска информации.

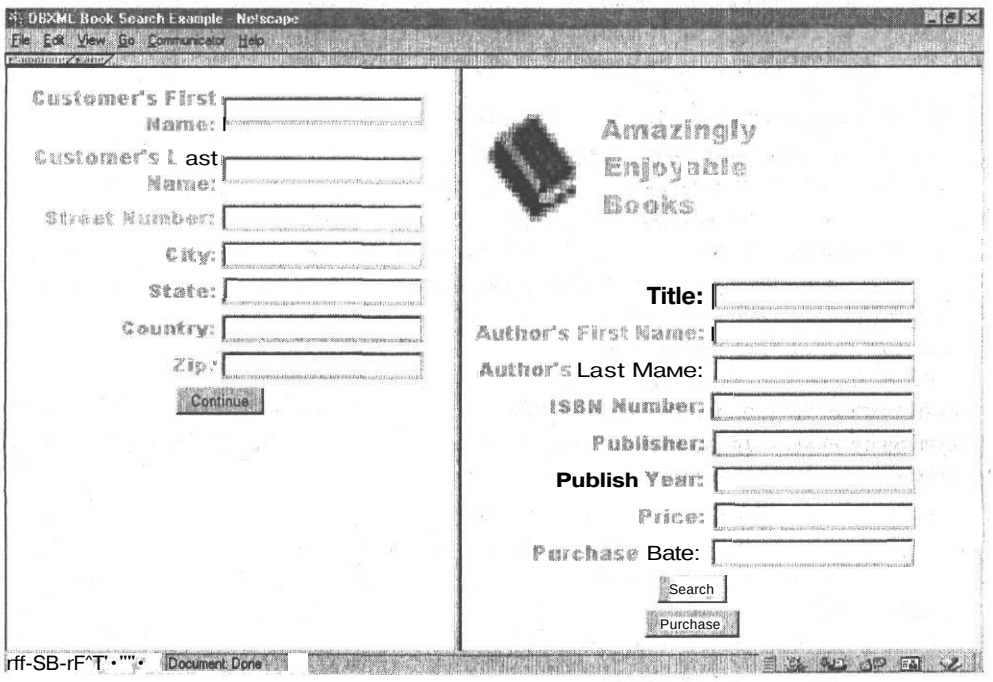


Рис. **1.2.** *Пример формы поиска книги*

В качестве примера приведем следующий XML-файл, который осуществляет поиск информации в таблицах базы данных, преобразование ее в XML-формат и отправку в другое приложение или на Web-сайт:

```
<?xml version="1.0"?>
<!DOCTYPE bookcustomerlist SYSTEM "bookcustomer.dtd">
<bookcustomerlist>
   <bookcustomer>
   <customer>
      <firstname>Juan</firstname>
      <lastname>Smith</lastname>
      <address>1 Oracle Parkway</address>
      <city>Redwood Shores</city>
      <state>California</state>
      <country>USA</country>
      <zip>94065</zip>
   </customer>
      <book>
         <title>Men are from Mars</title>
         <author>
            <firstname>James</firstname>
            <lastname>Collins</lastname>
         </author>
         <ISBN>99999</ISBN>
         <publisher>Oracle Press</publisher>
         <publishyear>2000</publishyear>
         <price>1.00</price>
         <purchasedate>January 1, 2000</purchasedate>
      </book>
         <totalsales>1000.00</totalsales>
   </bookcustomer>
</bookcustomerlist>
```

Этот тип приложения будет рассматриваться в этой книге неоднократно, обрастая новыми деталями. Он представляет собой область, где технология XML максимально влияет на разработку приложений.

Глава 2

Технологии XML CORE компании Oracle

ехнология XML и связанные с нею технологии являются открытым стандартом, и очень важно, что компоненты, библиотеки и приложения, построенные на базе этих стандартов, потенциально обладают высокой способностью к взаимодействию с другими продуктами и к повторному использованию. Это было основным принципом в решении Oracle строить свои собственные инфраструктурные компоненты и библиотеки. Данная функциональная возможность XML инкапсулирована в инструментарии разработчиков *XML Developer's Kits* (*XDK*)для четырех поддерживаемых языков — Java, C, C++ и PL/SQL.

Для эффективной работы с XML-документами придется иметь дело с несколькими компонентами и утилитами. Разработчик должен провести синтаксический анализ XML-документа, проверить его соответствие DTD или XML Schema, преобразовать XML-документ с помощью таблицы стилей, генерировать XML-документы из данных и т. д. В следующих параграфах обсуждаются программы-анализаторы синтаксиса, процессоры, генераторы, программы просмотра и утилиты, которые составляют инструментарий разработчика XDK, и приведены примеры их использования.

Анализатор XML Parser for Java V2

XML-анализатор от Oracle для языка Java позволяет Java-программистам легко расширять возможности уже имеющихся у них Java-приложений, чтобы они поддерживали технологию XML. Эта программа обрабатывает XML-документы и предоставляет доступ к информации, содержащейся в них, через целый ряд дружественных к пользователю API-интерфейсов. Этот XML-анализатор полностью поддерживает оба стандарта: и древовидное объектное представление документа *Document Object Model (DOM),* и построенный на базе событий *Simple API for XML (SAX).* Кроме того, он имеет встроенный XSLT-процессор, который очень легко преобразует XML-документы из одного формата в другой. Этот анализатор можно использовать в любой среде, поддерживающей инструментарий JDK версии 1.1.*x* и более новой, его можно также запускать в Oracle8i и Oracle9i JServer. Анализатор XML-синтаксиса полностью интернационализирован и корректно использует все наборы символов, поддерживаемых языком Java, а также целый ряд других. Такая поддержка означает, что анализатор будет выдавать сообщения об ошибках практически на каждом языке из тех, что поддерживаются в Oracle8i и Oracle9i. Это делает его неоценимым средством при написании XML-приложений для неанглоязычных пользователей.

Поддержка SAX

Часто приложения, требующие поддержки только стандарта SAX (Level 1 и 2), не хотят обременять себя анализатором, всегда строящим в памяти полноценное древовидное представление документа DOM. Высокопроизводительный анализатор синтаксиса Oracle XML SAX, построенный на базе событий и работающий в реальном времени, благодаря которому приложения могут использовать полную мощь SAX-модели для анализа синтаксиса очень больших документов, не расходуя чрезмерное количество ресурсов на память.

Следующий фрагмент кода демонстрирует использование API-интерфейсов SAX для извлечения полезной информации из XML-документа:

```
// This example demonstrates a simple use of the SAXParser.
// An XML file is parsed and some information is printed out.

import org.xml.sax.*;
import java.io.*;
import java.net.*;
import oracle.xml.parser.v2.*;

public class SAXHandler extends HandlerBase
{  public static void main (String[] argv)
   {  try
      {  // Get an instance of the parser
         Parser parser = new SAXParser ( ) ;

         // Create a SAX event handler and register it with the parser
         SAXHandler handler = new SAXHandler ( ) ;
         parser.setDocumentHandler(handler);

         // Convert file to InputSource and parse
         InputSource xmldoc = new InputSource (newFileInputStream(argv[0] ) ) ;
         parser.parse (xmldoc);
      }
      catch (Exception e)
      { System.out.println(e.toString() ); }
   }

   ////////////////////////////////////////////////////////////////////
   // Sample implementation of DocumentHandler interface.
   ////////////////////////////////////////////////////////////////////

   public void startElement (Stringname, AttributeList atts)
          throws SAXException
   { System.out.println("StartElement:"+name);
```

```
        for (int i=0;i<atts.getLength() ;i++)
        {  String aname = atts.getName(i) ;
           String type = atts.getType(i) ;
           String value = atts.getValue(i) ;

           System.out.println(" "+aname+"("+type+")"+"="+value);
        }
    }

    public void characters (char[] cbuf, int start, int len)
    {System.out.print ( "Characters: " ) ;
       System.out.println(new String (cbuf,start,len)) ;
    }
}
```

Чтобы использовать поддержку SAX, имеющуюся в этой программе-анализаторе, следует применить для анализа синтаксиса XML-документа класс **SAXParser.** Поэтому первое, что нужно сделать, это представить экземпляр этого класса:

```
Parser parser = new SAXParser ( ) ;
```

Затем надо зарегистрировать программу обработки SAX-событий в анализаторе, чтобы активизировать методы, соответственно происходящим событиям. Так как не все события могут представлять интерес, убедитесь, что зарегистрированная программа обработки распространяется на класс **org.xml.sax.HandlerBase**. Этот класс определяет для обрабатываемых событий поведение по умолчанию (обычно при этом не происходит никакой обработки). Затем можно заменить эти методы для событий, которые действительно интересуют. В предыдущем примере предполагалось, что интересующие события являются подмножеством событий, определяемых интерфейсом **org.xml.sax.DocumentHandler, т. е. startElement** и characters. Можно показать, что это самые важные генерируемые SAX-события, так как XML-документы обычно состоят из разметки и текста. Эту *программу-обработчик* можно зарегистрировать в анализаторе **SAXParser** с помощью простого вызова API-интерфейса:

```
parser.setDocumentHandler(handler);
```

Событие **startElement** происходит всякий раз, когда **SAXParser** встречает в XML-документе новый элемент. Когда такое событие происходит, вы можете распечатать имя этого элемента и его атрибуты:

```
public void startElement (String name, AttributeList atts)
        throws SAXException
{
    ...
}
```

Событие characters происходит всякий раз, когда анализатор SAXParser встречает немаркированный текст. Этот текст часто представляет собой значение элемента и его можно получить с помощью перехвата события:

```
public void characters(char[] cbuf, int start, int len)
{
    ...
}
```

После того как обработчик зарегистрирован, остается провести синтаксический анализ XML-документа с помощью **SAXParser:**

```
parser.parse(xmldoc);
```

Входной XML-документ может содержать список данных книги, например:

```
<booklist>
   <book isbn="1234-123456-1234">
      <title>C Programming Language</title>
      <author>Kernighan and Ritchie</author>
      <publisher>IEEE</publisher>
      <price>7.99</price>
   </book>
   <book isbn="1230-23498-2349879">
      <title>Emperor's New Mind</title>
      <author>Roger Penrose</author>
      <publisher>Oxford Publishing Company</publisher>
      <price>15.99</price>
   </book>
</booklist>
```

Тогда можно получить следующие выходные данные:

```
StartElement:booklist
StartElement:book
   isbn(CDATA)=1234-123456-1234
StartElement:title
Characters:C Programming Language
StartElement:author
Characters :Kernighan and Ritchie
StartElement:publisher
Characters:IEEE
StartElement:price
Characters:7.99
StartElement:book
   isbn(CDATA)=1230-23498-2349879
```

```
StartElement:title
Characters:Emperor's New Mind
StartElement:author
Characters:Roger Penrose
StartElement:publisher
Characters:Oxford Publishing Company
StartElement :price
Characters:15.99
```

Реализация API-интерфейса SAX Level 2 происходит главным образом в форме поддержки пространства имен Namespace и составления запросов или установки функций или свойств в программе-анализаторе. Благодаря поддержке пространства имен Namespace названия элемента и его атрибута могут теперь возвращать URI-индикатор Namespace, который следует за локальным именем, например **<foo:bar xmlns:foo="http://www.oracle.com/"/>, где http://www.oracle.com/** — это URI-индикатор Namespace, a bar — локальное имя. Кроме того, можно возвратить и уточненное имя (или **qName**) **foo:bar**. Без поддержки Namespace имена элемента и атрибута будут возвращать только локальное имя. Интерфейсы SAX Level 2, на которые влияет поддержка Namespace, это **XMLReader, Attributes и ContentHandler.** Пример поддержки SAX 2 Namespace, за которым следует код методов вызова **StartElement и endElement** в интерфейсе **ContentHandler,** может выглядеть следующим образом:

```
// This example demonstrates how SAX Level 2 Namespace
// support, followed by how the callback
// methods StartElement and endElement can be used.

import java.io.*;
import java.net.URL;
import java.net.MalformedURLException;
import org.xml.sax.*;
import org.xml.sax.helpers.*;
import oracle.xml.parser.v2.SAXParser;

public class SAX2Namespace {
   static public void main(String[] args){
      String fileName;
      // Get the file name
      fileName = args[0];
      try {
         // Create handlers for the parser
         // For all the other interface use the default provided by
         // Handler base
```

```
        DefaultHandler defHandler = new XMLDefaultHandler();
        SAXParser parser = new SAXParser();
        parser.setContentHandler (defHandler);
        parser.setErrorHandler (defHandler);
        parser.setEntityResolver (defHandler);
        parser.setDTDntHandler (defHandler);
        try
        { parser.parse(createURL(fileName)); }
        catch (SAXParseException e)
        { System.err. println(args[0] + ": " + e.getMassage()); }
        catch (SAXException e)
        { System.err.println(args[0] + *: " + e.getMassage()); }
      }
      catch (Exception e)
      { System.err.println(e.toString()); }
   }
   static URL createURL(String fileName) {
        URL url = null;
        try {
             url = new URL(fileName);
        } catch (MalformedURLException ex) {
          try {
               File f = new File(fileName);
               url = f.toURL();
           } catch (MalformedURLException e) {
              System.out.println("Cannot create url for: " + fileName");
              System.exit(0);
          }
        }
      return url;
}
private SAX2Namespace ( ) throws IOException
{ }
}
class XMLDef aultHandler extends DefaultHandler
{ public void XMLDef aultHandler ()
   { }
   public void startElement (String uri, String localName,
                             String qName, Attributes atts)
        throws SAXException
   { System.out .println("ELEMENT Qualified Name:" + qName);
     System.out .println("ELEMENT Local Name :" + localName);
     System.out.println("ELEMENT Namespace :" + uri) ;
```

```
        for (int i=0; i<atts.getLength(); i++)
        {  qName = atts.getQName(i);
           localName = atts.getLocalName(i);
           uri = atts.getURI(i);

           System.out.println(" ATTRIBUTE Qualified Name :" + qName);
           System.out.println(" ATTRIBUTE Local Name :" + localName);
           System.out.println(" ATTRIBUTE Namespace :" + uri);

           // You can get the type and value of the attributes either
           // by index or by Qualified Name.

           String type = atts.getType(qName);
           String value = atts.getValue(qName);

           System.out.println(" ATTRIBUTE Type :" + qName);
           System.out.println(" ATTRIBUTE Value :" + value);

           System.out.println();
        }
     }

     public void endElement (String uri, String localName, String qName)
              throws SAXException
     { System.out.println("ELEMENT QualifiedName: " +qName);
       System.out.println("ELEMENT Local Name :" + localName);
       System.out.println("ELEMENT Namespace :" + uri);
     }
}
```

Для интерфейса SAX Level 2 передаются дополнительные параметры: Namespace URI, локальное имя и **qName.** Остальные интерфейсы SAX Level 2 включают средства запросов и установки функций и свойств в программе-анализаторе, например методы извлечения/установки **getFeature, setFeature,** getProperty, **setProperty,** как это показано в следующем примере:

```
void process (String filename) throws SAXException, IOException
{  URL url =createURL(filename);

   // Validating, Namespace = true, NamespacePrefix = true
   parser.setFeature("http://xml.org/sax/features/validation",true);
   parser.setFeature("http://xml.org/sax/features/namespaces",true);
   parser.setFeature("http://xml.org/sax/features/namespace-prefix",true);

   try
   { parser.parse(url.toString()); }
```

```
    catch (XMLParseException e)
    { System.out.println();
      System.out.println(e);
    }

    // Non-validating, NamespacePrefix = false
    parser.setFeature("http://xml.org/sax/features/validation", true);
    parser.setFeature("http://xml.org/sax/features/namespace-prefix", true);

    try
    { parser.parse(url.toString()); }

    catch (XMLParseException e)
    { System.out.println();
      System.out.println(e);
    }
}
```

Для этого примера кода отметим, что при обработке данных с помощью интерфейсов SAX Level 2 можно управлять поддержкой параметра пространства имен Namespace. По умолчанию префиксы пространства имен имеют значение false (ложь), т. е. сообщение о значении **qName** является необязательным, а описания xmlns атрибутов Namespace не делаются вовсе. Однако в нашем примере пространствам имен и их префиксам присвоено значение true (истина), поэтому элемент:

```
<foo:bar xmlns:foo="http://wwww.oracle.com/" foo1="bar1"
foo:stock="wayout.com"/>,
```

будет иметь Namespace URI-адрес **"http://wwww.oracle.com/"**, локальное имя **"bar1"**, qName будет иметь значение **"foo:bar"**; один атрибут не будет иметь ни Namespace URI-адреса, ни локального имени, а значение **qName** будет **"xmlns:foo"**. Другой атрибут не будет иметь Namespace URI-адреса, у него будет локальное имя, а значение **qName** будет **"fool".** Последний атрибут будет иметь Namespace URI-адрес "http://wwww.oracle.com/", а локальное имя — **"stock".**

Поддержка DOM

Программа-анализатор Oracle XML поддерживает также спецификацию DOM (Level 1 и Level 2 CORE/Events/Traversal/Range). Мощь спецификации DOM состоит в ее способности обеспечивать доступ к древовидному представлению всего XML-документа, которое находится в памяти. С помощью DOM приложения могут выполнять поиск заданных данных в XML-документе, добавление или удаление элементов и атрибутов в XML-документе, а также преобразование DOM для совершенно другого документа. Анализатор синтаксиса Oracle DOM использует внутри себя быстродействующий анализатор SAX, который строит в памяти дерево DOM.

Он предоставляет идентификатор для корня дерева DOM (узел документа), из которого можно получить доступ ко всему дереву. XML-анализатор поставляется вместе с набором классов, которые реализуют API-интерфейсы DOM и расширяют их для выполнения других полезных функций, например печати фрагмента документа, получения информации о пространстве имен Namespace и выбора узлов из древовидного представления, определенного конфигурацией XPath.

В следующем коде продемонстрированы некоторые функциональные возможности DOM в программе-анализаторе XML Parser:

```
// This example demonstrates a simple use of the DOMParser
// An XML file is parsed and some information is printed out.

import java.io.*;
import java.net.*;
import oracle.xml.parser.v2.DOMParser;
import org.w3c.dom.*;
import org.w3c.dom.Node;

// Extensions to DOM Interfaces for Namespace support.
import oracle.xml.parser.v2.XMLElement;
import oracle.xml.parser.v2.XMLAttr;

public class DOMExample
{  public static void main(String[] argv)
   {  try
      {  // Generate a new input stream from given file
         FileInputStream xmldoc = new FileInputStream(argv[0]);

         // Parse the document using DOMParser
         DOMParser parser = new DOMParser();
         parser.parse(xmldoc);

         // Obtain the document.
         Document doc = parser.getDocument();

         // Print some information regarding attributes of elements
         // in the document
         printElementAttributes(doc);
      }
      catch (Exception e)
      {  System.out.println(e.toString());  }
   }
   static void printElementAttributes(Document doc)
   {  NodeList nl = doc.getElementsByTagName("*");
      Element e;
      XMLAttr nsAttr;
```

```
        String attrname, attrval, attrqname;
        NamedNodeMap nnm;

        for (int j=0; j < nl.getLength() ; j++)
        {  e = (Element) nl.item(j);
           System.out.println(e.getTagName() + ":");

           nnm = e.getAttributes();

           if (nnm != null)
           {  for (int i=0; i < nnm.getLength(); i++)
              {  nsAttr = (XMLAttr) nnm.item(i) ;

                 // Use the methods getQualifiedName(), getLocalName() ,
                 // getNamespace() and getExpandedName() in NSName
                 //  interface to get Namespace information.

                 attrname = nsAttr.getExpandedName();
                 attrqname = nsAttr.getQualifiedName();
                 attrval = nsAttr.getNodeValue();

                 System.out.println(" " + attrqname + "(" + attrname +
                    ")" + " = " +attrval);
              }
           }
           System.out.println();
        }
     }
  }
```

API-интерфейсы DOM в отличие от API-интерфейсов SAX могут использоваться только после того, как проведен полный синтаксический анализ XML-документа. Недостатком такого подхода является то, что большие XML-документы могут занимать очень много памяти. В конечном счете это снижает производительность работы приложения. Если же учитывать только функциональные возможности, то DOM представляет собой, безусловно, более мощный API-интерфейс. Первое, что нужно сделать перед использованием любого из API-интерфейсов DOM — провести синтаксический анализ документа с помощью новой копии **DOMParser**:

```
// Parse the document using DOMParser
DOMParser parser = new DOMParser() ;
parser.parse(xmldoc);
```

Затем нужно потребовать от программы-анализатора возврата обработчика к корню объектной модели документа Document Object Model, которая построена в памяти:

```
// Obtain the document,
Document doc = parser.getDocument();
```

С помощью предыдущей программы-обработчика можно получить доступ ко всем частям XML-документа, синтаксический анализ которого только что произведен. Класс **DOMExample** предполагает, что требуется доступ к элементам документа и их атрибутам. Для этого нужно получить список всех элементов документа, используя DOM-метод **setEIementsByTagName.** С его помощью можно рекурсивно получить все элементы, соответствующие заданному имени тега на определенном уровне. Он также поддерживает специальное имя тега "*", соответствующее любому тегу. С учетом вышесказанного нужно активизировать этот метод на верхнем уровне документа, применив программу обработки к корню:

```
NodeList nl = doc.setElementsByTagName("*");
```

Это обращение генерирует список всех элементов документа. Каждый из них содержит информацию о его атрибутах. Для того чтобы получить эту информацию, нужно пройти по всему списку:

```
len = nl.getLength();
for (int j=0; j < len; j++)
{ e = (Element) nl.item(j);
  ...
}
```

Чтобы получить атрибуты каждого элемента в этом цикле, можно использовать DOM-метод, который называется **getAttributes**. Этот метод генерирует специальный вид DOM-списка **NameNodeMap**. Получив этот список, проведите его обход, чтобы получить информацию об атрибутах:

```
for (int i=0; i < nnm.getLength(); i++)
 {  nsAttr = (XMLAttr) nnm.item(i);

 // Use the methods getQualifiedName() and getExpandedName() in NSName
 // interface to get Namespace information.

    attrname = nsAttr.getExpandedName();
    attrqname = nsAttr.getQualifiedName();
    attrval=nsAttr.getNodeValue();

    System.out.println(" * + attrqname + •(* + attrname +
                       ")" + " = " +attrval);
 }
```

В предыдущем фрагменте кода для получения дополнительной информации об атрибутах каждого элемента используется поддержка пространства имен Namespace программы-анализатора XML Parser. Этот вид кода может оказаться полезным, если XML-документ, требующий анализа, имеет элементы со многими атрибутами, принадлежащими к различным пространствам имен. Пусть, например, XML-документ **book** из предыдущего параграфа выглядит следующим образом:

```
<booklist xmlns:osborne="http://www.osborne.com"
          xmlns:bookguild="http://www.bookguild.com"
          xmlns:dollars="http://www.currency.org/dollars">
   <book osborne:isbn="1234-123456-1234" title="C Programming Language"
         author="Kernighan and Ritchie" bookguild:publisher="IEEE"
         dollars:price="7.99"/>
   <book osborne:isbn="1230-23498-2349879" title="Emperor's New Mind"
         author="Roger Penrose" bookguild:publisher="Oxford Publishing Company"
         dollars:price="15.99"/>
</booklist>
```

Тогда на выходе получается:

```
xmlns:osborne(http://www.w3.org/2000/xmlnls/:osborne)=http://www.osbo-rne.com
xmlns:bookguild(http://www.w3.org/2000/xmlnls/:bookguild) =
http://www.bookguild.com
xmlns:dollars(http://www.w3.org/2000/xmlnls/:dollars) =
http://www.currency.org/dollars

book:
 osborne:isbn(http://www.osborne.com:isbn) = 1234-123456-1234
 title(title) = C Programming Language
 author(author)= Kernighan and Ritchie
 bookguild:publisher(http://www.bookguild.com:publisher) = IEEE
 dollars:price(http://www.currency.org/dollars:price) = 7.99

book:
 osborne:isbn(http://www.osborne.com:isbn) = 1230-23498-2349879
 title(title) = Emperor's New Mind
 author(author) = Roger Penrose
 bookguild:publisher(http://www.bookguild.com:publisher) =
           Oxford Publishing Company
 dollars:price(http://www.currency.org/dollars:price) = 15.99
```

В числе функциональных возможностей DOM Level 2 есть методы, создающие итераторы и программы обхода дерева для определения узла и его дочернего узла в структуре документа (заданный порядок обхода соответствует порядку, в котором в текстовом представлении документа располагаются начальные теги); объекты, используемые для навигации по дереву документа или его поддереву с помощью представления документа, определенного его флагами **whatToShpw** и фильтром (если таковой существует), поэтому любой метод, выполняющий навигацию с помощью программы **TreeWalker,** будет автоматически поддерживать любое представление, определенное **TreeWalker;** методы итератора, используемые для выполнения шага через несколько узлов или через поддерево документа, управляемые отдельным узлом Node или результатами запроса или любым другим набором узлов; и объекты для отфильтровывания узлов.

Пример такого фрагмента кода показан ниже:

```
// This filter accepts everything
NodeFilter n1 =new nf1();
//Node iterator doesn't allow expansion of entity references
NodeIterator ni =
            doc.createNodeIterator(elems[0],NodeFilter.SHOW_ALL.n1,false);
// Move forward
XMLNode nn =(XMLNode)ni.nextNode();
while (nn !=null)
{ System.out.ptintln(nn.getNodeName() + • " + nn.getNodeValue());
   nn = (XMLNode)ni.nextNode();
}
// Move backward
nn =(XMLNode)ni.previousNode();
while (nn !=null)
{ System.out.ptintln(nn.getNodeName() + * " + nn.getNodeValue());
   nn = (XMLNode)ni.previousNode();
}

// Node iterator allows expansion of entity references
ni = doc.createNodeIterator(elems[0],NodeFilter.SHOW_ALL.n1,true);
// Move forward
nn =(XMLNode)ni.nextNode();
while (nn !=null)
{ System.out.ptintln(nn.getNodeName() + * • + nn.getNodeValue());
   nn = (XMLNode)ni.nextNode();
}
// Move backward
nn =(XMLNode)ni.previousNode();
while (nn !=null)
{ System.out.ptintln(nn.getNodeName() + * * + nn.getNodeValue());
   nn = (XMLNode)ni.previousNode();
}

// This filter doesn't accept expansion of entity references
NodeFilter n2 = new nf2();

//Node iterator allow expansion of entity references
ni = doc.createNodeIterator(elems[0],NodeFilter.SHOW_ALL.n2,true);
// Move forward
nn =(XMLNode)ni.nextNode();
while (nn != null)
{ System.out.ptintln(nn.getNodeName() + " " + nn.getNodeValue());
   nn = (XMLNode)ni.nextNode();  }
```

```
. // Move backward
 nn = (XMLNode)ni.previousNode();
 while (nn != null)
 { System.out.ptintln(nn.getNodeName() + * • +nn.getNodeValue());
   nn = (XMLNode)ni.previousNode();
 }

 // After detaching, all node iterator methods throw an exception
 ni.detach();
 try
 { nn = (XMLNode)ni.nextNode(); }
 catch (DOMException e)
 { System.out.ptintln(e.getMessage()); }
 try
 { nn = (XMLNode)ni.previousNode(); }
 catch(DOMException e)
 { System.out.ptintln(e.getMessage()); }
 // Treewalker allows expansion of entity references
 Treewalker tw = doc.createTreeWalker(elems[0],NodeFilter.SHOW_ALL.n1,true);
 nn = (XMLNode)tw.getRoot();
 // Traverse in document order
 while (nn != null)
 { System.out.ptintln(nn.getNodeName() + * * +nn.getNodeValue());
   nn = (XMLNode)tw.nextNode();
 }

 tw = doc.createTreeWalker(elems[0],NodeFilter.SHOW_ALL.n1,true);
 nn = (XMLNode) tw.getRoot();
 // Traverse the depth left
 while (nn != null)
 { System.out.ptintln(nn.getNodeName() + " • +nn.getNodeValue());
   nn = (XMLNode)tw.firstChild();
 }
 tw = doc.createTreeWalker(elems[0],NodeFilter.SHOW_ALL.n2,true);
 nn = (XMLNode)tw.getRoot();
 // Traverse in document order
 while (nn != null)
 { System.out.ptintln(nn.getNodeName() + " " +nn.getNodeValue());
   nn = (XMLNode)tw.nextNode();
 }
 tw = doc.createTreeWalker(elems[0],NodeFilter.SHOW_ALL.n2,true);
 nn = (XMLNode) tw.getRoot();
```

```
// Traverse the depth right
while (nn != null)
{  System.out.ptintln(nn.getNodeName() + " " + nn.getNodeValue());
   nn = (XMLNode)tw.lastChild();
}

...
class nf1 implements NodeFilter
{  public short acceptNode (Node node)
   {  return FILTER_ACCEPT;  }
}
class nf2 implements NodeFilter
{  public short acceptNode (Node node)
   {  short Type = node.getNodeType();
      if ((type == Node.ELEMENT_NODE) || (type == Node.ATTRIBUTE_NODE))
         return FILTER_ACCEPT; '
      if ((type == Node.ENTITY_REFERENCE_NODE))
         return FILTER_REJECT;
      return FILTER_SKIP;
   }
}
```

Поддержка XSLT

С помощью встроенного XSLT-процессора можно преобразовать XML-документ в другой XML-документ, в документ формата HTML или в документы других текстовых форматов. Процессор можно за'пускать программным образом (с помощью имеющихся API-интерфейсов) или из командной строки (с помощью утилиты **oraxsl).** В качестве входной информации процессор использует XML-документ (его нужно преобразовать) и таблицу стилей XSLT, которая будет действовать на этот документ.

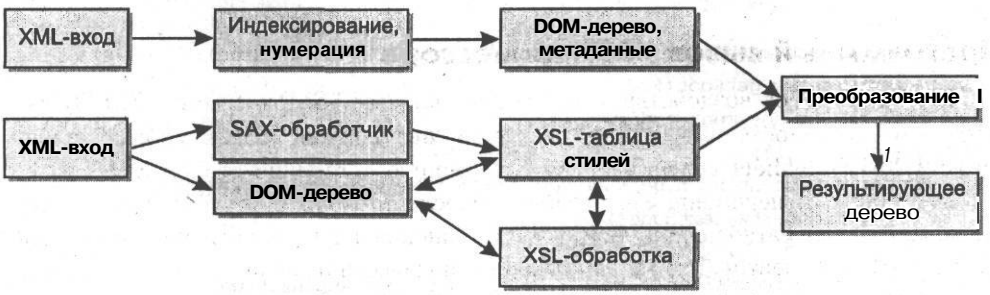


Рис. 2.1. *Архитектура Java XSLT-процессора*

Процессор выполняет преобразование, определяемое таблицей стилей XSLT, и генерирует либо результирующее древовидное представление DOM, либо текстовый выходной поток. Архитектура XSLT-процессора показана на диаграмме на рис. 2.1.

XSLT-процессор оперирует с двумя входами, это — XML-документ, который нужно преобразовывать, и XSLT-таблица стилей, используемая для этого преобразования. Если требуется согласование данных в соответствии с заданными схемами, процессор вызывает средство XPath. Оно часто требуется для прохода XML DOM-дерева с целью поиска узлов, которые передаются обратно в XSLT-процессор. Всякий раз, когда XSLT-процессору нужно генерировать результирующий узел, он генерирует специальное XSLT-событие. Это событие обрабатывается обработчиком XSLT-событий, функционирующим как агент промежуточного слоя с кэшем, который ждет последующих событий, влияющих на тот же самый результирующий узел. В качестве простого примера можно привести ситуацию, когда результирующим узлом является **XMLElement**. XSLT-процессором могут генерироваться несколько XSLT-событий, из которых одно создает элемент, а остальные описывают его атрибуты. Когда обработчик XSLT-событий получает полную информацию об узле, он генерирует соответствующее SAX-событие, которое затем будет обрабатываться зарегистрированным SAX-обработчиком.

В настоящее время процессор XSLT for Java поддерживает два механизма вывода: DOM-дерево и текстовый выходной поток. Оба механизма можно вызвать с помощью соответствующих API-обращений, сделанных с использованием класса XSLProcessor. Если DOM-дерево строится путем вызова API-интерфейса, оно регистрируется для процессора XSLT как обработчик SAX-событий. Аналогично, если API-интерфейс вызывается для получения текстового потока, то в качестве обработчика SAX-событий регистрируется механизм **OutputWriter**. Преимущество такой архитектуры в том, что DOM-дерево будет строиться только по требованию. Если же в XSLT-приложении нужно просто получить текстовый выход (например, HTML-файл), можно использовать механизм OutputWriter, который требует для выполнения этого процесса гораздо меньше памяти. Следует помнить, что если таблица стилей XSLT содержит инструкции **xsl:output**, то для корректной их интерпретации нужно использовать OutputWriter.

Программный вызов XSLT-процессора

XSLT-процессор воздействует на два Java-класса: XSLProcessor и **XSLStylesheet**, которые необходимо использовать для выполнения XSL-преобразования. Объект XSLStylesheet содержит всю информацию о таблице стилей XSLT, т. е. шаблоны, ключи, переменные и атрибуты. После того как этот объект построен, его можно многократно использовать для выполнения аналогичного преобразования других XML-документов. Его также можно периодически подправлять, устанавливая извне подходящие значения параметров таблицы стилей.

Нижеследующий пример кода демонстрирует использование API-интерфейсов XSLT:

```
import java.util.*;
import java.io.*;
import java.net.*;
import org.w3c.dom.*;
import oracle.xml.parser.v2.*;

/**
 * This is a simple example of how to use the XSL processing
 * capabilities of the Oracle XML Parser V2.0. An input XML document
 * is transformed using a given input stylesheet
 */

public class XSLExample
{  public static void main (String args[]) throws Exception
   {  DOMParser parser;
      XMLDocument xmldoc, xsldoc, out;
      FileInputStream xmlstream, xslstream;

      try
      {  // Create an instance of the DOMParser
         parser = new DOMParser();
         parser.setPreserveWhitespace(true);

         // parse input XML file
         xmlstream = new FileInputStream(args[0]);
         parser.parse(xmlstream);
         xmldoc = parser.getDocument();

         // parse input XSL file
         xslstream = new FileInputStream(args[1]);
         parser.parse(xslstream);
         xsldoc = parser.getDocument();

         // instantiate a stylesheet
         XSLStylesheet xsl = new XSLStylesheet(xsldoc, null);

         // Apply stylesheet
         XSLProcessor processor = new XSLProcessor();
         XMLDocumentFragment result = processor.processXSL(xsl, xmldoc);

         // print the transformed document
         result.print(System.out);
      }
```

```
        catch (Exception e)
        { e.printStackTrace(); }
    }
}
```

Приведенный здесь пример очень прост и нагляден. В качестве входных данных взят XML-файл, к нему применяется XSL-таблица стилей. Прежде всего следует провести синтаксический анализ полученного XML-файла с помощью анализатора **DOMParser** и получить корни соответствующих DOM-деревьев:

```
// Create an instance of the DOMParser
    parser = new DOMParser ( ) ;
    parser.setPreserveWhritespace(true);

    // parse input XML file
    xmlstream = new FileInputStream(args [0]) ;
    parser.parse(xmlstream);
    xmldoc=parser.getDocument ( ) ;

    // parse input XSL file
    xslstream = new FileInputStream(args[1] ) ;
    parser.parse(xslstream);
    xsldoc = parser.getDocument ( ) ;
```

Следует отметить, что программа-анализатор сохраняет все пробелы (по умолчанию эта опция отключена). Это очень важно, поскольку позволяет правилам, существующим в XSLT относительно пробелов, определять, как обрабатывать эти пробелы. На следующем этапе происходит сборка объекта таблицы стилей:

```
//instantiate a stylesheet
XSLStylesheet xsl = new XSLStylesheet (xsldoc, null);
```

В этом очень простом примере предполагается, что входная таблица стилей не имеет никакого отношения к внешним таблицам стилей, а также к другим внешним объектам. Здесь в конструктор XSLStylesheet передается только второй аргумент null. В противном случае придется создавать URL-адрес, чтобы сохранить в таблице стилей контрольную точку для внешних ссылок.

Следующий шаг — создание нового класса XSLProcessor и использование его для применения таблицы стилей к входному XML-документу:

```
// Apply stylesheet
XSLProcessor processor = new XSLProcessor ( ) ;
DocumentFragment result = processor.processXSL (xsl, xml) ;
```

Преобразованный выход сейчас доступен в качестве фрагмента документа, над которым в дальнейшем можно производить разные манипуляции с помощью обычных DOM API-интерфейсов. Для простоты приведем пример распечатки результатов:

```
        // print the transformed document
        result.print(System.out);
```

А теперь простая, но мощная тождественная таблица стилей:

```
<?xml version="1.0"?>
<!-- Identity transformation -->
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
   <xsl:template match="*|@*|comment()|processing-instruction()|text()">
      <xsl:copy>
         <xsl:apply-templates select="*|@*|comment()|processing-
         instruction()|text()"/>
      </xsl:copy>
   </xsl:template>
</xsl:stylesheet>
```

Применив эту таблицу стилей к любому XSL-документу, можно получить тот же самый документ. Можно протестировать вышеприведенный пример, применив его к файлу, содержащему тождественную таблицу стилей и (допустим) пример **booklist.** Результат окажется такой, как и ожидалось:

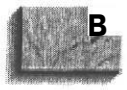
```
<booklist>
   <book isbn="1234-123456-1234">
      <title>C Programming Language</title>
      <author>Kernighan and Ritchie</author>
      <publisher>IEEE</publisher>
      <price>7.99</price>
   </book>
   <book  isbn="1230-23498-2349879">
      <title>Emperor's New Mind</title>
      <author>Roger Penrose</author>
      <publisher>Oxford Publishing Company</publisher>
      <price>15.99</price>
   </book>
</booklist>
```

Вызов процессора XSLT Processor из командной строки

XSLT-процессор поставляется вместе с утилитой командной строки **oraxsl.** Ее можно вызвать из программной оболочки (приглашение команды MS-DOS) и использовать для преобразования сразу нескольких XML-документов с помощью одной таблицы стилей. Переменную среды окружения CLASSPATH нужно установить на файл **xmlparserv2.jar;** а среду PATH настроить на Java-интерпретатор, который поставляется вместе с инструментарием JDK. При выполнении преобразований таким

способом параллельно используется сразу несколько потоков. Вот, например, синтаксис такого вызова:

oraxsl options* source? stylesheet? result?



Внимание

"" означает повторение 0 и более раз, о "?" означает повторение 0 или 1.*

Предполагается, что средству **oraxsl** нужны таблица стилей, XML-файл для преобразования и (необязательно) результирующий файл. Если последний не определен, утилита **oraxsl** преобразует документ таким образом, чтобы его можно было прочитать на стандартном выходном устройстве (обычно на экране монитора). Если с помощью одной таблицы стилей нужно преобразовать сразу несколько XML-документов, в командной строке следует использовать опции **-l** и -d вместе с опциями -s и **-r**. Эти и остальные опции описаны в таблице 2.1.

Таблица 2.1

Опции командной строки для ***oraxsl***

Опция	Назначение
-h	Режим помощи (печать синтаксиса вызова oraxsl).
-v	Вывод подробной информации (распечатывается некоторая отладочная информация, что может оказаться полезным при отслеживании проблем, встречающихся в процессе обработки).
-w	Показывать предупреждения (в режиме по умолчанию никакие предупреждения не выводятся).
-e <файл регистрации ошибок>	Запись ошибок в файл регистрации (указывается файл, в который будут записываться все ошибки и предупреждения системы).
-t <число потоков>	Число потоков, которые будут использоваться для обработки (использование нескольких потоков может увеличить производительность при обработке нескольких документов одновременно).
-l <список xml-файлов>	Список файлов для преобразования (указывается список файлов, которые нужно обработать).
-d <каталог>	Каталог с файлами, которые нужно преобразовать (по умолчанию обрабатываются все файлы в данном каталоге). Если в каталоге нужно преобразовать только некоторые файлы или один файл, следует использовать опцию -l и указать в списке только те файлы, которые нужно обработать. Можно также использовать опции -x и -j для выбора файлов в соответствии с их расширениями.

Таблица 2.1 (*продолжение*)

Опции командной строки для ***oraxsl***

Опция	Назначение
-x <расширение файла>	Расширение файлов, которые не нужно преобразовывать (используется в сочетании с опцией -d, все файлы с указанным расширением преобразовываться не будут).
-i <расширение файла>	Расширение файлов, которые нужно преобразовать (используется в сочетании с опцией -d, будут преобразованы все файлы с указанным расширением).
-s <таблица стилей>	Используемая таблица стилей (если определены опции -d или -l, то опция -s нужна для определения используемой таблицы стилей, причем необходимо указать полный путь к данной таблице).
-r <результирующее расширение>	Расширение, которое будет использоваться для результатов преобразования. Если определены опции -d или -l, необходимо определить опцию -r, чтобы задать расширение, которое будет использоваться для результатов преобразования. Если задать расширение out, то входной документ "foo" преобразуется в документ "foo.out". По умолчанию результаты преобразования помещаются в текущем каталоге, но это можно изменить с помощью опции -o, позволяющей определить каталог, куда будут отправляться результаты.
-o <каталог для результатов>	Каталог, в который помещаются результаты (эту опцию следует использовать только в сочетании с опцией -r).

Примеры использования утилиты oraxsl Рассмотрим следующую структуру каталогов:

C:\xml, C:\xml\input, C:\xml\output

Предположим, что каталог **C:\xml\input** содержит набор XML-документов (docl.xml, **doc2.xml** и т. д.), которые нужно преобразовать с помощью таблицы стилей **C:\Stylesheet.xsl**. Это делается с помощью утилиты oraxsl несколькими разными способами.

Чтобы преобразовывать каждый документ по очереди, можно ввести команду:

oraxsl **C:\xml\input\document1.xml C:\xml\Stylesheet.xsl document1.out**

и повторить ее для каждого преобразовываемого документа.

Для преобразования всех XML-документов (имеющих расширение **xml**) в каталоге **C:\xml\input** с помощью таблицы стилей **C:\xml\Stylesheet.xsl** и помещения результатов с расширениями **.out** в каталог **C:\xml\output** команда должна быть следующей:

```
oraxsl -s C:\xml\Stylesheet.xsl -d C:\xml\input -i xml -o
C:\xml\output -o C:\xml\results -r out
```

Отметим, что файл **doc1.xml** будет преобразован в **docl.xml.out, doc2.xml —** в файл **doc2.xml.out** и т. д. Если нужно преобразовать много документов, повысить производительность этого процесса можно, инициировав несколько параллельных преобразований. Для этого используется флаг -t, определяющий число потоков. Но следует помнить, что чем больше используемых потоков, тем больше потребляется памяти, поэтому выбирать это число нужно, хорошенько подумав. Например, если для выполнения преобразований решено использовать рять параллельных потоков, то в список параметров команды нужно добавить -t 5.

Предположим, необходимо преобразовать лишь несколько документов из каталога. Тогда нужно перечислить их, используя для этого опцию -1:

```
oraxsl -s C:\xml\Stylesheet.xsl -d C:\xml\input -1 docl.xml doc2.xml -o
C:\xml\output -o G:\xml\results -r out
```

При этом преобразуются только документы **docl.xml и doc2.xml,** и в результате будут получены файлы **docl.xml.out и doc2.xml.out** соответственно.

Поддержка XML Schema

Импортируя пакет XML Schema и активизируя метод **setValidationMode** с аргументом **SCHEMA_VALIDATION**, можно также проверить соответствие XML-документов правилам XML Schema (предшественником XML Schema являются определения типов документов DTD). Процессор Oracle XML Schema соответствует последней версии проекта рекомендации W3C Candidate Recommendation: Part 0 — Primer, Part 1 — Structures, Part 2 — Datatypes. Основными преимуществами XML Schema являются поддержка простых и сложных типов данных, наличие ограничений на структуры XML-документов, в том числе поддержка пространства имен Namespace, которой нет в DTD. Благодаря поддержке типов данных преобразование данных XML-документов в данные базы данных и обратно становится еще более простым и наглядным процессом. С помощью процессора XML Schema можно вызывать API-интерфейсы из библиотеки и инициировать преобразование из командной строки.

Вот пример Java-кода, выполняющего такое преобразование:

```
import oracle.xml.parser.schema.*;
import oracle.xml.parser.v2.*;
```

```
import java.net.*;
import java.io.*;
import org.w3c.dom.*;
import java.until.*;

public class XSDSetSchema
{  public static void main(String[]  args)  throws Exception
   {  if (args.length  != 2)
      { System.out.println ( "Usage: java XSDSample <schema_file> <xml_file>") ;
        return
      }

      XSDBuilder builder = new XSDBuilder ();
      URL url = createURL(args[0] ) ;

      // Build XML Schema Object
      XMLSchema schemadoc = (XMLSchema)builder.build(url);
      process(args[1] , schemadoc);
   }

   public static void process(String xmlURI, XMLSchema schemadoc)
         throws Exception
   {  DOMParser dp = new DOMParser();
      URL url = createURL(xmlURI);

      // Set Schema Object for Validation
      dp.setXMLSchema (schemadoc);
      dp.setValidationMode (XMLParser.SCHEMA_VALIDATION);
      dp.setPreserveWhitespace (true);

      dp.setErrorStream (System.out);

      try
      {  System.out.ptintln("Parsing " + xmlURI);
         dp.parse (url)
         System.out.prontln("The input file <"+xmlURI+">
                              parsed whithout errors");
      }
      catch (XMLParseException pe)
      {  System.out. ptintln ("Parser Exception: " + pe.getMessage());  }
      catch (Exception e)
      C  System.out.ptintln("NonParserException: " + e.getMessage()) ;  }
   }
```

```
    // Helper method to create a URL from a filename BCC
    static URL createURL (String fileName)
    {  URL url = null;
       try
       {  url = new URL(fileName) ;  }
       catch (MalformedURLException ex)
       {  File f = new File (fileName);
          try
          {  String path = f.getAbsolutePath ( ) ;
             // This is a bunch of weird code that is required to
             // make a valid URL on the Windows platform, due
             // to inconsistencies in what getAbsolutePath returns.
             String fs = System.getProperty ("file separator");
             if ( fs.length() == 1)
             {  char sep = fs.charAt (0);
                if (sep != '/')
                   path = path. replace (sep, '/');
                if (path.charAt(0) != '/')
                   path = '/' + path;
             }
             path = "file://" + path;
             url = new URL (path);
          }
          catch (MalformedURLException e)
          {  System.out.ptintln( "Cannot create url for: " + fileName);
             System.exit(0);
          }
       }
       return url;
    }
}
```

А вот примеры XML Schema, **catalogue.xsd** и XML-файла **catalogue.xml,** которые используют этот код:

```
<?xml version="1.0"?>
<schema  xmlns="http: //www.w3.org/2000/10/XMLSchema"
         targetNamespace="http: //www.publishing.org/namespace/Catalogue"
         elementFormDefault="qualified"
         xmlns:xsi= "http://www.w3.org/1999/XMLSchema-instance"
         xmlns:cat= "http: //www.publishing.org/namespace/Catalogue">
```

```
<complexType name="PublicationType">
   <sequence>
      <element name="Title" type="string" minOccurs="1"
       maxOccurs="unbounded"/>
      <element name="Author" type="string" minOccurs="1"
       maxOccurs="unbounded"/>
      <element name="Date" type="year" minOccurs="1" maxOccurs="1"/>
   </sequence>
</complexType>
<element name="Publication" type="cat:PublicationType" abstract="true"/>
<element name="Book" substitutionGroup="cat:Publication">
   <complexType>
      <complexContent>
         <extension base="cat:PublicationType">
            <sequence>
               <element name="ISBN" type="string" minOccurs="1"
                maxOccurs="1"/>
               <element name="Publisher" type="string" minOccurs="1"
                maxOccurs="1"/>
            </sequence>
         </extension>
      </complexContent>
   </complexType>
</element>
<element name="Magazine" substitutionGroup="cat:Publication">
   <complexType>
      <complexContent>
         <restriction base="cat:PublicationType">
            <sequence>
               <element name="Title" type="string" minOccurs="1"
                maxOccurs="unbounded"/>
               <element name="Author" type="string" minOccurs="0"
                maxOccurs="0"/>
               <element name="Date" type="year" minOccurs="1"
                maxOccurs="1"/>
            </sequence>
         </restriction>
      </complexContent>
   </complexType>
</element>
<element name="Catalogue">
   <complexType>
      <sequence>
```

```
            <element ref="cat:Publication" minOccurs="0"
             maxOccurs="unbounded" />
          </sequence>
        </complexType>
      </element>
    </schema>
```

Отметим, что в этой XML Schema существует тип данных "year" (год), ограничения на значения, а также поддержка пространства имен Namespace. Тогда XML-документ будет выглядеть так:

```
IP <?xml version="1.0"?>
   <Catalogue  xmlns="http://www.publishing.org/namespace/Catalogue"
               xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
               xsi:schemaLocation="http://www.publishing.org/namespaces/Catalogue
                                   catalogue.xsd">
          <Magazine>
                <Title>Natural Health</Title>
                <Date>1999</Date>
          </Magazine>
          <Book>
                <Title>Illusions The Adventures of a Reluctant Messiah</Title>
                <Author>Richard Bach</Author>
                <Date>1977</Date>
                <ISBN>0-440-34319-4</ISBN>
                <Publisher>Dell Publishing Co.</Publisher>
          </Book>
          <Book>
                <Title>The First and Last Freedom</Title>
                <Author>J. Krishnamurti</Author>
                <Date>1954</Date>
                <ISBN>0-06-064831-7</ISBN>
                <Publisher>Harper &amp;amp; Row</Publisher>
          </Book>
   </Catalogue>
```

Генератор Java-классов

Генератор классов XML Class Generator for Java создает исходные Java-файлы из XML DTD или из XML Schema. Он является вспомогательным средством для программного создания допустимых (valid) документов из динамических данных. Это может оказаться полезным, если необходимо, чтобы приложение посылало XML-сообщения в другое приложение в соответствии с заранее заданными DTD или

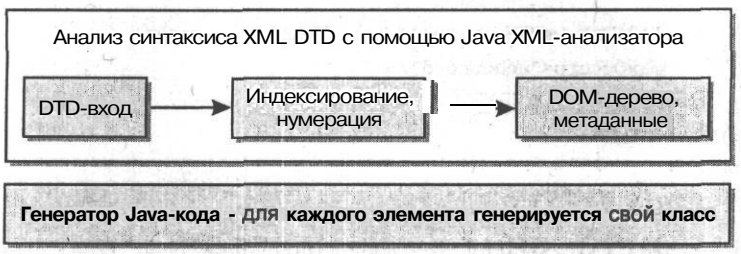


Рис. 2.2. *Архитектура генератора классов XML Class Generator*

XML Schema, или если нужна Web-форма для построения XML-документа. Можно построить, по желанию проверить допустимость и распечатать XML-документы, которые соответствуют введенным DTD или XML Schema в Java-приложениях с использованием сгенерированных классов. Генератор классов Class Generator работает вместе с программой-анализатором Oracle XML Parser for Java, анализирующей DTD или XML Schema и после анализа передающей их в генератор классов, как показано на рис. 2.2.

Затем генератор классов направляет запросы в DTD или XML Schema относительно всех элементов. Java-класс генерируется для каждого элемента. Эти классы имеют методы для установки атрибутов и добавления дочерних узлов с помощью соответствующей модели содержимого документа. DTD-класс, соответствующий корневому элементу, также имеет методы для определения допустимости и печати построенного XML-документа. В следующих разделах на примере показано использование XML Class Generator for Java для обработки DTD или XML Schema и генерирования классов для их элементов. Затем демонстрируется использование методов классов элементов для программного построения допустимого XML-документа.

Входной DTD

Следующий DTD-файл для информации о книгах bookcatalog.dtd используется в качестве входного для генератора классов. Здесь DTD определяет, что корень XML-документа — это BOOKCATALOG. Каталог книг BOOKCATALOG состоит из одной и более книг (BOOK). Каждая запись BOOK содержит в качестве уникального идентификатора атрибут ISBN, а также несколько необязательных атрибутов и дочерних элементов, например TITLE для названия книги, AUTHORNAME для имени автора, PUBLISHER для названия издательства и т. д. Необязательные атрибуты и дочерние элементы сопровождаются в определении элементов значками "?":

```
<!-- DTD for Book Data -->
<!ELEMENT BOOKCATALOG (BOOK)*>
<!ELEMENT BOOK (TITLE, AUTHORNAME, PUBLISHER?, PUBLISHYEAR?, PRICE?)>
<!ATTLIST BOOK ISBN CDATA #REQUIRED>
<!ATTLIST BOOK BOOKTYPE (Fiction|SciFi|Fantasy) #IMPLIED>
<!ELEMENT TITLE (#PCDATA)>
```

```
<! ELEMENT AUTHORNAME (#PCDATA)
<! ELEMENT PUBLISHER (#PCDATA)>
<! ELEMENT PUBLISHYEAR (#PCDATA)>
<! ELEMENT PRICE (#PCDATA)>
```

Обработка DTD для генерирования Java-классов

Ниже приводится пример кода, который обрабатывает DTD и генерирует соответствующие классы для элементов DTD. В этой программе кроме имени и корневого элемента используется внешний DTD-файл. Генератор классов может также проанализировать синтаксис XML-документа и использовать определенный в нем DTD. Применив генератор классов к этому DTD, мы получим Java-классы для каждого элемента (BOOKCATALOG, BOOK, TITLE и т. д.). Затем Java-приложение использует методы, определенные для этих классов, чтобы создать допустимый XML-документ.

```
public class SampleMain
{  public static void main (String args[])
   {                                          // validate arguments
      if (args.length != 1)
      C  System.out.println("Usage: java SampleMain "+
                            "-root <rootName> <fileName>");
         return ;
      }
      try                                     // to open the External DTD File
      {  // instantiate the parser
         XMLParser parser = new XMLParser();
         String doctype_name = null;
         parser.parseDTD(fileToURL(args[2]), args[1]);
         DTD dtd =  (DTD)parser.getDoctype();
         doctype_name = args[1];
         // generate the Java files...
         ClassGenerator generator = new ClassGenerator();
         // set generate comments to true
         generator.setGenerateComments(true);
         // set output directory
         generator.setOutputDirectory(".");
         // set validating mode to true
         generator.setValidationMode(true);
         // generate Java src
         generator.generate(dtd, doctype_name);
      }
```

```
        catch (Exception e)
        {  System.out.println ("XML Class Generator: Error " + e.toString());
           e.printStackTrace();
        }
     }
  }
```

Создание из Java-классов допустимого XML-документа

Ниже приведен Java-код, демонстрирующий использование сгенерированных методов. В этом примере создаются две записи BOOK: book1 и book2. Обязательные атрибуты вводятся путем добавления их в конструктор в качестве параметров. Здесь ISBN является обязательным атрибутом для элемента BOOK. Создаются также элементы для каждого столбца таблицы базы данных (title1, authorname1 и т. д.). Если какой-то элемент имеет перечисляемый атрибут, то для каждого значения в этом перечислении в классе определяется статическая константа. Для построения дерева XML-документа производится группировка нескольких элементов данных. Эти элементы ставятся в соответствие каждому элементу строки таблицы базы данных как узлы дерева. Затем каждый элемент BOOK добавляется в качестве узла в корневой элемент BOOKCATALOG документа. В этом примере классы, генерируемые генератором классов, записаны прописными буквами:

```
public class CreateBooks
{  public static void main (String args[])
   {  try
      {  BOOKCATALOG bookList = new BOOKCATALOG();
         // New book book1
         BOOK book1 = new BOOK("7654");          // create book1 and set ISBN
         TITLE title1 = new TITLE("The Adventures of Don Quixote");
         book1.setBOOKTYPE(BOOK.BOOKTYPE_FICTION);
         AUTHORNAME authorname1 = new AUTHORNAME("Miguel Cervantes");
         PUBLISHER publisher1 = new PUBLISHER("Oracle Press");
         PUBLISHYEAR publishyear1 = new PUBLISHYEAR("2000");
         PRICE price1 = new PRICE("1.00");

         // New book book2
         BOOK book2 = new BOOK("7788");          // create book2 and set ISBN
         TITLE title2 = new TITLE("The Iliad");
         book2.setBOOKTYPE(BOOK.BOOKTYPE_FICTION);
         AUTHORNAME authorname2 = new AUTHORNAME("Homer");
         PUBLISHER publisher2 = new PUBLISHER("Oracle Press");
         PUBLISHYEAR publishyear2 = new PUBLISHYEAR("1000");
         PRICE price1= new PRICE("");
```

```
            book1.addnode(isbn1);          // Add data as tree nodes to book1
            book1.addnode(title1);
            ...

            book2.addnode(isbn2);          // Add data as tree nodes to book2
            book2.addnode(title2);
            ...

            bookList.addNode(book1);       // Add book1 as tree node to
                                           // bookList doc root
            bookList.addNode(book2);       // Add book2 as tree node to
                                           // bookList doc root

            bookList.validateContent();
            bookList.print(System.out);
         }
         catch (Exception e)
         { System.out.println(e.toString());
            e.printStackTrace();
         }
      }
   }
```

XML-документ, созданный с помощью Java-приложения

Входные данные для вышеприведенного Java-приложения можно получить из нескольких источников, например из Web-формы, SAX-анализатора или набора результатов, полученного после запроса к базе данных с помощью JDBC. Это Java-приложение создает XML-документ, который затем можно преобразовать в HTML-файл с помощью XSLT-процессора или сохранить в базе данных с помощью XSQL Servlet. Созданный XML-документ будет выглядеть следующим образом:

```
<?xml version="1.0"?>
<!DOCTYPE BOOKCATALOG SYSTEM "bookcatalog.dtd">
<BOOKCATALOG>
   <BOOK ISBN = "7654" BOOKTYPE="Fiction">
      <TITLE>The Adventures of Don Quixote</TITLE>
      <AUTHORNAME>Miguel Cervantes</AUTHORNAME>
      <PUBLISHER>Oracle Press</PUBLISHER>
      <PUBLISHYEAR>2000</PUBLISHYEAR>
      <PRICE>1.00</PRICE>
   </BOOK>
```

```
    <BOOK ISBN = "7788" BOOKTYPE="FICTION">
      <TITLE>The Illiad</TITLE>
      <AUTHORNAME>Homer</AUTHORNAME>
      <PUBLISHER>Oracle Press</PUBLISHER>
      <PUBLISHYEAR>1000</PUBLISHYEAR>
      <PRICEx/PRICE>
    </BOOK>
</BOOKCATALOG>
```

Входные данные из XML Schema

Как уже говорилось, входные данные можно получить не только из DTD, но и из XML Schema. Механизмы этого процесса для Java-драйвера и для модели входных данных аналогичны. Ниже приведены примеры XML Schema и Java-программы, которая вызывает необходимые методы генератора классов.

```
<xsd:schema xmlns:xsd="http://www.w3.org/1999/XMLSchema">

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

<xsd:element name="PurchaseOrder">
   <xsd:complexType>
      <xsd:element name="shipTo" type="Address"/>
      <xsd:element name="billTo" type="Address"/>
      <xsd:element ref="comment" minOccurs="0"/>
      <xsd:element name="items" type="Items"/>
      <xsd:attribute name="orderDate" type="xsd:date"/>
      <xsd:attribute name="shipDate" type="xsd:date"/>
      <xsd:attribute name="receiveDate" type="xsd:date"/>
   </xsd:complexType>
</xsd:element>

<xsd:complexType name="Address">
   <xsd:element name="name" type="xsd:string"/>
   <xsd:element name="street" type="xsd:string"/>
   <xsd:element name="city" type="xsd:string"/>
   <xsd:element name="zip" type="xsd:decimal"/>
   <xsd:attribute name="country" type="xsd:NMTOKEN"
      use="fixed" value="US"/>
</xsd:complexType>

<xsd:complexType name="Items">
   <xsd:element name="item" minOccurs="0" maxOccurs="unbounded"/>
      <xsd:complexType>
         <xsd:element name="productName" type="xsd:string"/>
         <xsd:element name="quantity" type="xsd:int"/>
```

```
        <xsd:element name="price" type="xsd:decimal"/>
        <xsd:element name="shipDate" type="xsd:date" minOccurs='0'/>
        <xsd:element ref="comment" minOccurs="0"/>
        <xsd:attribute name="partNum" type="xsd:string"/>
      </xsd:complexType>
   </xsd:element/>
</xsd:complexType>

</xsd:schema>

import oracle.xml.classgen.*;
import oracle.xml.parser.v2.*;
import oracle.xml.parser.schema.*;
import java.io.*;
import java.util.*;

public class TestPo
{  public static void main(String arg[])
   {  TestPo testpo = new TestPo();
      try
      {  PurchaseOrder po = new PurchaseOrder();
         Address address = new Address();
         Items items = new Items();

         // Create shipTo address
         address.addName ("Mary Smith");
         address.addStreet ("Laurie Meadows");
         address.addCity ("San Mateo");
         address.addZip (new Float(98806));

         // Add shipTo address to the PurchaseOrderType
         po.addShipTo(address);

         // Create billTo address
         address.addName ("John Smith");
         address.addStreet ("1 North Broadway");
         address.addCity ("New York");
         address.addZip (new Float(11208));

         // Add billTo address to the PurchaseOrderType
         po.addBillTo(address);

         // Add comment to the PurchaseOrderType
         po.addComment(new Comment ("Happy Birthday"));

         // Create the items
         items.addProductName ("Perfume");
```

```
            items.addQuantity(1);
            items.addPrice(newFloat(75));
            items.addComment(newComment("Have a nice day!"));
            items.addShipDate(newString("Nov14, 2000"));

            // Add items to the PurchaseOrderType
            po.addItems(items);

            // Set purchase order date
            po.setOrderDate("December 17, 2000");
            po.setShipDate("December 19, 2000");
            po.setReceiveDate("December 21, 2000");

            //po.print(System.out);

            // Get Elements of Purchase Order
            Vector Elements = po.getElements();

            System.out.print("The child element node types of PurchaseOrder is:");
            for (int i=0; i < elements.size(); i++)
            { System.out.print(elements.elementAt(i));
              System.out.print(" ");
            }

            System.out.println();
            System.out.println("The attributes of PurchaseOrder is:");
            System.out.print("OrderDate = ");
            System.out.print(po.getOrderDate());
            System.out.println();
            System.out.print("ShipDate = ");
            System.out.print(po.getShipDate());
            System.out.println();
            System.out.print("ReceiveDate = ");
            System.out.print(po.getReceiveDate());
            System.out.println();
        }
        catch (InvalidContentException e)
        { System.out.println(e.getMessage());
          e.printStackTrace();
        }
        catch (Exception e)
        {System.out.println(e.toString());
          e.printStackTrace();
        }
    }
}
```

Просмотр и преобразование XML-файлов с помощью Java-программ

Многие XML-разработчики используют в качестве *интегрированной среды разработки* (integrated development environment, *IDE*) пакет Oracle JDeveloper. Это позволяет ускорить цикл разработки и воспользоваться преимуществами компонентного подхода при построении приложений, когда они создаются на базе ранее созданных компонентов. В Oracle JDeveloper полностью поддерживается компонентная модель Java-модулей (Java Beans). В JDeveloper можно строить Java-модули, а затем повторно использовать их в других проектах. Можно также устанавливать и использовать Java-модули, созданные ранее в Oracle и в других компаниях. Модули в JDeveloper можно установить в панели инструментов JDeveloper Tools, после чего настраивать в соответствии с конкретной ситуацией и включать в создаваемое приложение. После установки таких модулей в панель инструментов Tools их добавляют в разрабатываемое приложение, перетаскивая мышью по технологии drag-and-drop. При добавлении модуля в разрабатываемое приложение таким способом JDeveloper автоматически генерирует код, необходимый для его обработки и настройки. Эта процедура обычно предусматривает создание экземпляра этого модуля, установку его свойств для настройки работы модуля и добавление в него слушателей Action Listeners, чтобы модуль разрешил приложению обрабатывать генерируемые им события. Это очень мощная и простая в использовании технология, поэтому вполне естественно будет инкапсулировать в Java-модули основные функциональные возможности XML.

Модули XML TransViewer Bean представляют собой набор XML-компонентов для Java-приложений или апплетов, добавляющих в XML-приложение визуальный интерфейс. Эти визуальные и невизуальные Java-компоненты можно интегрировать в Oracle JDeveloper, чтобы позволить разработчикам быстро создавать и разворачивать XML-приложения баз данных. Имеются следующие модули:

- DOMBuilder
- XSLTransformer
- XMLSourceView
- XMLTreeView
- XMLTransformPanel
- DBView
- DBAccess

Если установить эти модули в среду JDeveloper, то при включении их в приложения JDeveloper будет автоматически генерировать соответствующий код. Если в работе используется не среда Jdeveloper, а среда JDK с интерфейсом командной строки,

можно применять модули для визуализации и преобразования XML-файлов. В таком случае модули используются как любые другие классы. Примеры, представленные в этой главе, разработаны с помощью JDeveloper. Три из вышеперечисленных модулей, а именно: **DOMBuilder, XSLTransformer и DBAccess** являются невизуальными. Остальные четыре модуля относятся к визуальным. Они расширяют возможности JPanel и их можно включать в приложения везде, где используется JPanel. Далее приведен обзор всех вышеупомянутых модулей, разберем работу и способы использования каждого из них.

Модуль DOMBuilder Bean

Модуль **DOMBuilder** Bean инкапсулирует анализатор синтаксиса Java XML Parser с интерфейсом модуля и расширяет его функциональные возможности таким образом, чтобы он поддерживал асинхронный анализ синтаксиса. После регистрации слушателя Java-приложения могут проводить анализ синтаксиса больших или следующих один за другим документов, возвращая функции управления в вызывающую программу сразу же после начала процесса анализа. Нижеследующий пример кода демонстрирует программу, которая берет в качестве параметров список XML-файлов и проводит анализ синтаксиса всех этих файлов одновременно:

```
package sample;

import java.awt.event.*;
import oracle.xml.async.*;
import oracle.xml.parser.v2.*;
import org.w3c.dom.*;
import java.net.*;
import java.io.*;

public class MParse extends Object
    implements DOMBuilderListener, DOMBuilderErrorListener{
    int numArgs,i;
    String Args[];
    DOMBuilder tParser;

    public MParse(String[] args) {
        Args=args;
    }
    public void parse() {
        for (i=0;i<Args.length;i++) {
            // new instance of the asynchronous parser
            tParser=new DOMBuilder(i);
            // add this object as Listener so we will be notified
            // when parsing is complete
            tParser.addDOMBuilderListener(this);
```

```
        // or when an error occur
        tParser.addDOMBuilderErrorListener(this);
        System.out.println("Start parsing "+Args[i]);
        try {
           tParser.parse(new URL("file:"+(String)Args[i]));
        } catch (Exception e) {
           System.out.println(e.toString());
        }
     }
     System.out.println("Multiple files parsed in background threads");
  }

  public static void main(String[] args) {
     MParse mParse = new MParse(args);
     mParse.parse();
  }

  // Implementing DOMBuilderListener Interface

  // this method is called by the DOMBuilder object when
  // the parsing of the document starts.
  public void domBuilderStarted(DOMBuilderEvent p0) {
  }
  // this method is called by the DOMBuilder object when
  // the parsing of the document produces an error.
  public void domBuilderError(DOMBuilderEvent p0) {
  }

  // this method is called by the DOMBuilder object when
  // the parsing of the document is completed.
  public void domBuilderOver(DOMBuilderEvent p0) {
     DOMBuilder parser;
     XMLDocument xmlDoc;
     int id;
     // Get a reference to the parser instance that finished parsing.
     parser=(DOMBuilder)p0.getSource();
     // get the parser id to identify the file being parsed
     id=parser.getId();
     System.out.println("Parse completed for file "+Args[id]);
     // get the dom tree
     xmlDoc=parser.getDocument();
     // Here we do whatever we would like to do with the
     // parsed document.
  }
```

```
    // Implementing DOMBuilderErrorListener Interface

    // This method is called when parsing error occurs
    public void domBuilderErrorCalled(DOMBuilderErrorEvent p0) {
        int id= ((DOMBuilder)p0.getSource()).getId();
        System.out.println("Parse error for "+Args[id]+": "+
        p0.getException().getMessage());
    }
}
```

Если эта программа запускается с помощью команды:

```
java Sample.mParse booklist1.xml booklist2.xml booklist3.xml
```

получаются следующие выходные данные (см. рис. 2.3).

```
Start parsing booklistl .xml
Start parsing booklist2.xml
Start parsing booklist3.xml
Multiple files parsed in background threads
Parse completed for file booklist2.xml
Parse completed for file booklistl .xml
Parse completed for file booklist3.xml
```

Рис. 2.3. *Пример выходных данных в JDeveloper при использовании модуля DOMBuilder Bean*

При необходимости провести синтаксический анализ большого числа файлов, можно воспользоваться модулем **DOMBuilder** Bean, чтобы существенно сэкономить время. В зависимости от системы и от количества параллельных потоков (экземпляров DOMBuilder), можно выполнить эту работу на 40% быстрее по сравнению с тем временем, которое потребуется для последовательного анализа файлов один за другим.

В интерактивных визуальных приложениях можно также использовать асинхронный синтаксический анализ в фоновом потоке, который реализован в DOMBuilder Bean. Если приложение проводит анализ больших файлов с помощью обычного анализатора, пользовательский интерфейс будет заморожен до тех пор, пока документ не будет полностью проанализирован. Этого можно легко избежать, использовав DOMBuilder Bean. После вызова метода синтаксического анализа из DOMBuilder Bean приложение снова получит все функции управления, и на экране монитора появится окно с сообщением "Parsing, please wait" ("Подождите, пожалуйста, идет синтаксический анализ"). В нем может быть также кнопка "Cancel" ("Отменить"), поэтому пользователь сможет прекратить эту операцию. Если же пользователь не предпринимает никаких действий, программа заканчивает свою работу, когда по завершении задачи синтаксического анализа в фоновом режиме модуль **DOMBuilder** Bean вызывает метод domBuilderOver.

Модуль XSLTransformer Bean

Модуль **XSLTransformer** Bean инкапсулирует процессор Java XML Parser XSLT и расширяет его функциональные возможности таким образом, чтобы он поддерживал асинхронное преобразование. Зарегистрировав слушателя, Java-приложения могут преобразовывать большие или последовательно поступающие документы, сразу же возвращая функции управления в вызывающую программу. Так как XSL-преобразования требуют довольно много времени для своего выполнения, стоит рассмотреть возможность использования асинхронного интерфейса. Модуль **XSLTransformer** Bean полезно использовать в приложениях, преобразующих большое количество файлов путем обработки нескольких файлов одновременно. Этот модуль также можно использовать в визуальных приложениях для получения более отзывчивого интерфейса. Рассуждения здесь можно привести те же самые, что и в случае с модулем **DOMBuilder** Bean. С точки зрения программирования предыдущий пример демонстрирует общий подход, который также применим к интерфейсу XSLTransformerListener: вызывающее приложение получает уведомление о завершении процесса преобразования, поэтому между запросом на преобразование и получением результатов оно может выполнять и другие задачи.

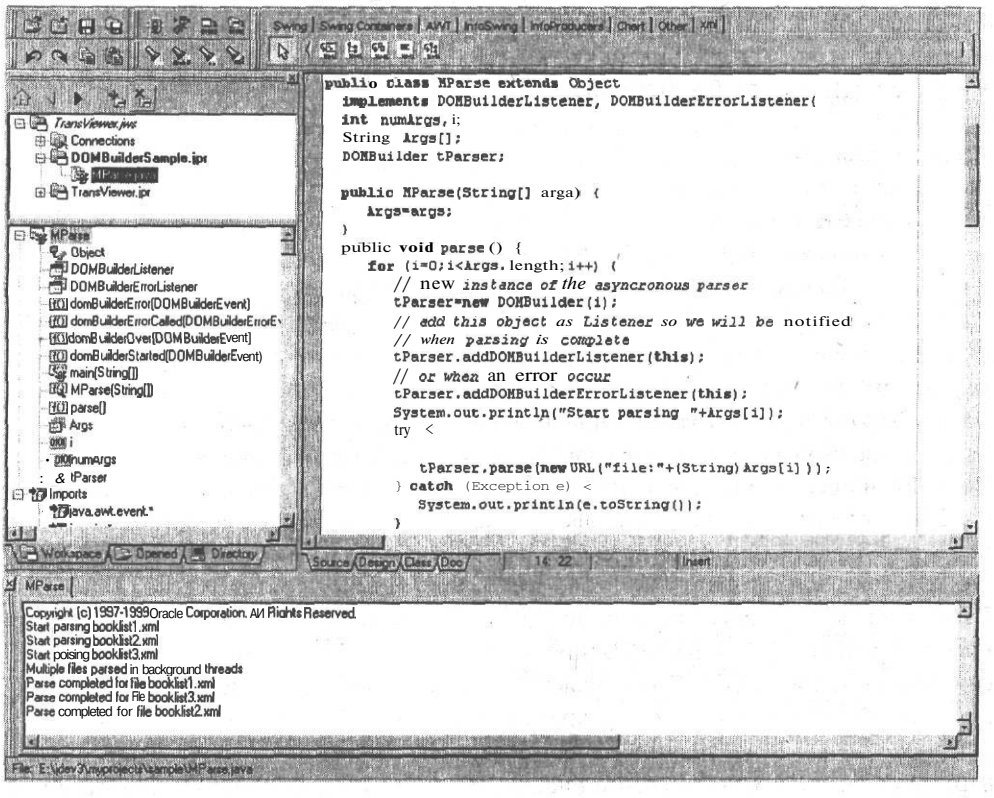


Рис. 2.4. *Пример программы, использующей модуль XML DOMBuilder Bean в среде JDeveloper*

Модуль XMLSourceView Bean

Модуль XMLSourceView Bean улучшает зрительное восприятие XML- и XSL-файлов, так как производит цветную подсветку XML/XSL-синтаксиса. Этот модуль поддерживает режим редактирования Edit. Он легко интегрируется с модулем **DOMBuilder** Bean, что позволяет визуализировать XML-файлы до и после их синтаксического анализа, а также проверять их соответствие заданному DTD. Ниже приведен пример простой программы, которая выводит на экран XML-файл с выделением синтаксиса XML. На рис. 2.4 показан вид экрана этой программы.

```
import java.awt.*;
import oracle.xml.srcviewer.*;
import oracle.xml.treeviewer.*;
import oracle.xml.parser.v2.XMLDocument;
import oracle.xml.parser.v2.*;
import org.w3c.dom.*;
import java.net.*;
import java.io.*;
import java.util.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.event.*;

public class ViewSample
{  public static void main(String[] args)
   {  String fileName = new String ("booklist.xml");
      if (args.length > 0) {
         fileName = args[0];
      }
      JFrame          frame          = setFrame ("XMLViewer");
      XMLDocument     xmlDocument    = getXMLDocumentFromFile (fileName);
      XMLSourceView   xmlSourceView  = setXMLSourceView (xmlDocument);
      XMLTreeView     xmlTreeView    = setXMLTreeView (xmlDocument);
      JTabbedPane     jtbPane        = new JTabbedPane ();

      jtbPane.addTab ("Source", null, xmlSourceView,
                      "XML document source view");
      jtbPane.addTab ("Tree", null, xmlTreeView, "XML document tree view");
      jtbPane.setPreferredSize (new Dimension(400,300));
      frame.getContentPane().add (jtbPane);
      frame.setTitle (fileName);
      frame.setJMenuBar (setMenuBar());
      frame.setVisible (true);
   }
```

```
static JFrame setFrame (String title)
{  JFrame frame = new JFrame (title);
   // Center the window
   Dimension screenSize = Toolkit.getDefaultToolkit() .getScreenSize();
   Dimension frameSize = frame.getSize();
   if (frameSize.height > screenSize.height) {
      frameSize.height = screenSize.height;
   }
   if (frameSize.width > screenSize.width) {
      frameSize.width = screenSize.width;
   }
   frame.setLocation ( (screenSize.width - frameSize.width) /2,
                       (screenSize.height - frameSize.height) /2);
   frame.addWindowListener(newWindowAdapter() {
      public void windowClosing (WindowEvent e) {
         System.exit(0);
      }
   });
   frame.getContentPane() .setLayout(newBorderLayout() );
   frame.setSize(new Dimension (400, 300));
   frame.setVisible (false);
   frame.setTitle (title);
   return frame;
}
static JMenuBar setMenuBar ()
{  JMenuBar menuBar = newJMenuBar ( ) ;
   JMenu menu = new JMenu ("Exit");
   menu.addMenuListener (new MenuListener () {
      public void menuSelected (MenuEvent ev) { System.exit (0); }
      public void menuDeselected (MenuEvent ev) {}
      public void menuCanceled (MenuEvent ev) {}
   });
   menuBar.add (menu);
   return menuBar;
}
/**
 * creates XMLSourceView object
 */
static XMLSourceView setXMLSourceView(XMLDocument xmlDocument)
{  XMLSourceView xmlView = new XMLSourceView();
   xmlView.setXMLDocument(xmlDocument);
   xmlView.setEditable(true);
```

```
    xmlview.setBackground(Color.yellow);
    return xmlview;
}
/**
 * creates XMLTreeview object
 */.
static XMLTreeview setXMLTreeView(XMLDocument xmlDocument)
{  XMLTreeview xmlview = new XMLTreeView();
   xmlView.setXMLDocument(xmlDocument);
   xmlView.setBackground(Color.yellow);
   return xmlview;
}
static XMLDocument getXMLDocumentFromFile (String fileName) {
   XMLDocument doc. = null;
   try {
      DOMParser parser = new DOMParser();

      try {
         FileInputStream in = new FileInputStream(fileName);
         parser.setPreserveWhitespace(false);
         parser.setBaseURL(newURL("file://"));
         parser.parse(in);
         in.close();
      } catch (Exception ex) {
         ex.printStackTrace();
         System.exit(0);
      }

      doc=(XMLDocument)parser.getDocument();

      try {
         doc.print (System.out);
      } catch (Exception ie) {
         ie.printStackTrace();
         System.exit(0); }:
      }

      catch (Exception e) {
         e.printStackTrace();
      }
      return doc;
   }
}
```

Модуль XMLTreeView Bean

Модуль **XMLTreeView** Bean выводит на экран визуальное представление XML-документа. При этом пользователь получает возможность легко манипулировать деревом документа с помощью мыши с целью сокрытия или просмотра отдельных ветвей дерева. Пример кода, приведенный выше, использует данный модуль вместе с модулем **XMLSourceView** Bean. На рис. 2.5 представлен вид экрана этого приложения, где для визуализации XML-файла использован модуль **XMLTreeView** Bean.

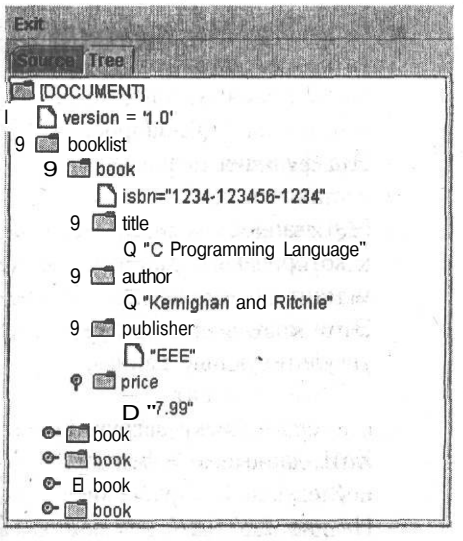


Рис. 2.5. *Пример использования модуля XMLTreeView Bean для вывода*

Модуль XMLTransformPanel Bean

С помощью модуля **XMLTransformPanel** Bean пользователи могут преобразовывать XML-документ в документ почти любого текстового формата (в том числе в документы XML- и HTML-форматов). Для этого к XML-документу применяется XSL-таблица стилей. Этот модуль можно включить в приложение, управляющее XML- и XSL-данными. **XMLTransformPanel** Bean имеет следующие функции:

- Импорт и экспорт XML- и XSL-файлов из файловой системы.
- Поддержка нескольких соединений с базой данных.
- Импорт и экспорт XML, XSL и HTML/текстовых файлов из баз данных Oracle8i. Этот модуль сохраняет в базе данных свои собственные таблицы *CLOB*. Каждая такая таблица имеет два столбца. Первый столбец — строкового типа, он используется для хранения имени данных, хранящихся во втором столбце. Второй столбец имеет тип CLOB, он используется для хранения реальных текстовых данных. Другими словами, первый столбец содержит имя файла, а второй — данные этого файла. Когда вы входите в систему, модуль XMLTransformPanel Bean составляет списки всех CLOB-таблиц в схеме пользователя. Если щелкнуть мышью по какому-нибудь названию таблицы, этот модуль составит список имен файлов в таблице. Можно создавать и удалять таблицы, а также извлекать, заменять и добавлять в них файлы. Таблицы можно использовать в базе данных в качестве файловой системы для организации информации в файлах.

- Создание XML-документов из результирующего набора данных, найденных в ответ на SQL-запрос. Эта функция позволяет составлять любые SQL-запросы к БД, к которой вы в данный момент подключены. Этот модуль преобразует результирующие данные в XML-документ и автоматически загружает XML-данные в буфер для последующей обработки. На рис. 2.6 и 2.7 показан интерфейс модульной БД и запрос, используемый для создания XML-данных из EMP-таблицы в БД.

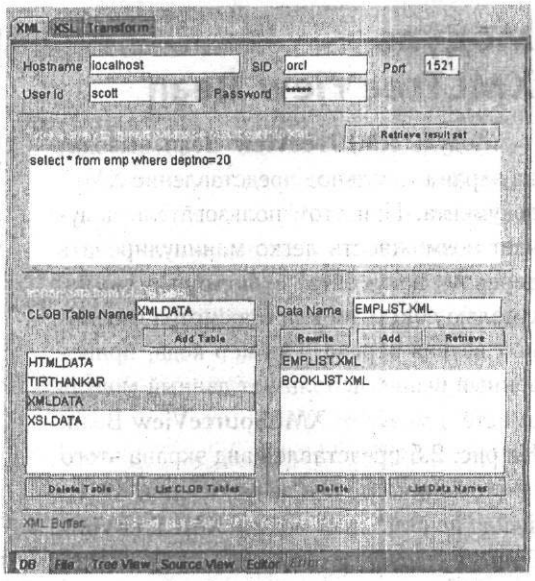


Рис. 2.6. *Получение результата запроса в виде XML-файла*

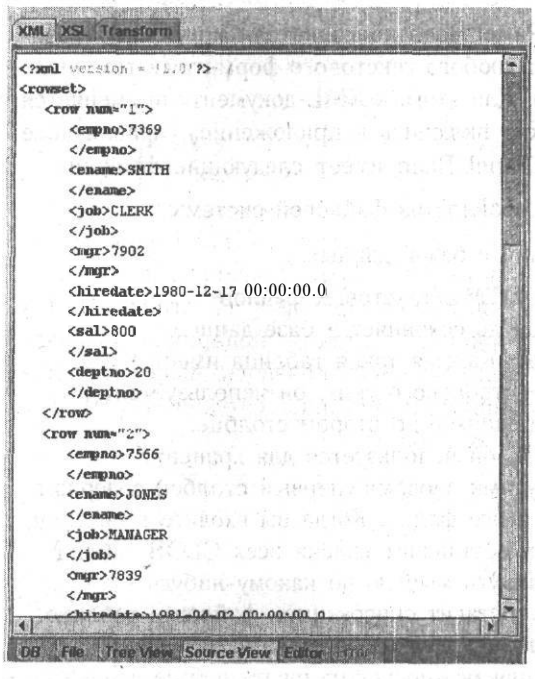


Рис. 2.7. *XML-документ, генерируемый модулем TransViewer Bean*

- Редактирование XML- и XSL-данных, загруженных в модуль XMLTransformPanel.
- Применение к XML-буферу XSL-преобразования и вывод на экран результата. Модуль XMLTransformPanel также позволяет экспортировать результаты в файловую систему или в CLOB-таблицу в БД. На рис. 2.8 показана результирующая страница, которая получается после того, как этот модуль применил таблицу стилей XSL к данным, показанным на рис. 2.7.

CLOB-таблицы, которые созданы модулем **XMLTansformPanel** Bean, могут быть использованы хранимыми процедурами, работающими по

триггерной схеме, для зеркального отображения таблиц или представлений в базе данных в HTML-данные, которые находятся в этих CLOB-таблицах. Например, с помощью модуля XMLTransformPanel Bean можно создать SQL-запрос, который выберет нужную информацию и сохранит результирующие XML-данные в CLOB-таблице. Затем с помощью этого же модуля можно разработать XSL-таблицу стилей и интерактивно применять ее к полученным XML-данным до тех пор, пока не будет получено удовлетворительное представление данных (выходной HTML-файл, полученный с помощью XSL-преобразования). Если используется язык запросов SQL (уровень селекции данных) и готовый уровень представления данных XSL, можно создать триггер для исходной таблицы или представления данных, используемый в SQL-запросе. Этот триггер запускает хранимую процедуру, которая, в свою очередь, инициирует запрос в базу данных, применяет таблицу стилей и сохраняет результирующий HTML-файл в CLOB-таблице. Этот процесс повторяется всякий раз при обновлении таблицы исходных данных. Поэтому HTML-файл, хранящийся в CLOB-таблице, всегда отражает последние данные, хранящиеся в таблицах, к которым направлялся запрос. Затем для вывода на экран полученного HTML-файла можно использовать простую JSP-страницу. При таком сценарии не возникает ситуация, когда множество конечных пользователей производят массу запросов, увеличивая тем самым нагрузку на базу данных. В описанной ситуации HTML-файл обновляется только при изменении данных в базе.

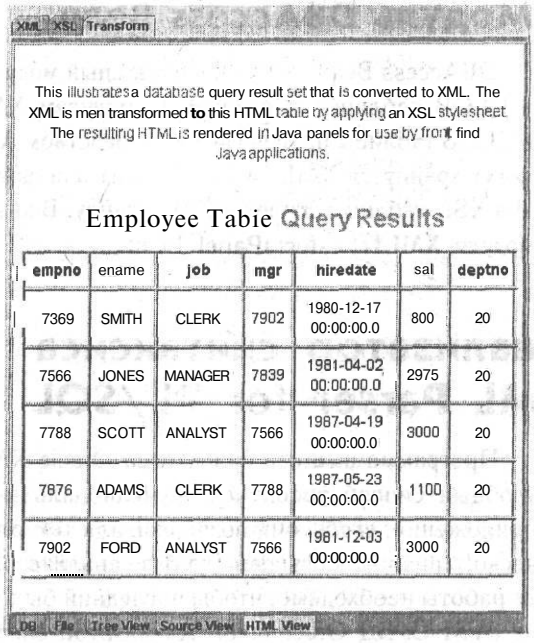


Рис. 2.8. *Модуль XML TransViewerBean показывает результат в окне просмотра HTML*

Модуль DBView Bean

Модуль **DBView** Bean — это визуальный модуль, который можно использовать для вывода на экран информации из базы данных, используя для этого XML- и XSL-преобразования. Это позволяет просматривать таблицы базы данных и столбцы данных в этих таблицах, как это показано в примере использования модуля XMLTransformPanel Bean.

Модуль DBAccess Bean

DBAccess Bean — это невизуальный модуль, который позволяет получать доступ в CLOB таблицы, управляемые модулем XMLTransformPanel Bean. Этот доступ в CLOB таблицы производится посредством API-интерфейса JDBC. В этих таблицах могут храниться XML- и XSL-файлы или даже файлы, полученные после применения XSL-таблицы стилей к XML-файлу. Все это показано в примере использования модуля **XMLTransformPanel** Bean.

Анализатор синтаксиса XML Parser for PL/SQL

Программа анализа синтаксиса Oracle XML Parser for PL/SQL предоставляет удобный способ расширить функциональные возможности имеющихся PL/SQL приложений, чтобы они поддерживали технологию XML. Эта программа синтаксического анализа построена на базе анализатора Oracle XML Parser for Java V2, для ее работы необходимо, чтобы последний был установлен в виртуальной Java-машине JVM СУБД Oracle8i/9i. Анализатор синтаксиса XML Parser for PL/SQL для проведения синтаксического анализа и манипуляций в XML-документе использует механизм хранимых Java-процедур Oracle8i/9i для вызова Java-анализатора. Поэтому анализатор нельзя запустить на исполнение в версии СУБД Oracle более ранней, чем Oracle8i/9i. XML Parser for PL/SQL полностью поддерживает спецификации XML, DOM, Namespace и XSLT, а поддержку SAX планируется реализовать в следующей версии этой программы.

Функции синтаксического анализа выполняют три PL/SQL-модуля, а именно: **xmlparser, xmldom и xslprocessor**. Модуль **xmlparser** содержит методы для синтаксического анализа XML-документа, в том числе отдельных строк, файлов или объектов CLOB. Он содержит и методы для анализа XML DTD, причем его можно настроить так, чтобы анализ XML DTD производился перед анализом самого XML-документа. Этот анализатор синтаксиса, подобно его Java-коллеге, можно настроить на проверку допустимости документа, а кроме того, в случае необходимости в нем можно установить опцию сохранения пробелов, что бывает нужно для обработки XML-документа XSLT-процессором. Модуль **xmldom** реализует DOM-интефейсы. Кроме того, он предоставляет методы для записи хранимых в памяти XML-документов или DTD в строку, файл или в CLOB. Модуль xslprocessor обеспечивает поддержку выполнения обработки XML-документов с помощью XSLT-процессора с использованием методов инкапсуляции XSLT API-интерфейсов в программу анализа синтаксиса Oracle XML Parser for Java.

Примеры

Часто представление типа документа DTD может храниться в базе данных в виде объекта CLOB. Обратиться к такому DTD непосредственно из XML-документа нельзя. Для проверки XML-документа на допустимость с помощью DTD нужно сначала провести синтаксический анализ DTD и зарегистрировать его в программе-анализаторе, а потом уже приступать к синтаксическому анализу самого XML-документа. Вот как это делается:

```
create or replace procedure dtdexample(dtd clob, root varchar2,
        inpfile varchar2, errfile varchar2) is
p xmlparser.Parser;
doc xmldom.DOMDocument;
dt xmldom.DOMDocumentType;
nl xmldom.DOMNodeList;

begin
-- new parser
   p := xmlparser.newParser;

-- set error log
   xmlparser.setErrorLog(p, dir || '/' || errfile);

-- parse dtd clob and register it
   xmlparser.parseDTDClob(p, dtd, root);
   dt := xmlparser.getDoctype(p);
   xmlparser.setDoctype(p, dt);

-- parse input file
   xmlparser.setValidationMode(p, true);
   xmlparser.parse(p, inpfile);

exception when others then
raise;
end dtdexample;
/
show errors;
```

В этом примере сначала создается новая копия программы-анализатора. Следует помнить о том, что для записи любых ошибок, которые могут иметь место во время синтаксического анализа, нужно определить файл, в который программа-анализатор должна отправлять сообщения об ошибках:

```
-- new parser
   p := xmlparser.newParser;

-- set some characteristics
   xmlparser.setErrorLog(p, dir || '/' || errfile);
```

Следующий шаг — синтаксический анализ DTD, его обработка и регистрация в программе-анализаторе:

```
— parse dtd dob and register it
   xmlparser.parseDTDClob(p, dtd, root);
   dt := xmlparser.getDoctype(p);
   xmlparser.setDoctype(p, dt);
```

Отметим, что при проведении синтаксического анализа DTD нужно определить корневой элемент XML-документа. До анализа XML-документа следует убедиться в том, что включена опция его проверки на допустимость. Затем можно вызывать метод синтаксического анализа, который впоследствии использует только что зарегистрированный DTD для проверки XML-документа на допустимость:

```
-- parse input file
   xmlparser.setValidationMode(p, true);
   xmlparser.parse(p, dir || '/' || inpfile);
```

В приведенном примере не генерируются никакие выходные данные, но в заданный файл будут записаны все ошибки (в том числе и ошибки проверки документа на допустимость).

Выше на примере программы-анализатора XML Parser for Java было показано, использование поддержки XSLT с помощью классов **XSLStylesheet и XSLProcessor.** В программе XML Parser for PL/SQL эта функциональная возможность обеспечивается с помощью интерфейсов **xslprocessor.Stylesheet и xslprocessor.Processor.** Кроме этих интерфейсов программа синтаксического анализа предоставляет методы использования выражений XPath для извлечения данных из дерева DOM. Это демонстрирует код:

```
create or replace procedure xslexample is

p xmlparser.Parser;
xmlbuf varchar2(512);
xslpat varchar2(512);
xmldoc xmldom.DOMDocument;
xmldocnode xmldom.DOMNode;
n xmldom.DOMNode;
begin
   xmlbuf := '<timegram><time>12:00</time></timegram>';
   xslpat := 'timegram/time/text()';

-- new parser
   p := xmlparser.newParser;

-- parse xml buffer
   xmlparser.parseBuffer(p, xmlbuf);
```

```
-- get document
   xmldoc := xmlparser.getDocument(p);

-- select a single node matching the XPath expression and print its value
   xmldocnode := xmldom.makeNode(xmldoc);
   n := xslprocessor.selectSingleNode(xmldocnode, xslpat);
   dbms_output.put_line('Value of the first selected node is: ' ||
                        xslprocessor.valueOf(n, '.'));
exception when others then
   raise;
end xslexample;
/
show errors;
```

Этот пример демонстрирует, как легко и просто можно использовать XPath с деревом DOM, построенным с помощью программы синтаксического анализа Oracle XML. Для иллюстрации этого утверждения определим небольшой XML-документ, содержащий время и выражение XPath, которое возвращает значение времени:

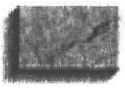
```
xmlbuf := '<timegram><time>12:00</time></timegram>';
xslpat := 'timegram/time/text()';
```

Обычно нужно провести синтаксический анализ XML-документа и обработку корня дерева DOM:

```
-- new parser
   p := xmlparser.newParser;
-- parse xml buffer
   xmlparser.parseBuffer(p, xmlbuf);
-- get document
   xmldoc := xmlparser.getDocument(p);
```

Теперь можно использовать методы, предоставляемые модулем xslprocessor для поиска в дереве DOM с использованием заданного выражения XPath. В этом примере для поиска первого узла, соответствующего заданной картине, используется метод **selectSingleNode**. Но если предполагается, что данной картине будут соответствовать сразу несколько узлов, и нужно найти все такие узлы, вместо метода selectSingleNode используется метод selectNodes. После получения узла будет получено и его значение:

```
-- select a single node matching the XPath expression and
-- print its value
   xmldocnode := xmldom.makeNode(xmldoc);
   n := xslprocessor.selectSingleNode(xmldocnode, xslpat);
   dbms_output.put_line('Value of the first selected node is: ' ||
                        xslprocessor.valueOf(n, '.'));
```



Внимание

Вместе с XPath-выражением '.' используется метод **valueOf**. *Это эквивалентно утверждению XSLT-процессора* **'<xsl:value-of select=".">'**.

В предыдущем примере сгенерированы следующие данные:

```
Value of the first selected node is: 12:00
```

что и следовало ожидать.

Анализатор синтаксиса XML Parser и процессор XSLT Processor for C

Программа-анализатор синтаксиса Oracle XML Parser с интегрированным процессором XSLT Processor for C поставляется в двух вариантах: в виде автономной исполняемой программы с интерфейсом командной строки и в виде библиотеки для приложений. Большинство пользователей пишут свои собственные приложения, поэтому им понадобится библиотека XML. А исполняемую программу пользователи-новички могут использовать для того, чтобы познакомиться с технологией XML и провести синтаксический анализ и проверку на допустимость своих тестовых документов. Кроме того, исполняемую программу можно использовать для применения таблиц стилей к XML-документу.

Сообщения программы-анализатора и XSLT-процессора сохраняются во внешнем файле (для глобализации), поэтому среда разработки должна сообщить анализатору и процессору, где этот файл находится. Если определен каталог **ORACLE_HOME**, в этом файле он будет выглядеть как **$ORACLE_HOME/mesg**. В противном случае в параметре **ORA_XML_MESG** нужно задать *полный* путь к каталогу, содержащему файл сообщений. Отметим, что этот файл имеет расширение .msb и содержит все текущие сообщения. Анализатор будет также генерировать файл **.msg**, в котором содержится информация об ошибках.

Автономный анализатор синтаксиса с интегрированным XSLT-процессором

Автономный анализатор синтаксиса с интегрированным XSLT-процессором с интерфейсом командной строки (под названием **xml**) поставляется в составе пакета языка C в подкаталоге bin/. Отметим, что при работе под OG Windows NT совместно используемые XML DLL-библиотеки должны находиться в том же каталоге, что и исполняемый файл анализатора синтаксиса, либо располагаться в каталоге, записанном в пути поиска. XML DLL-библиотеки устанавливаются по умолчанию в каталог bin/, и если запуск **xml** производится из него, никаких дополнительных действий не требуется.

Команда запуска автономного анализатора выглядит следующим образом:

xml [switches] *document*

где *document* — это название входного XML-документа, а *switches* — это необязательные ключи управления, которые позволяют менять параметры работы анализатора. Если в данной команде не задано никакого документа, если ключи указаны неправильно или указан ключ **-h** (режим помощи), будет напечатано сообщение:

```
% xml
No document specified.
Usage: xml [switches] [document]
   -c                Comformance check only, no validation
   -e <encoding>     Specify default input file encoding
   -f                File - Interpret <document> as filespec, not TOI
   -h                Help - show this usage help
   -1 <language>     Language for errorreporting
   -p                Print document/DTD structures after parse
   -s <stylesheet>   Stylesheet - specifies the stylesheet
   -r                Exercise the stream interface for XSLT
   -v                Version - show parser version and exit
   -w                Whitespace - preserve ALL whitespace
   -W                Warning - stop parsing after a warning
   -x                Exercise SAX interface for parser (prints document)
```

Если документ задан, а ключи на указаны, программа-анализатор проводит синтаксический анализ этого документа и его проверку на допустимость, после чего заканчивает свою работу с нулевым кодом возврата (что указывает на успешное выполнение задачи). Если произошла ошибка, печатается сообщение о ней и выдается код ошибки (ненулевой).

Ключи программы анализатора подробно описаны в таблице 2.2.

Таблица 2.2

Ключи командной строки анализатора синтаксиса xml

Ключ	Описание
-c	**Проверяется только соответствие.** По умолчанию анализатор синтаксиса XML проверяет документ на соответствие своему DTD (если он один). Когда задан ключ -c, документ синтаксически анализируется и тестируется на правильность, но не на допустимость.
-e *encoding*	**Кодировка.** Устанавливает кодировку входного файла *по умолчанию.* Для некоторых типов файлов входная кодировка может определяться автоматически (UCS-4, UTF-16, EBCDIC и т. д.) на основе маркера последовательности байтов, который стоит в начале такого файла. Для остальных файлов предполагаемый тип кодировки может определяться ключом **-e**. Если он не задан, по умолчанию принимается кодировка UTF-8. Кодировка ASCII

Таблица 2.2 (*продолжение*)

Ключи командной строки анализатора синтаксиса ***xml***

Ключ	Описание
-e *encoding* *(продолжение)*	встречается вдвое чаще, чем UTF-8, и если известно, что входной файл имеет кодировку ASCII, рекомендуется установить ключ -e ASCII. Более подробную информацию можно найти в спецификации XML 1.0 в приложении F (Автоматическое определение кодировки символов). Поддерживаются кодировки: US-ASCII, ASCII, UTF-8, UTF-16, ISO-10646-UCS-2, ISO-8859-1, ISO-8859-2, ISO-8859-3, ISO-8859-4, ISO-8859-5, ISO-8859-6, ISO-8859-7, ISO-8859-8, ISO-8859-9, ISO-8859-10, EUC-JP, SHIFT_JIS, BIG5, GB2312, KOI8-R, EBCDIC, EBCDIC-CP-{US,CA,NL,WT,DK,NO,FI,SE,IT, ES,GB,FR,HE,BE,CH,ROECE,YU,IS}.
-h	**Помощь.** Распечатывается краткий список опций, как было показано выше.
-1 *language*	**Язык.** Устанавливает язык, используемый для сообщений программы анализа синтаксиса. По умолчанию используется американский английский.
-p	**Печать.** После успешного завершения синтаксического анализа входного документа (и, возможно, его проверки на допустимость) распечатывается древовидная структура документа, начиная с его корневого *элемента.* Если заданы два ключа -p (например, -p -p или -pp), то печать документа начинается с корневого *узла.*
-s *stylesheet*	**Таблица стилей.** После успешного завершения синтаксического анализа документ пропускается через XSL-процессор с использованием заданной таблицы стилей.
-v	**Версия.** Распечатывается номер версии программы-анализатора, после чего ее работа заканчивается.
-w	**Пробелы.** Сообщение программе синтаксического анализа о сохранении *всех* пробелов. Этот анализатор отличается от стандартной версии 1.0, в которой по умолчанию опускались пробелы, используемые для форматирования. Если ключ **-w** не задан, то текст между элементами разметки, состоящий *только* из пробелов, будет опущен. Такие пробелы обычно ставятся в конце каждой строки и в начале строк для форматирования документов.

Библиотека анализатора синтаксиса/XSLT-процессора

Как уже отмечалось, анализатор синтаксиса с интегрированным XSLT-процессором может поставляться в виде библиотеки. В Oracle8i эта библиотека называется **libxml8.a** (или **libxml8.dll** для Windows NT), в Oracle9i — соответственно **libxml9.a** или **libxml9.dll.** Это та же самая библиотека, которая используется автономной программой-анализатором с XSLT-процессором. Она предназначается для запуска

в СУБД Oracle версии 8.1.6 или более поздней. Для того чтобы эту библиотеку можно было использовать с Oracle 8.1.5, поставляется специальная библиотека совместимости **libxmlc8.a** (или **libxml8c.dll**), которую нужно соответствующим образом подключить.

Эта библиотека содержит API-интерфейс для инициализации, синтаксического анализа файла или содержимого буфера, возврата в исходное состояние и завершения работы, плюс все средства для создания представления DOM, интерфейс SAX и XSLT-процессор. Обычно библиотека выполняет следующие действия: *инициализация*, *синтаксический анализ, завершение работы.* Если нужно проанализировать несколько документов, последовательность действий будет следующей: *инициализация, синтаксический анализ, синтаксический анализ, ..., завершение работы.* Все данные, представленные анализатором синтаксиса, остаются действительными до завершения работы или очистки буфера. Если же нужны результаты синтаксического разбора каждого файла без сохранения старых данных, то последовательность действий должна быть другой: *инициализация, синтаксический анализ, очистка, синтаксический анализ, очистка, ..., завершение работы.* Для применения к XML-документам таблиц стилей перед *завершением работы* программы или перед операцией *очистки* нужно добавить вызов функции xslprocess.

Инициализация

Программу синтаксического анализа перед использованием нужно инициализировать с помощью функции **xmlinit**. Каждое обращение к функции инициализации возвращает указатель контекста типа (xmlctx *), который должен быть передан во все последующие обращения. Можно сделать и несколько инициализирующих обращений, которые вернут несколько независимых контекстов. Выглядеть это будет следующим образом:

```
xmlctx *xmlinit(uword *err, const oratext *encoding,
                void (*msghdlr) (void *msgctx, const oratext *msg,
                                 uword errcode),
                void *msgctx, const xmlsaxcb *saxcb,
                void *saxcbctx, const xmlmemcb *memcb,
                void *memcbctx, const oratext *lang);
```

Параметры для функции xmlinit подробно описаны в таблице 2.3.

Таблица 2.3

Параметры обратного вызова для функции xmlinit

Параметр	Ввод/вывод	Описание
err	вывод	Возвращенный код ошибки. В случае успешного выполнения операции возвращается указатель контекста. В случае ошибки возвращается указатель NULL, а err принимает числовое значение кода ошибки, указывающего на существование конкретной проблемы. Является обязательным параметром.

Таблица 2.3 (*продолжение*)

Параметры обратного вызова для функции ***xmlinit***

Параметр	Ввод/вывод	Описание
encoding	ввод	Кодировка по умолчанию XML-файлов или буферов, если кодировку нельзя определить автоматически. Является эквивалентом ключа -в в описании вызова автономной программы синтаксического анализа (см. выше документацию для ключа -в, где приведен список поддерживаемых кодировок). Может принимать значение NULL, в этом случае кодировкой по умолчанию будет UTF-8.
msghdlr	ввод	Указатель на функцию обработки сообщения. Если его значение равно NULL, записываются сообщения о стандартных ошибках. Если этот параметр поддерживается, сообщения об ошибках форматируются и передаются в функцию msghdlr вместе с числовым кодом ошибки.
msgctx	ВВОД	Указатель контекста, который передается в вызовы функции msghdlr. Это необязательный параметр. Его значение просто передается в вызовы программы обработки сообщений. Для работы программы-анализатора он не имеет никакого значения и нужен только пользователю.
saxcb	ввод	Структура обратного вызова интерфейса SAX. Если этот параметр установлен, интерфейс SAX будет использоваться для возврата информации о синтаксически проанализированном документе. Если его значение равно NULL, будет использоваться интерфейс DOM. Следует отметить, что все функции обратного вызова SAX в этой структуре являются необязательными. См. следующий пункт, касающийся интерфейса SAX.
saxcbctx	ввод	Указатель контекста обратного вызова SAX. Как и указатель контекста сообщений, он является необязательным параметром, определяемым пользователем. Если этот параметр введен, он просто передается во все функции обратного вызова SAX. Он имеет значение только для пользователя, но не для программы синтаксического анализа.
memcb	ввод	Структура обратного вызова при обращении к памяти. Если этот параметр не задан, программа-анализатор сама распределяет и высвобождает память. Однако пользователь может при желании выбрать другие функции распределения, перераспределения и высвобождения памяти.

Таблица 2.3 (*продолжение*)

Параметры обратного вызова для функции ***xmlinit***

Параметр	Ввод/вывод	Описание
memcbctx	ввод	Контекст обратного вызова при обращении к памяти. Как и в случае с обратными вызовами интерфейса SAX, пользователь может сам определить указатель контекста, который должен передаваться во все функции обратного вызова при обращении к памяти.
lang	ввод	Язык для сообщений об ошибках. В текущей версии этот параметр игнорируется; по умолчанию предполагается, что этим языком является американский английский.

Анализ синтаксиса

Анализом синтаксиса XML-документов непосредственно занимаются две функции: xmlparse и xmlparsebuf. Функция xmlparse производит анализ документа, который хранится во внешнем файле, а xmlparsebuf делает то же самое с документом, находящимся в буфере памяти. Третья функция xmlparsedtd проводит синтаксический анализ только внешних DTD и используется генератором классов (более подробно об этом см. ниже). Все три функции возвращают числовой код ошибки: ноль в случае успешного завершения операции и ненулевой в случае ошибки. Обращение к функциям выглядит следующим образом:

```
uword xmlparse(xmlctx *ctx, const oratext *filename,
               const oratext *encoding, ub4 flags);
```

Параметры функции xmlparse подробно описаны в таблице 2.4.

Таблица 2.4

Параметры функции xmlparse

Параметр	**Ввод/вывод**	**Описание**
ctx	ввод	Указатель контекста, возвращаемый функцией xmlinit
filename	ввод	Путь к XML-документу, который нужно синтаксически проанализировать
encoding	ввод	Кодировка документа по умолчанию, если эту кодировку нельзя определить автоматически. Заменяет кодировку по умолчанию, передаваемую в функцию xmlinit.
flags	ввод	Шаблон для флаговых битов, которые изменяют поведение программы-анализатора. См. следующую таблицу.

Флаговые биты программы синтаксического анализа показаны в таблице 2.5.

Таблица 2.5

Флаговые биты функции ***xmlparse***

Название флага	Функция (после установки)
XML_FLAG_VALIDATE	Проверяет проанализированный документ на соответствие его DTD (если он один).
XML_FLAG_DISCARD_WHITESPACE	Отбрасывает пробелы, используемые только для форматирования (в конце строк и в их начале).
XML_FLAG_DTD_ONLY	Анализирует синтаксис только внешнего DTD. Установка этого флага аналогична вызову функции xmlparsedtd.

Функция **xmlparsebuf** аналогична функции **xmlparse**, но она направляет указатель в буфер **(buffer),** а вместо имени файла в качестве параметра в ней фигурирует длина буфера (len). Остальные параметры те же самые, что в функции **xmlparse:**

```
uword xmlparsebuf (xmlctx *ctx, const oratext *buffer, size_t len,
                   const oratext *encoding, ub4 flags);
```

Очистка

После завершения синтаксического анализа документа указатели на данные, возвращаемые через вызовы DOM или SAX, остаются допустимыми до явной очистки или завершения вызова (т. е. выделенная память не освободилась). Отметим, что интерфейс DOM может внимательно рассмотреть и модифицировать только *текущий* документ (последний из синтаксически проанализированных). Если между вызовами функций инициализации и завершением работы проводится анализ синтаксиса нескольких документов, но старые данные из предыдущих операций анализа синтаксиса не нужны, следует после каждой такой операции вызывать функцию очистки **xmlclean.** Это освобождает всю выделенную память, но нужно помнить, что в этот момент указатели на старые данные становятся недействительными. Вызов функции **xmlclean** записывается так:

```
void xmlclean(xmlctx *ctx);
```

Параметр ctx здесь — это тот самый указатель контекста, который был возвращен функцией xmlinit.

Завершение работы

Функция завершения **xmlterm** прекращает работу программы синтаксического анализа и освобождает всю выделенную память. Программу-анализатор нельзя использовать до тех пор, пока не будет сделан еще один вызов функции **xmlinit.** Если нужно провести анализ синтаксиса нескольких файлов за один сеанс, то между

файлами используется функция xmlclean, чтобы высвободить память, либо всякий раз выполняются функции инициализации и завершения работы **xmlinit/xmlterm**. Вызов этих функций выглядит так:

```
uword xmlterm(xmlctx *ctx);
```

Эта функция возвращает код ошибки, который в случае удачного завершения равен нулю, а в случае ошибки — отличен от нуля.

Пример

Здесь представлена простейшая скелетная программа, демонстрирующая использование функций программы синтаксического анализа верхнего уровня.

```
main()
{  xmlctx *ctx;
   uword ecode;

   if (!(ctx = xmlinit(&ecode, NULL, NULL, NULL, NULL,
                       NULL, NULL, NULL, NULL)))
   {  /* An error occurred, ecode holds the error code */
      return 1;
   }
   if (ecode = xmlparse(ctx, "testdoc.xml", NULL, XML_FLAG_VALIDATE))
   {  /* Parse or validation failed, error code in ecode */
      return 1;
   }
   /* Manipulate parsed data with DOM interface here */
   (void) xmlterm(ctx);
   return 0;
}
```

Если в предыдущем примере требовалось провести анализ дополнительных файлов, одновременно сохраняя данные от предыдущих операций анализа, нужно повторить вызов функции xmlparse, прежде чем завершить работу. Но помните, что через интерфейс DOM можно получить доступ только к самому *последнему* (текущему) документу, и никакие указатели на данные для предыдущих документов уже не будут допустимыми. Если нужно провести анализ нескольких файлов, но без сохранения старых данных, вызывайте всякий раз функцию xmlclean. Вот пример такого вызова (проверки кода возврата здесь пропущены):

```
for (i = 1; i <= 9; i++)
{  sprintf(filename, "file%d.xml", i);
   (void) xmlparse (ctx, filename, NULL, XML_FLAG_VALIDATE);
   xmlclean (ctx);
}
```

API-интерфейс Document Object Model (DOM)

API-интерфейс DOM используется для составления запросов и проведения манипуляций с синтаксически проанализированным документом или для создания нового документа с самого начала. Интерфейс DOM выбирается, пока в функцию xmlinit не отправляется структура обратного вызова SAX (при этом происходит отключение DOM и включение SAX). DOM — это просто набор функций, составляющих API-интерфейс. После того как документ прошел синтаксический анализ (и, возможно, был признан допустимым), можно вызывать функции DOM для отбора данных, поиска, редактирования и т. д.

DOM — это сложный API-интерфейс со многими вызовами. Здесь они не описаны по причине большого объема, но подробности содержатся в спецификации DOM Level 1 консорциума W3C по адресу **http://www.w3c.org/TR/REC-DOM-Level-1/**.

Так как DOM — это объектно-ориентированная спецификация, а язык C таковым *не является,* нужно произвести некоторые изменения. В частности, пространство имен функции C является плоским, поэтому имена методов DOM, одинаковые в нескольких различных классах, нужно изменить, чтобы они стали уникальными, как описано в таблице 2.6.

Таблица 2.6

API-интерфейсы DOM

DOM-имя	C=имя
`Attr::getName, . . .`	`getAttrName, ...`
`CharacterData::getData, ...`	`getCharData, . . .`
`DocumentType::getName, . . .`	`getDocTypeName, ...`
`Entity::getPublicId, ...`	`getEntityPublicId, . . .`
`NameNodeMap::item`	`getChildNode`
`NameNodeMap::getLength`	`getNodeMapLength`
`NodeList::item`	`getChildNode`
`NodeList::getLength`	`getNodeMapLength`
`Notation::getPublicId, ...`	`getNotationPublicId, ...`

Подробную информацию о заголовочном файле oraxml.h программы синтаксического анализа можно получить в документации по всем функциям DOM.

Simple API for XML (SAX)

Simple API for XML (*SAX*)— это альтернатива API-интерфейса для доступа к синтаксически проанализированному документу. Для использования интерфейса SAX в функцию xmlinit передается набор функций обратного вызова, после чего

программа-анализатор вызывает эти функции, когда ей встречаются соответствующие части документа. Можно сравнить эту технологию с DOM, где после синтаксического анализа документа в памяти строится дерево узлов, куда потом можно отправлять запросы и которое можно модифицировать через API-интерфейс DOM. Функции интерфейса SAX вызываются в процессе синтаксического анализа документа. Каждая функция SAX возвращает код ошибки (**sword**). Если этот код не равен нулю, выдается сообщение об ошибке, и синтаксический анализ документа сразу же прекращается. Структура обратного вызова SAX (xmlsaxcb) определяется следующим образом:

```
struct xmlsaxcb
{
   sword (*startDocument) (void *ctx);
   sword (*endDocument) (void *ctx);
   sword (*startElement) (void *ctx, const oratext *name,
                          const struct xmlnodes *attrs);
   sword (*endElement) (void *ctx, const oratext *name);
   sword (*characters) (void *ctx, const oratext *ch, size_t len);
   sword (*ignorableWhitespace) (void *ctx, const oratext *ch,
                                 size_t len);
   sword (*processingInstruction) (void *ctx, const oratext *target,
                                   const oratext *data);
   sword (*notationDecl) (void *ctx, const oratext *name,
                          const oratext *publicId,
                          const oratext *systemId);
   sword (*unparsedEntityDecl) (void *ctx, const oratext *name,
                                const oratext *publicId,
                                const oratext *systemId,
                                const oratext *notationName);
   sword (*nsStartElement) (void *ctx, const oratext *qname,
                            const oratext *local,
                            const oratext *nsp,
                            const struct xmlnodes *attrs);
}
```

Можно определить любую из функций обратного вызова и даже все такие функции, но ни одна из них не является обязательной. Можно также задать необязательный указатель контекста, который будет передаваться во все функции обратного вызова. Его использование полностью зависит от желания самого пользователя.

Функции обратного вызова подробно описаны в таблице 2.7.

Таблица 2.7

Функции обратного вызова SAX

Функция обратного вызова	Описание
startDocument	Вызывается непосредственно перед началом синтаксического анализа.
endDocument	Вызывается сразу же после успешного окончания синтаксического анализа.
startElement	Вызывается, когда встречается начальный тег элемента. Если имеется версия этой функции для пространства имен, то вызывается именно она.
endElement	Вызывается, когда встречается конечный тег элемента.
characters	Вызывается для каждого блока текстовых данных CDATA или #PCDATA.
ignorablewhitespace	Вызывается при каждом игнорировании пробела, если специально не оговорено сохранение всех пробелов (в случае вызова функции characters).
processingInstruction	Вызывается для каждой команды обработки.
notationDecl	Вызывается для каждого описания NOTATION в DTD.
unparsedEntityDecl	Вызывается для каждого не анализированного объекта.
nsStartElement	Версия пространства имен функции startElement, которая выдает имя элемента, разбитого на пространство имен, локальный элемент и т. д.

Следующие программные фрагменты показывают, как описываются, регистрируются и используются функции обратного вызова SAX:

```
/* declare SAX callback functions */
sword startdocument(void *ctx);
sword enddocument(void *ctx);
sword startelement(void *ctx, const oratext *name,
                   const xmlnodes *attrs);
sword endelement(void *ctx, const oratext *name);
sword characters(void *ctx, const oratext *ch, size_t len);
sword whitespace(void *ctx, const oratext *ch, size_t len);
sword pi(void *ctx, const oratext *target,
         const oratext *data);
sword notation(void *ctx, const oratext *name,
               const oratext *publicId,
               const oratext *systemId);
```

```
sword entity(void *ctx, const oratext *name,
             const oratext *publidId,
             const oratext *systemId,
             const oratext *notationName);
/* declare SAX callback context */
typedef struct
   uword    depth;        /* nested element level, for indenting */
} sax_context;

/* declare SAX callback structure */
xmlsaxcb sax_callback = {
   startdocument, enddocument, startelement, endelement,
   characters, whitespace, pi, notation, entity
};
/* declare SAX context and initialize */
sax_context saxctx = { 0 };          /* depth = 0 */
/* initialize parser specifying SAX callbacks */
xmlinit(&ecode, NULL, NULL, NULL, NULL,
        &sax_callback, (void *) &saxctx, NULL, NULL);

/* ----- SAXCALLBACKS ----- */
sword startdocument (void *context)
{puts ("StartDocument");
   return 0;    /* success */
}

sword enddocument (void *context)
{puts ("EndDocument");
   return 0;    /* success */
}

sword startelement (void *context, const oratext *name, const xmlnodes *attrs)
{  sax_context *saxctx = (sax_context *) context;
   indent(saxctx->depth);
   printf("<%s", name);
   if (attrs)
   {  for (i = 0; i < numAttributes(attrs); i++)
      {  attr = getAttributeIndex(attrs, i);
         printf("%s=\"%s\"",getAttrName(attr),getAttrValue(attr));
      }
   }
   puts(">");
   saxctx->depth++;
   return 0;    /* success */
}
```

```
sword endelement (void *context, const oratext *name)
{  sax_context *saxctx = (sax_context *) context;
   indent(-saxctx->depth);
   printf ("</%s>\n", name);
   return 0;    /* success */
}

sword characters (void *context, const oratext *ch, size_t len)
{  sax_context *saxctx = (sax_context *) context;
   indent(saxctx->depth);
   putchar( ' "');
   print_string ( (oratext *) ch, (sword) len);
   puts("\"");
   return 0;    /* success */
}

sword whitespace (void *context, const oratext *ch, size_t len)
{  sax_context *saxctx = (sax_context *) context;
   indent(saxctx->depth);
   putchar( ' \ ' ' ) ;
   print_string ( (oratext *) ch, (sword) len);
   puts ("'");
   return 0;    /* success */
}

sword pi (void *context, const oratext *target, const oratext *data)
{  sax_context *saxctx = (sax_context *) context;
   indent (saxctx->depth);
   fputs( "PI", stdout);
   if (target)
      printf (" target=\"%s\"", target);
   if (data)
      printf (" data=\"%s\"", data) ;
   putchar ('\n') ;
   return 0;    /* success */
}

sword notation(void *context, const oratext *name, const oratext *publicId,
               const oratext *systemId)
{  sax_context *saxctx = (sax_context *) context;
   indent(saxctx->depth);
   printf ("NOTATION '%s'", name) ;
   if (publicld)
      printf C PUB:%s", publicld);
```

```
    if (systemId)
      printf(" SYS:%s", systemId);
    putchar('\n');
    return 0;    /* success */
}

sword entity(void *context, const oratext *name, const oratext *publidId,
             const oratext *systemId, const oratext *notationName)
{  sax_context *saxctx = (sax_context *) context;
    indent (saxctx->depth);
    printf("ENTITY '%s'", name);
    if (publidId)
      printf(" PUB:%s", publidId);
    if (systemld)
      printf(" SYS:%s", systemId);
    if (notationName)
      printf(" NAME:%s", notationName);
    putchar('\n');
    return 0;    /* success */
}
```

Вот пример XML-документа:

```
<?xml version="1.0"?>
<!DOCTYPE PLAY [
   <!ELEMENT top      (second*)>
   <!ELEMENT second   (third*)>
   <!ELEMENT third    (#PCDATA)*>
   <!NOTATION note1 SYSTEM "foo.exe*>
   <!NOTATION note2 PUBLIC "bar" "bar.ent">
   <!ENTITY ent SYSTEM "http://www.w3.org/" NDATA n>
]>
<?dummy this is a sample processing instruction?>
<top>
   <second>
      <third>third level</third>
   </second>
</top>
```

А вот что получится при обработке данного документа программой, представленной выше:

```
StartDocument
NOTATION 'note1' SYS:foo.exe
NOTATION 'note2' PUB:bar SYS:bar.ent
```

```
ENTITY 'ent' SYS:http://www.w3.org/ NAME:n
PI target=dummy data=this is a sample processing instruction
<top>
   '\n '
   <second>
      •\n '
      <third>
         "third level"
      </third>
      '\n '
   </second>
   '\n'
</top>
EndDocument
```

Поддержка процессора XSLT

Как уже говорилось, поддержка процессора XSLT интегрирована в программу синтаксического анализа, и этот процессор можно вызвать с помощью команды *xslprocess*. Пример такого использования:

```
/* Parse the XML document */
if ( ! (xctx= xmlinit (&ecode, (const oratext *) 0,
                  (void (*) (void *, const oratext *, uword) ) 0,
                  (void *) 0, (const xmlsaxcb *) 0, (void *) 0,
                  (const xmlmemcb *) 0, (void *) 0, (const oratext *) 0)))
{
   ...
}

if (ecode = xmlparse(xctx, (oratext *) argv[1] , (oratext *) 0,
                  XML_FLAG_VALIDATE I XML_FLAG_DISCARD_WHITESPACE) )
{
   ...
}

/* Parse the XSL document */
if ( ! (xslctx= xmlinit (&ecode, (const oratext *) 0,
                  (void (*} (void *, const oratext *, uword)) 0,
                  (void *) 0, (const xmlsaxcb *) 0, (void *) 0,
                  (const xmlmemcb *) 0, (void *) 0, (const oratext *) 0)))
{
   ...
}
```

```
if (ecode = xmlparse(xslctx, (oratext *) argv[2], (oratext *) 0,
                     XML_FLAG_VALIDATE | XML_FLAG_DISCARD_WHITESPACE)) .
{
   ...
}

/* Initialize the result context */
if (!(resctx = xmlinit(&ecode, (const oratext *) 0,
                       (void (*) (void *, const oratext *, uword)) 0,
                       (void *) 0, (const xmlsaxcb *) 0, (void *) 0,
                       (const xmlmemcb *) 0, (void *} 0,
                       (const oratext *) 0)))
{
   ...
}

/* XSL processing */
if (ecode = xslprocess(xctx, xslctx, resctx, &result)
{
   ...
  }

/* Print the result tree */
printStream(stdout, result, 4, 0);

/* Call the terminate functions */
(void)xmlterm(xctx);
(void)xmlterm(xslctx);
(void)xmlterm(resctx);
```

Поддержка XML Schema

Кроме поддержки DTD в инструментарии разработчика Java XML Development Kit появилась поддержка проверки XML-документов на соответствие XML Schema. В комплекте поставки программного пакета языка C в подкаталоге **bin/** имеется автономный процессор XML Schema с интерфейсом командной строки (он называется **schema).** При работе под ОС Windows NT совместно используемые XML DLL-библиотеки должны находиться в том же каталоге, что и исполняемый файл процессора XML Schema. XML DLL-библиотеки устанавливаются по умолчанию в каталог bin/, и если запуск **xml** производится из этого каталога, то никаких дополнительных действий не требуется.

Команда запуска автономного процессора выглядит следующим образом:

schema [*switches*] *document* [*schema*] *[working directory*]

где *document* — это название входного XML-документа (обязательный параметр), *switches* — необязательные ключи управления, которые позволяют менять параметры работы анализатора, *schema* — процессор XML Schema. Что такое *working directory* (рабочий каталог) понятно без объяснений. Если в данной команде не задано никакого документа, ключи указаны неправильно или указан ключ **-h** (режим помощи), будет напечатано сообщение:

```
% schema
Usage: schema [flags] <instance> [schema] [working dir]
Where:
      <instance>          is the XML instance document to validate (required)
      [schema]            is the default schema (optional)
      [workingdir]        is the working directory for processing (optional)
Flags:
      -0                  Always exit with code 0 (success)
      -p                  Print, instance document to stdout on success
      -v                  Show version numbers
```

Библиотека процессора XML Schema

API-интефейсы процессора XML Schema могут поставляться в виде библиотеки под именем **libxsd8.a** (или **libxsd8.dll** для Windows NT), в Oracle9i — **libxsd9.a** или **libxsd9.dll** соответственно. Это та же библиотека, которая используется автономной программой-анализатором с XSLT-процессором. Она предназначается для запуска в СУБД Oracle версии 8.1.6 или более поздней.

Эта библиотека содержит API-интерфейсы для инициализации (*schemaInitialize*), загрузки XML Schema (*schemaLoad*), проверки XML-документа на соответствие XML Schema (*schemaValidate*) и завершения этой операции *(schemaTerminate)*. Обычно используются следующие функции: *xmlinit*, *xmlparse*, *schemaInitialize*, *schemaValidate* и *schemaTerminate*. Например:

```
/* Parser */
if (!(ctx= xmlinit(&ecode, (const oratext *) 0,
                   (void (*) (void *, const oratext *, uword) ) 0,
                   (void *) 0, (const xmlsaxcb *) 0, (void *) 0,
                   (const xmlmemcb *) 0, (void *) 0,
                   (const oratext *) 0)))
{
   ...
}
if (ecode = xmlparse (ctx, (oratext *) doc, (oratext *) 0,
                      XML_FLAG_DISCARD_WHITESPACE))
{
   ...
}
```

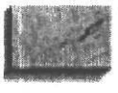
```
/* Schema */
if (!(scctx= schemaInitialize(ctx, &ecode) ) )
{
   ...
}

/*Validating XML document against XML Schema*/
if (ecode = schemaValidate (scctx, root, (oratext *) schema))
{
   ...
}

schemaTerminate(scctx);
```

XML Parser, XSLT Processor, XML Schema Processor for C++

Программа-анализатор синтаксиса Oracle XML Parser с интегрированным XSLT-процессором Processor и процессор XML Schema Processor for C++ представляют собой написанную на языке С программу синтаксического анализа с процессорами XSLT и XML Schema с упаковщиком, позволяющим получить к ней доступ из C++. Это ПО находится в той же XML-библиотеке **libxml8.a** (или **libxml9.a** в Oracle9i) и **libxsd8.a** или **libxsd9.a** для процессора XML Schema Processor. Все, что говорилось ранее по поводу С-анализатора, XSLT-процессора и процессора XML Schema, остается в силе и для версии C++, за исключением того, что упаковщики C++ могут предоставлять DOM API-интерфейс в объектно-ориентированном виде, поэтому переименований функций не потребуется. В C++ DOM-функции поименованы и используются точно так же, как в спецификации DOM. Более подробную информацию о функциях, имеющихся в версии C++, можно получить в заголовочном файле C++ DOM, который называется **oraxmldom.h**.

Генератор классов C++

Генератор классов C++ Class Generator — это средство, которое генерирует классы C++ (исходный и заголовочный файл) на основе DTD. Затем эти классы используются в программе для построения документов, которые проще согласовать с DTD.



Внимание

Генератор классов АЛЯ языка С не существует, так как С не является объектно-ориентированным языком и в нем нет концепции класса.

Генератор классов поставляется либо в автономном виде с интерфейсом командной строки (**xmlcg**), либо в виде библиотеки (**libxmlg8.a** или **libxmlg9.a**). Один класс определяется в заголовочном файле oraxmlcg.h, он называется **XMLClassGenerator**. Он имеет один метод generate:

```
class XMLClassGenerator
{ public;
  uword    generate(DocumentType *dtd, char *outdir = (char *) 0);
};
```

Для того чтобы использовать этот метод, нужно провести синтаксический анализ документа и DTD с помощью программ xmlparse или xmlparsebuf, а затем загрузить DTD с помощью метода **XMLParser::getDocType** и вызвать generate. Если выходной каталог (outdir) не определен, генерируемые файлы будут помещаться в текущий каталог. Если нужно генерировать классы на базе внешнего DTD, а не полного документа, следует провести синтаксический анализ DTD с помощью программы xmlparsedtd, а затем действовать, как уже было показано.

Из каждого определенного элемента генерируется один класс. Он представляет методы создания узлов элемента, соответствующих определению этого элемента.

Существуют два метода построения: все сразу с помощью конструкторов C++ или постепенно, с помощью пустого конструктора и дополнительных функций. В первом случае за один вызов создается элемент и его дочерний элемент, а во втором сначала создается пустой элемент, а его дочерние элементы создаются по одному с помощью вызовов addNote и addData.

Для каждой возможной комбинации дочерних элементов для заданного элемента, который использует * (ноль и более) или + (одна и более) модификаций, предоставить конструкторы невозможно, поскольку таких возможных комбинаций существует бесконечное множество. Поэтому для каждого типа * или + предоставляется только один конструктор. Если нужно сделать элемент с более сложным набором дочерних элементов, следует начать с пустого элемента и надстраивать его.

Например, задан элемент со следующим определением:

```
<!ELEMENT B (#PCDATA | F)*>
```

Будет генерироваться следующий класс:

```
class B : public Element
{ public:
      // Constructors
      B(Document *doc);
      B(Document *doc, String data);
      B(Document *doc, F *theF);
      // Assemblers
      void addNote (F *theF);
```

```
        // Add data after construction
        void addData (Document *doc, String data);
};
```

Отметим, что все конструкторы берут в качестве первого аргумента **(Document** *), так как класс **Document** имеет методы для построения узлов. Можно получить класс **Document** с помощью метода **XMLParser::getDocument**.

Точно так же в анализатор синтаксиса вызывается исполняемая программа генератора классов с интерфейсом командной строки:

```
xmlcg [switches] document
(xmlcg [ключи] документ)
```

Если заданы неправильные ключи либо только ключ -h (для вызова справки), вместо документа распечатывается следующее сообщение:

```
haifa% xmlcg
Error: No document specified
Usage: xmlcg [switches] <document>
   -d <name>          DTD - input is external DTD (must specify name)
   -o <directory>     Output - specify outpit directory
   -e <encoding>      Encoding - specify input file encoding
   -h                 Help - show this usage help
   -v                 Version - show parser version and exit
```

Ключи подробно описаны в таблице 2.8.

Таблица 2.8

Ключи генератора классов с интерфейсом командной строки

Ключ	Описание
-d name	Определяет, что входным файлом является внешний DTD с заданным именем, а не XML-документ.
-o *directory*	Определяет выходной каталог, куда будут помещаться сгенерированные файлы. Если этот каталог не задан, выходные файлы окажутся в текущем каталоге.
-e *encoding*	Определяет кодировку входного файла по умолчанию. Эта опция работает точно так же, как опция -e в программе синтаксического анализа XML; см. предыдущие комментарии.
-h	Распечатывается сообщение со справочной информацией.
-v	Распечатывается версия генератора классов, после чего работа заканчивается.

Глава 3

Разработка приложений для СУБД Oracle9i

Начиная с Oracle8i, компания Oracle поддерживает в своих СУБД два основных языка программирования: PL/SQL и Java. Многие клиенты Oracle используют сейчас для построения приложений баз данных оба эти языка. И если уж для программирования базы данных используются два языка, возникает естественный вопрос: как лучше использовать PL/SQL и Java для построения приложений, поддерживающих технологию XML?

Коротко ситуацию с этими языками можно описать следующим образом. PL/SQL предоставляет разработчикам приложений СУБД от Oracle громадные возможности. Этот язык обладает высокой производительностью, прост в использовании, он практически прозрачно для пользователя интегрирован языком SQL, кроме того, он очень устойчив. Современный язык PL/SQL — это мощный процедурно-ориентированный язык для разработки приложений баз данных, который идеально подходит для построения SQL-приложений с интенсивной обработкой информации. Начиная с СУБД Oracle8i, компания Oracle добавила в сервер базы данных поддержку языка Java, в результате чего этот чрезвычайно популярный универсальный язык получил надежную масштабируемую платформу. Язык Java используется для разработки многослойных компонентно-ориентированных приложений, использующих модули Enterprise Java Beans и *API-интерфейсы брокера объектных запросов* (Common Object Request Broker API, *CORBA*), а также для разработки традиционных хранимых процедур баз данных. В Oracle8i имеется несколько средств, позволяющих упростить процесс создания приложений с поддержкой технологии XML, используя для этого оба указанных языка. Кроме того, Oracle8i позволяет довольно легко компоновать приложения, написанные на двух языках — PL/SQL и Java. В Oracle9i эти возможности еще больше расширены за счет встроенных функций XML SQL и PL/SQL, а также за счет средств обмена сообщениями на базе XML-технологии.

Полное Java-решение от Oracle предлагает простоту, универсальность и свободу выбора, не ограничивая при этом мощность, необходимую для работы приложений масштаба предприятия. Это Java-решение объединяет в себе лучшее из традиционного мира баз данных и новые Интернет-стандарты, что позволяет создавать мощные промышленные Java-приложения.

Компания Oracle выпускает несколько компонентов, утилит и интерфейсов, которые можно использовать для реализации в приложениях БД всех преимуществ технологии XML (см. главу 2). Какие именно продукты использовать, зависит от требований приложения, предпочтений в методах программирования и от имеющихся сред разработки и разворачивания приложений. СУБД Oracle9i имеет расширенную

по сравнению с Oracle8i поддержку технологии XML за счет включения функций XML SQL и PL/SQL, встроенной поддержки типов данных XML, интегрированного средства организации очередей сообщений, а также встроенной поддержки стандартов Internet Java и XML. Можно запускать компоненты Oracle XML и построенные с их помощью приложения внутри базы данных с помощью Oracle JServer — встроенной виртуальной Java-машины от Oracle. В устройствах и приложениях, требующих меньшего объема базы данных, для хранения, и извлечения XML-данных используется версия Oracle Lite.

В данной главе обсуждаются основные принципы архитектуры Oracle JServer и использование в Oracle JServer компонентов Oracle Java XML. Рассматриваются различные подходы к хранению и извлечению XML-документов с помощью XML-компонентов, в том числе с помощью появившегося в Oracle9i нового компонента *XMLType*. Кроме того, рассматриваются представление локальных данных в базе данных с помощью новой функции поддержки идентификатора URI с использованием технологии XPath; интеграция XML- и SQL-запросов с помощью новых функций XML SQL и PL/SQL; использование интегрированных функциональных возможностей по организации очередей XML-сообщений, В конце главы обсуждаются методы использования XML-компонентов для разработки простого Web-сайта книжного магазина.

Oracle9i — СУБД с поддержкой технологии XML

Базы данных и технология XML в деле хранения информации обладают дополняющими друг друга функциональными возможностями. Базы данных хранят информацию для ее эффективного извлечения, а XML предлагает простой способ обмена информацией, позволяющий реализовать взаимодействие между различными приложениями. СУБД Oracle8i и, в еще большей степени, Oracle9i позволяют хранить XML-документы и строить приложения, поддерживающие технологию XML. Хранение в базе данных XML-документов позволяет использовать целый ряд полезных средств администрирования и процедур СУБД, в том числе средства создания резервных копий данных. Эти средства и процедуры используются для усиления правил операций с данными и защиты информации, а также для блокирования операций, которые могут нарушить целостность данных. Это осуществляется путем встраивания в базу данных нужных правил и логических схем. Преобразование таблиц базы данных в XML-документы позволит воспользоваться преимуществами XML-функций. Можно представить XML-документы в виде HTML-страниц с помощью таблиц стилей XSLT, вести поиск в этих документах с помощью языков запросов, поддерживающих технологию XML, или использовать эти документы для обмена данными с другими приложениями.

Объектно-реляционные функции Oracle позволят собирать сложные структуры XML-данных. Можно оперировать и управлять XML-данными с желаемым уровнем их детализации и легко приспосабливать их для эффективного построения динамичных XML-документов из ранее полученных фрагментов. Можно также хранить XML-документы в виде единого документа с тегами в файле *XMLType*, в *большом символьном объекте* (Character Large Object, *CLOB*) или в виде данных, распределенных без всяких тегов и разметки в объектно-реляционных таблицах, Файловая система Oracle Internet File System (см. главу 5) может получать доступ к XML-документам, хранимым во внешних файлах или в Web. Средство Oracle Text из СУБД Oracle9i (см. главу 6) используется для ведения поиска информации в XML-документах. XML-документы индексируются как плоский текст или как разделы документов для более точного поиска информации (тогда запрос может выглядеть так: "Oracle WITHIN title", где title — конкретный раздел документа). Для хранения и извлечения документов можно использовать новый XML-тип данных *uri-ref*, появившийся только в Oracle9i. Он позволяет проводить операции с документами, указания на которые дает единый индикатор ресурсов (Uniform Resource Indicator, URI). Имеющееся в OracleQi средство Oracle Text может производить полнотекстовое индексирование документов. В нем есть функции для выполнения в документах SQL-запросов и механизм поиска типа XPath. Наконец, функция Advanced Queuing (AQ) из Oracle9i теперь поддерживает организацию очередей в базе данных с использованием технологии XML. Это позволяет производить синхронный и асинхронный обмен XML-сообщениями по стандарту internet Data Access Presentation (iDAP), определенному и для сервера, и для клиента.

JServer и Java XML-компоненты Oracle

Компания Oracle предлагает очень широкий набор конкурентоспособных Java-решений — целый ряд серверов для запуска Java-приложений, стандартные интерфейсы программирования (API) для создания на языке Java приложений масштаба предприятия и набор эффективных средств для разворачивания Java-приложений. Oracle предлагает две разные среды исполнения Java-приложений — Oracle9i и Oracle internet Application Server. Оба сервера используют общую модель разработки с общими API-интерфейсами, а также общую структуру разворачивания и управления, позволяющую строить Java-приложения, которые можно легко распределить по слоям в приложениях, имеющих многослойную архитектуру. Можно развертывать бизнес-логику с интенсивной обработкой информации на сервере базы данных и проводить компьютерную обработку на сервере приложений Application Server.

Oracle9i JServer является самым надежным в индустрии Java-сервером масштаба предприятия. Он был разработан как первая в мире платформа программирования на языке Java для систем масштаба предприятия. В Oracle9i JServer решены технические проблемы, которые ограничивали широкомасштабное использование языка Java для приложений масштаба предприятия. Этот сервер предоставляет для запуска

Java-приложений высокопроизводительную, высокомасштабируемую, надежную и управляемую среду. Oracle9i JServer полностью совместим с Java-стандартами, поддерживаемыми в комплекте разработчика Java Development Kit. Этот сервер прошел все тесты, необходимые для получения сертификата "standard Java". В настоящее время СУБД Oracle9i поставляется с поддержкой языка Java для целого ряда операционных систем и аппаратных платформ, в том числе Solaris, Windows NT, HP-UX, DEC, AIX, Sequent и др.

Основы JServer

Сервер JServer, который тесно интегрирован с базой данных Oracle, состоит из нескольких компонентов: компилятора байт-кода, программы очистки памяти от ненужных данных (сборщик мусора), интегрированного загрузчика Java-классов и компилятора "Java-через-С". Все эти компоненты настроены на оптимальную производительность и масштабируемость при работе в среде базы данных (см. рис. 3.1). JServer работает в том же операционном и адресном пространстве, что и ядро самой базы данных, совместно используя несколько областей памяти для оптимизации производительности. Сеансы работы JServer происходят практически так же, как и сеансы работы с традиционными базами данных Oracle. После подключения к СУБД Oracle начинается сеанс связи с базой данных. Когда в течение этого сеанса в первый раз вызывается Java-метод, на период сеанса создается частная виртуальная Java-машина (JVM). В течение сеанса живут все объекты, на которые ссылаются

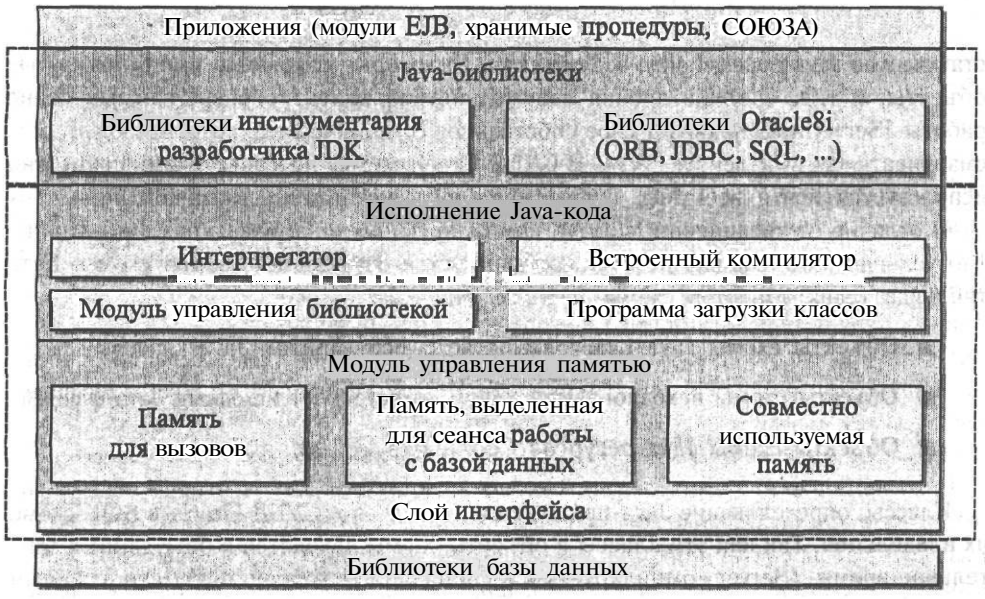


Рис. 3.1. *Архитектура Oracle JServer*

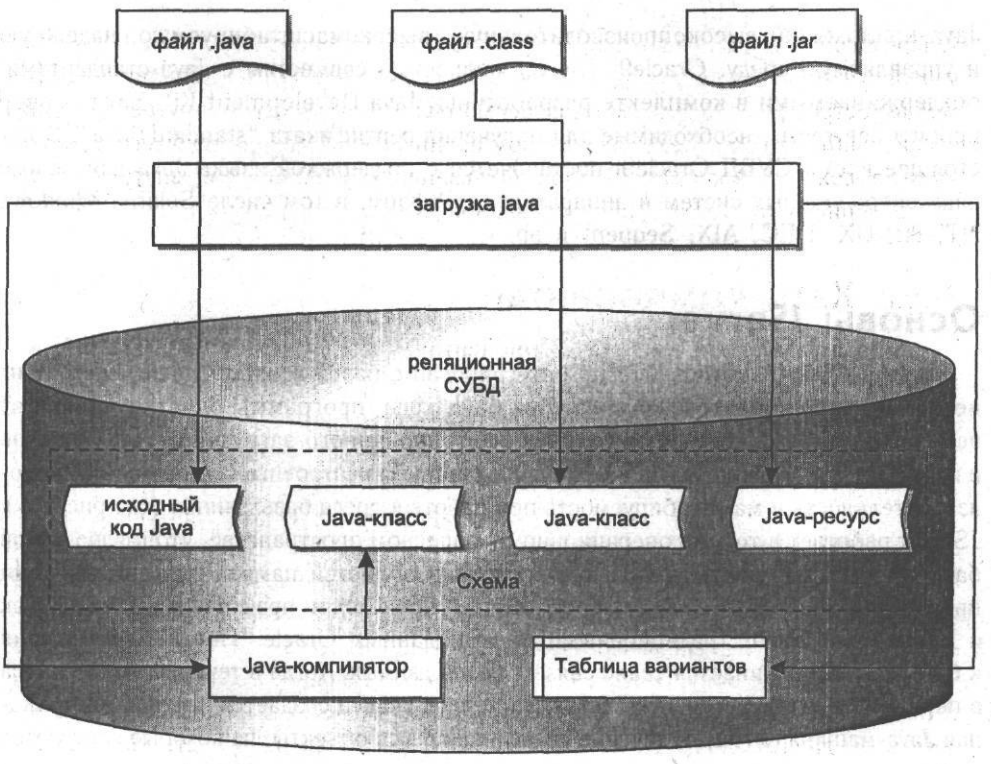


Рис. 3.2. *Объекты Java-схемы*

статические Java-переменные, все объекты, на которые ссылаются вышеупомянутые объекты, и т. д. С точки зрения взаимодействия клиент/сервер, каждый сеанс работы JServer поддерживает свое собственное состояние Java, которое можно рассматривать как выделенную JVM. В СУБД Oracle между пользователями совместно используется почти весь код, инфраструктура и метаданные активной JVM.

В отличие от традиционных JVM, компилирующих и загружающих Java-файлы, виртуальная Java-машина JServer компилирует и загружает объекты схемы. Есть три вида таких объектов схемы Java (см. рис. 3.2):

- **Объекты схемы Java-классов** Соответствуют файлам Java-классов
- **Объекты схемы исходного Java-кода** Соответствуют исходным Java-файлам
- **Объекты схемы Java-ресурсов** Соответствуют Java-файлам ресурсов

Классы, определяющие Java-приложение, хранятся в СУБД Oracle в SQL-схеме их владельцев. Так как эти классы и объекты схемы находятся в базе данных длительное время, JServer компилирует их в специальный хорошо оптимизированный код, используя для этого *опережающую компиляцию WAT* (way ahead of time).

Опережающая компиляция транслирует стандартные бинарные файлы Java-классов, генерируя специализированные C-программы, которые затем компилируются (с помощью встроенного платформно-зависимого C-компилятора) во внутренние динамические библиотеки. Опережающая компиляция существенно ускоряет исполнение Java-приложений благодаря полной ликвидации интерпретатора, внутритекстовому кодированию и объектно-ориентированной оптимизации.

Java XML-компоненты

Oracle поставляет целый ряд компонентов, утилит и интерфейсов, которые используются для реализации преимуществ XML-технологии в приложениях баз данных (см. главу 2). Среди этих компонентов есть XML-анализатор синтаксиса, XSL-средство преобразования документов, генератор классов XML, модули XML TransViewer, сервлет XSQL Servlet и утилита XML SQL Utility. XML-анализатор синтаксиса является автономной XML-программой, производящей синтаксический анализ XML-документов, которые затем обрабатываются приложением. Этот анализатор синтаксиса поддерживает интерфейсы *Document Object Model (DOM)* и *Simple API for XML (SAX), а* также пространства имен XML. XML-анализатор может проводить синтаксический анализ XML-документов в двух режимах: с проверкой документов на допустимость (validation) и без таковой. Если проверка на допустимость не производится, анализатор просто проверяет, правильно ли (well formed) составлен данный XML-документ. В режиме проверки на допустимость документ проверяется еще и на соответствие DTD или XML-схеме, а кроме того, проверяется правильность налагаемых на документ ограничений. В XML-анализатор интегрировано XSL-средство преобразования документов. С его помощью производится преобразование XML-документов с применением XSL-таблиц стилей. Генератор классов XML Class Generator из XML DTD или XML-схемы создает набор исходных Java-файлов. Сгенерированные исходные файлы можно использовать для построения, проверки на допустимость и печати XML-документа, который соответствует входным XML DTD или XML-схеме.

Сервлет XSQL Servlet — это средство для обработки SQL-запросов и выдачи результирующих данных в виде XML-файла. Этот процессор реализован как Java-сервлет, в качестве входного файла он берет XML-файл, содержащий встроенные SQL-запросы. XSQL Servlet для выполнения своих операций использует все предыдущие компоненты, после чего применяет утилиту XML SQL Utility для возврата результатов запроса в виде XML-файла. Модули XML TransViewer представляют собой набор XML-компонентов для Java-приложений или апплетов. Эти модули могут использоваться для асинхронного синтаксического анализа XML-файлов или для просмотра XML/XSL-файлов с цветной подсветкой XML/XSL-синтаксиса. Можно применить XSL-таблицу стилей для преобразования XML-документа *в* документ почти любого текстового формата, в том числе XML, HTML и DDL, и сразу же увидеть результаты преобразования. Можно также интегрировать эти визуальные

и невизуальные Java-компоненты в инструментальный пакет Oracle JDeveloper, с помощью которого разработчики сумеют быстро создавать и разворачивать приложения баз данных, поддерживающие технологию XML.

Java XML-компоненты достаточно универсальны, чтобы работать в качестве автономных Java-приложений с использованием драйвера JDBC для взаимодействия с базой данных или чтобы эффективно функционировать внутри Oracle JServer с использованием точно настроенного внутреннего серверного JDBC-драйвера. Автономное Java-приложение может постоянно находиться на клиентской стороне или в промежуточном слое сервера Oracle internet Application Server. Обычно в автономных Java-приложениях для установки связи с базой данных используется JDBC и класс DriverManager, управляющий набором JDBC-драйверов. После загрузки этих драйверов можно вызывать метод getConnection, который возвращает объект Connection, являющийся эквивалентом сеанса связи с базой данных. Все операторы SQL исполняются в рамках контекста данного сеанса связи. Однако внутренний серверный JDBC-драйвер запускается в сеансе и в контексте транзакции по умолчанию. Поэтому всегда существует связь с базой данных, и все SQL-операции являются частью контекста транзакции по умолчанию. Для того чтобы получить объект Connection в хранимой Java-процедуре, используется следующий оператор:

```
Connection c = DriverManager.getConnection("jdbc:default:connection");
```

При разработке приложений хранимых Java-процедур следует знать, что внутренний серверный JDBC-драйвер нельзя использовать для подключения к удаленной базе данных. Можно подключиться только к серверу, на котором исполняется хранимая Java-процедура. Для соединений типа "сервер-сервер" или "клиент-сервер" нужно использовать серверный или клиентский драйвер JDBC Thin.

Как уже говорилось в начале главы, применять язык Java для разработки приложений баз данных и использования их в этой базе данных в качестве хранимых процедур можно, начиная с версии Oracle8i. Хранимые процедуры, разработанные на языке Java, запускаются в том же адресном пространстве, что и SQL и PL/SQL, поэтому они должны без проблем взаимодействовать с существующими PL/SQL-приложениями. Так как Java-программы работают в том же адресном пространстве, что и сервер базы данных, а кроме того, используют встроенные внутренние серверные JDBC-драйверы, им не нужно производить полный обход сети для получения доступа к SQL-данным. В результате эти программы позволяют добиться высокой производительности и снизить интенсивность сетевого трафика. Для разворачивания хранимых Java-программ в Oracle8i и Oracle9i нужно загрузить в базу данных Java-программы и опубликовать Java-методы в SQL. Указанные СУБД поддерживают несколько разных форм, в которых можно загружать Java-программы. Можно загрузить исходный текст Java, стандартные файлы с Java-классами или Java-архивы (jar). Исходный код Java после загрузки в базу данных автоматически компилируется имеющимся в этой СУБД компилятором Java байт-кода. Java-объекты в СУБД Oracle загружаются несколькими разными способами. Но сначала следует выдать

новую *DDL-команду типа* "CREATE JAVA..." из SQL*Plus для загрузки в базу данных исходного кода Java, бинарных файлов или файлов ресурсов:

```
# Create a directory object on the server's file system
SQL> CREATE DIRECTORY bfile_dir as '/home/user/oracle/xml/parser/v2';
# Then load the java class using "CREATE JAVA CLASS ..." statement
SQL> CREATE JAVA CLASS USING BFILE (bfile_dir, 'XMLParser.class');
```

Чтобы упростить процесс загрузки, у Oracle имеется написанная на языке Java утилита LOADJAVA, автоматизирующая процесс загрузки Java в базу данных. Так как эта утилита использует для связи с БД и загрузки в нее Java-кода драйверы Oracle JDBC, можно загрузить модули Java-программ в базу данных через сеть.

```
loadjava -user scott/tiger@myhost:1521:orcl xmlparserv2.jar
```

Публикация и вызов Java XML-компонентов

Важно отметить, что появление хранимых процедур, написанных на языке Java, вовсе не означает, что хранимые процедуры, написанные на PL/SQL, пора отправлять в отставку. Используются и уже имеющиеся библиотеки хранимых процедур на PL/SQL, и новые серверные хранимые процедуры на Java. Такой подход вполне оправдан, поскольку и те, и другие процедуры имеют одну и ту же спецификацию вызова. Поэтому на уровне приложения для пользователя абсолютно не важно, какая технология используется.

Java-методы в репозитории СУБД Oracle публикуются до того, как они будут вызваны из SQL. При загрузке какого-нибудь Java-класса в БД его методы не публикуются автоматически, поскольку СУБД Oracle не знает, какой из этих методов является надежной точкой входа для обращений из SQL. Для публикации этих методов нужно написать спецификацию обращения, в которой должны быть описаны имена Java-методов, типы параметров и типы результатов для их SQL-эквивалентов. Спецификация вызова функции или процедуры задается для каждого Java-метода с помощью операторов SQL CREATE FUNCTION или CREATE PROCEDURE. Java-методы, вычисляющие значения, публикуются как функции, а пустые Java-методы — как процедуры. Приложения вызывают Java-метод, ссылаясь на имя спецификации вызова. Система, работающая в реальном времени, отыскивает определение этой спецификации вызова в репозитории СУБД Oracle и исполняет соответствующий Java-метод, как показано на рис. 3.3.

Спецификация вызова и Java-метод должны находиться в одной и той же схеме (если только Java-метод не имеет синоним PUBLIC). Можно описывать спецификацию вызова как:

- Автономную PL/SQL функцию верхнего уровня или процедуру
- Упакованную PL/SQL функцию или процедуру
- Метод-член объектного SQL-типа

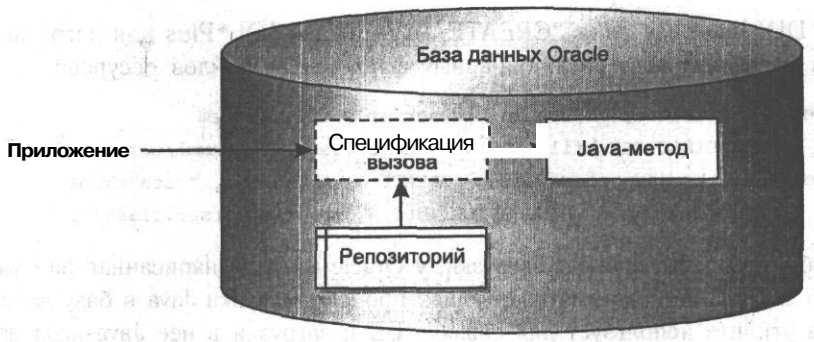


Рис. 3.3. *Вызов Java-метода*

Спецификация вызова открывает для СУБД Oracle входную точку верхнего уровня Java-метода. Можно публиковать только открытые статические методы, за одним исключением: методы экземпляра публикуются как методы-члены объектного SQL-типа. Соответствующие SQL- и Java-параметры (и результаты функций) должны иметь в спецификации вызова совместимые типы данных. А преобразования между SQL-типами и Java-классами СУБД Oracle производит автоматически.

Можно загрузить следующий исходный Java-код или его классификационный файл в базу данных Oracle и опубликовать его методы как PL/SQL-функции. Для публикации этих методов требуется следующая спецификация вызова:

```
public class CaseConvert
{  public static String toUpper ( String str)
    {  return str.toUpperCase() ;  }
   public static String toLower(String str)
    {  return str.toLowerCase();  }
}

CREATE OR REPLACE FUNCTION toUpper (str VARCHAR2) return VARCHAR2AS LANGUAGE JAVA '
NAME 'CaseConvert.toUpper (java.lang.String) return java.lang.String';

CREATE OR REPLACE FUNCTION toLower (str VARCHAR2) return VARCHAR2AS LANGUAGE JAVA
NAME 'CaseConvert.toLower (java.lang.String) return java.lang.String';
```

Аналогичным образом загружается Java XML-анализатор синтаксиса и пишется следующий исходный Java-код для анализа и проверки на допустимость XML-документа, хранящегося в таблице в качестве большого символьного объекта CLOB.

```
public class Validate {
   public static String validateCLOB (String xmldoc) {
      SAXParser parser = new SAXParser ( ) ;
```

```
    try {
      parser parse(new StringInputStream (xmldoc));
      return "Valid XML Document";
    } catch (SaxException e) {
      return "Invalid XML Document" + e.getMessage();
    }
  }

CREATE OR REPLACE FUNCTION validateCLOB(xmldoc CLOB)
  return VARCHAR2AS LANGUAGE JAVA NAME
'Validate.validateCLOB (oracle.sql.CLOB) returns java.lang.String';
```

Опубликованная хранимая Java-процедура будет выглядеть как хранимая PL/SQL-процедура. К ней можно обращаться, используя SQL-имя ее спецификации из нескольких различных контекстов:

- Из любого SQL-оператора:

```
SELECT validateCLOB(XMLDOC) FROM XMLSTORE;
```

- Из верхнего уровня с использованием функции CALL:

```
CALL validateCLOB(:xmldoc ) INTO :x;
```

- Из PL/SQL-процедуры, модуля или безымянных блоков с помощью того же синтаксиса, который используется для вызова других PL/SQL-процедур:

```
DECLARE
result VARCHAR2 [1000];
xmldoc CLOB;
BEGIN
...
result :=validateCLOB(:xmldoc);
dbms_output.put_line(result);
...
END;
```

До сих пор мы обсуждали, как разрабатывать и использовать хранимые Java-процедуры в базе данных Oracle. На нескольких простых примерах мы продемонстрировали шаги, которые необходимо предпринять для загрузки, разрешения, публикации и вызова Java-программ в базе данных. Одним из основных приложений хранимых Java-процедур является использование базы данных Oracle в качестве репозитория для XML-документов. В следующем разделе рассмотрим различные методы отображения схемы базы на XML-документы и обратно.

Схема базы данных и XML-документы

XML-документы — это текст, соответствующий иерархии древовидной структуры, определяемой DTD или XML-схемой. Эти иерархические данные можно хранить в оптимальной внутренней форме с использованием объектно-реляционных таблиц. Все нынешние и будущие внутренние приложения БД могут работать с этой информацией наиболее эффективным способом. Извлекая информацию для совместного использования ее с партнерами или с другими приложениями, можно представить данные и содержимое документов в виде, специально предназначенном для выполнения конкретной задачи. Объектное представление данных в Oracle8i и Oracle9i позволяет представлять данные с помощью любых логических комбинаций, скрывая при этом всю информацию об их физическом хранилище. Можно преобразовывать структуру одной или нескольких базовых таблиц в более полезную или более подходящую для требований конкретного приложения структуру. При связывании представления информации с другими подобными представлениями последние довольно естественно образуют деревья или графы. Когда информация предоставляется в базу данных в XML-формате, предыдущие связанные представления образуют базу для нескольких разных древовидных XML-документов.

Вот простой пример составления схемы таблицы из базы данных в соответствии с XML DTD:

```
<!ELEMENT table (rows)*>
<!ELEMENT rows (column1, column2, ...)>
<!ELEMENT coluitml (#PCDATA)>
<!ELEMENT column2 (#PCDATA)>
...
```

Однако база данных предоставляет для формулирования правил еще больше возможностей, чем DTD. С помощью DTD нельзя определить тип информации, отличный от строкового. А схема базы данных определяет тип информации и ограничения на нее, например допустимые диапазоны изменения значений. Схема базы данных позволяет определять взаимоотношения или зависимости данных. Например, на сайт электронной коммерции могут придти заказы в виде XML-документов. С помощью базы данных можно связать клиента и информацию о его заказе и определить правило не обрабатывать заказы клиентов, имеющих закрытые счета. Несмотря на существующие в DTD ограничения, отображение схемы базы данных на DTD представляет эту БД в виде виртуального XML-документа для тех средств, которые используют в качестве входных данных XML-документы.

Формат для DTD — широко распространенный в мире стандарт, и в течение ближайших лет он, скорее всего, будет жить и развиваться. Однако из-за внутренних ограничений DTD и бытующего мнения, что в условиях развития электронного бизнеса и электронной коммерции XML должен все более ориентироваться на данные, консорциум W3C пропагандирует новый стандарт, называемый XML-схема, и не

пытается продвигать и развивать существующие стандарты на DTD. С появлением XML-схемы были преодолены ограничения, существующие в DTD. XML-схемы позволяют определять и тип информации, и ограничения на нее. С помощью XML-схемы можно отобразить простую таблицу БД на XML-схему следующего вида:

```
<schema targetNamespace="someNSURI">
  <element name="table">
    <element name="rows" minOccurs="0" maxOccurs="*">
      <element name="column1">
        <datatype source="string">
          <length value="1000"/>
        </datatype>
      </element>
      <element name="column2" type="decimal">
        ...
    </element>
  </element>
</schema>
```

Для переноса данных между XML-документом и базой данных нужно отобразить структуру документа на схему базы данных и наоборот. Структура любого XML-документа связана с DTD или с XML-схемой. DTD используется для описания элементов и атрибутов. Как и DTD, схемы описывают данные, но обладают дополнительным преимуществом, поскольку позволяют определить тип данных из таких простых примитивных типов, как строки, даты и целые числа для сложных структур, например почтовых адресов и точек графика.

Отображение XML-документов на схему базы данных

Для отображения XML-документов, связанных с XML DTD или XML-схемой, на схему базы данных для сохранения их в СУБД Oracle9i существуют четыре основные стратегии:

- Отображение всего XML-документа как единого цельного объекта, например большой символьный объект CLOB
- Отображение элементов XML-документов на объектно-реляционные таблицы и колонки в схеме базы данных
- Отображение фрагментов XML-документов как больших символьных объектов, а оставшихся частей этих документов — как объектно-реляционных таблиц
- Отображение XML-документов на Oracle XMLType

Можете выбрать один из этих методов в зависимости от структуры XML-документа и операций, выполняемых приложением. Можно также сохранить XML DTD или XML-схему в базе данных для проверки допустимости XML-документов.

XML-документы в больших символьных объектах

Хранить цельный XML-документ в большом символьном объекте (CLOB) или *большом двоичном объекте (BLOB)* предпочтительней, когда он содержит статическое содержимое, которое если и будет обновляться, то все целиком, путем замены всего документа. В качестве примера можно привести текстовые документы типа статей, рекламных объявлений, книг, юридических документов или контрактов. Эти документы, централизованные по документам, извлекаются из базы данных целиком. Хранение их целиком внутри СУБД Oracle9i дает преимущества проверенной индустрией и надежной базы данных над простой системой хранения файлов. Выбрав для хранения XML-документа место за пределами базы данных, тем не менее можно использовать функции Oracle9i для индексирования, составления запросов и извлечения документов с помощью внешних бинарных файлов, URL-адресов и индексирования текста.

XML-документы как объектно-реляционные данные

Если XML-документ имеет четко определенную структуру и содержит данные, которые обновляются или используются какими-то другими способами, его можно назвать централизованным по данным (data-centric). Обычно такой XML-документ содержит элементы или атрибуты, имеющие сложные структуры. В качестве примера можно привести заказы на продажу, счета-фактуры и расписания самолетов. СУБД Oracle9i с ее объектно-реляционными расширениями может фиксировать информацию о структуре данных в базе, используя для этого типы объектов, объектные ссылки и совокупности данных. Существуют два варианта хранения структуры XML-данных в объектно-реляционной форме:

- Хранение атрибутов элементов в реляционной таблице и определение представлений объекта для хранения структуры XML-элементов.
- Хранение структурированных XML-элементов в объектной таблице.

После сохранения в объектно-реляционной форме данные можно легко обновлять, составлять к ним запросы, перегруппировывать или переформатировать с помощью языка SQL.

Затем с помощью сервлета XSQL Servlet или утилиты XML SQL Utility можно отобразить XML-документ на базовое объектно-реляционное хранилище, и с их же помощью извлекать из базы объектно-реляционные данные в виде XML-документа.

Если XML-документ структурирован, но его структура несовместима со структурой базовой схемы базы данных, то, прежде чем записывать данные в базу, их нужно преобразовать в правильный формат. Это можно сделать с помощью XSL-таблиц стилей или другими программными методами. Но, возможно, потребуется сохранить централизованный по данным XML-документ в виде цельного единого объекта. Можно также задать представления объекта, соответствующие различным структурам XML-документа, и определить триггеры для выполнения соответствующего преобразования и обновления базовых данных.

XML-документы как фрагменты документов и объектно-реляционные данные

Для просмотра и оперирования комбинацией структурированных и неструктурированных XML-данных как единого целого можно использовать представления СУБД Oracle9i. Эти представления позволяют строить объект на лету, объединяя XML-данные, хранимые несколькими разными способами и в разных местах. Так можно хранить структурированные данные (например, информацию о сотрудниках, клиентах и т. д.) в одном месте в объектно-реляционных таблицах, а связанные с ними неструктурированные данные (например, описания и комментарии) — в объекте CLOB. Когда нужно извлечь данные целиком, следует просто построить структуру из различных фрагментов данных с помощью конструкторов типов в операторе выбора этого представления. Затем с помощью сервлета XSQL Servlet можно получить построенные данные из представления в виде единого XML-документа.

XML-документы как XMLType

Наконец, можно хранить XML-документы в Oracle9i *XMLType,* что позволяет вести поиск и организовывать запросы с помощью XPATH-подобного синтаксиса, а также использовать *XMLType* для создания таблицы базы данных и просмотра столбцов (поскольку он представляет собой скрытый столбец CLOB) или в качестве параметра и возвращаемого типа SQL-, PL/SQL- и Java-функций. Например, Oracle SQL-функции *SYS_XMLGEN* и *SYS_XMLAGG,* генерирующие XML-документы и группирующие несколько таких документов, используют в качестве параметра и возвращают объект *XMLType.* Эти функции можно встроить в SQL-запросы, как это делается в простом операторе *SELECT,* и получить XML-документ:

```
SELECT SYS_XMLGEN(book) FROM bookcatalog WHERE title LIKE '%ELLISON%';
```

и

```
SELECT SYS_XMLAGG(SYS_XMLGEN(book)).getCloVal() book_list FROM
bookcatalog Group BY title;
```

где первый оператор вернет XML-документ с введенной информацией о книге, а второй — список заголовков всех введенных книг. Аналогично, новый PL/SQL-

модуль *DBMS_XMLGEN*, включающий номера новых процедур, преобразует результирующий набор данных из SQL-запросов в XML-документ, хранящийся *в XMLType,* например:

```
SELECT SYS_XMLAGG(SYS_XMLGEN(book)).getCloVal() book_list FROM
bookcatalog Group BY title
```

и выдает на выходе распечатку всех названий книг в XML-формате.

Наконец, таблицы можно создавать с помощью XMLType-столбцов и операций языка манипулирования данными DML (data manipulation language) по вставке, обновлению и удалению значений. Это осуществляется с помощью нового типа данных, а также с помощью функций-членов *existsNode()* и *extract(),* которые берут аргументы с XPATH-подобным синтаксисом и возвращают фрагменты XML-файлов в XMLType:

```
SELECT book.extract('//title/text()').getStringVal() FROM bookcatalog;
```

и

```
SELECT * FROM bookcatalog where book.existNode('//book/title') != 0;
```

Отображение схемы базы данных на виртуальные XML-документы

Для отображения схемы базы данных на виртуальные XML-документы есть два основных метода:

- Отображение полной базы данных как виртуального XML-документа
- Отображение результата запроса в виде виртуального XML-документа

В зависимости от приложения выбирается один из этих методов. Рассмотрим приложение, выполняющее копирование базы данных из одной базы во вторую, с другой схемой. Это приложение может произвести XSL-преобразование виртуального документа, представляющего собой первую базу данных, после чего вставить результат преобразования во вторую базу данных.

Полная схема

База данных состоит из схемы, ассоциированной с каждым пользователем базы данных. Каждая схема, ассоциированная с пользователем, представляет собой коллекцию объектов схем, доступных этому пользователю. При отображении схемы базы данных на DTD каждый пользователь отображается как дочерний элемент элемента верхнего уровня, определяемого идентификационным номером данного экземпляра БД. Элемент, представляющий схему пользователя, и его дочерние элементы используют уникальное пространство имен, чтобы избежать конфликтов с объектами схемы, определенными в другой схеме пользователя.

DTD генерируется из реляционной схемы с помощью следующей упрощенной процедуры:

1. Для каждой таблицы создается элемент.
2. Для каждого столбца таблицы создается дочерний элемент в формате PCDATA.
3. Для каждого объекта или вложенного столбца таблицы создается дочерний элемент с содержимым ELEMENT с дочерними элементами в виде атрибутов или вложенных столбцов.

Например, простой базе данных соответствует следующий DTD:

```
<!ELEMENT dbschema (sys, scott, ...)>
<!ATTLIST dbschema
      xmlns CDATA #FIXED "http://www.oracle.com/xml/dbschema"
      sid   CDATA #REQUIRED>
<!ELEMENT scott (BookList, ...)>
<!ATTLIST scott
      xmlns CDATA #FIXED "http://www.oracle.com/xml/dbschema/scott">
<!ELEMENT BookList (Book)*>
<!ATTLIST Book row_num CDATA #IMPLIED>
<!ELEMENT Book (Title, ISBN, Author, Publisher, (Review)*)>
...
```

К сожалению, в отображении схемы базы данных на DTD есть целый ряд недостатков. Например, из DTD нельзя сколько-нибудь точно спрогнозировать типы данных и длину столбцов. Решение этой проблемы состоит в использовании типов данных в XML-документах с применением XML-схем.

Запрос

Результат запроса можно отобразить на виртуальный XML-документ точно так же, как это делается при отображении базы данных на XML-документ. Результат такого запроса содержит несколько строк, которые отображаются на XML-документ с корневым элементом ROWSET, а каждая строка такого документа инкапсулирована в элемент ROW. Каждый столбец, выбранный с помощью запроса, добавляется в качестве дочернего элемента ROW. Можно определить альтернативные теги для элемента ROWSET и теги для элемента ROW. Отображение простого запроса на XML-документ иллюстрируется примером:

```
SQL> select * from scott.BookList;
<!ELEMENT BookList (Book)*>
<!ATTLIST BookList
      xmlns CDATA #FIXED "http://www.oracle.com/xml/dbschema/scott">
```

```
<AQXMLSend xmlns=http://www.oracle.com/xml/schemas/AQ/access>
  <!-- mandatory -->
  <producer_options>
    <!-- mandatory -->
    <destination>BOOKLIST.BOOK_QUEUE</destination>
  </producer_options>
```

```
<!ELEMENT Book (Title, ISBN, Author, Publisher, (Review)*)>
<!ATTLIST Book row_num CDATA #IMPLIED>
...
```

Для определения структуры XML-документа и типов данных используется также

Конечно же, СУБД Oracle9i и инструментарий Oracle XML Developer's Kit предоставляют все основные технологии, необходимые разработчикам для реализации этого решения. Кроме того, для автоматизации использования имеющихся в этом ПО компонентов XML-технологии применяется средство Oracle XSQL Pages, позволяющее обойтись без программирования почти во всех ситуациях. Oracle XSQL Pages представляет собой набор шаблонов, с помощью которого любой специалист, знакомый с языком SQL, может:

- Собирать динамические XML-страницы с данными на базе одного или нескольких параметризованных SQL-запросов.
- Преобразовывать эти страницы для получения окончательного результата в нужном XML, HTML или другом текстовом формате с помощью средства XSLT Transformation.

Например, следующий URL-запрос использует список сегодняшних авиарейсов в любой заданный пункт назначения. Результаты этого запроса показаны на рис. 3.4.

http://yourcompany.com/AvailableFlightsToday.xsql?City=SFO

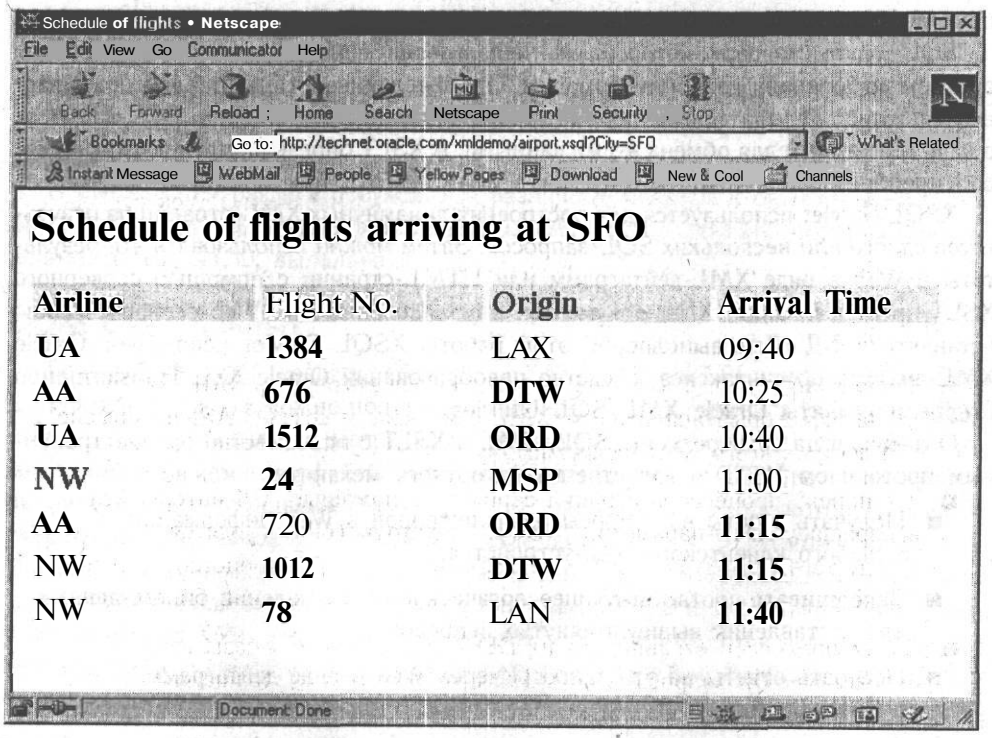


Рис. 3.4. *Вид экрана с результатами запроса* ***airport.xsql***

Можно написать следующую XSQL-страницу, чтобы в ответ на этот запрос получить показанный на рис. 3.4 список из базы данных.

```
<?xml version="1.0"?>
<xsql:query connection="demo" xmlns:xsql="urn:oracle-xsql">
    SELECT Carrier, FlightNumber, Origin,
           TO_CHAR(ExpectedTime, 'HH:MI') ASDue
      FROM FlightSchedule
     WHERE TRUNC(ArrivalTime) = TRUNC(SYSDATE)
       AND destination = '{@City}'
  ORDER BY ExpectedTime
</xsql:query>
```

Архитектура XSQL

XSQL Servlet — это средство для обработки SQL-запросов и выдачи результатов в формате XML. Этот процессор реализован в виде Java-сервлета. В качестве входных данных он использует XML-файл, содержащий встроенные SQL-запросы. А для выполнения многих операций XSQL Servlet использует инструментарий разработчика Oracle XML Developer's Kit.

Этот сервлет запускается из любого Web-сервера, поддерживающего Java-сервлеты. На рис. 3.5 показаны потоки данных от клиента к сервлету и обратно к клиенту. При этом события происходят в следующей последовательности:

- Пользователь вводит через браузер URL-адрес, который интерпретируется и передается в XSQL Servlet через Java Web-сервер. Этот URL содержит название целевого XSQL-файла (**.xsql**) и в качестве необязательных параметров значения и имя таблицы стилей XSL. Пользователь также может запустить XSQL Servlet из командной строки в обход браузера и Java Web-сервера.
- Сервлет передает XSQL-файл в анализатор синтаксиса XML Parser for Java, который анализирует XML-синтаксис и создает API-интерфейс для доступа к его XML-содержимому.
- Компонент процессора страниц сервлета с помощью API-интерфейса возвращает XML-параметры и SQL-операторы (они находятся между тегами **<xsql:query>** и **</xsql:query>**). Этот процессор страниц также передает в XSQL-процессор все операторы по XSL-обработке.
- Затем процессор страниц строит DOM-представление базы данных, отправляя SQL-запросы в СУБД Oracle9i, откуда возвращаются результаты. Потом результаты вставляются в XML-файл, в то же самое место, что и первоначальные теги **<xsql:query>**.

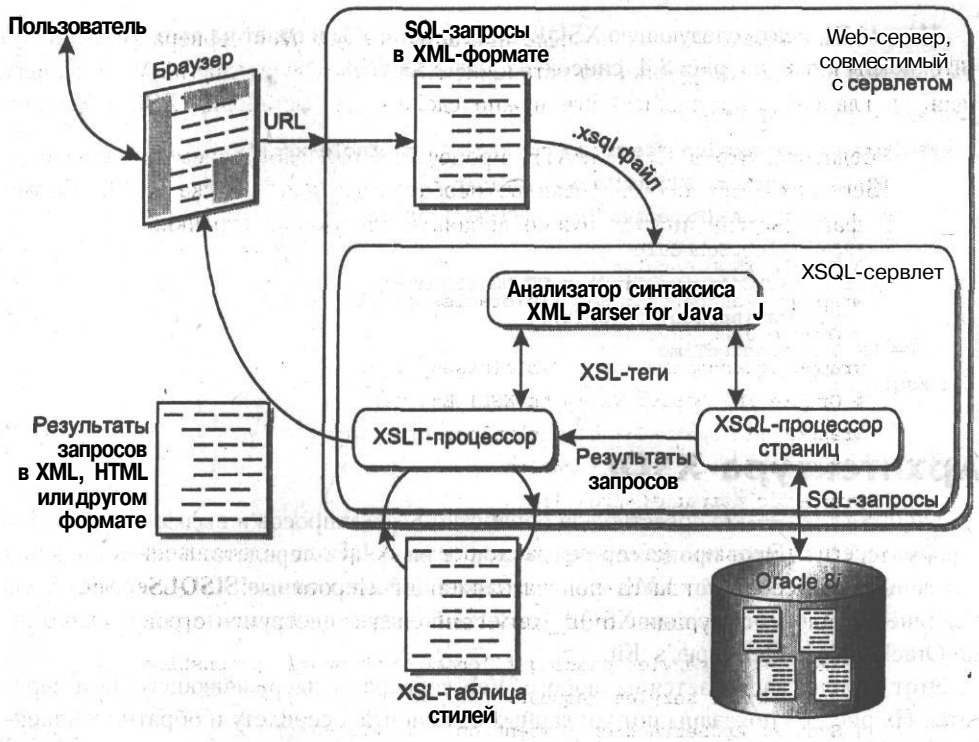


Рис. 3.5. *Архитектура XSQL-страниц*

- По желанию пользователя результаты запроса и любые другие XML-данные преобразуются XSLT-процессором с использованием заданной XSL-таблицы стилей. Данные можно преобразовать в HTML или любой другой формат, определенный этой таблицей стилей. XSLT-процессор может селективно применять разные таблицы стилей в зависимости от типа клиента, с которого пришел первоначальный URL-запрос. Информация о типе клиента (HTTP_USER_AGENT) приходит от клиента в результате HTTP-запроса.

XSLT-процессор передает оформленный документ обратно на клиентский браузер для представления пользователю.

Установка XSQL Servlet

XSQL Servlet можно установить и сконфигурировать на самых разных Web-серверах, в том числе на Oracle9i Lite Portal-to-Go Server, Apache 1.3.9 с JServ 1.0 и Sun JavaServer Web Development Kit (JSWDK) 1.0.1 Web Server.

В данном разделе описан процесс инсталляции для Web-сервера Apache 1.3.9. Более подробные инструкции по установке XSQL Servlet на других Web-серверах даны в главе 8. Следующие шаги нужно сделать для установки XSQL Servlet:

1. Убедиться, что в CLASSPATH процессора реального времени Apache JServ находятся все JAR-файлы, необходимые для запуска XSQL Servlet. В файл **jserv.properties** нужно добавить следующие строчки:

```
# Oracle XSQL Servlet
wrapper.classpath=C:\xql\lib\oraclexsql.jar
# Oracle JDBC (8.1.5)
wrapper.classpath=C:\xql\lib\classes111.zip
# Oracle XML Parser V2 (with XSLT Engine)
wrapper.classpath=C:\xql\lib\xmlparserv2.jar
# XSQLConfig.xml File location -
wrapper.classpath=C:\xql\lib
```

2. Зарегистрировать файловое расширение **.xsql** для отображения на Java-сервлет класса под названием **oracle.xml.xsql.XSQLServlet**. В файл конфигурации **mod_jserv.conf** следует добавить строки:

```
# Executes a servlet passing filename with proper extenstion
# property of servlet request.
# Syntax: ApJServAction [extention] [servlet-uri]
# Defaults: NONE
# Notes: This is used for external tools.
#ApJServAction .jsp /servlets/nl.nmg.jsp.JSPServlet
ApJServAction .xsql /servlets/oracle.xml.xsql.XSQLServlet
```

После регистрации файлового расширения **.xsql** нужно заново запустить Web-сервер и просмотреть какой-нибудь XSQL-файл, чтобы увидеть выходные XML-данные или преобразованные выходные данные в формате HTML.

Динамические XML-документы из SQL-запросов

Oracle XSQL-страницы представляют собой XML-страницы со встроенными SQL-запросами о возврате или вставке данных. Можно построить XSQL-страницу, включив тег **<xsql:query>** в XML-файл, в то место, где должен исполняться SQL-запрос. В результате выполнения запроса вместо элемента **<xsql:query>** появится выходной XML-файл.

Для доступа и аутентификации связи с базой данных XSQL Servlet использует файл конфигурации **XSQLConfig.xml.** Пример такого файла конфигурации:

```
<?xml version="1.0" ?>
<XSQLConfig>
   <connectiondefs dumpallowed="no">
      <connection name="demo">
         <username>scott</username>
         <password>tiger</password>
         <dburl>jdbc:oracle:thin:@localhost:1521:ORCL</dburl>
         <driver>oracle.jdbc.driver.OracleDriver</driver>
      </connection>
   <connectiondefs/>
   :
</XSQLConfig>
```

Дополнительные элементы связи определяются для идентификации разных пользователей или для использования разных JDBC-драйверов. Предполагается, что XSQL Servlet найдет в корневом элементе XML-документа атрибут "connection", значение которого должно соответствовать названию соединения, определенного в файле конфигурации. Самое простое применение тега **<xsql:query>** выглядит так:

```
<?xml version="1.0"?>
<xsql:query xmlns:xsql="urn:oracle-xsql" connection="demo">
     SELECT 'Hello World' AS "GREETING" FROM DUAL
</xsql:query>
```

Эта XSQL-страница порождает следующий динамически созданный канонический XML-документ:

```
<?xml version = '1.0'?>
<ROWSET>
   <ROW id="1">
      <GREETING>Hello World</GREETING>
   </ROW>
</ROWSET>
```

Этот канонический XML-документ можно преобразовать в другую XML-форму или в HTML-файл. Можно также данной XSQL-странице сопоставить XSL-таблицу стилей с помощью следующей команды:

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

В XSQL-страницах можно использовать объектно-реляционные запросы, а также посылать в эти страницы параметры, используя URL-адрес. Например, объектно-реляционные возможности Oracle9i помогают создать определяемый пользователем тип объекта POINT. Этот новый тип POINT можно использовать в качестве типа данных столбца ORIGIN в таблице LOCATION с помощью следующих DDL-операторов:

```
CREATE TYPE POINT AS OBJECT (X NUMBER, Y NUMBER) ;
CREATE TABLE LOCATION (
   NAME     VARCHAR2 (80) ,
   ORIGIN    POINT
);
```

В эту таблицу LOCATION можно вставить какую-нибудь строку. Для этого с конструктором POINT() используется оператор INSERT:

```
SQL> INSERT INTO LOCATION VALUES ('Someplace', POINT(11,17) );
SQL> COMMIT;
```

После чего XSQL-страница используется подобно следующему файлу **point.xsql** для запроса к таблице LOCATION с помощью параметра *x*-coord.

```
<xsql:query connection="demo" xmlns:xsql="urn:oracle-xsql">
      SELECT name, origin
        FROM location loc
       WHERE loc.origin.x = {@x-coord}
</xsql:query>
```

Следующий URL может использоваться для возврата всех ячеек, где значение *x*-coord равно 11.

```
http://yourmachine.com/xsql/demo/point.xsql?x-coord=11
<ROWSET>
   <ROW num="1">
      <NAME>Someplace</NAME>
      <ORIGIN>
         <X>11</X>
         <Y>17</Y>
      </ORIGIN>
   </ROW>
</ROWSET>
```

Этот пример демонстрирует, что вложенные атрибуты *X* и *Y* в структуре типа данных POINT столбца ORIGIN появляются автоматически как вложенные элементы <X> и <Y> в выходном XML-файле.

Поддержка условных SQL-команд в XSQL

Oracle XSQL-страницы представляют собой шаблоны, с помощью которых собираются динамичные XML-страницы на базе одного или нескольких параметризованных SQL-запросов. XSQL-процессор затем использует спецификацию XSLT для преобразования этой страницы и получения окончательного результата в XML, HTML или любом другом текстовом формате.

Представленная ниже XSQL-страница демонстрирует использование программы обработки операций *<xsql:ref-cursor-function>* и параметрического значения для получения краткой или подробной информации о списке книг с помощью хранимой функции из Oracle Reference Cursor:

```
<?xml version="1.0"?>
<xsql:ref-cursor-function connection="o817" MyPara="1" xmlns:xsql="urn:oracle-
xsql" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
   bookutils.bookdata({@MyPara});
</xsql:ref-cursor-function>
```

Хранимая функция в СУБД Oracle:

```
create or replace package Bookutils as
type BookCurType is ref cursor;
function BookData(p_Parameter IN NUMBER DEFAULT 1) return BookUtils.BookCurType;
end Bookutils
/
show errors;
create or replace package body Bookutils as
   function BookData(p_Parameter IN NUMBER) return BookUtils.BookCurType is
      bookcur BookCurType;
   begin
      if p_Parameter=1 then
         open bookcur for select* from scott.booklist;
      else
         open bookcur for select title, author from scott.booklist;
      end if;
   return bookcur;
   end BookData;
end Bookutils;
/
show errors;
```

Пример с продажей книг

В этом разделе демонстрируется создание Web-сайта с использованием XSQL-страниц и СУБД Oracle9i в качестве репозитория данных. Здесь будут использоваться концепции, обсуждавшиеся в этой главе для иллюстрации реального примера с управлением онлайновым книжным каталогом. Рассмотрим пример от самого начала и до конца, от проектирования до реализации. Этой информации вполне достаточно, чтобы начать строить собственные Web-сайты с поддержкой технологии XML.

Проектирование схемы базы данных

Наша цель — разработать простой Web-сайт для управления книжным каталогом. Прежде всего следует определить модель хранения каталога в базе данных. В нашем примере используется модель, показанная на рис. 3.6. Она иллюстрирует основные входные данные: название книги, авторы, отзывы.

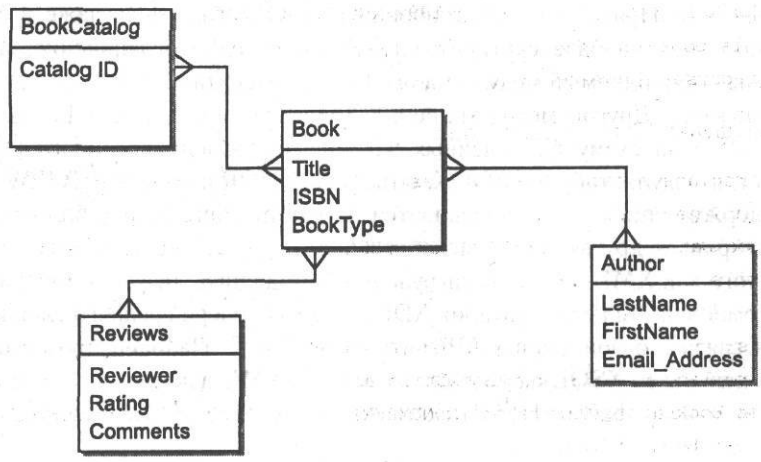


Рис. 3.6. *Модель данных для системы управления книжным каталогом*

Любая книга имеет связь с автором типа "многие-ко-многим", поскольку у нее может быть много авторов (или хотя бы один), а каждый автор может написать несколько книг. Книга также имеет связь с отзывами типа "один-ко-многим". Эта связь является необязательной, поскольку у книги вообще может не оказаться ни одного отзыва. Эту модель можно преобразовать в следующий DTD:

```
<!-- DTD for Book Data -->
<!ELEMENT BOOKCATALOG (BOOK)*>
<!ELEMENT BOOK (TITLE, AUTHOR+, PUBLISHER?, PUBLISHYEAR?, PRICE?, REVIEWS*)>
<!ATTLIST BOOK ISBN CDATA #REQUIRED>
<!ATTLIST BOOK BOOKTYPE (Fiction|SciFi|Fantasy) #IMPLIED>
<!ELEMENT TITLE (#PCDATA)>
<!ELEMENT AUTHOR (LASTNAME, FIRSTNAME, EMAIL_ADDRESS)>
<!ELEMENT LASTNAME (#PCDATA)>
<!ELEMENT FIRSTNAME (#PCDATA)>
<!ELEMENT EMAIL_ADDRESS (#PCDATA)>
<!ELEMENT PUBLISHER (#PCDATA)>
<!ELEMENT PUBLISHYEAR (#PCDATA)>
<!ELEMENT PRICE (#PCDATA)>
<!ELEMENT REVIEWS (REVIEWER, RATING, COMMENTS)>
```

```
<! ELEMENT REVIEWER (#PCDATA)>
<! ELEMENT RATING (#PCDATA)>
<! ELEMENT COMMENTS (#PCDATA) >
```

XML-документ (каталог книг), соответствующий этому DTD, можно сохранить в базе данных несколькими способами. Во-первых, его можно сохранить как CLOB-объект, а для поиска и извлечения информации из этого документа использовать Oracle Text. Правда, модель хранения в CLOB-объекте, и даже в *XMLType*, который построен на базе скрытого CLOB, имеет одно ограничение, поскольку минимальный обновляемый модуль такого CLOB-объекта — это весь сформатированный документ. Другой метод хранения XML-документа в базе данных — отображение DTD на схему базы данных. Схема для вышеприведенного DTD будет содержать три модуля: таблицу BOOK, объект AUTHOR и объект REVIEW. Таблица BOOK содержит список книг и ссылается на две подчиненные таблицы: одна для списка авторов, а другая для списка отзывов.

После того как XML-документ загружен в базу данных, нужно установить XSQL Servlet и компоненты инструментария XDK на Web-сервере среднего слоя и использовать для связи с базой данных API-интерфейс JDBC. Наконец, нужно построить Web-сайт, используя XSQL-файлы для получения XML-документов и преобразования их в выходные файлы HTML-формата.

Проектирование Web-сайта с использованием XSQL

Есть разные способы проектирования и построения Web-сайта с использованием JavaScript, Java Server Pages и динамического языка HTML. Но в этом разделе представлен очень простой способ, в котором используется только HTML. Мы по шагам рассмотрим использование XSQL-страниц для получения информации из базы данных. XSQL-страницы можно использовать в качестве строительных блоков при разработке более сложного Web-сайта. Но из данных, содержащихся в каталоге BOOKCATALOG, возьмем только две Web-страницы:

- Страницу, содержащую список всех книг в каталоге
- Страницу, содержащую подробное описание книги, определяемое номером ISBN

Создание Web-страницы происходит в два этапа. Во-первых, с помощью XSQL нужно выбрать данные, необходимые для создания Web-страницы, а затем провести HTML-форматирование с использованием XSLT-преобразования. Если проектируется относительно сложная Web-страница, для преобразования данных в JavaScript применяется XSLT. Можно также включить XSQL-страницу в Java Server Pages и использовать для извлечения данных API-интерфейсы DOM.

Просмотр каталога

Для страницы просмотра каталога не нужна полная информация о каждой книге. Сюда стоит выводить только некоторые атрибуты книг, например номер ISBN, название книги, фамилию автора, издательство и цену. Для этого используется следующая XSQL-страница:

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/html" href="catalog.xsl"?>
<xsql:query xmlns:xsql="urn:oracle-xsql"
            connection="demo"
            rowset-element="BOOKCATALOG"
            row-element="BOOK">
    SELECT ISBN, Title, Author.Lastname, Publisher, Price
      FROM scott.BookCatalog
</xsql:query>
```

Эта XSQL-страница возвращает канонический XML-документ, содержащий список книг. XSQL-процессор преобразует XML-документ с помощью XSLT-таблицы стилей "catalog.xsl", поставленной ему в соответствие в команде по обработке документа xml-таблицей стилей.

```
<?xml version="1.0"?>
<BOOKCATALOG>
   <BOOK>
      <ISBN>1234-123456-1234</ISBN>
      <TITLE>C Programming Language</TITLE>
      <AUTHOR_LASTNAME>Kernighan</AUTHOR_LASTNAME>
      <PUBLISHER>EEE Editions</PUBLISHER>
      <PRICE>7.99</PRICE>
   </BOOK>
   <BOOK>
      <ISBN>3456-34567890-3456</ISBN>
      <TITLE>C++ Primer</TITLE>
      <AUTHOR_LASTNAME>Lippmann</AUTHOR_LASTNAME>
      <PUBLISHER>McGraw Hill</PUBLISHER>
      <PRICE>4.99</PRICE>
   </BOOK>
   <BOOK>
      <ISBN>2137-598354-65978</ISBN>
      <TITLE>Twelve Red Herrings</TITLE>
      <AUTHOR_LASTNAME>Archer</AUTHOR_LASTNAME>
      <PUBLISHER>Harper Collins</PUBLISHER>
      <PRICE>12.95</PRICE>
   </BOOK>
```

```
    <BOOK>
       <ISBN>237864-4787834-3459</ISBN>
       <TITLE>The Eleventh Commandment</TITLE>
       <AUTHOR_LASTNAME>Archer</AUTHOR_LASTNAME>
       <PUBLISHER>Harper Collins</PUBLISHER>
       <PRICE>3.99</PRICE>
    </BOOK>
    <BOOK>
       <ISBN>1230-23498-2349879</ISBN>
       <TITLE>Emperor's New Mind</TITLE>
       <AUTHOR_LASTNAME>Penrose</AUTHOR_LASTNAME>
       <PUBLISHER>Oxford Publishing Company</PUBLISHER>
       <PRICE>15.99</PRICE>
    </BOOK>
</BOOKCATALOG>
```

Далее можно использовать простую таблицу стилей для преобразования сгенерированного XML-документа в таблицу формата HTML. Это показано в следующем примере, а результат преобразования можно увидеть на рис. 3.7.

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

<xsl:template match="/">
   <HTML>
      <body bgcolor="#FFFFFF" text="#000000">
      <img align="center" src="mylogo.gif" border="0"/>
      <h1 align="center"> Scott's book qatalog </h1>
      <xsl:apply-templates/>
      </body>
   </HTML>
</xsl:template>

<xsl:template match="BOOKCATALOG">
   <table align="center" border="10" cellspacing="10" cellpadding="2">
      <th>ISBN</th>
      <th>Title</th>
      <th>Author</th>
      <th>Publisher</th>
      <th>Price</th>
      <xsl:apply-templates select="BOOK"/>
   </table>
</xsl:template>
```

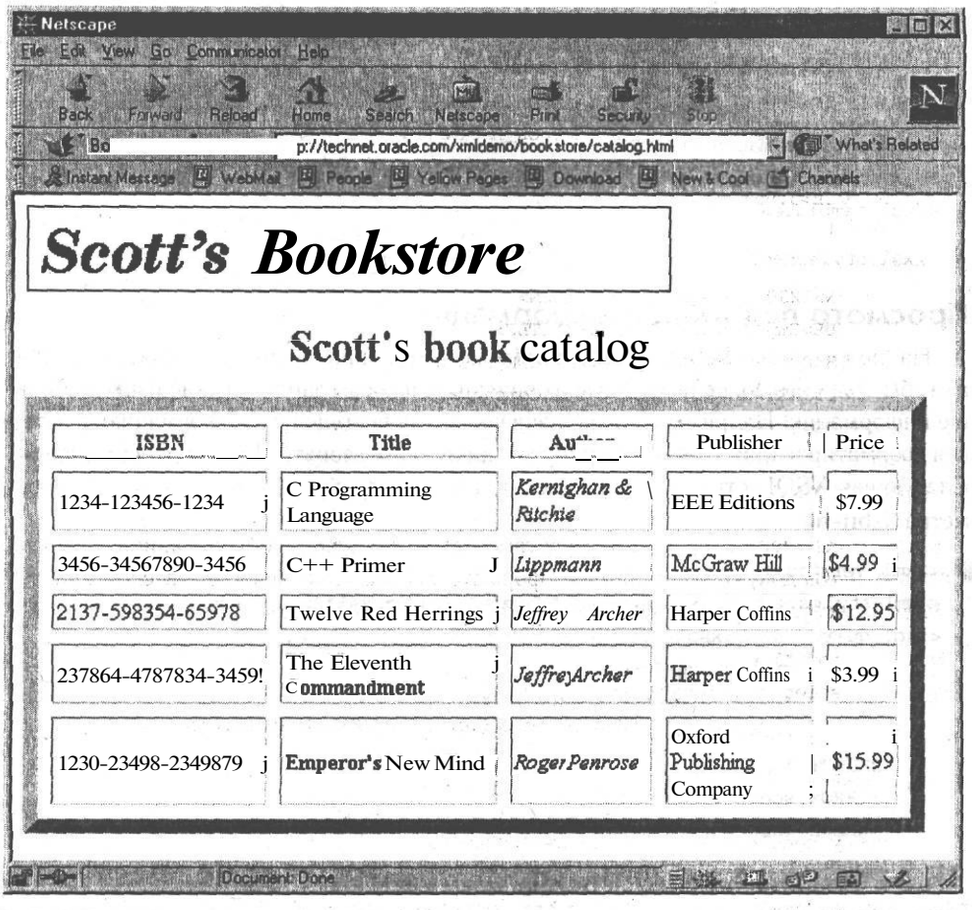


Рис. 3.7. *Вид экрана Web-страницы с каталогом книг*

```
<xsl:template match="BOOK">
   <trxxsl :apply-templates/></tr>
</xsl:template>

<xsl:template match="TITLE">
   <td><b><xsl:apply-templates/></b></td>
</xsl:template>

<xsl:template match="AUTHOR_LASTNAME">
   <td><i><xsl:apply-templates/></i></td>
</xsl:template>

<xsl:template match="PUBLISHER">
   <td><xsl:apply-templates/></td>
</xsl:template>
```

```
<xsl:template match="ISBN">
   <td><xsl:apply-templates/></td>
</xsl:template>

<xsl:template match="PRICE">
   <td>$<xsl:apply-templates/></td>
</xsl:template>

</xsl:stylesheet>
```

Просмотр подробной информации

На полученной Web-странице каталога есть лишь некоторые атрибуты книг, а чтобы получить более подробные их описания, из базы данных нужно извлечь больше информации. Например, можно извлечь оттуда вообще всю информацию, а затем для форматирования нужных атрибутов книг использовать XSLT-преобразование. Следующая XSQL-страница извлекает из БД все атрибуты книги с помощью параметра **isbn-no:**

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/html" href="catalog.xsl"?>
<xsql:query xmlns:xsql="urn:oracle-xsql"
            connection="demo"
            rowset-element="BOOKCATALOG"
            row-element="BOOK">
    SELECT *
      FROM scott.BookCatalog
     WHERE ISBN = {@isbn-no}
</xsql:query>
```

Тогда будет извлечен следующий XML-документ, который можно сформатировать в виде HTML-страницы, показанной на рис. 3.8.

```
<?xml version="1.0"?>
<BOOKCATALOG>
   <BOOK>
      <ISBN>2133-23700-5978</ISBN>
      <TITLE>My favorite title</TITLE>
      <AUTHOR>
         <LASTNAME>Smith</LASTNAME>
         <FIRSTNAME>Mark</FIRSTNAME>
         <EMAIL>mssmith@mydomain.com</EMAIL>
      </AUTHOR>
      <BOOKTYPE>Fiction</BOOKTYPE>
      <PUBLISHER>Harper Collins</PUBLISHER>
      <PUBLISHYEAR>1982</PUBLISHYEAR>
```

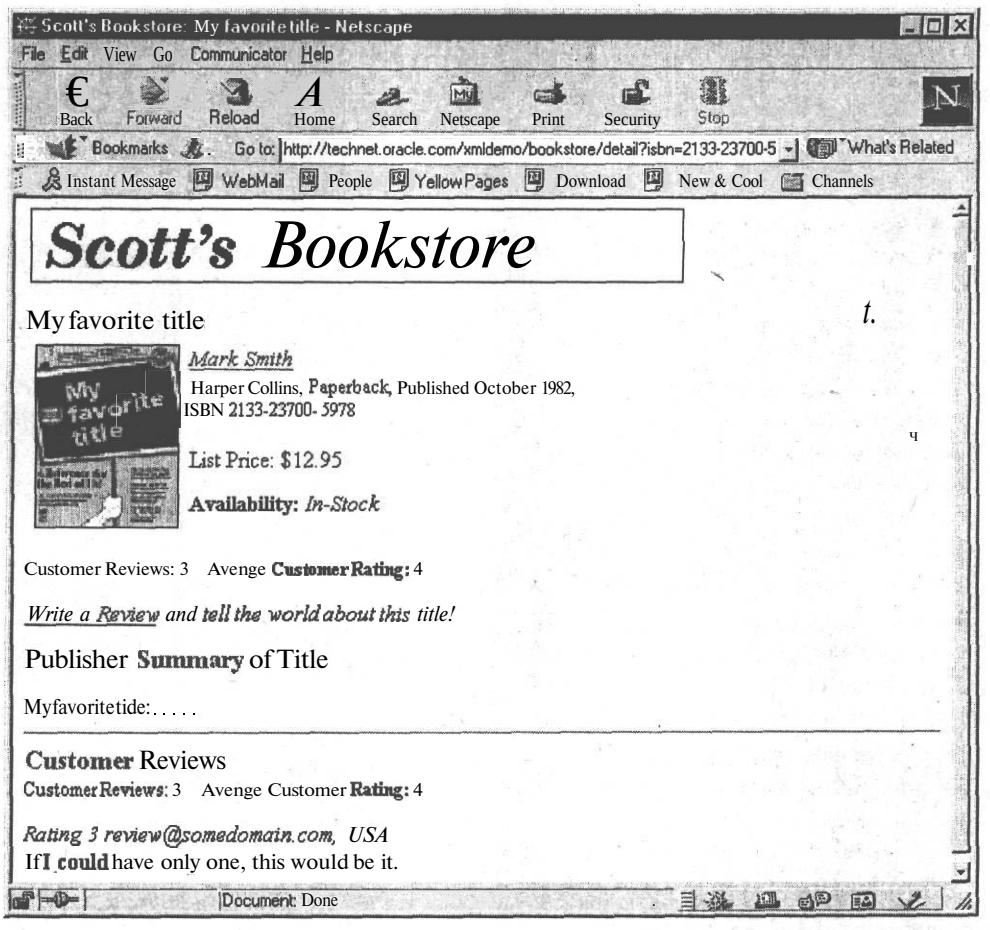


Рис. 3.8. *Вид экрана Web-страницы с подробной информацией о книге*

```
        <PRICE>12.95</PRICE>
        <REVIEW_LIST>
           <REVIEW>
              <REVIEWER>review@somedomain.com</REVIEWER>
              <RATING>3</RATING>
              <COMMENTS>
                 ...
              </COMMENTS>
           </REVIEW>
           ...
        </REVIEW_LIST>
     </BOOK>
  </BOOKCATALOG>
```

овсеместное использование Интернет-технологий смогло преодолеть врожденные ограничения традиционных клиент-серверных приложений и свойственной им двухуровневой модели компьютерных вычислений. Еще на заре существования Интернета все пользователи сети, от ИТ-менеджеров до разработчиков, быстро подхватили идею использования универсального Web-браузера. Теперь по одному щелчку мышью пользователи могут получать доступ к тексту, видео, графике и звуковым записям, хранящимся на HTML-страницах. *Протокол Hypertext TransportProtocol (HTTP),* исполняемый поверх протоколов TCP/IP, позволяет получать доступ к HTML-страницам, хранящимся на любом сервере, подключенном к системе *World Wide Web* (или попросту Web). А появление в Интернете массы рядовых потребителей привело к разработке Web-приложений, которые в противном случае вообще никогда не были бы созданы.

Появление языка Java только ускорило этот процесс. Платформно-независимые Java-приложения теперь можно запускать на любом компьютере, где есть виртуальная Java-машина (Virtual Java Machine). Толстые клиенты оказались не нужными, а для получения окончательного результата работы приложения стали использоваться клиентские Java-программы. Web-браузеры могут загружать Java-апплет с центральных серверов и запускать его на исполнение. Таким способом платформно-независимые приложения и данные распределяются по клиентским машинам по мере необходимости. Кроме того, многие пользователи уже оценили возможность распространения приложений масштаба предприятия на более широкую аудиторию, и работающие на базе браузеров клиенты функционируют сегодня не только в *локальных сетях* (local area network, *LAN),* как того требовали традиционные двухуровневые приложения типа клиент-сервер, но и в *глобальных сетях* (wide area network, *WAN).*

Как уже было сказано, двухуровневая модель компьютерной обработки с толстым сервером исторически оказалась самой распространенной в системе Web. Согласно этой модели почти вся логика приложения, кроме HTTP-серверов с их статичными HTML-страницами и *CGI*-приложениями, находится на уровне сервера. Аналогично, приложения баз данных типа клиент-сервер были традиционно разделены на серверный слой (база данных) и относительно тонкий клиентский слой, и каждый из этих слоев содержит и данные, и логику приложения в форме хранимых процедур.

Хотя Web-приложения, построенные по модели клиент-сервер, часто хорошо работают в средних и больших группах пользователей, тем не менее они плохо масштабируются. Когда счет пользователям и транзакциям начинает идти на миллионы,

клиент-серверные системы быстро исчерпывают свои возможности по росту производительности. Другим недостатком архитектур типа клиент-сервер является их неспособность поддерживать постоянное соединение между клиентом и базой данных, что особенно важно в современных Web-приложениях. Сама природа клиент-серверного соединения требует, чтобы клиент устанавливал новое соединение для каждой транзакции, из-за чего растет сетевой трафик и падает общая производительность. Проблемы, связанные с масштабируемостью, сопровождением и постоянством соединения, стали реальным ограничением для архитектур клиент-сервер, особенно для приложений, распределенных по разнородным аппаратным платформам с разными операционными системами.

Недавно появилась новая трехуровневая модель, изолирующая большую часть логики приложения на среднем слое, который находится между клиентским слоем (браузером) и серверным слоем (база данных). С увеличением числа промежуточных слоев архитектура становится еще более объектно-ориентированной, что также улучшает инкапсуляцию важнейшей логики приложения. Конечным результатом этого процесса является *n-уровневая архитектура,* позволяющая разбивать логику приложения на компоненты, которые можно легко модифицировать и повторно использовать, а также разворачивать на нескольких физических машинах, работающих на базе разных платформ. Серверы промежуточного слоя также могут управлять соединениями с БД и делать их *зависимыми от состояния,* чтобы клиенты и серверы могли находиться в состоянии соединения во время выполнения нескольких транзакций, а приложения можно было распространять по нескольким машинам и платформам. Все это еще больше увеличивает масштабируемость, производительность и общую гибкость работы приложения.

Использование n-уровневой архитектуры — первый шаг в создании надежных Web-приложений масштаба предприятия. Для того чтобы использовать преимущества n-уровневой архитектуры, разработчикам нужна интегрированная среда для разработки и развертывания компонентов промежуточного уровня. Поэтому появилась новая категория продуктов — серверы приложений. *Сервер приложения* является центральной частью промежуточного слоя. Он предоставляет единую платформу для совместно используемых приложений масштаба предприятия. Сервер приложений обеспечивает работу таких жизненно важных служб, как система безопасности, брокер сообщений, взаимодействие с базой данных, управление транзакциями и изолирование процессов, не забывая при этом о качестве обслуживания.

Серверы приложений Oracle Application Server (*OAS*) и *Oracle9i Internet Application Server* (*iAS*) интегрируют все основные службы и функции, необходимые для построения, разворачивания и администрирования высокопроизводительных n-уровневых ориентированных на транзакции Web-приложений на основе открытых стандартов. Эти серверы обладают следующими важными возможностями:

- HTTP Web Server и поддержка других популярных Web-серверов
- База данных и унаследованное промежуточное ПО доступа для связи со всеми основными базами данных

- Поддержка брокера объектных запросов CORBA для масштабируемой межплатформной распределенной объектной компьютерной обработки
- *Монитор транзакций* Transaction Processing *(TP)* Monitor для балансировки нагрузки, организации пула ресурсов и обработки транзакций
- Сетевые службы Network Services: система безопасности и служба каталогов
- Ориентированное на сообщения промежуточное ПО для расширяемости, позволяющее упростить ' разворачивание новых приложений

Серверы OAS и iAS обеспечивают межплатформную поддержку сетевых клиентов (HTML и Java), Web-серверов и баз данных, что позволяет сохранить инвестиции в унаследованные клиент-серверные системы.

В начале этой главы рассматриваются фундаментальные особенности архитектуры OAS и те особенности архитектуры iAS, которых нет в OAS. Затем на примере приложения Bookstore (книжный магазин), реализованного в виде OAS-сервлета, показано, как применить преобразования таблицы стилей XSLT к XML-документам, полученным из базы данных. Приложение Bookstore также иллюстрирует разные способы доступа в базу данных из сервера OAS. Наконец, один из разделов посвящен конфигурированию сервлета для LAS.

Архитектура Oracle Application Server

По определению, *Oracle Application Server* (OAS) — это совокупность связующих средств и служб, а также платформа для распределенных исполняемых на сервере приложений. OAS реализует промежуточный слой с использованием CORBA — стандартной модели для распределенного объектно-ориентированного программирования. Технология CORBA определяет поведение брокеров объектных запросов *Object Request Brokers (ORBs).* Брокер объектных запросов ORB отвечает за направление запросов от клиентов и других ORB-брокеров к объектам CORBA. Клиенты взаимодействуют с ORB-брокерами с помощью протокола *Internet Inter-ORB Protocol (ПОР),* а не по протоколу HTTP.

Сервер OAS поддерживает доступ к клиентам двух типов: Web-браузерам по протоколу HTTP и CORBA-клиентам, получающим доступ к приложениям JCORBA и EJB по протоколу ПОР. Для CORBA-клиентов сервер OAS предоставляет ORB-брокер, совместимый с CORBA 2.O. Эти клиенты могут быть Java-апплетами, картриджами, автономными приложениями или другими JCORBA- или EJB-объектами в том же самом или в других приложениях.

Группа тесно связанных машин, на которых работает сервер OAS, образует *сайт.* В каждом сайте одна из машин называется *главным узлом,* на ней находится ORB-брокер, модуль доступа *администратора ресурсов* Resource Manager (*RM*) и информация о конфигурации всего вычислительного центра. Остальные машины сайта называются *удаленными узлами.*

Сервер OAS состоит из трех основных компонентов:

- Программы, прослушивающие соединения по протоколу HTTP
- Компоненты OAS
- Приложения-картриджи

Программы, прослушивающие соединения по протоколу HTTP

Прослушивающая программа — это HTTP-сервер, обрабатывающий запросы на статичные страницы и CGI. Сервер OAS имеет свою собственную прослушивающую программу Oracle Web Listener, но поддерживает и другие программы того же назначения от других разработчиков, а именно:

- Netscape FastTrack Server
- Netscape Enterprise Server
- Microsoft Internet Information Server
- Apache (не-NT версия)

Все запросы, кроме запросов на статичные страницы и CGI, передаются в диспетчеры OAS для обработки. В сервере OAS есть специальные адаптеры, позволяющие прослушивающим программам работать в тесной интеграции с диспетчерами, что оптимизирует производительность работы сервера.

Компоненты OAS

Сервер OAS предоставляет полный набор служб для приложений, развернутых на сервере в виде картриджей. В число этих служб входят система балансировки нагрузки, службы транзакций, обработки ошибок и восстановления неправильно выполненных процессов, система безопасности и служба каталогов. Диспетчеры сервера OAS обрабатывают запросы к приложению от программ, прослушивающих HTTP-соединения, и CORBA/IIOP-запросы. Получив HTTP-запрос, который не идентифицируется как запрос на статичную HTML-страницу или CGI-программу, прослушивающая программа направляет его своему диспетчеру, назначающему для его выполнения копию картриджа соответствующего типа.

Диспетчер OAS Dispatcher обрабатывает HTTP-запросы, управляя пулом копий картриджа, работающих на одном или нескольких узлах. Когда приходит запрос, диспетчер направляет его на одну из копий находящихся в кэше картриджей соответствующего типа. Если у диспетчера нет копии такого картриджа, он отправляет на сервер OAS запрос о создании нового сервера картриджей с копией нужного картриджа. Затем диспетчер добавляет эту копию картриджа в свой кэш и назначает ей обработку отложенного запроса. После выполнения запроса диспетчер сохраняет новую копию картриджа, чтобы она была доступна для обработки нового запроса.

Proxy-компонент администратора ресурсов OAS RM обрабатывает CORBA/IIOP-запросы, получая JCORBA объектные ссылки от имени внешних клиентов. Когда клиент запрашивает объектную ссылку, proxy-компонент RM при необходимости обрабатывает запрашиваемый объект в сервере JCORBA и возвращает ссылку клиенту. Затем клиент использует эту объектную ссылку для вызова методов непосредственно к объекту.

На каждом Web-сайте существует один proxy-компонент RM, а на каждом узле Web-сайта есть один диспетчер, связанный с каждой прослушивающей программой.

Картриджи приложений

Картридж — это исторически сложившийся в Oracle термин для бизнес-компонентов. В контексте сервера OAS оба термина семантически эквиваленты, но термин *"картридж"* используется для того, чтобы добавить некий дух Oracle в описываемую тему. Картриджи — это объекты, управляемые и разворачиваемые на сервере OAS, это совместно используемые библиотеки, реализующие программную логику и предоставляющие доступ к программной логике, хранимой в каком-то другом месте, отличном от сервера, например в базе данных. *Сервер картриджей* — это процесс, в котором запускаются на исполнение один или несколько копий картриджа. *Копия картриджа* — это объект CORBA, который выполняется внутри процесса сервера картриджей, исполняющего код картриджа.

OAS — это платформа для самых разных языковых сред, поэтому большинство картриджей, связанных с сервером OAS, ориентированы на API-интерфейс совершенно определенного языка программирования и поддерживаемую библиотеку. В следующих разделах описываются картриджи, входящие в состав сервера OAS 4.0.8.

PL/SQL-картридж

PL/SQL-картридж запускает в БД Oracle хранимые процедуры PL/SQL. При этом для определения местоположения базы данных, с которой нужно установить соединение, используется *дескриптор доступа к базе данных* Database Access Descriptor (DAD). Этот картридж запускает на исполнение файлы, содержащие исходный код PL/SQL. Он загружает содержимое такого файла в БД и исполняет этот код. PL/SQL-картридж поставляется в составе инструментария PL/SQL Web Toolkit, с помощью которого можно получать информацию о запросе, определять значения таких HTTP-заголовков, как тип содержимого и cookies, и генерировать HTML-теги.

Jweb-картридж

Jweb-картридж запускает на исполнение Java-приложения. В его состав входит виртуальная Java-машина, которая интерпретирует байт-коды для Java-приложений. JWeb картридж поддерживает два разных метода для связи с базой данных и доступа к хранимой в ней информации. Первый метод состоит в использовании Java-упаковщиков хранимых процедур и функций PL/SQL. Для генерации интерфейсного Java-класса используется средство pl2java, а хранимые PL/SQL процедуры вызываются через их прототипные методы. Второй механизм доступа к базе данных можно реализовать с помощью интерфейса *Java Database Connectivity (JDBC),* который позволяет Исполнять SQL-опраторы непосредственно из приложения. Jweb-картридж поставляется в составе инструментария PL/SQL Web Toolkit, с помощью которого можно получать информацию о запросе, определять значения таких HTTP-заголовков, как тип содержимого и cookies, и генерировать HTML-теги.

Jservlet-картридж

Jservlet-картридж специализируется на обработке HTTP-запросов. Он отличается от большинства других сервлетов тем, что совместно используется многими JWeb-службами. Основное достоинство OAS Jservlet-картриджа над остальными сервлетами, работающими на одной хост-машине, состоит в том, что у него есть функция автоматического распределения одного и того же JServlet-приложения по нескольким хост-машинам на O AS- сайте.

C-картридж

C-картридж запускает на исполнение C-приложения, поддерживающие функции обратного вызова, активизируемые сервером OAS. C-картридж поставляется в составе API-интерфейса WRB, который содержит функции и структуры данных, а они, в свою очередь, поддерживают возврат информации о запросе, определяют значения таких HTTP-заголовков, как тип содержимого и cookies, и отправляют запросы на другие картриджи.

LiveHTML-картридж

LiveHTML-картридж интерпретирует *документы-вставки на стороне сервера* (server-side includes, *SSI*). SSI — это стандарт, позволяющий вести поиск данных в HTML-документах. Для маркировки мест, где этот картридж заменяет динамические данные, в LiveHTML-документах используются специальные теги. Этот картридж также позволяет использовать для генерации динамических данных встроенные Perl-скрипты.

Perl-картридж

Perl-картридж запускает на исполнение Perl-скрипты. Он поставляется вместе с некоторыми Perl-модулями, в том числе с DBI/DBD, которые позволяют получать доступ к базам данных Oracle.

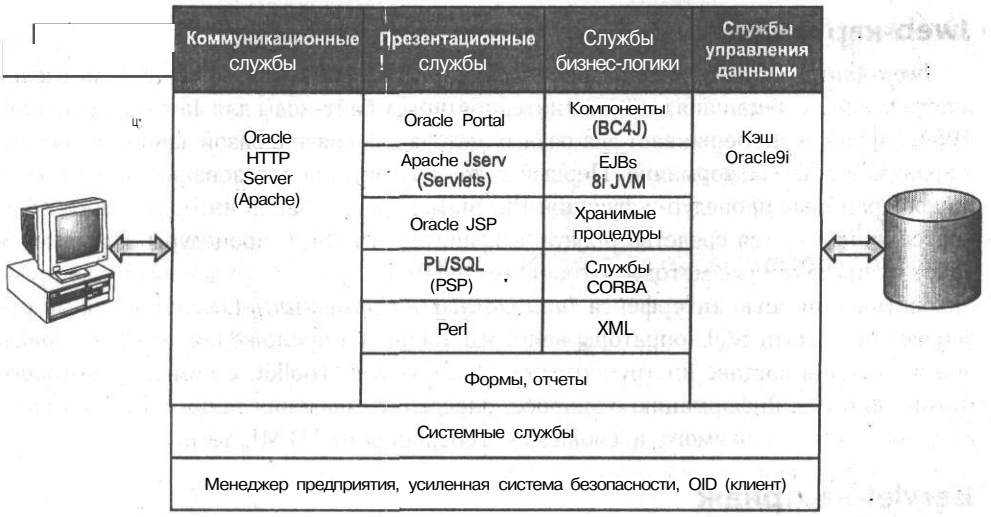


Рис. 4.1. *Архитектура служб Oracle iAS*

Архитектура Oracle Internet Application Server

Как и в случае с сервером OAS, сервер iAS представляет собой коллекцию служб и средств для разворачивания на среднем слое приложений уровня предприятия. Но на этом вся аналогия заканчивается, так как iAS — это полная переработка большинства служб, которые можно разделить на пять категорий:

- Коммуникационные службы
- Презентационные службы
- Службы бизнес-логики
- Службы управления данными
- Системные службы

На рис. 4.1 показана структурная схема служб сервера iAS и их компоненты.

Коммуникационные службы iAS

Коммуникационные службы обрабатывают входящие запросы, получаемые сервером Oracle Internet Application Server. Запросы обрабатываются сервером Oracle HTTP Server или направляются для обработки в другие службы iAS. Эти службы состоят из компонентов, обеспечивающих поддержку технологий ПОР, RMI и Net8. Кроме того, сервер Oracle iAS способен дополнительно поддерживать мобильные и беспроводные технологии, в том числе Wireless Access Protocol (WAP).

Oracle HTTP Server

Oracle HTTP Server построен на базе сервера Apache. Apache — это стандарт компьютерной индустрии для прослушивающих Web-программ в Интернете. Этот сервер установлен более чем на 60% Интернет-сайтов всего мира. Поэтому включение в iAS HTTP-сервера на базе Apache создает масштабируемую доступную платформу, которая уже тщательно протестирована.

Модули сервера Oracle HTTP Server

Модули представляют собой дополнительные сменные программы для HTTP Server, расширяющие его функциональные возможности, предлагая специальные службы (например, mod_ssl) или производя диспетчеризацию запросов к внешним процессам (например, mod_jserv направляется на Apache JServ). Кроме скомпилированных модулей Apache, предоставляемых Oracle HTTP-сервером, Oracle имеет несколько усовершенствованных стандартных модулей и дополнительные модули, специфичные только для Oracle. Они описаны в следующих разделах.

mod_plsql Модуль mod_plsql направляет HTTP-запросы к хранимым процедурам в СУБД Oracle8i для обработки. Он исполняется внутри процесса HTTP Server и представляет собой обновление для картриджа PL/SQL в сервере приложений Oracle Application Server.

mod_ose Этот модуль направляет HTTP-запросы к приложениям Java-сервлета и к страницам Java Server Pages (JSP), сконфигурированным для сервлетного механизма Oracle Servlet Engine (OSE), на одну из копий этих приложений или страниц, работающих на сервере iAS. Применяя HTTP-туннелирование по протоколу Net8 между сервером и сервлетным механизмом, mod_ose может использовать менеджер соединений, функцию балансировки нагрузки и поддержку сетевого экрана.

mod_ssl Этот модуль обеспечивает работу стандартного сервера HTTP Server, который полностью поддерживается Oracle. Он позволяет устанавливать защищенные соединения между HTTP-сервером и клиентом-браузером, используя для этого поставляемый Oracle механизм шифрования поверх протокола Secure Socket Layer (SSL). Его можно также использовать для аутентификации через Интернет по технологии цифровых сертификатов.

mod_perl Этот модуль направляет запросы Perl-приложения на Perl-интерпретатор, встроенный в Oracle HTTP Server. Встроенный Perl-интерпретатор сохраняет передаваемую информацию с момента запуска внешнего интерпретирующего процесса. Благодаря функции кэширования кода, которая предполагает только однократную загрузку и компиляцию модулей и скриптов, сервер запускает ранее загруженный и откомпилированный код. В результате этого экономятся ресурсы и время на исполнение повторяющихся Perl-программ. Эта функция более подробно описана в разделе "Perl-интерпретатор".

mod_jserv Этот модуль направляет все запросы Java-сервлета и Java Server Page (JSP) на сервлетный процессор Apache JServ по протоколу Apache JServ Protocol (AJP). A OracleJSP будет управлять исполнением страницы из среды Apache JServ. Этот процесс подробнее описан в разделе "Apache JServ".

Презентационные службы iAS Presentation Services

Презентационные службы для выходных данных Oracle iAS представляют собой целый ряд различных графических представлений, большинство из которых имеет формат HTML. Сервер Oracle iAS поддерживает самые разные способы генерации презентаций, передающихся на клиентскую машину. Это и программирование низкого уровня с использованием Perl-скриптов и Java-сервлетов, и структуры высокого уровня, в которых используются службы Oracle Portal. Ниже описываются все существующие в Oracle LAS презентационные службы.

Portal-службы (Oracle Portal)

Oracle Portal — это приложение, работающее в реальном времени, и средство для построения порталов и их управления. Portal предоставляет сотрудникам компании или рядовым пользователям возможность публиковать в Web самую разную информацию и управлять ею. Администрирование опубликованных ими данных может быть централизованным, чтобы на всем сайте выдерживалось единство стиля. Кроме того, Oracle Portal обеспечивает внутренний доступ к содержимому сайта, к выполнению транзакций и к бизнес-приложениям. Помимо всех этих функциональных возможностей Oracle Portal использует технологии работы по одному паролю, персонализации и портлета, чтобы обеспечить единый способ просмотра внешнего контента, корпоративной информации и приложений. Для интегрирования других приложений и синдицирования контента, в которых задействованы технологии Java и XML, можно воспользоваться API-интерфейсами инструментария разработчика Portal Developer's Kit.

Apache JServ

Apache JServ — это процессор Java-сервлета, который полностью соответствует спецификации Java Servlet 2.0 API компании Sun Microsystems. Как и Oracle Portal, Apache JServ реализован в виде двух основных частей. Первая часть — это сам сервлетный процессор, представляющий собой чистый Java-код. Он работает на виртуальной Java-машине версии 1.1 или более поздней. Вторая часть — это расширение HTTP-сервера mod_jserv, работающее в рамках процесса HTTP-сервера и направляющее запросы HTTP-сервлета на копии сервлетного процессора. Это расширение взаимодействует с сервлетным процессором по протоколу Apache JServ Protocol (AJP), который работает поверх стека протоколов TCP/IP и позволяет запускать сервлетный процессор на HTTP-сервере либо локально, либо удаленно.

OracleJSP

OracleJSP — это интерпретатор и процессор реального времени для страниц Java Server Page (JSP). JSP-страницы, в свою очередь, определяют простой способ разделения представлений и бизнес-логики. Можно создавать очень динамичные Web-страницы, зная язык Java и не зная языка HTML. OracleJSP представляет собой JSP-контейнер, т. е. программный модуль, который транслирует, исполняет и обрабатывает JSP-страницы и доставляет на них запросы. OracleJSP запускается на процессоре Apache JServ или на любом другом стандартном сервлетном процессоре, поддерживающем спецификацию сервлета версии 2.0 или более позднюю. OracleJSP транслирует JSP-страницу в сервлетный код, который затем компилируется, после чего его можно вызывать двумя способами: динамически, с помощью JSP-контейнера, как результат пользовательского **.jsp** URL-запроса, или явным образом (это делает разработчик JSP). Кроме того, OracleJSP предоставляет исключительные функциональные возможности приложениям БД благодаря расширенной поддержке SQLJ, национальных языков и дополнительных библиотек тегов.

Страницы PL/SQL

Страницы PL/SQL Server Pages очень похожи на страницы Java Server Page, за исключением того, что они используют для создания серверных скриптов язык PL/SQL, а на Java. В состав Oracle PSP входит PSP-компилятор и инструментарий PL/SQL Web Toolkit. Благодаря PSP легко создавать и запускать Web-страницы, активно использующие базы данных, а также интегрировать уже существующие приложения, написанные на языке PL/SQL.

Perl-интерпретатор

Perl-среда реального времени встроена в процесс Oracle HTTP Server, что позволяет сэкономить на повторной передаче данных при запуске внешнего интерпретатора или при вызовах на обработку данных для исполнения Perl-скриптов. Когда Oracle HTTP Server получает HTTP-запрос на исполнение Perl-скрипта, этот запрос направляется в модуль mod_perl, который, в свою очередь, направляет запрос для обработки на Perl-интерпретатор.

Службы бизнес-логики сервера iAS

Сервер Oracle iAS позволяет разрабатывать бизнес-логику несколькими способами, используя для этого и технологию Java, и модели высокого уровня. Такие способы предусматривают использование Java-технологий, например J2EE, EJB и Oracle Business Components for Java, а также методов, ориентированных на графические интерфейсы GUI, например Oracle Forms and Reports. В следующих разделах описаны основные элементы, обеспечивающие работу служб бизнес-логики в Oracle Internet Application Server.

Java-процессор внутри сервера Oracle iAS — это виртуальная Java-машина Oracle8i JServer JVM. Виртуальная Java-машина JServer изначально была создана как часть СУБД Oracle8i.

Oracle Business Components for Java

Oracle Business Components for Java (BC4J) — это структура, созданная только на языке Java и усиленная поддержкой технологии XML. С ее помощью можно разрабатывать, переносить на другие платформы и настраивать в соответствии с требованиями пользователей многоуровневые приложения с поддержкой баз данных. Причем все это можно делать из повторно используемых бизнес-компонентов. Эта среда используется для проектирования и тестирования бизнес-логики в компонентах, которые автоматически интегрируются в БД. Такую бизнес-логику можно потом повторно использовать в нескольких SQL-представлениях данных, поддерживающих разные задачи, выполняемые приложением. Получать доступ и обновлять хранимые данные могут сервлеты, страницы Java Server Pages и тонкие клиенты Java Swing. Среда BC4J поддерживает технологию XML и во внутренней структуре своих файлов свойств, и при работе с XML-данными и документами.

Поддержка модели Java-разработки

Созданный как хорошо масштабируемая серверная Java-платформа, сервер iAS представляет собой на 100% совместимую с технологией Java серверную среду масштаба предприятия, которая поддерживает технологии J2EE (включая сервлеты, JSP-страницы и Enterprise JavaBeans), CORBA и хранимые процедуры базы данных. Oracle9i JServer достигает высокой масштабируемости благодаря своей уникальной архитектуре, в которой сложности управления памятью удалось минимизировать даже при большом числе одновременно работающих на сервере пользователей.

Oracle9i PL/SQL-процессор

Oracle9i PL/SQL-процессор — это масштабируемая среда для исполнения PL/SQL хранимых процедур, PL/SQL Web-приложений и страниц PL/SQL Server Pages (PSPs). Oracle8i PL/SQL-процессор работает и внутри процесса СУБД Oracle, и на промежуточном слое, т. е. в сервере Oracle iAS. PL/SQL хранимые процедуры базы данных и PSP-страницы можно кэшировать в среднем слое и исполнять в Oracle8i PL/SQL-процессоре на сервере Oracle iAS, вынося обработку данных из процессора самой базы данных.

Службы Oracle Forms Services

Службы Oracle Forms Services входят в состав сервера iAS. Они позволяют запускать в Интернете или в корпоративной интрасети приложения, построенные с помощью Oracle Forms Developer. Службы Oracle Forms Services состоят из слушающей программы и процессора реального времени, который работает на сервере

iAS на промежуточном уровне. На клиентском уровне Oracle Forms Services состоят из тонкого клиентского Java-апплета Forms, обеспечивающего работу пользовательского интерфейса приложения. Клиент и сервер могут взаимодействовать друг с другом по закрытому протоколу непосредственно по каналам двусторонней связи, как правило существующим в корпоративных интрасетях. Сообщения между клиентом и сервером можно также передавать по протоколу HTTP через сетевые экраны для использования в Интернет-средах.

Службы Oracle Reports

Oracle Reports — это служба реального времени для запуска приложений Oracle Reports. Это ПО можно использовать для быстрого разворачивания приложений, которые могут динамически генерировать отчеты, содержащие большое количество данных, и публиковать их в стандартных Интернет-форматах через Web. В Oracle Reports есть также опция вывода отчетов в XML-формате, Эти отчеты можно преобразовывать, фильтровать или приводить к нужному формату с помощью XSL-таблиц стилей.

Discoverer Viewer

Discoverer Viewer — это среда реального времени для запуска через Web рабочих журналов Discoverer. С помощью этого ПО пользователи могут из окна стандартного браузера делать динамические специально подобранные запросы и анализировать информацию. Для настройки формата представления данных в среде Discoverer активно используются технологии XML и XSL.

Службы управления данными сервера iAS

Чтобы снизить нагрузку на сервер, на котором работает база данных, и избежать многократной пересылки туда и обратно данных, предназначенных только для чтения, в Oracle Internet Application Server имеется Oracle9i Cache.

Oracle9i Cache

Oracle9i Cache — это кэш для приложений и данных, предназначенных только для чтения. Он является компонентом сервера Oracle Internet Application Server и располагается на промежуточном уровне. Он увеличивает производительность и масштабируемость приложений, имеющих доступ в базы данных Oracle, за счет кэширования часто используемых данных и хранимых процедур на компьютере, обслуживающем промежуточное ПО. Благодаря Oracle9i Cache приложения могут в несколько раз увеличить число обрабатываемых запросов по сравнению со своей изначальной производительностью. Обработка запросов к базе данных на среднем уровне заметно уменьшает время на отправку и прием данных в сети. Это снижает нагрузку на серверный уровень базы данных, что дает возможность обслужить больше пользователей при том же самом оборудовании.

Хорошим примером приложения, которое может существенно выиграть от применения Oracle9i Cache, является уже знакомый нам Web-сайт книжного магазина. На этом Web-сайте находится база данных всех книг магазина с указанием названия, авторов, аннотаций и прочей информации. Данные относительно статичны, их обновление может происходить раз в день или раз в неделю. Неизменяемыми страницами и графикой на этом сайте можно манипулировать из файловой системы на промежуточном уровне с помощью iAS HTTP-сервера. В самой базе данных хранится вся каталожная информация. Многие запросы к такой базе данных будут динамическими. Это запросы на поиск книг по имени автора, названию, теме или какому-то другому параметру. То есть ответ на многие запросы будет извлекаться из копии базы данных. Однако с ростом числа пользователей базы данных на нее будет одновременно поступать все больше запросов. В этом случае существование лишь одной копии базы данных станет узким местом в работе всей системы, и при высокой нагрузке ответы на запросы будут приходить все позднее. Эту проблему можно решить, перенеся обработку запросов на промежуточный уровень путем кэширования каталога в Oracle9i Cache. Тогда все запросы к каталогу будут обрабатываться локально на узле промежуточного слоя, и данные для каждого запроса не будут путешествовать по сети туда и обратно, за счет чего удастся повысить производительность работы всей системы. Результатом будет снижение нагрузки на базу данных и, следовательно, увеличение ее масштабируемости.

Системные службы iAS

В состав сервера приложений Oracle Internet Application Server входят программное обеспечение системного администрирования Oracle Enterprise Manager и служба безопасности Oracle Advanced Security. Эти системные службы образуют систему управления для всей среды Oracle и сетевой безопасности по протоколу шифрования SSL (Secure Socket Layer) с использованием средств аутентификации.

Oracle Enterprise Manager

Oracle Enterprise Manager (EM)— это средство системного администрирования для управления вашей платформой Oracle. На своей графической консоли менеджер EM объединяет серверы администрирования Oracle Management Server, программы-агенты Oracle Intelligent Agents, службы общего назначения и средства администрирования. Oracle Enterprise Manager представляет собой законченную платформу системного администрирования для управления и сервером Oracle iAS, и СУБД Oracle8i.

Oracle Advanced Security

Работу средств безопасности в iAS обеспечивает пакет Oracle Advanced Security. В его функции входит и сетевая защита, и защита сетей ма'сштаба предприятия, и защищенное расширение корпоративных сетей в Интернете, и защита служб каталогов, чтобы пользователь мог управлять сетью масштаба предприятия по одному паролю входа в систему.

Клиентские компоненты сервера iAS

Комплект разработчика Oracle Database Client Developer's Kit

Комплект разработчика Oracle Database Client Developer's Kit содержит клиентские библиотеки для СУБД Oracle9i. Кроме того, этот инструментарий поддерживает Java2 Enterprise Edition (JMS, SQLJ, JDBC и JNDI). Разработчики могут включать эти библиотеки в разрабатываемые ими приложения. Приложения будут работать на сервере Oracle iAS и иметь доступ к базе данных Oracle9i.

Комплекты разработчика Oracle XML Developer's Kits

В состав сервера iAS входят и инструментальные комплекты для разработчика Oracle XML Developer's Kits (XDK), которые мы обсуждали в предыдущих главах. Все пять комплектов XDK — Java, JavaBeans, C, C++ и PL/SQL — включены в этот сервер для разворачивания и распределения приложений. Эти компоненты используются многими службами iAS и внутри, и вне сервера (в последнем случае им приходится взаимодействовать с внешними XML-интерфейсами).

Клиентскийинструментарий Oracle LDAP Client Toolkit

Инструментарий Oracle LDAP Client Toolkit используется для разработки и мониторинга работы приложений, поддерживающих технологию облегченного протокола службы каталогов (Lightweght Directory Access Protocol, LDAP). Он поддерживает клиентские обращения к службам каталогов, которые можно использовать для доступа к данным каталога. Приложения, построенные с помощью этого инструментария, могут использовать все преимущества служб Oracle Internet Directory (OID) и других версий служб 3 LDAP.

Программа онлайнового книжного магазина как OAS-сервлет

Онлайновый книжный магазин Bookstore можно представить в виде приложения сервера OAS, которое работает на промежуточном уровне. Код такого приложения приведен ниже. Вся информация из каталога книжного магазина хранится в базе данных Oracle. Логика приложения реализована в виде Java-сервлета, который находится на сервере OAS и получает информацию о книгах из базы данных в виде

XML-документа. Клиентские программы представляют собой Web-браузеры, посылающие свои запросы на информацию о книгах в OAS-сервлет и получающие ответы в виде HTML-страниц. Упомянутый сервлет для преобразования XML-документа БД в клиентскую HTML-страницу использует таблицы стилей XSLT.

Нижеследующий код иллюстрирует типичную реализацию сервлета, применяющего XSLT-таблицы стилей к XML-документам. *Сервлет* — это Java-программа, получающая входные данные с сервера и на этот же сервер отправляющая выходную информацию. Сервлет запускается выбором определенного URL-адреса сервлета из клиентского браузера. Такой подход довольно часто используется для установки логики приложения на сервер.

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import java.util.*;
import java.sql.*;
import java.net.*;
import oracle.xml.parser.v2.*;
import org.w3c.dom.*;

public class BookstoreServlet extends HttpServlet
{  ServletContext context;
   Connection conn = null;

   public void init (ServletConfig config) throws ServletException
   {  super.init(config);

      context = config.getServletContext ();

      // initialize/open database connection
      initDB ();
   }

   public void doGet (HttpServletRequest req, HttpServletResponse res)
         throws ServletException, IOException
   {  InputStream xmlIn, xslIn;
      String docHTML;

      res.setContentType ("text/html");
      PrintWriter out = new PrintWriter (res.getOutputStream());

      String xslName = getXSLName (req);

      // print client HTML header
      out.println ("<html>");
      out.println ("<head><title>Booklist</title></head>");
      out.println ("<body>");
```

```
    try
    {  xmlIn = getXMLfromDB (xmlName);
       xslIn = getXSL (xslName);
       docHTML = doXSLTransform (xmlIn, xslIn);
    }
    catch (Exception e)
    {  docHTML = *<H2>Error: * + e.getMessage() + "</H2>";  }

    out.print (docHTML);
    out.println ("</body></html>");
    out.close();
}

public void destroy () {
    // close DB connection
    closeDB ();
}

String doXSLTransform (InputStream xmlIn, InputStream xslIn)
        throws Exception
{  DOMParser parser;
   XMLDocument xml, out;
   XSLStylesheet xsl;

   try
   {  parser = new DOMParser();
      parser.setPreserveWhitespace(true);

      // parse input XSL file
      parser.parse (xslIn);
      xsl = new XSLStylesheet (parser.getDocument(), null);

      // parse input XML document
      parser.parse(xmlIn);
      xml = parser.getDocument();

      XSLProcessor processor = new XSLProcessor();

      // display any warnings that may occur
      processor.showWarnings (true);
      processor.setErrorStream (System.err);

      // Process XSL
      DocumentFragment result = processor.processXSL (xsl, xml);

      // create an output document to hold the result
      out = new XMLDocument() ;
```

```
            // create a dummy document element for the output document
            Element root = out.createElement("root");
            out.appendChild (root);

            // append the transformed tree to the dummy document element
            root.appendChild(result);

            // print the transformed document
            StringWriter strWriter = new StringWriter ();
            out.print (new PrintWriter (strWriter));
            return strWriter.toString();
        }
        catch (Exception e)
        { return new String ("<H2>Error: " + e.getMessage() + "</H2>"); }
    }

    // helper functions
    ...
}
```

Приложение BookstoreServlet

Связь с БД устанавливается во время инициализации сервлета. Доступ к БД осуществляется двумя способами, описанными в этой главе ниже. Клиентские запросы принимаются сервлетными методами **doGet()** или **doPut().** Эти методы передают два параметра: клиентский запрос и ответ на него. Сначала устанавливается тип ответа:

```
res.setContentType ("text/html");
```

То есть клиент получает информацию в виде HTML-страницы. Имя XSLT-таблицы стилей, преобразующей XML-документ базы данных в HTML-страницу, получается из запроса. Ответом на запрос будет заголовок HTML-файла, а XML-документ берется из базы данных. Таблица стилей XSLT считывается из запоминающего устройства сервлета, после чего вызывается метод **doXSLTransform()**.

```
PrintWriter out = new PrintWriter (res.getOutputStream());
...
try
{ xmlIn = getXMLfromDB (xmlName);
  xslIn = getXML (xslName);
  docHTML = doXSLTransform (xmlIn, xslIn);
}
catch (Exception e)
{ docHTML = "<H2>Error: " + e.getMessage() + "</H2>"; }
```

Метод **XSLTransform()** приписывает значение XSL-процессору и выполняет преобразование:

```
XSLProcessor processor = new XSLProcessor ();
...
DocumentFragment result = processor.processXSL (xsl, xml);
```

Если возникает ошибка, сообщение о ней отправляется обратно к клиенту.

Входной XML-документ может также содержать список данных о книгах, подобный следующему:

```
...
<booklist>
   <book isbn="1234-123456-1234">
      <title>C Programming Language</title>
      <author>Kernighan and Ritchie</author>
      <publi sher>IEEE</publisher>
      <price>7.99</price>
       ...
   </book>
    ...
   <book isbn="1230-23498-2349879">
      <title>Emperor's New Mind</title>
      <author>Roger Penrose</author>
      <publisher>Oxford Publishing Company</publisher>
      <price>15.99</price>
       ...
   </book>
    ...
</booklist>
```

После применения к этому XML-документу клиентской таблицы стилей будет сгенерирован HTML-документ, содержащий только заголовки книг и цены.

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

   <xsl:template match="/">
      <xsl:apply-templates/>
   </xsl:template>

   <xsl:template match="booklist">
      <BODY BGCOLOR="#CCFFFF">
         <H1 align="center">List of books</Hl>.
         <table border="0" width="100%" cellspacing="5">
```

```
            <tr>
               <td width="83%" bgcolor="FFFF80" align="left">
                  <font face="Arial" size="5" color="000080">
                     <strong>BOOK</strong>
                  </font>
               </td>
               <td width="17%" bgcolor="FFFF80" align="left">
                  <font face="Arial" size="5" color="000080">
                     <strong>Price</strong>
                  </font>
               </td>
            </tr>
            <xsl:apply-templates/>
         </table>
      </BODY>
   </xsl:template>

   <xsl:template match="booklist/book">
      <tr>
         <xsl:apply-templates/>
      </tr>
   </xsl:template>

   <xsl:template match="booklist/book/title">
      <td width="83%" bgcolor="FFFF80" align="left">
         <font face="Arial" size="5" color="000080">
            <strong><xsl:apply-templates/></strong>
         </font>
      </td>
   </xsl:template>

   <xsl:template match="booklist/book/price">
      <td width="17%" bgcolor="FFFF80" align="left">
         <font face="Arial" size="5" color="000080">
            $<xsl:apply-templates/>
         </font>
      </td>
   </xsl:template>

   <xsl:template match="booklist/book/author">
   </xsl:template>
   <xsl:template match="booklist/book/publisher">
   </xsl:template>

</xsl:stylesheet>
```

Регистрация приложения BookstoreServlet и картриджа

Приложение BookstoreServlet предназначено для использования картриджа OAS JServlet и всех связанных с этим преимуществ. Никакого запуска или закрытия виртуальной Java-машины при каждом запросе не происходит, поскольку связь с базой данных остается открытой на протяжении всего сеанса, так что время и ресурсы системы не тратятся на повторные подключения. На одной и той же виртуальной Java-машине можно запускать сразу несколько картриджей JServlet, если они принадлежат к одному и тому же приложению. То есть вместо создания новых копий приложения используются те его копии, которые свободны в данный момент. Картридж JServlet использует такие функции служб OAS, как балансировка нагрузки, масштабируемость, мониторинг, регистрация событий и администрирование сеанса связи с базой данных. Для генерации HTML-страниц и доступа к базе данных можно также использовать инструментарий JServlet Toolkit.

После создания сервлетного приложения следует зарегистрировать его с помощью OAS Manager. OAS Manager соединяет картридж JServlet с приложением и передает информацию о конфигурации, в том числе виртуальный и физический путь к сервлету, переменные окружения и данные для аутентификации. Затем надо перезагрузить сервер для активизации регистрации сервлета. OAS Manager перезагружает все процессы, слушающие программы и приложения сервера Oracle Application Server.

Запуск приложения BookstoreServlet

Приложение BookstoreServlet можно запустить с помощью следующего URL-адреса, который нужно набрать в строке клиентского браузера:

```
http://<host>:<port>/<servlet_virtual_path>/BookstoreServlet
```

Здесь параметры host и port идентифицируют слушающую программу, в которой есть информация о данном картридже. Это может быть любая слушающая программа на сервере приложений за исключением программы Node Manager, например программа *www listener,* по умолчанию находящаяся на порту 80.

Запустить картридж можно и другим способом — из HTML-страницы, например:

```
...
<HTML>
<HEAD><TITLE>Bookstore </TITLE></HEAD>
<BODY>
...
   <FORM METHOD = "GET"
ACTION=HREF="http://jsmith.us.oracle.com:80/oas/jservlets/HelloServlet">
   <A>Run BookstoreServlet</A>
...
```

Здесь **jsmith.us.oracle.com** — это хост-машина, 80 — номер порта, **oas/jservlets** — виртуальный путь, а входным методом сервлета будет **doGet().**

Компонент слушающей программы OAS HTTP получает с клиентского браузера запрос к картриджу JServlet. Диспетчер сервера OAS определяет, что запрос предназначается для картриджа и направляет его *брокеру запросов О AS Web* Request Broker (*WRB*). Брокер WRB исследует URL-адрес и определяет, что этот запрос предназначается для картриджа JServlet, так как картриджу назначен виртуальный путь **/oas/jservlets.** Затем запрос передается на картридж JServlet. Этот картридж, работающий в серверном процессе, получает запрос, исследует URL-адрес и обычно в конце его находит имя приложения (класса) Java-сервлета, который нужно запустить на исполнение. В данном примере приложение называется BookstoreServlet. Затем картридж JServlet загружает класс **BookstoreServlet** и вызывает его метод точки входа. Для сервлетов точки входа — это методы **doGet() и doPut().** Потом приложение Java-сервлета генерирует ответ, в который входит HTTP-заголовок и тело ответа, и возвращает его через специальный выходной поток (HtmlStream). Картридж JServlet получает этот ответ и возвращает его брокеру запросов WRB, который, в свою очередь, направляет ответ на клиентский браузер, запустивший приложение.

Доступ к базе данных

Обычно приложения JServlet могут получать доступ к базам данных двумя способами: посредством интерфейса JDBC и с помощью утилиты *pl2java.*

Первый метод доступа предполагает использование JDBC-модуля JdbcBeans. JDBC предоставляет стандартный интерфейс для доступа к базам данных разных поставщиков с использованием в качестве аргументов методов доступа встроенные SQL-операторы. Сервлетное приложение обращается к операторам PL/SQL напрямую. В примере ниже показано приложение BookstoreServlet с доступом к БД через интерфейс JDBC:

```
...
import java.sql.*;
import oracle.html.*;

public class BookstoreServlet extends HttpServlet
                              implements SingleThreadModel {
   private Connection conn = null;

   public void init( ServletConfig config ) throws ServletException {
      super.init (config) ;
      ...
      initDB ();
   }
```

```
    initDB () {
       try {
          // Load the Oracle JDBC driver
          Class.forName ("oracle.jdbc.driver.OracleDriver");

          String   username = "scott",
                   password = "tiger",
                   dbURL = "jdbc:oracle:oci8:@";
          // Connect to the database. To connect to a remote database,
          // insert the connect string after the @ sign in the connection URL.
          conn = DriverManager.getConnection (dbURL, username, password);
       } catch ( SQLException & ) {
          System.err.println ("Could not establish connection.");
       } catch ( ClassNotFoundException e ) {
          System.err.println ("Could not load database driver.");
       }
    }
    ...

    InputStream getXMLfromDB( ) throws Exception
    {  try {
          // Create a Statement
          Statement stmt = conn.createStatement ();

          // Select the ENAME column from the EMP table
          ResultSet rset = stmt.executeQuery (
                                   "select num, xmldoc from bookstore");

          // Iterate through the result
          while ( rset.next () )
          {  buffer.add (rset.getString(2));
       }
    } catch ( SQLException e ) {
       buffer = *A database error occurred";
    }
    return new StringBufferInputStream (buffer);
}

void closeDB () {
    try {
       // close the database connection
       if ( conn != null)
          conn.close();
    }
```

```
    catch (SQLExceptione) {
       System.err.println ("Error closing database connection.");
    }
  }
  ...
}
```

Второй механизм доступа к БД — это вызов хранимых процедур и функций PL/SQL непосредственно из Java-приложений, работающих в контексте картриджа JServlet. Генерация интерфейсных Java-классов для процедур в PL/SQL-модуле происходит с помощью утилиты *pl2java*. Для каждого конкретного PL/SQL-модуля утилита *pl2java* генерирует один интерфейсный Java-класс, который содержит интерфейсный метод для каждой процедуры или функции в этом модуле. Сигнатуры интерфейсных методов для PL/SQL-процедур и функций будут одинаковыми. Это позволяет прозрачно для пользователя вызывать хранимые процедуры PL/SQL.

Вот, например, как выглядит вызов *pl2java* утилиты из командной строки:

```
> pl2java -class bookstore bookstore.sql scott/tiger ...
```

При этом из PL/SQL-модуля **bookstore.sql** создается интерфейсный Java-класс с именем "bookstore". Кроме того, нужно задать пару имя пользователя/пароль. Эти интерфейсный классы принадлежат модулю **oracle.plsql,** они являются производными от базового класса **PValue,** который инкапсулирует неопределенный атрибут PL/SQL-значений. Для передачи данных между Java и PL/SQL утилита *pl2java* преобразует типы данных PL/SQL в интерфейсные Java-классы.

Следующий пример кода демонстрирует приложение BookstoreServlet, которое для доступа к базе данных использует интерфейсный класс.

```
...
import oracle.rdbms.*;

public class BookstoreServlet extends HttpServlet
                              implements SingleThreadModel
{  private Connection conn = null;

   public void init ( ServletConfig config ) throws ServletException
   {  super.init(config);
      ...
      initDB();
   }

   initDB () {
      ...
      // Define ORACLE_HOME
      Session.setProperty("ORACLE_HOME", System.getProperty("oracleHome"));
```

```
        // Create a new database session and logon
        Session session = new Session ();
        session.logon ("scott", "tiger", "bookstore");
    }

    public void doPost (...
    {
        ...
    }

    InputStream getXMLfromDB () throws Exception
    {  ...

        // Instantiate Bookstore wrapper class:
        Bookstore store = new Bookstore (session);

        // Instantiate a PStringBuffer to get a string from the PL/SQL procedure
        PStringBuffer pBuffer = new PStringBuffer (30);

        // Invoke the PL/SQL procedure
        PBuffer = store.getXML ();

        // Retrieve the return value
        if(!pBuffer.isNull())
           buffer = pBuffer.intValue ();

        return new StringBufferInputStream (buffer);
    }

    ...
}
```

Связь с базой данных инкапсулируется классом **Session** в модуле oracle.rdbms. *Класс Session* предоставляет методы для выполнения обычных для базы данных задач: вход *в* базу, выход из нее, фиксация транзакции, откат к предыдущей точке и т. д. Перед установкой соединения с БД Oracle нужно определить свойства среды. В частности, с помощью метода **System.getProperty()** параметру ORACLE_HOME сервера приложений назначается в качестве системного свойства картриджа значение "oracleHome". Затем создается объект Session и выполняется вход в БД.

Для вызова хранимых процедур PL/SQL нужно провести преобразование объекта интерфейсного класса. При этом в качестве аргумента используется объект Session. Если же сеансов связи с базой данных несколько, для каждого из них нужно создать свой объект интерфейсного класса. Значение параметра PL/SQL-процедуры устанавливается с помощью метода **setValue()** интерфейсного объекта. Для извлечения возвращаемых значений используются методы **getValue()** интерфейсных классов. Например, метод **intValue()** класса **PInteger** используется для возвращения целого

значения. Соответствующий PL/SQL-таблице Java-объект (и входной, и выходной параметры) представляет собой Java-массив вместе с элементами этого массива. Это проиллюстрировано следующим фрагментом Java-кода:

```
// Create a PL/SQL table parameter
PStringBuffer pBooks[] = new PStringBuffer[80];
for(int = 0; i < pBooks.length; i++) {
   pBooks[i] = new PStringBuffer (200);
}

// Invoke a PL/SQL procedure
bookstore.get_books (pBooks);
```

Все интерфейсные классы, инкапсулирующие PL/SQL-значения, имеют метод **toString(),** следовательно, их можно связать с помощью Java Strings. Например, в сцеплении строк можно использовать объект pBook:

```
for (int i = 0; i < pBook.length; i++) {
   buffer.add(pBook[i]);
}
```

Использование службы транзакций

С помощью картриджа JServlet можно также использовать службу транзакций сервера OAS. Эта служба для координации распределенных транзакций работает с разными API-интерфейсами доступа к базе данных. В данном случае API-интерфейсами доступа к базе данных являются JDBC-классы и классы, сгенерированные с помощью утилиты pl2java. Служба транзакций позволяет выполнять транзакции, которые соединяют запросы между менеджерами ресурсов и картриджами. Работает эта служба на базе технологии *Java Transaction Service* (*JTS*).

Типичный сценарий работы службы транзакций сервера OAS начинается с инициализации, а для этого делается обращение к методу **initTS()**. Связь с менеджером ресурсов устанавливается с помощью метода **TransactionService.connectRM()**. Затем с помощью объекта Session в классах pl2java регистрируется транзакционный DAD. Если доступ к базе данных осуществляется с помощью интерфейса JDBC, имя DAD отправляется в интерфейсный объект в виде параметра **connectRM()**. Объект Current получается через оператор **TransactionService.getCurrent()**. Этот объект предоставляет для разграничения транзакций методы **begin(), commit() и rollbackQ.** Транзакция начинается с метода **begin()**. Соединение с БД устанавливается с помощью метода **DriverManager.getConnection()**. Затем выполняются операции в базе данных. Фиксация и откат транзакции выполняются с помощью методов **commit() и rollback()** соответственно. JServlet отключается от менеджера ресурсов с помощью метода **disconnectRM()**. Наконец, завершение работы и очистка службы транзакций производится вызовом метода **termTS()**.

Активизация компонентов сервера OAS

При запуске сервлетного приложения в среде картриджа JServlet возможен вызов других приложений, расположенных на том же сервере OAS. Например, приложение Bookstore может соединяться с БД через объект ECO/Java. Объекты ECO/Java — это объекты CORBA, работающие в процессах OAS. Точно так же с помощью приложений JServlet можно вызвать приложения *Enterprise Java Beans* (*EJB*)и C++ CORBA, если они находятся на том же сервере OAS.

Приложение Bookstore как iAS-сервлет

В сервере iAS регистрация и конфигурация сервлетов производится совсем другим способом, поскольку в нем в HTTP-службах используется платформа Apache. Как указывалось выше, в состав сервера iAS входит сервлетный процессор JServ. Ниже будет показано, как сконфигурировать Apache и JServ для запуска демонстрационной программы Bookstore.

Конфигурация Apache

Сервер Apache поставляется в уже настроенном и сконфигурированном виде, поэтому на нем можно использовать сервлетный процессор JServ без всякой переконфигурации. Для запуска приложения Bookstore нужно создать виртуальный каталог. Это можно сделать, добавив в файл под именем *http.conf*конфигурации Apache следующую строку (предполагается, что приложение устанавливается в каталог **C:/Bookstore**):

```
Alias /Bookstore/ "C:/Bookstore/"
```

Конфигурация JServ

Теперь нужно сконфигурировать процессор JServ, чтобы он мог находить и запускать классы Bookstore. Это делается путем регистрации их в CLASSPATH, используемом процессором JServ. Файл под именем **jserv.properties** конфигурации JServ нужно дополнить следующими строками:

```
wrapper.classpath=C:\Bookstore\lib\xmlparserv2.jar
tt Bookstore Servlet wrapper class
wrapper.classpath=C:\Bookstore\lib\Bookstore.class
```

После повторного запуска сервера Apache доступ к приложению можно получить через стартовую страницу со следующим URL-адресом:

```
http://localhost/Bookstore/index.html
```

Глава 5

Файловая система Oracle Internet File System (iFS)

Oracle Internet File System (iFS) — это файловая система и среда разработки, которая работает внутри базы данных Oracle8i/9i или на промежуточном уровне в сервере Oracle9iAS. Oracle iFS предоставляет средства для создания, хранения и управления разными типами информации в одном репозитории. Для конечного пользователя iFS выглядит как обычный файл-сервер, организующий и представляющий данные в виде иерархической структуры папок и документов. Работа *i*FS проходит совершенно прозрачно для пользователей. Они могут так никогда и не узнать, что их документы хранятся в базе данных, поскольку извне создается впечатление, что iFS — это типичный файл, Web- или почтовый сервер.

Документы в iFS вовсе не обязательно должны быть текстовыми. Это могут быть цифровые данные любой формы: файлы форматов электронных таблиц и текстовых редакторов, графические и звуковые файлы, презентации, файлы структурированных форматов HTML и XML, реляционные данные и т. д. В репозитории iFS можно хранить все типы данных и документов.

Сила системы iFS в ее согласованности: все данные можно хранить в едином репозитории, доступ к которому можно получить с любого компьютера корпоративной сети, а кроме того, администрирование и управление контентом в этом репозитории производится довольно простыми методами.

Характеристики

Отметим, что представленное здесь описание характеристик системы iFS, ее функциональных возможностей и платформ, на которых ее можно разворачивать, относится к версии 1.1.9 (если не указано другое), т. е. к текущей версии на момент написания этой книги. Система iFS работает внутри сервера Oracle8i/9i Server в виде Java-приложения, запускающего виртуальную Java-машину. Она тесно интегрирована с другими функциями Oracle8i/9i, такими как объектные типы, interMedia Text/Oracle Text, и средством администрирования Oracle Enterprise Manager. Она также работает в виртуальной Java-машине, поставляемой в составе HTTP-сервера Oracle9iAS, и взаимодействует с БД посредством интерфейса JDBC. Поскольку система iFS функционирует внутри серверов Oracle8i/9i и Oracle9iAS, она одновременно с сервером автоматически увеличивает масштабируемость, производительность и уровень безопасности. Общая схема архитектуры системы iFS и взаимоотношения ее компонентов показана на рис. 5.1.

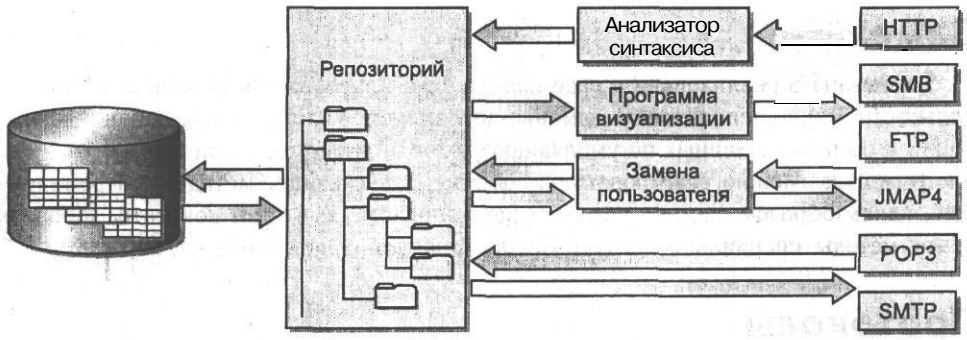


Рис. 5.1. *Схема архитектуры системы iFS*

Хранилище таблиц

Система *i*FS существует внутри СУБД Oracle8i/9i или может с ней взаимодействовать, поэтому основным механизмом хранения данных для нее являются таблицы. Все *i*FS-данные хранятся в таблицах, содержащих и документы, и метаданные с информацией об этих документах. Документы с большим объемом неструктурированных данных (текст, графика, видео и т. д.) хранятся в объектах *Oracle8i/9i Large Objects (LOB)*. Размер одного такого объекта может достигать 4 Гбайт. Схема базы данных iFS является частью репозитория *i*FS. В ней хранятся все постоянные объекты, и *i*FS управляет ими как совокупностью таблиц и полей. Они являются неким набором SQL-представлений файловой системы, что изолирует пользователя от непосредственного взаимодействия с базой данных.

Анализаторы синтаксиса

Когда документ помещается в систему *i*FS, анализатор синтаксиса автоматически производит разбор документа, отделяя содержимое от метаданных и сохраняя эти части в разных столбцах записи базы данных. В *i*FS есть несколько встроенных анализаторов синтаксиса (в том числе и XML-анализатор), но пользователь может создать и свои собственные анализаторы.

Программы визуализации

Когда нужно просмотреть хранимый документ, z'FS строит (визуализирует) его из базы данных. *Программы визуализации* представляют собой механизмы, которые определяют, как именно должны быть представлены и анализированные, и не анализированные документы. Причем неважно, каков был исходный формат этого документа, когда он попал в *i*FS,— программы визуализации преобразуют документ в формат, который наилучшим образом подходит для средства или протокола, используемого для доступа к документу.

Замены

Система *i*FS реализована *в* виде набора Java-классов. Для ее большей универсальности можно использовать замены, позволяющие изменять или добавлять операции к обработке данных по умолчанию, которая происходит при вызове того или иного метода. Можно реагировать на событие, запускающее метод и контролирующее, каким образом система обрабатывает определенные типы контента, добавляя в эти методы специально созданный для каждого конкретного случая код.

Протоколы

Когда система *i*FS работает внутри СУБД Oracle8i или на промежуточном уровне, мы воспринимаем ее как обычную файловую систему. Доступ к информации, хранящейся в *i*FS, можно получить с помощью нескольких протоколов:

- HTTP и HTTPS (с помощью любого стандартного Web-браузера)
- SMB (для доступа с помощью клиентов Microsoft Windows 95, Windows NT, 98 или 2000)
- FTP (File Transfer Protocol) — протокол передачи файлов
- SMTP, IMAP4, POP3 (протоколы электронной почты, с помощью которых пользователи могут получать доступ к файлам через стандартные почтовые клиентские программы, например Eudora, Netscape Mail и Microsoft Outlook)
- Custom Protocol Servers — настраиваемые серверы протоколов, созданные самим пользователем или сторонними разработчиками.

Преимущества

Файловая система Internet File System (*i*FS) имеет ряд преимуществ по сравнению с традиционными методами хранения и доступа к корпоративной информации:

- Интегрированное хранилище Упрощает жизнь конечных пользователей, системных администраторов и разработчиков, так как все типы данных хранятся в одном репозитории.
- Знакомый интерфейс *i*FS представляет хранящиеся в репозитории данные в хорошо знакомой иерархической форме "папка/файл"
- Универсальный доступ Доступ к данным, хранящимся в системе *i*FS, можно осуществлять по нескольким протоколам из Web-браузеров, чат-клиентов, FTP-клиентов и т. д.
- Управление контентом В состав *i*FS входят встроенные средства управления версиями, поэтому все хранимые документы доступны в нескольких версиях, соответствующих каждому своему обновлению.

Интегрированная программа блокировки, работающая по принципу регистрации и выписки, не позволит произвести перезапись данных поверх старой информации.

- Функциональные возможности Так как система *iFS* тесно интегрирована с СУБД Oracle8i/9i и сервером 9iAS, ей автоматически становятся доступными все возможности базы данных, в том числе автоматическое индексирование, быстрый и контекстный поиск, резервное копирование и восстановление данных.
- Простота разработки В систему iFS встроены многие полезные функции приложений, централизованные по данным (защита данных, их распределенное хранение, возможности поиска информации и т. д.), поэтому все они сразу же становятся доступными разработчику без дополнительного кодирования. Благодаря единому API-интерфейсу разработки и единой организации хранения данных приложения для *i*FS можно разрабатывать довольно быстро и без особых проблем.

Компоненты

Доступ к данным осуществляется через несколько протоколов, а технологии Java и PL/SQL через Java-модули предоставляют API-интерфейсы программирования для разработки приложений.

Администрирование осуществляется через *Oracle Enterprise Manager (OEM)* (это же средство использовалось для администрирования сервера Oracle8i).

В состав iFS включены следующие программные компоненты:

- Сервер *i*FS
- Java API-интерфейс *i*FS
- *i*FS Manager для администрирования (для платформ Solaris 2.6 и NT 4.0)
- iFS Web Manager
  (для администрирования на любых платформах через браузер)
- СУБД Oracle 8.1.6 или более поздней версии
  (обязательный компонент, но вместе с iFS он не поставляется)
- Windows-клиент
- Web-клиент

XML

Система *i*FS обеспечивает встроенную поддержку XML-файлов. В iFS также включены анализатор XML-синтаксиса и программа визуализации XML-файлов, а для определения новых типов файлов, созданных на базе XML, в iFS есть специальный механизм расширения.

После регистрации нового типа файлов на базе XML выдается дескриптор документа, определяющий XML-структуру типа и вид хранения в БД. Для описания структуры (схемы) документов этого типа дескрипторы используют синтаксис, созданный на базе XML. Когда документ записывается для последующего хранения в *i*FS, он распознается как один из определенных пользователем типов файлов, после чего проводится синтаксический анализ документа, и данные записываются в таблицы в соответствии с указаниями дескриптора. Этот же дескриптор потом используется для визуализации (компоновки) XML-документа на выходе.

С помощью дескриптора можно выбрать, каким способом будет храниться XML-документ — как серия таблиц и столбцов реляционной базы данных, как единый текстовый объект CLOB или как некая комбинация этих двух вариантов. Например, на рис. 5.2 показан документ с заявлением о выплате страхового возмещения, в котором есть и структурированные данные (получатель, дата и т. д.), и структурированная текстовая разметка (DamageReport). В этом примере структурированные данные будут храниться в таблицах, а структурированная текстовая разметка — в виде больших двоичных объектов.

```
<?xml version="1.0"?>
      <InsuranceClaim>
         <ClaimID>12345</ClaimID>
         <LossCategory>7</LossCategory>
         <Settlements>
            <Payment>
               <Payee>Borden Real Estate</Payee>
               <Date>12-OCT-1998</Date>
               <Amount>200000</Amount>
               <Approver>JCOX</Approver>
            </Payment>
         </Settlements>
         <Damage Report>
            A massive <Cause>Fire</Cause> ravaged the building and
<Casualities>12</Casualities> people were killed. Early FBI reports
indicate that <Motive>arson</Motive> is suspected.
         </Damage Report>
         :
         </InsuranceClaim>
```

Рис. 5.2 *XML-документ с заявлением о выплате страхового возмещения*

После того как копии типов файлов *i*FS (в том числе и файлы, созданные на базе XML) будут сохранены в базе данных, можно производить поиск их содержимого с помощью стандартных SQL-запросов. Эти файлы можно организовывать, просматривать и создавать их новые версии, используя стандартные средства, например Windows Explorer.

Части этого документа, хранящиеся в базе данных как CLOB-объекты, можно по желанию проиндексировать с помощью поискового средства Oracle8i interMedia Text или Oracle9i Text.

interMedia Text/Oracle Text

Поисковое средство Oracle8i *interMedia Text* (раньше оно называлось *Context)* и последовавшее за ним Oracle9i Text были усовершенствованы, чтобы разработчики могли точно указывать тот раздел документа, в котором следует вести поиск данных. Для документов, созданных на базе технологии XML, раздел неявно определяется XML-тегами. Так как средство interMedia Text интегрировано в БД и в язык SQL, последний используется для выполнения запросов на поиск информации и в структурированных данных, и в индексированном тексте.

Например, на рис. 5.3 показан запрос на документы с заявлениями о выплате страхового возмещения (аналогичное заявление приведено на рис. 5.2): "Сколько денег Джим Кокс распорядился выплатить к настоящему моменту в качестве возмещения за ущерб от пожаров, связанных с поджогами?"

```
SELECT   SUM(Amount)
         FROM Claim_Header ch,
              Claim_Settlements cs,
              Claim_Settlement_Payments csp
  WHERE csp.Approver='JCOX'
    AND CONTAINS (DamageReport, 'Arson WITHIN Motive') > 0
    AND CONTAINS (DamageReport, 'Fire WITHIN Cause' ) > 0
    AND ... /* Join Clauses */
```

Рис. 5.3. *Пример запроса на документ о страховом возмещении*

Этот пример демонстрирует, как вести поиск информации и производить манипуляции, пользуясь уже известными для баз данных методами, в структурированных документах XML-формата, введенных в базу данных Oracle8i/9i.

Модель документа

Система *i*FS была создана на базе документов и специально для работы с документами — все в *i*FS является документом.

Документы состоят из элементов структурированных данных, называемых *свойствами* (properties), внутри неструктурированного тела документа. Все традиционные файловые системы хранят дополнительную информацию о файлах: имя файла, дата последнего изменения и размер файла. Система *i*FS расширяет эту модель, чтобы обеспечить управление содержимым документа (контроль версий и блокировка), защиту информации и гибкую структуру каталогов.

Тело документа — это его содержимое (текст, графика, аудиоданные или видео). Тело документа является свойством, но его нельзя установить через API-интерфейс и хранить как набор каких-то частей, это — единый неделимый блок данных.

В *i*FS поддерживается несколько типов документов:

- Составные документы содержат ссылки на файлы и обеспечивают работу виртуальных каталогов.
- Файлы конфигурации — это XML-документы, в которых хранятся метаданные о типах документов *i*FS.
- Документы типа (**.typ**) расширяют базовое определение документа, чтобы определить заданные пользователем свойства для форматов файлов.
- Документы расширений файлов (.fex) определяют расширения файлов, которые, в свою очередь, определяют документы типа.
- Документы анализатора синтаксиса (**.parse**) и программы визуализации (**.renderer**) определяют Java-классы, используемые для хранения и вывода на экран данных конкретного типа.
- Документы списка контроля доступа *Access Control List (ACL)* определяют привилегии чтения/записи для пользователей и их групп.

Свойства документов

В системе *i*FS используются четыре типа свойств, которые характеризуют: номер версии, стандартность, связи и пользователя. Все эти типы описываются ниже.

Таблица 5.1

Список свойств, описывающих версии документа

Название свойства	Тип данных	Допускается Null?	Системное?	Описание
Creation_Date	Дата	Нет	Да	Дата создания документа.
Max_Untagged	Целое	Нет	Нет	Количество непомеченных редакций документа для сохранения (определяет порядок удаления версий документа).
Project_ID	DocID	Да	Да	Идентификационный номер документации проекта, которому принадлежит данный файл. Файлы могут принадлежать только одному проекту. Если документ не является частью какого-то проекта, это значение равно null.
Total_Revisions	Целое	Нет	Да	Общее количество редакций документа, существующих на данный момент.
Volume_ID	Целое	Нет	Да	Номер идентификатора тома документа.

Свойства, описывающие версии документа

Свойства, описывающие версии документа, всегда одинаковы, независимо от того, какая версия документа используется. Например, дата создания документа не изменяется после того, как был создан изначальный документ. Ниже в таблице 5.1 представлен список свойств, описывающих версии документа.

Свойства, связанные со стандартностью документа

Свойства, связанные со стандартностью документа, всегда одни и те же для каждой конкретной версии документа, независимо от того, как и кем они были получены. Но они могут изменяться в последующих версиях этого документа. Описание этих свойств представлено в таблице 5.2.

Таблица 5.2

Список свойств, связанных со стандартностью документа

Название свойства	Тип данных	Допускает Null?	Индекс	Описание
1	2	3	4	5
Abstract	Строковый(512)	Да	Нет	Краткое описание документа.
ACL	DocID	Нет	Нет	DocID для списка контроля доступа этого документа.
Author	Строковый(60)	Да	Нет	Первый автор (или отправитель) документа.
Abstract	Строковый(512)	Да	Нет	Краткое описание документа.
ACL	DocID	Нет	Нет	DocID для списка контроля доступа этого документа.
Author	Строковый(60)	Да	Нет	Первый автор (или отправитель) документа.
Body_ID	Численный(38)	Нет	Да	Первичный ключ, который идентифицирует этот документ в таблице тела документа.
Branch_Tags	Массив	Да	Да	Список массивов всех сегментов дерева каталогов, которым назначена эта редакция документа.
Checked_Out_To	DocID ,	Да	Да	DocID пользователя, который проверил этот документ.
Content_Type	Строковый(24)	Нет	Нет	Название типа содержимого документа.

Таблица 5.2 (*продолжение*)

Список свойств, связанных со стандартностью документа

1	2	3	4	5
Docid	Численный(38)	Нет	Нет	Уникальный идентификатор ID этой версии документа.
Generator	Строковый(128)	Да	Нет	Название приложения, которое сгенерировало тело документа.
ID	Численный(38)	Нет	Да	Уникальный идентификатор ID документа.
Invalid	Логический	Нет	Да	Присваивается значение истина (true), если документ неправильный (т. е. в нем есть синтаксическая ошибка или нарушены заданные ограничения).
Is_Current	Логический	Нет	Да	Присваивается значение истина (true), если документ является самой последней версией.
Mime_Type	Строковый(48)	Нет	Нет	MIME (Multipurpose Internet Mail Extensions) тип, описывающий формат документа.
Mod_Date	Дата	Нет	Нет	Дата последней модификации данного документа.
Modified_By	DocID	Нет	Да	Идентификатор DocID пользователя, который создал эту версию документа.
Num_Links	Целое	Нет	Да	Общее число связей документа.
Owner	DocID	Нет	Да	Идентификатор DocID пользователя (владельца документа).
Read_Only	Логический	Нет	Нет	Присваивается значение истина (true), если документ не подлежит изменениям.
Release_Tags	Массив	Да	Да	Список тегов, связанных с данной версией документа.
Saved_Links	Целое	Нет	Да	Количество сохраненных ссылок на документ. Документ с такими ссылками нельзя удалить, пока не будут удалены эти ссылки.
Size	Целое	Нет	Да	Длина тела документа в байтах.
System	Логический	Нет	Нет	Присваивается значение истина (true), если документ является файлом, предназначенным для внутреннего использования в системе.

Таблица 5.2 (*продолжение*)

Список свойств, связанных со стандартностью документа

1	2	3	4	5
Title	Строковый(80)	Нет	Нет	Название документа.
Version	Строковый(20)	Нет	Да	Назначенный системой идентификатор версии документа.

Свойства, описывающие связи документа

Свойства, описывающие связи документа, используются составными документами и хранятся в том контейнере, где появился данный документ. Все эти свойства описаны в таблице 5.3.

Таблица 5.3

Список свойств, описывающих связи документа

Название свойства	Тип данных	Допускается Null?	Системное?	Описание
Archive	Логический	Нет	Нет	Присваивается значение истина (true), если версия документа была архивирована программой резервного копирования.
Expires	Дата	Да	Нет	Дата, когда эта связь должна быть автоматически удалена.
Filename	Строковый(128)	Нет	Нет	Название файла для данного документа в конкретном контейнере (т. е. имя ссылки на этот файл).
Hidden	Логический	Нет	Нет	Присваивается значение истина (true), если документ нельзя обычным образом выводить на экран в списках каталогов.
Link_Serial_ Number	Целое	Нет	Да	Серийный номер, назначаемый ссылкам в порядке их поступления в контейнер.
Flags	Необработанные данные RAW(1)	Нет	Нет	Флаги, используемые внутри iFS.
Saved	Логический	Нет	Нет	Присваивается значение истина (true), если данная ссылка сохранена. Документ с сохраненной ссылкой не может быть удален, пока не будут удалены все ссылки на него.

Пользовательские свойства

Пользовательские свойства сохраняются для каждого индивидуального пользователя документа, и для каждого такого пользователя они могут быть различными. Например, флагу read будет присваиваться значение истина (true) для всех пользователей, прочитавших документ, и ложь (false) для всех, кто его не читал.

Обработка документа

Когда документ помещается в *i*FS, он разбивается на куски, соответствующие его компонентам, и в таком виде хранится в нескольких таблицах базы данных.

После вставки документа в *i*FS расширение файла документа сравнивается с расширениями файлов, которые определены на данный момент в *i*FS. Если не найдено ни одно подходящее расширение, то для извлечения основной информации о документе используется стандартный анализатор синтаксиса. Эта информация извлекается из протокола клиента (например, имя файла берется из FTP-программы) или путем экстраполяции (например, размер документа можно вычислить при его чтении). Свойства документа записываются в таблицу документов данного рода.

Если же расширение файла данного документа уже определено в *i*FS, вызывается анализатор синтаксиса данного типа файлов, который и извлекает из документа дополнительные свойства. Затем эти свойства сохраняются в таблице для данного типа документов.

Весь документ вместе со своими свойствами загружается в один столбец таблицы, используемой для хранения только тел документов. Свойство **BodyID** в таблице типа документа дает уникальный ключ для записи тела документа.

При вставке документа можно также провести операцию его индексирования для последующего полнотекстового поиска с помощью средства interMedia Text.

Хранение содержимого документов и их метаданных в таблицах — очень эффективная технология их обработки, но при таком способе хранения есть опасность порчи документов пользователями. Пользователи и разработчики могут повредить структуры данных при изменении содержимого таблицы, и система *i*FS даже не узнает об этом. Поэтому для каждого типа документов поставляются программы Object Views. Такие программы выводят документы на экран в виде единой виртуальной таблицы, что гарантирует безопасное управление данными. Данные в системе *i*FS должны изменяться *только с* помощью ее стандартных API-интерфейсов или с помощью программ Object Views.

Определение типов документов

Документы типов (расширение **.typ**) — это XML-файлы, определяющие свойства, которые устанавливают соответствие между конкретными форматами файлов или типами и схемами баз данных. Это определение используется для создания таблиц

базы данных, в которых хранятся части документа. Кроме того, оно идентифицирует Java-классы, используемые для создания, манипуляции и сохранения копий данного типа документа.

Документ типа выполняет три функции:

- Определяет имя типа
- Идентифицирует Java-классы, используемые для взаимодействия копии этого типа документа с репозиторием *i*FS
- Определяет свойства типа документа

Пример определения типа

Ниже представлен пример простого определения типа:

```
<?xml version ="1.0" standalone="yes"?>
<type>
   <title>Simple Type Definition</title>
   <typename>simpletype</typename>
   <dbi_classname>oracle.ifs.demo.simple.SimpleDocumentDBI
   </dbi_classname>
   <properties>
      <propdef>
         <propname>SimpleProperty</propname>
      </propdef>
   </properties>
</type>
```

Что означает каждая строка в этом примере, описывается в таблице 5.4.

Таблица 5.4

Описание файла определения типа

Строка	Описание
<?xml version ="1.0" standalone="yes"?>	Стандартное объявление для XML-файлов
<type>	Корневой элемент, который начинает описание типа
<title>Simple Type Definition</title>	Дает содержательное имя типа
<typename>simpletype</typename>	Идентификатор уникального имени типа
<dbi_classname>oracle.ifs.demo.simple.SimpleDocumentDBI</dbi_classname>	Дает название Java-классу, который используется для взаимодействия с динамической копией документа данного типа в репозитории *i*FS
<properties>	Начинает определение свойств типа

Таблица 5.4 (*продолжение*)

Описание файла определения типа

Строка	Описание
<propdef>	Начинает определение свойства
<propname>SimpleProperty</propname>	Определяет свойство с именем **SimpleProperty**
</propdef>	Завершает определение свойства
</properties>	Завершает блок определения свойства
</type>	Завершает определение типа

Стандартные свойства типа

Документы типов сами по себе являются определенным типом *i*FS документа со своим собственным набором уникальных свойств. Информация о родовом типе хранится в таблице **content_type**.

Таблица 5.5

Список всех стандартных типов свойств и их значений по умолчанию

Свойство	Java-тип	Тип БД	Столбец БД	Значение по умолчанию
typename	Строковый	varchar2(64)	typename	Значения по умолчанию нет — это поле обязательно для заполнения
inherits_from	Строковый	varchar2(64)	inherits_from	Document
final_type	Логический	char(1) [T или F]	final_type	False
tablename	Строковый	varchar2(256)	shd_table	IFS_*typename*
use existing_ table	Строковый	char(1) [T или F]	use existing_ table	False
chunk_size	Целый	integer	chunk_size	*inherits_ from*.chunk_size
user_visible	Логический	char(1) [T или F]	user_visible	True
classname	Строковый	varchar2(1024)	classname	*inherits_ from*.classname
dbi_classname	Строковый	varchar2(1024)	dbi_classname	*inherits_from*.dbi_ classname

В таблице 5.5 перечислены стандартные типы свойств документа и их значения по умолчанию. Обязательным является только свойство **typename.** В большинстве случаев достаточно значений по умолчанию. Ниже приведены подробные описания каждого свойства:

- **typename** Уникальный идентификатор типа документа. Его максимальная длина составляет 64 символа, но лучше уложиться в 15 символов. Этот тип наследует свойства своего родительского типа (а также родительского по отношению к его родительскому типу и т. д.). Это предполагает, что название **typename** должно быть уникальным для всего родословного дерева.
- **inherits_from** Название типа, расширением которого является текущий тип документа. При расширении существующего типа его свойства становятся доступными для нового типа, но для последнего определяются еще и дополнительные свойства.
- **final_type** Тип, определяемый как конечный (T), не может быть расширен другими определениями типа. Таблица типа, определенного как конечный, не имеет столбца **derived_rowid.**
- **tablename** Название имеющейся таблицы (используется в сочетании со *свойством* **use_existing_table**). Если **use_existing_table** имеет значение истина (true), то в свойстве **tablename** должно быть названо имя таблицы, уже определенной в репозитррии *i*FS.
- **use_existing_table** Если это свойство имеет значение истина (true), *iFS* не создает таблицу для данного типа документа. Вместо этого предполагается, что такая таблица уже определена и идентифицирована свойством **tablename.**
- **chunk_size** Оптимальное число байтов для чтения или записи в тело документа данного типа. Значение этого свойства следует устанавливать, только если интерфейсный класс базы данных для документа такого типа (идентифицируемый свойством **dbi_classname)** аннулирует механизм сохранения тела документа, действующий по умолчанию.
- **user_visible** Флаг, позволяющий разработчикам скрывать от пользователей некоторые документы на период выполнения операции. По умолчанию это свойство имеет значение истина (true), и в таком случае документы нового типа будут видимыми для пользователей. Если установить в качестве значения этого свойства ложь (false), документы данного типа будут скрыты от пользователей.
- **classname** Java-класс, который используется для Создания в памяти копии документа данного типа на период выполнения операции. Этот класс для данного документа должен явным образом расширять класс **oracle.ifs.sdk.Document** или неявно делать это путем расширения одного из своих подклассов.

- **dbi_classname** Это свойство идентифицирует Java-класс, являющийся интерфейсом между базой данных и копией документа данного типа, которая находится в памяти. Для типов, которые являются расширением типа Document, этот класс должен иметь расширение **oracle.ifs.sdk.override.DocumentDBI**. Для типов, являющихся расширением типа Container, данный класс должен иметь расширение **oracle.ifs.sdk.override.ContainerDBI**.

Свойства настраиваемого типа

Определение типа также может определять новые свойства, помимо тех, что были перечислены в списке стандартных свойств. Определение типа имеет свойство, названное "*properties*";оно является массивом записей с определениями свойств. Каждая запись в таком массиве описывает одно свойство, и таким способом можно описать не более 1000 свойств. Значения атрибутов этих свойств хранятся в таблице **propdef.**

Ниже представлен пример возможного объявления свойств определением типа, а затем в таблице 5.6 приведены построчные объяснения к этому примеру.

```
<properties>
  <propdef>
    <propname>sampleDate</propname>
    <colname>sampleDateColumn</colname>
    <datatype> date </datatype>
  </propdef>
  <propdef>
    <propname>sampleProperty </propname>
  </propdef>
</properties>
```

Таблица 5.6

Список объявлений свойств и описаний тегов

Код	Описание
<properties>	Начинает блок описания свойства
<propdef>	Начинает описание нового свойства
<propname>sampleDate</propname>	Определяет свойство с именем sampleDate
<colname>sampleDateColumn</colname>	Создает новый столбец sampleDateColumn в таблице propdef
<datatype>date</datatype>	Описывает тип данных, которые будут храниться в данном столбце таблицы
</propdef>	Завершает определение первого свойства
<propdef>	Начинает описание второго свойства

Таблица 5,6 (*продолжение*)

Список объявлений свойств и описаний тегов

Код	Описание
<propname>sampleProperty</propname>	Определяет свойство с названием sampleProperty. По умолчанию название столбца будет совпадать с названием свойства, а тип данных по умолчанию • будет строковым.
</propdef>	Завершает определение второго свойства
</properties>	Завершает блок объявления свойств

Атрибуты свойств

Атрибуты, используемые для настраиваемых пользователем свойств, перечислены в таблице 5.7. Единственным обязательным атрибутом здесь является **propname.** Все остальные могут быть производными от их значений по умолчанию.

Таблица 5.7

Список свойств атрибутов и их значений по умолчанию

Свойства	Java-тип	Тип БД	Столбец БД	Значение по умолчанию
propname	Строковый	varchar2(40)	Propname	Обязательный параметр, значения по умолчанию нет
datatype	Строковый	целый	Type	"String"
max_length	Целый	целый	max_length	120
colname	Строковый	varchar2(256)	Colname	propname
user_visible	Логический	char(1) [T или F]	user_visible	True
tablename	Строковый	varchar2(256)	std_table	propname с добавлением целого числа, чтобы сделать это значение уникальным
can_be_null	Логический	char(1) [T или F]	can_be_null	True
synthetic	Логический	char(1) [T или F]	Synthetic	False
use existing_ table	Логический	char(1) [T или F]	use existing_ table	False
properties	Массив	целый	Properties	Не определено
Refers_to	Строковый	varchar2(256)	Refers_to	None

- **propname** Имя, которое *i*FS присваивает данному свойству. Все ссылки на это свойство в *i*FS должны использовать именно такое имя. Это имя должно быть уникальным в контексте текущего описания свойств.
- **datatype** Определяет тип информации, которая хранится в данном свойстве. Константы для этих значений определяются в классе **oracle.ifs.sdk.Type.** Ниже в таблице 5.8 перечислены типы данных свойств, определенных для системы *i*FS.

Таблица 5.8

Список типов данных свойств в системе iFS

Тип данных **ifsType**	Тип данных столбца	Описание
ARRAY	Целый	Массив значений
RECORD	Целый	Массив значений свойств
DOCID	Целый	Уникальный идентификатор, который может использоваться вместо полного пути для ссылки на *i*FS документ
BOOLEAN	char(1)	Хранится в базе данных в виде *T или F.* Разработчики могут определять этот тип как логический или строковый, а *i*FS затем преобразует это значение к нужному значению уже в процессе выполнения операции
BIT	char(1)	0 или 1
INTEGER	Целый	Любые целые числа
NUMBER	Числовой	Самый универсальный числовой тип; можно записать любое значение
TIMESTAMP	Дата	Значение даты
DATE	Дата	То же, что и в TIMESTAMP, значение даты
TIME	Дата	То же, что и в TIMESTAMP или в DATE, значение даты
SHORT	Целый	То же, что и в INTEGER
LONG	Целый	То же, что и в INTEGER
DOUBLE	число(38*12)	Число с плавающей точкой с удвоенной точностью
FLOAT	число(19,6)	Число с плавающей точкой с одинарной точностью
VARCHAR	varchar2	Строка, максимальная длина которой определяется атрибутом **max_length**
STRING	varchar2	То же, что и в VARCHAR

- **max_length** Определяет максимальную длину (в символах) строки свойства. Это свойство применяется только к строковому типу данных. После того как строковое свойство сохранено в базе данных,

длина его строки будет сравниваться со значением **max_length**. Если длина строки больше значения **max_length**, то перед сохранением в базе данных она обрезается.

- **colname** Определяет название столбца для данного свойства. По умолчанию столбец будет назван **<propname>.** Если это свойство будет находиться в таблице, определяемой разработчиком, он должен назвать столбец так, чтобы его имя соответствовало столбцу в создаваемой таблице.
- **user_visible** Если атрибут имеет значение ложь (false), доступ к этому свойству осуществляется только программным образом, и оно не может запрашиваться или определяться конечным пользователем.
- **tablename** Определяет таблицу, в которой будет храниться определение данного свойства.
- **can_be_null** Определяет, можно ли вообще не вводить значение этого свойства. Если этот атрибут имеет значение истина (true), столбец базы данных не должен оставаться пустым, т. е. пользователь для создания документа данного типа должен вставить в этот столбец какое-либо значение.
- **synthetic** Определяет, имеет ли данное свойство хранимое или синтезированное значение. Синтезированные свойства рассчитываются во время выполнения программы путем отмены метода в классе **DB1** для данного типа документа.
- **use_existing_table** Это значение определяет, генерирует ли класс **TypeDBI** новую таблицу для хранения свойств составного свойства или же использует таблицу, уже созданную разработчиком.
- **properties** Массив свойств может содержать другой массив свойств. Такой прием используется для создания другой записи или свойства массива. Нижеследующий пример демонстрирует синтаксис, создающий стандартное свойство для имени супруга, за которым идет свойство записи, используемое для хранения имен и возраста детей:

```
<properties>
   <propdef>
      <propname> spousename </propname>
   </propdef>
   <propdef>
      <propname>child </propname>
      <datatype>record </datatype>
      <properties>
         <propdef>
            <propname>firstname</propname>
         </propdef>
```

```
            <propdef>
               <propname> age </propname>
               <datatype> integer </datatype>
            </propdef>
         </properties>
      </propdef>
   </properties>
```

- **refers_to** Это свойство используется для массива обращений или свойств записи, созданной с помощью конструкции **<declarations>**. Объявления имеют те же самые атрибуты, что и свойства, но определяются отдельно из других свойств. Объявляя массивы отдельно от использующих их записей, можно взять ту же таблицу для хранения аналогичной информации для более чем одного свойства. Например, в программе электронной почты свойства адреса есть у основного получателя письма (строка To:), у получателей копий (строка Cc:) и у получателей скрытых копий (Bcc:). Информация здесь одна и та же, отличается только способ ее использования в данный момент времени. Свойства для строк To, Cc и Bcc можно определить ссылкой на один и тот же массив, но объявить об этом следует отдельно. Эта ситуация описана в следующем примере:

```
<properties>
   <propdef>
      <propname> to </propname>
      <colname> to_addr </colname>
      <datatype> array </datatype>
      <refers_to> addresses </refers_to>
   </propdef>
   <propdef>
      <propname> cc </propname>
      <colname> cc_addr </colname>
      <datatype> array </datatype>
      <refers_to> addresses </refers_to>
   </propdef>
   <propdef>
      <propname> bcc </propname>
      <colname> bcc_addr </colname>
      <datatype> array </datatype>
      <refers_to> addresses </refers_to>
   </propdef>
</properties>
<declarations>
   <propdef>
      <propname> addresses </propname>
      <tablename> addresses </tablename >
      <datatype> record </datatype>
```

```
        <use_existing_table> T </use_existing_table>
        <properties>
          <propdef>
            <propname> address </propname>
          </propdef>
        </properties>
      </propdef>
    </declarations>
```

Расширения файлов

Все файлы в системе *iFS* обрабатываются в соответствии с определением типа документа. Принадлежность файлов к какому-то конкретному типу определяется по их расширениям. Определение типа документа можно использовать для обработки документов с несколькими разными расширениями файлов. Расширения файлов для того или иного типа документов определяются в документе расширения файла (сам он имеет расширение .fex). Документы расширений файлов также являются XML-документами.

Как именно используются XML-теги в определениях расширений файлов, описывается в таблице 5.9.

Таблица 5.9

XML-теги расширений файлов и их описания

XML-тег	Описание
<file_extension>	Самый внешний тег, который охватывает определение расширения.
<title>	Содержательное название определяемого расширения.
<type_of_extension>	Тип содержимого файла, определенный в *iFS*. Он должен точно соответствовать свойству **typename**, определенному в файле типа.
<mime_type_of_file>	Тип MIME описывает тип информации в документе в стандартном формате *general/specific*(общее/частное). Например, для файла HTML тип MIME равен *text/html.*
<extensions_list>	Охватывает объявления отдельных расширений.
<extension>	Определяемое расширение файла. Если таких расширений несколько, используется несколько тегов **<extension>**.

Пример определения расширения файла:

```
<?xml version="1.0" standalone="yes"?>
<file_extension>
  <title>Purchase Order Extensions </title>
```

```
<type_of_extension> PurchaseOrder </type_of_extension>
<mime_type_of_file> text/xml </mime_type_of_file>
<extensions_list>
   <extension> ipo </extension>
   <extension> opo </extension>
   <extension> purchaseorder </extension>
</extensions_list>
</file_extension>
```

В таблице 5.10 представлены построчные описания всех операций в вышеприведенном примере.

Таблица 5.10

XML-теги и их описания

XML-тег	Описание
<?xml version="1.0" standalone="yes"?>	Стандартное объявление, используемое всеми XML-файлами.
<file_extension>	Корневой элемент, который начинает определение расширения файла.
<title>Purchase Order Extensions</title>	Устанавливает содержательное имя для набора расширений файлов.
<type_of_extension>PurchaseOrder </type_of_extension>	Имя типа из файла определения типа.
<mime_type_of_file>text/xml </mime_type_of_file>	MIME-тип файла (плоский текстовый XML-файл).
<extensions_list>	Открывает список расширений.
<extension>ipo</extension>	Определяет расширение **.ipo**. /FS распознает любой документ с таким расширением как документ типа **PurchaseOrder.**
<extension>opo</extension>	Определяет второе эквивалентное расширение **.opo.**
<extension>purchaseorder</extension>	Определяет третье эквивалентное расширение **.purchaseorder**.
</extensions_list>	Закрывает список расширений.
</file_extension>	Завершает объявление расширения файла и документа.

Новые расширения файлов определяются путем вставки файлов их описаний .fex в любое место *i*FS репозитория. *i*FS распознает расширение **.fex** и обрабатывает файл, добавляя вновь определенные расширения в системный список. Если затем файл с описанным расширением попадает *в* репозиторий, он обрабатывается в соответствии с заданным ранее типом документа.

Использование системы /FS

До сих пор в этой главе обсуждались API-интерфейсы, к которым открывает доступ система *i*FS. Эти API-интерфейсы могут использоваться внешними Java-приложениями или когда *i*FS запускается внутри Oracle8i, чтобы предоставить приложению файловые службы. Ниже приведены примеры, иллюстрирующие несколько самых распространенных задач, выполняемых *i*FS-службами.

Пример 1. Создание и сохранение приветствия миру "Hello World"

Среди самых распространенных задач, которые выполняются приложениями для поиска файлов конфигурации или файлов с данными, можно назвать создание и сохранение файлов в заданных каталогах (например, в текущем домашнем каталоге пользователя). Приведенный здесь код демонстрирует использование API-интерфейсов *i*FS для создания Java-класса, который будет создавать файл **HelloWorld.txt,** добавлять в него содержимое "Hello World", определять местоположение этого файла и добавлять его в текущий домашний каталог пользователя.

```
public static void example1() throws IfsException
{ LibraryService ifsService = new LibraryService();
  LibrarySession ifsSession;
  ifsSession = ifsService.connect("user","pass","Local");

  DirectoryUser thisUser = ifsSession.getDirectoryUser();
  PrimaryUserProfile userProfile;
  userProfile = ifsSession.getPrimaryUserProfile(thisUser);
  Folder homeFolder = userProfile.getHomeFolder();

  DocumentDefinition newDoc;
  newDoc = new DocumentDefinition(ifsSession);
  newDoc.setName("HelloWorld.txt");
  newDoc.setContent("Hello World");

  Document doc;
  doc = (Document) ifsSession.createPublicObject(newDoc);
  homeFolder.addItem(doc);

  ifsSession.disconnect();
}
```

Доступ в текущую домашнюю папку пользователя производится с помощью метода **Profile.getHomeFolder() для PrimaryUser**. Чтобы получить объекты пользователя, класс **LibrarySession** создает методы **getPrimaryUserProfile() и getDirectoryUser()**. Метод **getPrimaryUserProfile()** получает объект **PrimaryUserProfile,** а метод

getDirectoryUser() получает объект **DirectoryUser**, который используется для получения текущего профиля пользователя. Этот пример иллюстрирует процедуру создания нового документа с помощью метода **DocumentDefinition()**, а также процедуру создания его имени и содержимого с помощью методов **setName()** и **setContent()**. Затем этот документ можно в качестве объекта добавить в домашнюю папку с помощью методов **createPublicObject()** и **addItem()**.

Пример 2. Создание более качественного приветствия миру "Hello World"

На базе первого примера можно построить более сложный пример, где формат документа будет устанавливаться на основе его расширения, для создания документа и добавления его в домашнюю папку создается единый логических блок, а в конце добавлена дополнительная функция составления отчета о любых произошедших ошибках.

```
public static void addToFolder (LibrarySession ifsSession,
                                DocumentDefinition newDoc,
                                Folder homeFolder)
                                throws IfsException
{ IfsException.setVerboseMessage(true);
   Transaction t = ifsSession.beginTransaction();
   try {
      Collection c = ifsSession.getFormatExtensionCollection();
      Format textFormat = (Format) c.getItems("txt");
      newDoc.setFormat(textFormat);
      Document doc;
      doc = (Document) ifsSession.createPublicObject(newDoc);
      homeFolder.addItem(doc);
      ifsSession.completeTransaction(t);
   }
   catch (IfsException e)
   {e.printStackTrace();
      ifsSession.abortTransaction(t);
   }
}
```

Предположим, что системе *i*FS нужно узнать mime-тип документа для Правильного индексирования его содержимого. Это выполняется путем передачи объекта Format, который определяет mimeType, в метод **DocumentDefinition.setFormat()**. Так как *i*FS поставляется вместе с набором наиболее часто встречающихся определений объектов Format, можно использовать расширение файла для поиска формата с помощью метода **getFormatExtensionCollection()**.

В качестве логического тела задачи используются транзакции. Если указать, чтобы в процессе выполнения одной транзакции выполнялись сразу несколько операций, как это сделано в данном примере между методами **beginTransaction()** и **completeTransaction()**, весь процесс можно провести более эффективно. Начиная с Oracle9iFS версии 1.2, можно встраивать собственные SQL-операторы в транзакцию доступа к файлу *iFS.*

Все ошибки возвращаются в класс **ifsException** и обрабатываются кодом, использующим стандартный синтаксис **try л catch.** В данном случае все ошибки распечатываются методом **printStackTrace(),** после чего метод **abortTransaction()** прекращает транзакцию, все изменения отменяются, память освобождается и производится блокировка ресурсов.

Пример 3. Работа с файлами

Приложениям приходится выполнять не только операции создания и сохранения файлов. Удобным контейнером для файлов являются папки. В приведенном ниже примере показаны операции поиска папки по заданному пути, ее открытие и чтение ее содержимого.

```
public static void readFile (LibrarySession ifsSession,
                             Folder folder, String name)
                             throws IfsException, java.io.IOException
{  if (folder.checkExistenceOfPublicObjectByPath (name,File.separator) )
   {  PublicObject po;
      po = folder.findPublicObjectByPath(name,File.separator) ;
      if (po.getClassname ( ) .equals(Document.CLASS_NAME) ) {
         Document doc = (Document) po;
         BufferedReader r;
         r = new BufferedReader (doc.getContentReader ());
         for (String nextLine = r . readLine ();
              nextLine != null;
              nextLine = r. readLine () )
         { System. out.println (nextLine);  }
      }
   }
}
```

Перед тем как открывать папку, нужно проверить, существует ли она вообще. Для этого вызывается метод **checkExistenceOfPublicObjectByPath()**, возвращающий ответ — существует ли папка (true или false). Затем метод **findPublicObject-ByPath()** возвращает файловый объект. И уже после этого можно получить Доступ и прочитать файловый объект с помощью метода **getContentReader()**. Затем содержимое папки распечатывается.

Пример 4. Поиск файлов

Последний пример является иллюстрацией того, как можно провести простой поиск в коллекции файлов и получить пути к файлам, соответствующим заданным критериям.

```
public static void findMatchingFiles(LibrarySession
                                     ifsSession, String name)
                                     throws IfsException
{  String search = PublicObject.NAME_ATTRIBUTE + "= '" + name +"'";
   Selector mySelector = new Selector(ifsSession,Document.CLASS_NAME,search);

   if (mySelector.getItemCount() == 0)
   System.out.println("Search did not find any documents.");
   else {
      for (int i=0;i<mySelector.getItemCount();i++) {
         Document myDoc = (Document)
         mySelector.getItems(i);
         String path = myDoc.getAnyFolderPath();
         System.out.println("Found a Document at: " + path);
      }
   }
}
```

Отметим, что строка поиска связана с атрибутом **name,** который был объявлен в формате:

NAME_ATTRIBUTE+"="+*filename*+""

Класс **Selector** для процесса отбора файлов предоставляет несколько методов, аналогичных тем, что использовались для обработки списка папок. В данном случае для создания списка используются методы **getItems()** и **getItemCount()**, после чего производится вывод списка на печать.

Использование iFS с XML-файлами

*i*FS имеет встроенные средства обработки XML-файлов. Хотя *i*FS знает, как обрабатывать и хранить все типы файлов, функции по обработке XML-файлов обладают максимальной универсальностью и расширяемостью. Их можно разбить на две большие категории — функции для работы с не анализированными XML-файлами и функции для работы с синтаксически разобранными XML-файлами. Рассмотрим их более подробно.

Хранение синтаксически разобранных XML-файлов

Репозиторий *iFS* распознает файлы XML-формата по их расширению **.xml**. Однако файлы этого формата могут иметь и другие расширения, для этого можно определить подкласс расширения **.xml**. Например, описанный выше заказ на покупку может иметь расширение .po, а FAQ-файл с ответами на часто встречающиеся вопросы — расширение .faq. Но и такие файлы могут распознаваться как файлы XML-формата, поскольку репозиторий iFS сначала проверяет, зарегистрировано ли данное расширение в качестве подкласса в любой из его определений типов. В этом случае репозиторий *i*FS будет искать файл формата XML, и если в репозиторий есть соответствующий файл конфигурации, то создаются новые объекты, например подклассы, пользователи или их группы. Наконец, при необходимости провести синтаксический анализ документа репозиторий вызывает XML-анализатор синтаксиса и проверяет корневой элемент документа, находит соответствующий подкласс и создает новый *i*FS-объект, если документ признан допустимым.

Используемый анализатор синтаксиса можно также определить через XML Parser Framework. Например, вместо использования анализатора SimpleXMLParser, поставляемого вместе с *i*FS, можно зарегистрировать анализатор, поставляемый с инструментарием разработчика XDK for Java, который полностью поддерживает стандарты консорциума W3C. Регистрация выполняется через опцию Register в меню Object из /FS Manager (см. рис. 5.4).

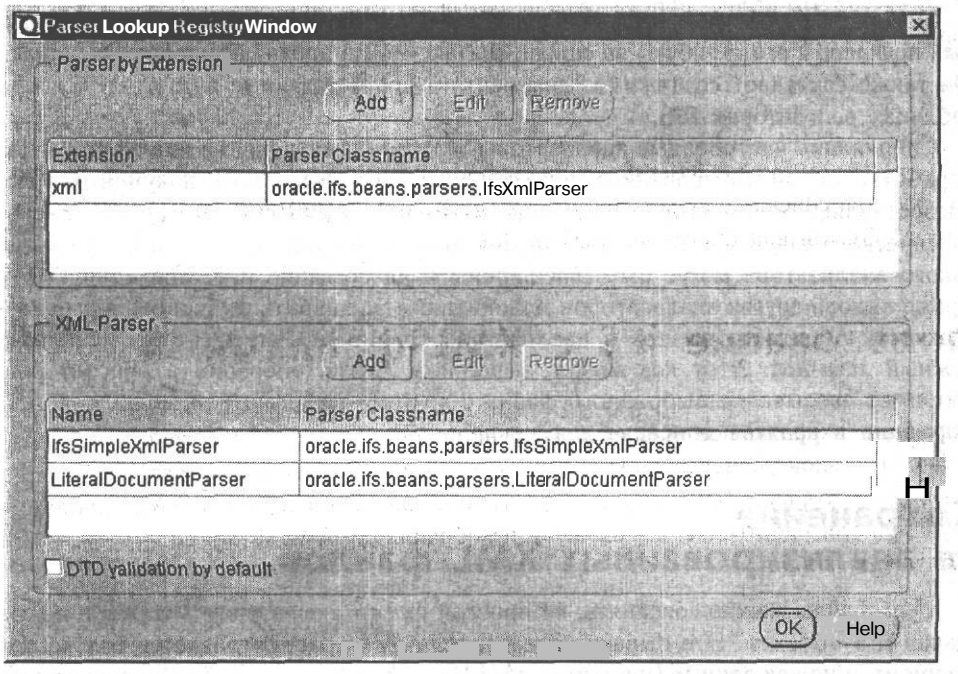


Рис. 5.4. *Регистрация анализатора синтаксиса*

Кроме анализаторов от Oracle и других сторонних поставщиков можно задать специально настроенные анализаторы, которые будут последовательно работать с XML-файлами. Для начала нужно быть уверенным, что данные XML-файлы действительно имеют как минимум два анализатора синтаксиса, связанные с каждым расширением,— *Dispatching Parser* и *Document Parser*. Первым используется Dispatching Parser. Он должен подтвердить, что XML-файл является правильным (well formed) и допустимым (valid) в плане соответствия своему файлу описания типа. Он может также соответствовать своему DTD. Проверка на правильность производится по умолчанию, а вот проверка на допустимость, т. е. на соответствие DTD — нет. Так сделано потому, что пользователю обычно не нужно проводить проверку большого количества файлов, особенно если этим занимаются приложения. Проверка на допустимость запускается из того же меню Object менеджера *i*FS. Проверка на соответствие DTD — довольно трудоемкая затея, поэтому в качестве Dispatching Parser лучше всего использовать анализатор SAX, встроенный в Oracle Parser for Java. Его основанные на событиях функции обрабатывают XML-файл, особо не загружая память и без задержек при запуске, свойственных DOM-анализатору. Провести проверку файла на допустимость можно и через встроенный DTD или с помощью внешнего DTD, на который имеется ссылка.

По поводу проверки на соответствие XML-файла своему DTD важно помнить: если файл не соответствует DTD, то не будет создано никакого подкласса и никакого *i*FS объекта. Однако можно сделать так, чтобы в этой ситуации пользователь получал сообщение о необходимости исправить ошибку, после чего нужно попробовать провести проверку еще раз.

Document Parser относится к тем анализаторам, которые могут создавать заданный подкласс и его атрибуты на основе синтаксически проанализированных данных. Он также сохраняет содержимое документа и его метаданные в соответствующих таблицах репозитория *i*FS.

Специально настроенные анализаторы могут вызывать другие анализаторы для обработки специализированных или сложных составных XML-документов. Это предоставляет пользователю возможность полного контроля над процессом синтаксического анализа и следующим за ним процессом создания подкласса. Функции такого анализатора могут быть расширены и на создание представления DOM. Кроме вызова других анализаторов можно также расширить функциональные возможности обратного вызова в анализаторе Document Parser, чтобы исполнять нужный Java-код. Этот код может выполнять другие операции в репозитории, например связывание выбранного файла с другими файлами или добавление информации в краткое описание документа.

Сохранение не анализированных XML-файлов

XML-файлы можно сохранять, не проводя их синтаксического анализа и не создавая *i*FS-объекты. Это бывает полезно, если файл рассматривается только как документ, а не как данные (например, xHTML, JavaDoc и т. д.). Тогда репозитории

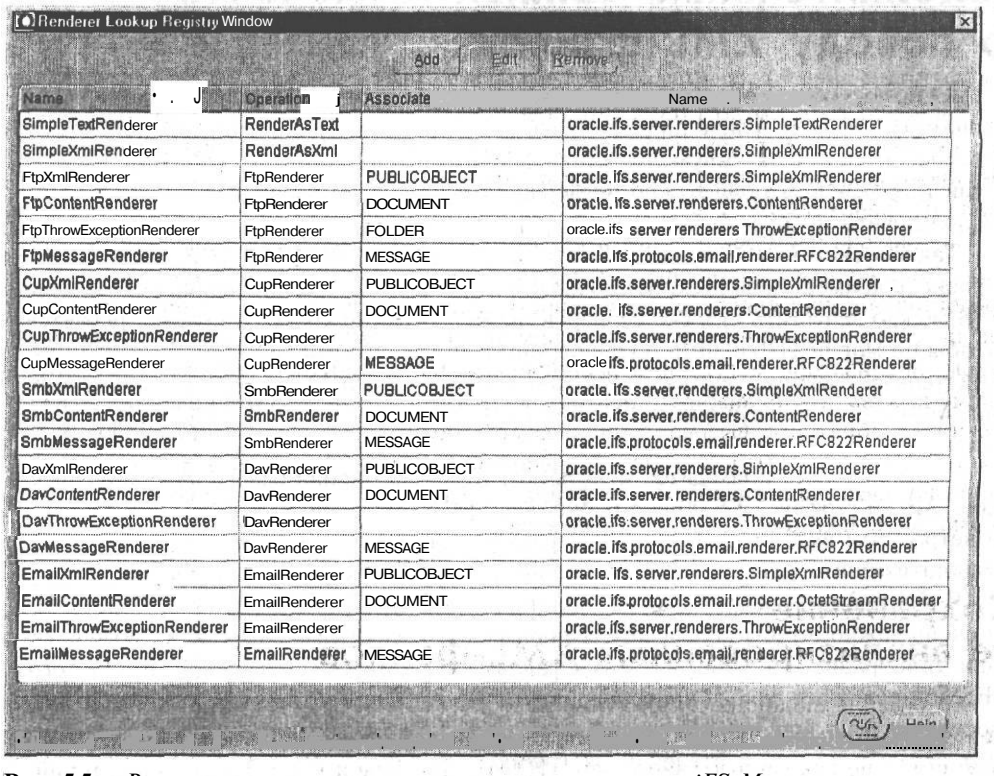
iFS можно сконфигурировать так, чтобы он принимал XML-файлы без их синтаксического анализа. Сделать это можно двумя разными способами.

Если не определен подкласс, связанный с расширением XML-файла, файл не будет анализироваться как файл XML-формата. Тем не менее репозиторий сохранит его, но как CLOB-объект. Затем его можно проиндексировать с помощью *inter*Media Text.

Если никакой анализатор синтаксиса не связан с данным расширением XML-файла, то файл также не будет анализироваться и будет сохранен как CLOB-объект. Это произойдет, даже если поставить ему в соответствие какой-то подкласс. Этот файл также можно проиндексировать с помощью *inter*Media Text.

Визуализация XML-файлов

Как и в случае с синтаксическим анализом, *i*FS имеет расширенную структуру визуализации файлов, позволяющую восстанавливать XML-документы. В эту структуру включена базовая программа визуализации XML-файлов, запускаемая при запросе пользователем XML-документа из репозитория. Например, если запрос был подан через браузер Internet Explorer, /FS использует базовую программу визуализации для восстановления XML-документа из его хранимых подкомпонентов.



Name	Operation	Associate	Name
SimpleTextRenderer	RenderAsText		oracle.ifs.server.renderers.SimpleTextRenderer
SimpleXmlRenderer	RenderAsXml		oracle.ifs.server.renderers.SimpleXmlRenderer
FtpXmlRenderer	FtpRenderer	PUBLICOBJECT	oracle.ifs.server.renderers.SimpleXmlRenderer
FtpContentRenderer	FtpRenderer	DOCUMENT	oracle.ifs.server.renderers.ContentRenderer
FtpThrowExceptionRenderer	FtpRenderer	FOLDER	oracle.ifs server renderers ThrowExceptionRenderer
FtpMessageRenderer	FtpRenderer	MESSAGE	oracle.ifs.protocols.email.renderer.RFC822Renderer
CupXmlRenderer	CupRenderer	PUBLICOBJECT	oracle.ifs.server.renderers.SimpleXmlRenderer
CupContentRenderer	CupRenderer	DOCUMENT	oracle. ifs.server.renderers.ContentRenderer
CupThrowExceptionRenderer	CupRenderer		oracle.ifs.server.renderers.ThrowExceptionRenderer
CupMessageRenderer	CupRenderer	MESSAGE	oracle ifs.protocols.email.renderer.RFC822Renderer
SmbXmlRenderer	SmbRenderer	PUBLICOBJECT	oracle. ifs.server.renderers.SimpleXmlRenderer
SmbContentRenderer	SmbRenderer	DOCUMENT	oracle.ifs.server.renderers.ContentRenderer
SmbMessageRenderer	SmbRenderer	MESSAGE	oracle.ifs.protocols.email.renderer.RFC822Renderer
DavXmlRenderer	DavRenderer	PUBLICOBJECT	oracle.ifs.server.renderers.SimpleXmlRenderer
DavContentRenderer	DavRenderer	DOCUMENT	oracle.ifs.server. renderers.ContentRenderer
DavThrowExceptionRenderer	DavRenderer		oracle.ifs.server.renderers.ThrowExceptionRenderer
DavMessageRenderer	DavRenderer	MESSAGE	oracle.ifs.protocols.email.renderer.RFC822Renderer
EmailXmlRenderer	EmailRenderer	PUBLICOBJECT	oracle. ifs. server.renderers.SimpleXmlRenderer
EmailContentRenderer	EmailRenderer	DOCUMENT	oracle.ifs.protocols.email.renderer.OctetStreamRenderer
EmailThrowExceptionRenderer	EmailRenderer		oracle.ifs.server.renderers.ThrowExceptionRenderer
EmailMessageRenderer	EmailRenderer	MESSAGE	oracle.ifs.protocols.email.renderer.RFC822Renderer

Рис. 5.5. *Регистрация программы визуализации с помощью iFS Manager*

Индивидуально настроенные программы визуализации можно регистрировать точно так же, как и настроенные пользователем анализаторы синтаксиса. Эта операция производится с помощью функции Object Registry в *i*FS Manager. При этом используется окно Renderer Registry Lookup Window, как показано на рис. 5.5.

Возможности расширения данной программы визуализации используются для создания настраиваемых пользователем программ, которые применяют XSL-таблицы стилей к запрашиваемым XML-документам, преобразовывая их в форматы, совместимые с запрашивающим устройством. Например, один и тот же XML-документ можно представить в форматах HTML, XML, WML и RTF, применив к нему соответствующую таблицу стилей. Таблицы стилей можно использовать для фильтрации или добавления информации в выходной документ.

Можно создать специально настроенную программу визуализации для сборки документов из динамического контента, например, для создания отчета о продажах из таблицы счетов-фактур. Такой отчет реализуется в виде XML-файла, преобразуется и посылается по протоколу HTTP на запрашивающий браузер. А поддержка средств безопасности в *i*FS должна гарантировать, что пользователь, отправивший запрос, имеет право на просмотр подобной информации.

Дополнительные важные замечания по поводу XML-файлов

XML-документы могут иметь самую разную кодировку символов. Однако важно понимать, что если нужно поддерживать несколько языков и кодировок, необходимо произвести некую разумную селекцию набора символов в базе данных. Как правило, если в *i*FS-репозитории нужно поддерживать несколько кодировок, рекомендуется использовать кодировку UTF8.

Так как система *i*FS может работать и внутри базы данных, и на промежуточном уровне, она имеет довольно гибкую архитектуру. При масштабировании репозитория *i*FS можно запустить столько серверов промежуточного уровня, сколько требуется, чтобы он мог поддерживать большое число пользователей. Все они будут взаимодействовать с одной и той же базой данных. Еще большей масштабируемости можно добиться, используя опцию Partitioning (разбиение) в СУБД Oracle8i Enterprise Edition.

Довольно часто приходится менять поведение всех копий одного из набора классов. API-интерфейс *i*FS имеет набор так называемых Tie-классов. Они хранятся в нетронутом виде на самом верху иерархии классов сразу под Library Object, и их можно изменить или расширить, чтобы выполнить указанную операцию.

В *i*FS практически все файлы свойств и конфигурации построены на базе XML, и очень важно, чтобы пользователь производил только разрешенные их модификации. Для правильной работы с этими файлами существует *i*FS Manager. Кроме того, *i*FS поддерживает списки контроля доступа Access Control Lists (ACLs), чтобы предоставлять или аннулировать разрешения на работу с заданными файлами и папками для отдельных пользователей и их групп.

Глава 6

Поиск XML-документов с помощью Oracle Text

азы данных переживают революционный период. Данные в традиционных реляционных базах данных жестко структурированы, и запросы на доступ к ним всегда должны соответствовать схеме конкретной базы данных. Такие базы данных хорошо использовать для хранения структурированной информации, но они теряют все свои достоинства при обработке и/или поиске данных, представленных в других формах. Классический язык SQL хорош для выполнения неструктурированного списка в небольших столбцах, но он не обладает достаточной мощью для работы с длинными текстовыми столбцами. Поиск данных в неструктурированных *больших объектах* типа *LOB* (Large Data Object) традиционными методами считается неэффективным и медленным. Поэтому многие современные поставщики баз данных ставят перед собой цель добиться высокопроизводительного доступа к таким сложным типам данных. По мере того, как снимаются жесткие ограничения на доступ к данным, базы данных также ослабляют жесткие требования к структуре данных.

Информация, хранящаяся в современных Интернет-базах данных, имеет самые разные формы. Видео, графика, текст — и все это в больших количествах. Да и размер файлов этих типов часто на несколько порядков превышает этот параметр для структурированных баз данных. Сама сеть Интернет предъявляет совсем другие требования для доступа к данным. Этот доступ бывает в Интернете эпизодическим и незапланированным, поэтому наиболее приемлемым механизмом для обмена текстовыми данными становятся XML- и HTML-документы. Следовательно, современные Web-сайты с базами данных должны обеспечивать простой поиск и получение этих документов с высоким уровнем масштабируемости и производительности.

Oracle Text — это компонент СУБД Oracle9i, благодаря которому пользователи могут управлять самыми разными типами данных, используя для этого стандартный SQL-доступ. Средство Oracle Text, ранее известное под названиями interMedia Text и ConText, обрабатывает текстовые документы, изображения, аудио- и видеоданные. Оно позволяет легко управлять такими данными, в том числе обеспечивать Интернет-поддержку для средств Web-разработки и Web-серверов. В частности, Oracle text повышает эффективность использования БД при обработке текстовой информации, предоставляя возможности поиска в рамках стандартного SQL-запроса, обрабатываемого целиком внутри ядра БД, а не в дополнительном текстовом процессоре.

Эта глава посвящена архитектуре индексирования текста Oracle Text, причем особое внимание уделяется следующим функциям, связанным с технологией XML: XML_SECTION_GROUP, AUTO_SECTION_GROUP и PATH_SECTION_GROUP (новая функция), операторам HASPATH() и INPATH(), представленным в Oracle9i, а также их зонам, полям и атрибутам. Особенности использования Oracle Text иллюстрируются на примере приложения Bookstore.

Oracle Text как средство текстового поиска нового поколения

Современные реляционные СУБД могут легко справляться со всеми аспектами индексирования простых типов данных, например целыми числами и небольшими строковыми данными. Однако сложные типы данных — структурированный текст, пространственные данные, графика, видео и звук — требуют организовывать поиск на основе их содержимого. Они имеют собственные форматы, специфичные для каждого конкретного приложения, свои требования к индексированию и свои предикаты для выбора данных. Документы с разметкой, например HTML- и XML-файлы, могут потребовать проведения поиска по тегам или атрибутам. Такая ситуация повышает спрос на индексирование сложных типов данных и на специализированные методы индексирования.

В ПО от Oracle введена концепция индексного типа *(indextype),* которая наряду с расширяемой структурой позволяет определять индексные типы по требованию. *Indextype* является аналогом сортированных или побитных индексных типов, встроенных в Oracle Server. Но здесь есть одно важное отличие. Индексные типы не являются встроенными, разработчик приложения должен их определить сам. Суть типов индексирования построена на идее коллективного индексирования, которая предполагает кооперацию средств interMedia и Oracle9i для построения и сопровождения индексов текстовых и пространственных данных, а также применение *аналитической обработки данных в реальном времени* (online analytical processing, *OLAP).* Средство interMedia отвечает за определение структуры индекса, сохранение его содержимого во время операций загрузки и обновления и за поиск в индексе во время обработки запросов. Сама структура индекса может храниться либо в СУБД Oracle в виде *индексно-организованной таблицы* (index-organized table, *ЮТ), либо* вне базы данных в виде файла операционной системы. Разработчик определяет конкретный вид реализации индексного типа, тогда как ядро СУБД Oracle реализует встроенные сортированные или побитные индексы.

Индексы Oracle Text могут создаваться практически для любого столбца базы данных. Когда в один из таких столбцов загружается какой-нибудь документ, это изменение детектируется автоматически и подается сигнал об индексировании. Во время индексирования, которое может выполняться сразу же или по какому-то определенному пользователем расписанию, все слова и темы документа записываются в индексные таблицы для последующего извлечения. В этом индексе выполняются текстовые запросы, и в качестве ответа на них возвращаются списки соответствующих запросам документов. Индексы хранятся в базе данных, и все запросы выполняются внутри этой базы данных с помощью единого API-интерфейса.

Программа-оптимизатор Oracle Cost Based Optimizer (*CBO*) оценивает и выбирает для заданных свойств данных наиболее быстрый план выполнения запроса. Оптимизатор CBO может передавать идентификаторы строк, удовлетворяющие

текстовому предикату, а может сначала проверять, удовлетворяет ли этому предикату строка с заданным идентификатором. Второй метод использует для доступа к инвертированному текстовому индексу с целью поиска заданной строки специальный алгоритм, впервые введенный в Oracle8i.

Oracle8i стала первой реляционной базой данных, в которой средства текстовых запросов интегрированы в ядро базы данных. Эта полностью интегрированная система использует для обработки всего запроса ядро реляционной СУБД, поэтому в названии этого раздела и указано, что Oracle Text — это средство текстового поиска нового поколения.

Архитектура индексирования Oracle Text

Процесс обработки текста с помощью Oracle Text можно представить в виде ленты конвейера, которая движется от одних модулей к другим, причем на каждом этапе конвейера возможны разные варианты работы. В конце конвейера должен получиться *инвертированный индекс,* представляющий собой список слов из документа. Каждому слову из списка сопоставлен список документов, в которых оно встречается. Такой индекс называется инвертированным, потому что он противоположен нормальному виду текста, который обычно представляется в виде списка документов, где каждый документ содержит список слов. Каждый этап конвейера индексирования можно настроить в соответствии с нуждами пользователя с помощью опций Preference System и/или Section Groups. Их можно создать, используя специальные API-интерфейсы:

```
ctx_ddl.create_preference(...);
ctx_ddl.create_secrtion_group(...);
```

которые подключаются к индексирующему конвейеру с помощью следующего параметрического оператора:

```
create index xml_index on Bookstore (xml_text)
        indextype is ctxsys.index
        parameters ('...');
```

Следующие четыре раздела посвящены подробному описанию этапов, или модулей, индексирующего конвейера, информации, проходящей по этому конвейеру, а также опциями и командам, которые можно применять на каждом этапе. Общая схема передачи информации по конвейеру представлена на рис. 6.1.

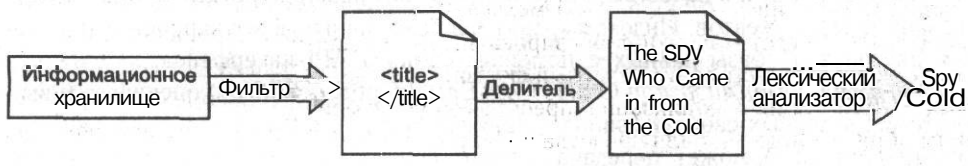


Рис. 6.1. *Информационный поток в индексирующем конвейере*

Информационное хранилище

Индексирующий конвейер берет свое начало в *информационном хранилище,* где происходит связывание строк таблицы и считывание данных из столбца. Строки считываются не как попало, а в совершенно определенном порядке. Текст, проиндексированный с помощью Oracle Text, можно хранить в БД или в файловых системах. Управление этими данными осуществляется из СУБД. URL-адрес информационного хранилища позволяет базе данных управлять документами, хранящимися удаленно на других серверах, и получать к ним доступ по протоколам HTTP или FTP. Обычно это данные из столбцов, но в некоторых информационных хранилищах они используются в качестве указателя на данные документов. Например, URL_DATASTORE использует в качестве URL-адреса данные из столбца таблицы, выполняет оператор GET и выдает возвращаемые данные. Такие информационные хранилища создаются и встраиваются в конвейер с помощью следующего фрагмента кода:

```
ctx_ddl.create_preference ('xml_urls', 'URL_DATASTORE');
ctx_ddl.set_attribute('xml_urls', 'HTTP_PROXY',
                      'www.proxy.us.oracle.com');
...
create index xml_index on bookstore (xml_text) ...
   parameters ('datastore xml_urls ... ' ) ;
```

Программа фильтрации

Программа фильтрации берет данные документа из информационного хранилища и преобразует их в один из видов текстового представления, например в файл форматов XML и HTML или в плоский текстовый файл. Это нужно делать при хранении таких бинарных документов, как файлы форматов Microsoft Word или Adobe Acrobat. В программе Oracle Text есть фильтры для более чем сотни форматов файлов, которые могут быть смешаны в одном столбце таблице. Во фрагменте кода, приведенном ниже, показан, правда, только один выходной фильтр — для HTML-формата.

Кроме того, можно подключить к работе индексирующего конвейера собственный *специальным образом настроенный фильтр,* а также фильтры производства сторонних разработчиков. Специально настроенный фильтр представляет собой исполняемую программу или скрипт с двумя аргументами, первый из них — это файл, содержащий входной отформатированный текст, а второй — название файла, куда следует записать все, что окажется на выходе фильтрующей программы. Фильтрующие программы создаются и встраиваются в конвейер точно так же, как это делается с информационными хранилищами, и фрагмент кода для выполнения этой операции выглядит аналогично:

```
ctx_ddl.create_preference ('xml_filter', 'URER_FILTER');
ctx_ddl.set_attribute ('xml_filter', 'EXECUTABLE', 'xmlfilter.exe');
...
```

```
create index xml_index on bookstore (xml_text)
   ...
   parameters ( 'datastore xml_urls filter xml_filter ... ' ) ;
```

Программа разбиения на секции

Программа разбиения на секции (sectioner) берет данные на выходе фильтра и преобразует их в плоский текст. Процесс преобразования управляется классом группы секций, который сообщает программе о том, каким образом производить синтаксический разбор введенной информации. Введенные данные с фильтра могут быть синтаксически проанализированы как XML- или HTML-документ, *а* также как плоский текст. Группа секций обычно содержит одну или несколько секций. Определив простое отображение, разработчики приложений могут дать каждой секции имя по своему выбору. Программа разбиения на секции распознает их для каждого блока текста и автоматически индексирует текст как часть этих секций. Секции могут быть также HTML-, XML- или любой другой секцией вида *<StartTag>...</EndTag>* в группе AUTO_SECTION_GROUP. Ниже приведен пример кода для проведения поиска в проиндексированной секции, в котором используется оператор CONTAINS:

```
SELECT pk FROM bookstore
   WHERE CONTAINS (xmltext, '(The Spy) WITHIN title') > 0;
```

По этому запросу будет найден документ, содержащий следующую XML-разметку:

```
<book>
   <id> 4 </id>
   <author> John LeCarre </author>
   <title> The Spy Who Came in from the Cold </title>
</book>
```

Если группа секций пуста, программа разбиения на секции преобразует отформатированный текст в плоский текст, но не будет индексировать никаких границ между секциями. На выходе программы разбиения на секции мы получим границы секций и содержимое документов в виде плоского текста. Все это отправляется на лексический анализатор. Операция преобразования в плоский текст также предусматривает поиск важных тегов секции, удаление невидимой информации и переформатирование текста. Границы секции затем направляются в поисковое средство. Группы секций создаются и подключаются с помощью API-интерфейсов Section Group:

```
ctx_ddl.create_section_group ( 'group_name', 'group_type' );
...
create index
   ...
   parameters ( 'sectiongroup xml_group ...');
```

Программа разбиения на секции учитывает тип группы при преобразовании входных данных, поступающих с фильтра, в плоский выходной текст и при расчете индексов для границ секций. Для XML-документов важны следующие группы секций:

- **BASIC_SECTION_GROUP** Распознаются открывающие и закрывающие теги. DOCTYPE, объекты и атрибуты тегов не поддерживаются. Производится только удаление тегов разметки из выходного плоского текста.
- **HTML_SECTION_GROUP** Очевидно, что эта группа секций предназначается для HTML-документов. Она поддерживает не только теги языка HTML 4.0, но также обрабатывает неизвестные теги. Удаляется содержимое и комментарии секций SCRIPT и STYLE, а содержимое секции TITLE сохраняется.
- **XML_SECTION_GROUP** Программа разбиения на секции проводит синтаксический разбор входного текста как XML-документа и обрабатывает внутренние объекты, секции DOCTYPE и атрибуты тегов. Теги удаляются, а внутренние объекты обрабатываются, чтобы на выходе получился плоский текст. Индексируются определенные пользователем секции из данной группы секций.
- **AUTO_SECTION_GROUP** Аналогично HTML_SECTION_GROUP, входной текст обрабатывается как XML-документ, но без секций, определенных пользователем. Вместо этого все непустые теги автоматически индексируются как секция зоны, и имя этой секции будет совпадать с именем тега.
- **PATH_SECTION_GROUP** Эта группа соответствует AUTO_SECTION_GROUP в том, что касается тегов и атрибутов внутри секции. Здесь программа разбиения на секции также позволяет новым операторам HASPATH() и INPATH() выполнять поиск информации, аналогичный тому, что производит оператор XPATH. Кроме того, запросы в этой секции зависят от имен атрибутов и тегов.

Эти секции имеют три важных атрибута: тег, имя и тип. Тег указывает программе разбиения на секции, как именно распознавать эту секцию. Когда программа находит текстовую последовательность *<tag>...</tag>,* она автоматически индексирует ее для поиска. Теги во всех секциях одной группы секций должны быть уникальными. В запросах должна быть ссылка на имя секции. Одному и тому же имени можно поставить в соответствие несколько тегов, и тогда они рассматриваются как экземпляры одной и той же секции. Для XML-документа важными являются следующие типы секций:

- **ZONE** Секции зоны могут повторяться, и каждый такой экземпляр будет рассматриваться отдельно в семантике запроса. Эти секции могут содержать и другие секции, в том числе и секции зоны.

Кроме того, они могут сами входить в состав других секций. В случае групп AUTO_SECTION_GROUP или PATH_SECTION_GROUP программа разбиения на секции автоматически индексирует все непустые теги как секции зоны, и такой секции присваивается то же имя, что и тегу.

- FIELD Программа разбиения выделяет из этой секции все содержащиеся в ней элементы и индексирует их отдельно. За счет этого запросы к секции поля *FIELD* могут выполняться в три раза быстрее, чем запросы к секции зоны, особенно если теги этой секции присутствуют в каждом документе. Правда, такая скорость достигается за счет потери универсальности. Секции поля предназначаются для неповторяющихся и неперекрывающихся секций. Если секция поля повторяется в том же документе, она рассматривается как продолжение секции, а не как отдельный ее экземпляр. Если секция поля перекрывается самой собой или другой секций поля, она закрывается в той точке, где начинается другая секция. Кроме того, отметим, что в каждой группе секций могут существовать максимум 64 секции поля.
- ATTRIBUTE Секции атрибутов могут быть добавлены только в группы XML_SECTION_GROUP. Программа разбиения на секции индексирует содержимое атрибута секции и делает его доступным для запросов. В случае групп AUTO_SECTION_GROUP или PATH_SECTION_GROUP программа разбиения на секции автоматически индексирует значение атрибутов как секции атрибутов с именами *<tag>@<attribute>*.
- STOP Секции останова (stop) используются только в группах AUTO_SECTION_GROUP и PATH_SECTION_GROUP. Эта секция указывает, что соответствующий *<tag>* следует игнорировать и не индексировать как секцию. Количество секций останова в каждой группе AUTO_SECTION_GROUP или PATH_SECTION_GROUP не ограничено.

Лексический анализатор

Лексический анализатор получает от программы разбиения на секции плоский текст и разбивает его на отдельные лексемы или слова. В программе Oracle Text есть лексические анализаторы для языков, в которых слова разделяются пробелами, а также для азиатских языков, где сегментация слов сложнее. Лексический анализатор basic_lexer, используемый по умолчанию, имеет функцию создания унифицированных индексов текста и тем. Аналогично информационным хранилищам, лексические анализаторы создаются и подключаются к индексирующему конвейеру с помощью системы предпочтений:

```
ctx_ddl.create_preference ('my_basic_lexer', 'basic_lexer');
...
```

```
create index xml_index on bookstore (xml_text)
   ...
   parameters ('lexer my_basic_lexer ...');
```

Наконец, Oracle9i берет все лексемы от лексического анализатора и все секции из программы разбиения и строит инвертированный индекс. Затем в него записываются лексемы и документы, в которых имеются эти лексемы.

Работа с Oracle Text

Вот типичная последовательность SQL-операторов, используемых в Oracle Text для работы со структурированными документами:

```
create table bookstore (
   pk       NUMBER PRIMARY KEY ,
   xml_text CLOB
);
insert into bookstore values (111,
   '<Book>
      <Id>4<\Id>
      <Author>John LeCarre</Author>
      <Title>The Night Manager</Title>
   </Book>');
insert into bookstore values (112,
   '<Book>
      <Id>5<\Id>
      <Author>John Grisham</Author>
      <Title>The Client</Title>
   </Book>');

/* ... insert the rest of the books into the Bookstore */

commit;

begin
   ctx_ddl.create_section_group('xml_sections', 'XML_SECTION_GROUP');
   ctx_ddl.add_zone_section    ('xml_sections', 'titlesec', 'title');
   ctx_ddl.add_zone_section    ('xml_sections', 'authorsec', 'author');
   ctx_ddl.create_preference   ('my_basic_lexer', 'basic_lexer');
   ctx_ddl.set_attribute       ('my_basic_lexer', 'index_text','true');
   ctx_ddl.set_attribute      ('my_basic_lexer', 'index_themes','false');
end;
```

```
create index xml_index on bookstore (xml_text)
   indextype is ctxsys.context
   parameters ( 'lexer my_basic_lexer SECTION GROUP xml_sections');

select pk from bookstore
   where contains (xml_text, 'LeCarre WITHIN authorsec') > 0 ;
```

Этот пример Bookstore иллюстрирует создание и заполнение таблицы, создание Section Group и индекса текста, а также простой запрос.

Сначала создается таблица. Для идентификации каждого документа используется первичный ключ. XML-документ определяется как *большой символьный объект* (Oracle Character Large Object, *CLOB).* Его значение состоит из символов, принадлежащих к набору символов базы данных Oracle9i. Таблица заполняется документами с информацией о книгах, число которых может достигать десятков тысяч.

Затем создается группа XML_SECTION_GROUP с двумя секциями зоны для тегов <author> и <title>:

```
ctx_ddl.create_section_group   ('xml_sections', 'XML_SECTION_GROUP');
ctx_ddl.add_zone_section       ('xml_sections', 'titlesec', 'title');
ctx_ddl.add_zone_section       ('xml_sections', 'authorsec', 'author');
```

Эта группа секций отображается на столбец xml_text. Это указывает поисковой программе Oracle Text на то, что данный текст является структурированным XML-документом. Открывающие и закрывающие теги и внутренний DTD распознаются как отдельные элементы. Кроме того, элементы <title> и <author> подготавливаются к индексированию для поиска текста. Затем строится текстовый индекс и подключаются basic_lexer и группа секций:

```
create index xml_index on bookstore (xml_text)
   indextype is ctxsys.context
   parameters ( 'lexer my_basic_lexer SECTION GROUP xml_sections');
```

Оператор *indextype* дает команду СУБД Oracle9i о создании текстового индекса вместо обычного индексного дерева. После создания такого индекса можно отправлять контекстные запросы к базе данных из двух документов с помощью функции contains:

```
select pk from bookstore
   where contains (xml_text, 'LeCarre WITHIN authorsec') > 0 ;
```

По этому запросу будут получены все строки в таблице bookstore, где текстовый столбец содержит текст LeCarre в элементе <author>. Oracle SQL пока не поддерживает возврат *логических переменных,* поэтому содержимое скобок всегда больше нуля. В ответ на этот запрос будет получен ответ:

```
PK
---------
111
```

В следующих разделах рассмотрен индексирующий конвейер на примере приложения Bookstore, иллюстрирующий применение различных методов работы Oracle Text.

Информационные хранилища

Процессом считывания информации из столбцов базы данных управляет информационное хранилище Oracle Text.

Информационное хранилище с прямым доступом к данным является самым простым хранилищем, работающим с данными, содержащимися в индексированных столбцах. Оно не имеет никаких задаваемых пользователем атрибутов. Информационное хранилище с прямым доступом к данным — это DEFAULT_DATASTORE, поэтому его не нужно явным образом подключать к Oracle Text, оно уже существует. В примере Bookstore как раз используется информационное хранилище с прямым доступом к данным.

Файловое информационное хранилище считывает данные столбцов из файла. Оно открывает и считывает содержимое файла и интерпретирует его как индексированный столбец. Ниже приведен пример все того же книжного склада Bookstore, модифицированный для работы с файловым информационным хранилищем:

```
create table bookstore (
   id       number primary key,
   xml_text varchar2(2000)
);
insert into bookstore values (111, 'book1.xml');
insert into bookstore values (112, 'book2.xml');
...
begin
   ctx_ddl.create_preference   ('xml_files', 'FILE_DATASTORE');
   ctx_ddl.set_attribute       ('xml_files', 'PATH','/xml/files');
   ...
end;
...
create index xml_index on bookstore (xml_text)
   indextype is ctxsys.context
   parameters ( 'datastore xml_files lexer my_basic_lexer SECTION
                GROUP xml_sections' );
```

Информационное хранилище URL считывает данные из столбцов как URL-адреса. Поддерживаются HTTP, FTP и файловые протоколы. Чтобы в примере Bookstore книги хранились в виде URL-адресов, код должен выглядеть так:

```
...
begin
   ctx_ddl.create_preference ('xml_urls','URL_DATASTORE') ;
   ctx_ddl . set_attribute   ('xml_urls', 'HTTP_PROXY' , '<your proxy server>');
   ctx_ddl . set_attribute ( ' xml_urls ' , ' Timeout ' , '300');
...
end;
insert into bookstore values (111, 'file://xml/files/book1.xml') ;
insert into bookstore values (112, 'file://xml/files/book2.xml');
...
create index xml_index on bookstore (xml_text)
   indextype is ctxsys . context
   parameters ( 'datastore xml_urls lexer my_basic_lexer SECTION
                GROUP xml_sections ' ) ;
```

Информационное хранилище пользователя построено на базе хранимых процедур, которые и синтезируют документ. Если данные о книгах из примера Bookstore хранятся в виде плоского текста, информационное хранилище пользователя можно использовать для преобразования этих данных в XML-документ. Нижеследующий код показывает, как в данном случае будет выглядеть пример с книжным складом Bookstore:

```
create table bookstore (
   pk       number primary key,
   id       integer,
   author   varchar2 (80),
   title    varchar2(80) ,
   xmltext  clob                        /* placeholder for user datastore */
) ;
insert into bookstore (pk, id, author, title) values (
 111,
 4, 'John LeCarre', 'The Spy Who Came in from the Cold');
insert into bookstore (pk, id, author, title) values (
 112,
 5, 'John Grisham' , 'The Client');
...

/* the stored procedure */
create procedure toXML (rid in rowid, tlob in out tlob)
   offset number :=1;
```

```
begin
   for book in (select id,author,title,xmltext from bookstore
                where rowid = rid)
   loop
      synthesize_XML (tlob, book.id, book.author, book.title, offset);
      dmbs_lob.append(tlob,book.xmltext);
   end loop;
end toXML;

grant execute on toXML to public;
connect xml/bookstore

begin
   ctx_ddl.create_preference   ('xml_proc', 'user_datastore') ;
   ctx_ddl.set_attribute       ('xml_proc', 'procedure','toXML' );
   ...
end;
...
create index xml_index on bookstore (xml_text)
   indextype is ctxsys.context
   parameters ('datastore xml_proc lexer my_basic_lexer SECTION
               GROUP xml_sections') ;
```

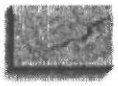
Здесь не показана вспомогательная процедура synthesize_XML. Она берет данные о книгах и помещает их во временный XML-документ *(tlob)*.

```
'<Book><Id>' I I book.id I I '</Id><Author>' I I book.author I I
        '</Author><Title>' I I book.title '</Title></Book>'
```

Затем tlob копируется в bookstore xml_text. Потом идет оставшаяся часть SQL-запроса, почти идентичная предыдущим версиям примера Bookstore. Главное отличие этого варианта состоит в создании информационного хранилища пользователя вместо файлового или URL-хранилища.

Секции поля и зоны

Этот же пример Bookstore можно немного модифицировать, чтобы проиллюстрировать его работу с секциями поля и зоны.



Внимание

Здесь рассматриваются только SQL-операторы.

```
...
insert into bookstore values (
   111,
```

```
    ' <Book>
       <Id>4<\Id>
       <Authors>
          <Author1>John Kay</Author1>
          <Author2>Mary Powell</Author2>
       </Authors>
       <Title>Don Juan </Title>
    </Book>'
 );
insert into bookstore values (
    112,
    '<Book>
       <Id>5<\Id>
       <Authors>
          <Author1>Juan Smith</Author1>
       </Authors>
       <Title>One Fine Day </Title>
    </Book>'
);
...
begin
    ...
    ctx_ddl.add_zone_section('xml_sections' , 'authors', 'authors');
    ctx_ddl.add_zone_section( 'xml_sections', 'authorsec', 'author1');
    ctx_ddl . add_zone_section ( ' xml_sections ' , ' authorsec2 ' , ' author2 ' ) ;
    ...
end;

create index xml_index on bookstore (xml_text)
    indextype is ctxsys . context
    parameters ( 'SECTION GROUP xml_sections ...');
```

По запросу:

```
select pk from bookstore where
    contains (xml_text, 'Mary within authorsec') > 0;
```

мы получим этот документ. Хотя Магу и является автором, но ее имени нет в секции **'authorsec'**. В отличие от первоначального варианта примера Bookstore секции **<author1> и <author2>** отображаются на одно и то же имя секции:

```
begin
    ...
    ctx_ddl.add_zone_section ( 'xml_sections', 'authorsec', 'author1');
    ctx_ddl.add_zone_section ('xml_sections', ' authorsec2 ' , 'author2');
    ...
end;
```

По этому запросу находится первый документ, поскольку и **<author1>**, **и <author2>** интерпретируются как **'authorsec'**.

Как уже указывалось в этой главе, каждый экземпляр секции зоны интерпретируется отдельно. Другими словами, по запросу типа:

```
select pk from bookstore where
   contains (xml_text, '(Mary and Powell) within authorsec') > 0;
```

этот документ находится, а по следующему запросу — нет:

```
select pk from bookstore where
   contains (xml_text, '(Mary and John) within authorsec') > 0
```

Книга из первого документа имеет авторов с именами Mary и John, находящимися в секции **'authorsec'**, но не в одной и той же секции с таким названием. Если и внешнюю секцию **<author>** отобразить на секцию **'authorsec'**:

```
ctx_ddl.add_zone_section ('xml_sections', 'authosec', 'authors');
```

то последний запрос будет успешным, так как оба имени будут находиться во внешней секции **'authosec'**.

Если создателя приложения интересуют только имена и неважно, в какой секции они находятся, данный фрагмент будет выглядеть так:

```
begin
   ...
   ctx_ddl.add_zone_section ('xml_sections', 'authors', 'authors');
   ctx_ddl.add_zone_section ('xml_sections', 'names', 'author1');
   ctx_ddl.add_zone_section ('xml_sections', 'names', 'author2');
   ctx_ddl.add_zone_section ('xml_sections', 'names', 'title');
   ...
end;
```

Теперь по запросу:

```
select pk from bookstore where
   contains (xml_text, 'Juan within names') > 0
```

будут найдены оба документа, так как Juan находится в первом документе **<title>** и во втором **<author1>**. Отметим, что при этой установке делаются различия между именами в **<title>** и именами в **<author1>**. Так получается при использовании вложенного запроса:

```
select pk from bookstore where
   contains (xml_text, '(Juan within names) within authors') > 0;
```

По вложенному запросу находится второй документ, поскольку только он содержит имя Juan в секции **<authors>**. Вложенный запрос работает исключительно

в секциях зон. Сама технология индексации секций зон в Oracle Text такова, что секции с одинаковыми названиями не различаются. Например, на запрос:

```
select pk from bookstore where
   contains (xml_text, '(Juan within authors) within names') > 0;
```

тоже будет получен ответ, потому что границы секций совпадают. Вложенный запрос не подразумевает четких отношений родительского и дочернего элементов. Это могут быть отношения родителя и внука или даже правнука. Как уже отмечалось в этой главе, каждый экземпляр секции поля в документе считается продолжением этой секции. Если в примере Bookstore установить секции полей следующим образом:

```
begin
   ...
   ctx_ddl.add_field_section ('xml_sections', 'authors', 'authors');
   ctx_ddl.add_field_section ('xml_sections', 'authorsec', 'author1');
   ctx_ddl.add_field_section ('xml_sections', 'authorsec', 'author2');
   ...
end;
```

то на запрос:

```
select pk from bookstore where
   contains (xml_text, '(Mary and John) within authorsec') > 0
```

документ будет найден, кроме случая, когда секция зоны будет **'authorsec'**. Для каждого документа программа разбиения на секции объединяет экземпляры поля секции. Лучше, когда секции поля имеют вид **<title>,** т. е. не повторяются. Они извлекаются из содержимого документа и индексируются отдельно.

Это означает, что вложенный запрос:

```
select pk from bookstore where
   contains (xml_text, '4') > 0
```

не сможет найти документ, даже если он содержит **'4'**. Но если бы использовались секции зон, запрос был бы успешным. Ситуацию можно изменить, сделав секцию поля видимой, т. е. установив значение четвертого необязательного *логического* аргумента **add_field_section** равным TRUE (истина):

```
ctx_ddl.add_field_section ('xml_sections', 'authorsec', 'author1', TRUE);
```

Теперь содержимое секции поля будет видимым для невложенных запросов. Это можно осуществить путем двойной индексации слова, один раз — как части извлеченной секции, а другой раз — как части документа. Конечно, в таком случае индекс занимает больше места, чем обычно.

Внутренние типы документов DOCTYPE анализируются для групп секций XML_SECTION_GROUP и PATH_SECTION_GROUP. В этом случае создаются поля секций, ограниченные данным документом, чтобы можно было различить теги

из разных DOCTYPE. Синтаксис тегов в такой ситуации должен выглядеть так: (<*doctype_name*>) <*tag_name*>:

```
begin
      ctx_ddl.create_section_group('xml_sections', 'XML_SECTION_GROUP');
      ctx_ddl.add_field_section('xml_sections ','(Bookstore)'authorsec',
                                'author');
      ctx_ddl.add_field_section('xml_section'','(MyBookstore)names',
                                'author');
      ctx_ddl.add_field_section ('xml_section'', 'authorsec',
                                 'author');
   ...
end
```

Теперь, когда программа разбиения на секции видит тег **<author>**, она индексирует его как секцию **'authorsec'**, если DOCTYPE имеет значение **Bookstore,** и как секцию **'names'**, если DOCTYPE имеет значение **MyBookstore**. В группе секций могут сосуществовать обе секции, и ограниченные, и не ограниченные пространством doctype. Ограниченная секция применима только к своему DOCTYPE, а для всех остальных DOCTYPE используется неограниченная секция. На выполнение запросов это никак не влияет, поскольку запрос обрабатывается в секции name.

Секции останова

В группах секций AUTO_SECTION_GROUP и PATH_SECTION_GROUP программа разбиения на секции автоматически индексирует все непустые теги как секцию зоны, и имя такой секции совпадает с именем тега. Но не все теги важны для приложения. Если они не используются в запросах, их индексирование будет пустой тратой времени. Для таких тегов в группах секций AUTO_SECTION_GROUP и PATH_SECTION_GROUP существуют секции останова, указывающие на то, какие теги следует игнорировать. Например, если в примере Bookstore назначить группу секций PATH_SECTION_GROUP:

```
begin
   ctx_ddl.create_section_group ('xml_sections', 'PATH_SECTION_GROUP');
   ctx_ddl.add_stop_section ('xml_section', 'Id')
   ...
end;
```

то тег **<Id>** игнорируется и не будет индексироваться как секция. Количество секций останова в группе может быть любым. Кроме того, поддерживается ограничение данных doctype. Например:

```
ctx_ddl.add_stop_section('xml_section'', '(MyBookstore)author');
```

игнорирует тег **<author>** только внутри DOCTYPE **'MyBookstore'**.

Секции атрибутов

Секции атрибутов в группах XML_SECTION_GROUP и PATH_SECTION_GROUP позволяют проводить индексацию и поиск в рамках значений атрибутов. Кроме того, внутри PATH_SECTION_GROUP в запросах учитывается зависимость от имен атрибутов и тегов. Чтобы использовать секцию атрибутов, пример Bookstore можно видоизменить:

```
insert into bookstore values (
   111,
   '<Book   id       = "4"
            author   = "John LeCarre"
            title    = "The Night Manager* />'
);
insert into bookstore values (
   112,
   '<Book   id       = "5"
            author   = "Juan Smith"
            title    = -The Night Club"/>'
);
 ...
begin
   ctx_ddl.create_section_group('xml_sections','XML_SECTION_GROUP');
   ctx_ddl.add_attr_section('xml_sections','authorsec','book@author');
   ctx_ddl.add_attr_section('xml_sections', 'titlesec', 'book@title');
   ...
end;
create index xml_index on bookstore (xml_text)
   indextype is ctxsys.context
   parameters ( ' SECTION GROUP xml_sections ...');
```

Имя атрибута будет зависеть от имени тега, которому он принадлежит:

<tag_name>@<attribute_name>.

Если в БД Bookstore направляется следующая последовательность запросов:

```
select pk from bookstore where
   contains (xml_text, 'Juan within author') > 0;
select pk from bookstore where
   contains (xml_text, '(The Night Manager) within title') > 0;
select pk from bookstore where
   contains (xml_text, '(Juan and John) within author') > 0;
```

то по первому и второму запросу документы будут найдены, а по третьему — нет. Не обнаружатся они потому, что секции атрибутов в запросах могут отличаться в зависимости от их местонахождения. Отметим, что и первый, и второй запросы

тоже останутся без ответа в секции PATH_SECTION_GROUP, если имена тегов будут написаны прописными буквами:

```
select pk from bookstore where
   contains (xml_text, 'Juan within AUTHOR') > 0;
select pk from bookstore where
   contains (xml_text, '(The Night Manager) within TITLE') > 0;
```

Текст атрибута считается невидимым, поэтому по не вложенному запросу, например:

```
select pk from bookstore where
   contains (xml_text, 'Juan') > 0;
```

никакой документ найден не будет. Имена секций атрибутов не должны совпадать с именами секций зоны и файла. Причем разрешается отображать несколько имен атрибутов на одно имя секции. При синтаксическом разборе DTD определенные в нем значения атрибутов по умолчанию не поддерживаются. Другими словами, если в документе пропущено действительное значение атрибута, то при индексировании значение по умолчанию, определенное для данного DTD, не вставляется.

В группах секций AUTO_SECTION_GROUP и PATH_SECTION_GROUP программа разбиения на секции автоматически индексирует значения атрибутов как секции атрибутов с именем *<tag>@<attr>*. Например, запрос:

```
select pk from bookstore where
contains (xml_text, '(The Night Manager) within title') > 0;
```

ведет поиск внутри области значений атрибутов заголовка, поскольку атрибут заголовка автоматически индексируется как секция атрибута.

Поиск XPATH внутри группы PATH_SECTION_GROUP

В СУБД Oracle9i впервые были введены операторы INPATH() и HASPATH(), позволяющие вести поиск в группе PATH_SECTION_GROUP по XPATH-подобным запросам. Например, вместо предыдущего примера, где велся поиск в секции "title", можно составить запрос с XPATH-выражением для поиска слова "Night" в одной из дочерних секций "book" хранилища bookstore, т. е. в секции "title":

```
select pk from bookstore where
   contains (xml_text, 'Night within INPATH(//bookstore/book)') > 0;
```

Разрешив подставлять выражения типа XPATH вместо имен секций, мы расширили возможности оператора WITHIN в Oracle9i. Кроме того, теперь разрешены и проверки атрибутов, и если слово "Night" встречается в нескольких секциях "title", но "title" имеет атрибуты "number" для поиска, то запрос на заголовки можно составить с проверкой соответствия атрибутов:

```
select pk from bookstore where
   contains (xml_text, 'Night within INPATH(//bookstore/book)
title[@number="one"]') > 0 ;
```

Подобно тому как оператор INPATH() расширил возможности оператора WITHIN, оператор HASPATH() делает то же с оператором CONTAINS: он разрешает вести поиск в XML-документах с помощью выражений типа XPATH. Если в документе есть заданный путь path, то в ответ на запрос HASPATH() получим значение 100. Например, в примере bookstore получим в ответ 100, составив запрос следующим образом:

```
select pk from bookstore where
   contains(xml_text, 'HASPATH(bookstore/book/title') > 0;
```

Оператор HASPATH() может искать в XML-документах элементы с определенным значением. Например, по запросу, указывающему, что заголовок должен иметь значение The Night Manager', будет получено значение 100 для всех документов, в которых встречается такой заголовок. Этот запрос должен выглядеть так:

```
select pk from bookstore where
contains(xml_text, 'HASPATH(Book@Title="TheNight Manager"]) ') > 0;
```

Секция динамических добавлений

Если в хранилище поступают документы с новыми тегами или с новыми DOCTYPE, очень удобно добавлять новые секции в уже существующий индекс без его переконструирования. Например, вот SQL-последовательность, которая добавляет новую секцию зоны с именем "names", использующую тег <title>:

```
alter index xml_index rebuild
   parameters ('add zone section names tag title')
```

А эта SQL-последовательность добавляет секцию поля с именем "authors", которая будет использовать тег <author>:

```
alter index xml_index rebuild
   parameters ('add field section authors tag author')
```

Секция динамических добавлений модифицирует только метаданные индекса и не перестраивает сам индекс. То есть эти секции действуют на любой документ, проиндексированный после данной операции, и не действуют ни на какие уже существующие документы. Если в индексе уже есть документы с этими секциями, их нужно вручную промаркировать для повторного индексирования (обычно с помощью команды обновления индексированного столбца).

Внимание

Текущая версия Oracle Text не поддерживает поиск и извлечение фрагментов XML-документа, которые удовлетворяют условиям запроса. Другими словами, перед *проведением поиска нужно разбить большой XML-документ на несколько меньших по размерам значимых документов. Но в будущем, конечно, появятся более интеллектуальные операторы, которые смогут вести контекстный поиск значимых фрагментов во всем XML-документе, содержащем десятки тысяч элементов* **<book>**.

Глава 7

Службы электронного бизнеса Oracle E-Business XML Services

акет приложений Oracle E-Business Suite предлагает компаниям новый усовершенствованный способ ведения бизнеса. В пакете E-Business Suite имеются средства управления взаимодействием с клиентами Customer Relationship Management (CRM), системой поставок и внутренними операциями в виде интегрированного решения. Компонент Oracle E-Business XML Services, который является частью Oracle CRM и входит в состав Oracle Applications версии 11.5.6, предоставляет базу и системную инфраструктуру для разворачивания, управления и исполнения в реальном времени служб XML Services. Он также является основой для разработки приложений электронного бизнеса нового поколения, функционирующих в Web.

Обзор служб XML Services

Компонент XML Services — новая технология, позволяющая CRM-разработчикам обобщать функциональные возможности CRM-приложений, чтобы представить их в виде XML-служб, доступных из любого авторизованного источника с помощью XML-сообщений, передаваемых по протоколу SOAP. Эти приложения создаются для того, чтобы следующим образом использовать достоинства XML-служб:

- Каждое приложение создает и публикует некий набор XML-служб. Обычно такие приложения проявляют основные достоинства интеграции именно в виде XML-служб.
- Каждое приложение создает и публикует некий набор событий.
- Приложения вызывают службы, опубликованные другими приложениями.
- Приложения абонируют все события, которые они желают отслеживать и обрабатывать.
- Приложения сигнализируют о произошедшем событии компоненту XML Services, а он затем вызывает на исполнение все абонированные службы, которые должны обрабатывать это событие.

Каркас XML Services до определенной степени абстрактная вещь, которая скрывает детали, связанные с передачей сообщений по протоколу SOAP, и позволяет разработчикам сосредоточиться на разработке служб, построенных на базе технологии Java. Этот каркас также определяет модель авторизации для пользователей, которые получают доступ к службам.

Компоненты XML Services

XML Services имеют пять основных компонентов.

Интерфейс администратора

С помощью консоли управления CRM Administrative Console разработчики CRM-приложений, системные интеграторы и разработчики Oracle-приложений могут создавать, удалять и конфигурировать службы, точки вызова, события и подписчиков этих событий.

SOAP Server

Движок реального времени SOAP Server отвечает за исполнение всех запросов на службы, которые развернуты в репозитории XML Services. Серверный движок SOAP реализован в виде сервлета, который обрабатывает все поступающие SOAP XML-запросы. SOAP-сервлет транслирует входящие SOAP XML-запросы в вызовы Java-методов. После исполнения запрошенного метода полученный Java-объект кодируется как SOAP XML-отклик и отсылается обратно, в то место, откуда пришел вызов.

Клиентские API-интерфейсы

Разработчики используют клиентские API-интерфейсы XML Services для вызова XML-служб и подачи сигнала о наступлении событий. API-интерфейс события используется для публикации относящейся к данному событию информации для всех подписавшихся на это событие. Можно также публиковать информацию о событии для целого подмножества подписчиков, выбранных на основе информации, связанной с этим событием.

Репозитории служб и событий

XML Services поддерживают работу центрального репозитория, в котором хранится полная информация о всех зарегистрированных службах, точках вызова, событиях и их подписчиках. Он используется движком реального времени XML Services и управляется с помощью интерфейса администрирования XML Services.

Терминология

При описании основных функций XML Services используются следующие термины.

XML-служба

XML-служба — это единая логическая единица работы, имеющая четко определенные входные и выходные данные. XML-службу можно вызвать из любого авторизованного приложения, отправив соответствующее XML-сообщение и получив в качестве отклика другое XML-сообщение. Каждая XML-служба реализуется в виде Java-метода. XML-службы специфичны для каждого конкретного приложения. Как правило, они представляют собой четко определенные блоки бизнес-логики.

Web Service

Web Service — это XML-служба, которая развернута в системе World Wide Web.

Группа служб

Группа служб — это группа, связанная с XML-службами. Каждая группа служб реализована как Java-класс. Все открытые методы в этом Java-классе могут быть службами, принадлежащими к этой группе служб.

Основная точка интеграции

Основная точка интеграции отображает функциональные возможности приложения, доступные через Интернет посредством XML-службы. Обычно основные точки интеграции реализуются в виде служб, демонстрирующих все бизнес-функции, которые приложение предлагает для использования другим приложениям. Основная точка интеграции — это бизнес-интерфейс API, который используется для программного запуска функций приложения.

Точка вызова

Точка вызова — то место приложения, где последнее вызывает XML-службы, предлагаемые другим приложением. Клиентский API-интерфейс XML Services отделяет логику приложения от деталей вызова службы, как то: где именно развернута данная служба, какой протокол используется для передачи данных, каковы параметры аутентификации для доступа к службе. Для вызова службы приложение использует имя логической службы, указывающее на запись о вызове, в которой есть вся вышеупомянутая информация. Даже если провайдер службы поменяется, в точке вызова не придется делать никаких изменений.

Запись вызова

В *записи вызова* хранится вся информация, необходимая для вызова данной службы. В этой записи указывается URL-адрес, где развернута данная служба, имя группы служб, имя службы, идентификатор пользователя и пароль. В этой записи

также хранится подробная информация о том, какие типы параметров направляются в данную службу, какое отображение типа используется для каждого параметра и т. п. Каждая запись вызова идентифицируется уникальным образом своей строкой вызова. Эта строка также называется логическим именем службы. Когда программа приложения использует для вызова службы API-интерфейс XML Services, приложение направляет логическое имя службы на службу, которая вызывается в качестве первого параметра. Эта строка используется в режиме реального времени для возвращения записи вызова, в которой полностью определена процедура вызова данной службы. Поэтому запись вызова является инструментальным средством при динамическом связывании, используемом клиентом XML-службы при вызове XML-служб. Благодаря использованию динамического связывания и логического имени службы как кода указателя на запись вызова код приложения может сохраняться даже при изменении важных деталей вызова, например, если меняется провайдер данной службы и права доступа к нему.

Событие

Событие — это четко определенный Java-объект, который можно синтезировать в конкретной точке приложения при выполнении определенных условий. Все события регистрируются в репозитории событий XML Services и доступны тем, кто на него подписан. В XML Services каждое событие имеет уникальное имя. Для подачи сигнала компоненту XML Services о совершении данного события приложения используют вызов API-интерфейса XML Services. С его помощью приложение передает имя события и его объект в инфраструктуру XML Services. Затем инфраструктура XML Services обрабатывает задачу по вызову всех текущих подписчиков данного события и передает объект события каждому из них.

Подписчик события

Подписчик события работает по отношению к данному событию как слушающая программа. Подписчики события реализуются в виде XML-служб. Каждая служба подписчика события — это обычная XML-служба, имеющая один входной параметр. Этот параметр принадлежит к такому же Java-типу, как и объект события, который будет обрабатываться. При наступлении какого-то события и передаче сигнала об этом исходным приложением будут вызваны все XML-службы, являющиеся подписчиками данного события, после чего объект события будет передан в виде входного параметра. Следует отметить, что XML-служба подписчика не обязательно должна иметь возвращаемое значение.

XML-службы и протокол SOAP

XML-службы построены на базе протокола SOAP. Поэтому для разработки, разворачивания и использования XML-служб следует понимать, что такое SOAP.

Что такое SOAP

Simple Object Access Protocol (SOAP) — это упрощенный протокол для обмена информацией в децентрализованной распределенной среде, построенный на базе технологии XML. Протокол SOAP состоит из трех частей:

- SOAP Envelope, который определяет общую схему того, что находится в сообщении, кто должен его обрабатывать и является ли эта обработка обязательной.
- Набор правил кодирования для отображения экземпляров определяемых приложением типов данных. Эти правила определяют механизм преобразования типов данных приложения в XML и обратно.
- Соглашение SOAP RPC, которое определяет удаленные вызовы процедуры и отклики на них.

Основное преимущество протокола SOAP — его простота и расширяемость. SOAP имеет более слабую связь между клиентом и сервером, чем аналогичные протоколы для распределенной компьютерной обработки, например CORBA/IIOP. Все это придает протоколу SOAP дополнительные достоинства. SOAP не зависит от транспортного протокола, и его можно использовать с любым транспортным протоколом. В то же время протокол SOAP используется в качестве стандарта де-факто для доставки содержимого программы, когда для удаленного вызова службы через Интернет применяется протокол HTTP. В настоящее время спецификация SOAP 1.1 принята к рассмотрению консорциумом W3C, а для выработки стандарта, который заменит протокол SOAP, создана специальная рабочая группа W3C XML Protocol Working Group.

Поскольку протокол SOAP построен на базе технологии XML, он не зависит ни от платформы, ни от операционной системы. Он поддерживает связь между клиентом и сервером, которые используют разные языки программирования. Сгенерировать SOAP-запрос очень просто, и такие запросы удобны для обработки клиентом. С помощью протокола SOAP одно приложение может становиться программным клиентом для слу кб другого приложения, при этом два приложения могут обмениваться полноценной структурированной информацией. Протокол SOAP предоставляет разработчикам устойчивую модель программирования, позволяющую объединять мощные распределенные Web-службы, чтобы превратить Интернет в платформу разработки приложений будущего.

Как работает протокол SOAP

Спецификация SOAP описывает стандартный, основанный на технологии XML способ кодирования запросов и ответов на них, в том числе:

- Запросов на вызов метода как службы, включенной в параметры
- Ошибок, передаваемых службой

- Откликов на метод службы, в том числе возвращаемого значения и выходных параметров

. Рассмотрим пример. SOAP-запрос GetLastTradePrice отправлен в службу Stock-Quote. Этот запрос берет строковый параметр (код ценных бумаг данной компании) и возвращает плавающий курс акций. SOAP-сообщением является XML-документ. Для устранения неоднозначности SOAP-идентификаторов и идентификаторов, специфичных для данного приложения, используются пространства имен XML. В этом примере в качестве транспортного протокола используется протокол HTTP. Правила, управляющие форматом XML-содержимого в .SOAP-запросе, совершенно не зависят от того, что содержимое передается по протоколу HTTP (поскольку SOAP не зависит от транспортного протокола). Сообщение с SOAP-запросом, встроенное в HTTP-запрос, выглядит следующим образом:

```
POST /StockQuote HTTP/1.1
Host: www.stockquoterserver.com
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn
SOAPAction: "Some-URI"
<SOAP-ENV: Envelope xmlns:SOAP- ENV="http://schemas.xmlsoap.org/soap/
envelope/" SOAP-
ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
<SOAP-ENV:Body>
<m:GetLastTradePrice xmlns:m="Some-URI">
<symbol>ORCL</symbol>
</m:GetLastTradePrice>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Далее приведено ответное HTTP-сообщение, содержащее XML-сообщение, в котором в качестве полезной нагрузки имеется SOAP:

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn

<SOAP-ENV: Envelope xmlns:SOAP-
ENV=http://schemas.xmlsoap.org/soap/envelope/ SOAP-
ENV: encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
<SOAP-ENV:Body>
<m:GetLastTradePrice xmlns:m="Some-URI">
<Price>34.5</Price>
</m:GetLastTradePrice Response>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Что делает SOAP-клиент?

SOAP-клиент должен выполнять следующие операции:

- Собирать все параметры, необходимые для вызова службы.
- Создавать сообщение с SOAP-запросом к службе. Это XML-сообщение, которое построено в соответствии с протоколом SOAP и содержит все значения всех входных параметров, закодированных в XML-формате. Этот процесс называется *преобразованием параметров в последовательную форму* (serialization).
- Отправлять запрос на SOAP-сервер по одному из протоколов, поддерживаемых этим сервером.
- Получать сообщение с SOAP-ответом.
- Определять, успешно ли выполнен запрос. Для этого используется элемент SOAP Fault.
- Преобразовывать возвращенный параметр из XML-формата к исходному типу данных. Этот процесс называется *преобразованием из последовательной формы в параллельную* (deserialization).
- В случае необходимости использовать полученный результат.

Чтобы избавить пользователя от работы с XML и SOAP на самом нижнем уровне, разработано так много SOAP-клиентов, что все они никогда не потребуются. Для упрощения разработки приложений и избавления разработчиков от необходимости изучать все тонкости работы SOAP в XML Services включен API-интерфейс SOAP-клиента. Этот API-интерфейс предоставляет простой способ вызова SOAP-служб из оболочки XML Services. API-интерфейс SOAP-клиента XML Services поддерживает для запросов и откликов модель синхронного вызова.

Что делает SOAP-сервер!

Любой SOAP-сервер при выполнении запроса на SOAP-службу выполняет следующие шаги:

- Сервер получает запрос на службу.
- После синтаксического разбора XML-запроса сервер должен решить, выполнять ли это сообщение или отбросить его.
- Если сообщение принято *к* исполнению, сервер должен определить, существует ли запрашиваемая служба.
- Сервер преобразует все входные параметры из XML-формата в типы данных, понятные данной службе.

- Сервер вызывает службу.
- Возвращаемый параметр преобразуется в формат XML, и генерируется ответное SOAP-сообщение.
- Ответное сообщение направляется обратно в вызывающую программу.

XML Services использует Oracle-версию SOAP, которая входит в состав сервера Oracle iAS 1.0.2.2 в качестве SOAP-процессора реального времени. Следует отметить, что Oracle-версия SOAP построена на базе Apache-версии SOAP.

Поэтому XML-службы являются SOAP-службами, и их можно вызывать из любого SOAP-совместимого Apache-клиента.

Руководство по группам служб

Для объединения служб в группы служб нужно помнить следующие правила:

- Логически связанные службы, охватывающие специфические области бизнеса, следует организовывать в группу служб.
- Группа служб используется в рамках какого-то приложения.
- С точки зрения реализации группа служб — это Java-класс, и все его открытые методы можно использовать в качестве служб. Хотя все открытые методы Java-класса, представляющего собой группу служб, можно использовать в качестве XML-служб, с управляющей консоли эти службы можно отключать или делать недоступными.
- Поскольку все открытые методы являются кандидатами в службы, разработчику приложения следует построить классы групп служб таким образом, чтобы все открытые методы соответствовали разрешенным службам.
- Группа служб представляет собой Java-класс, однако следует помнить, что метод в этом Java-классе нужно исполнять в рамках RPC-модели, для которой подходят не все объектно-ориентированные концепции. Ниже представлен список *запрещенных* действий:
  - Метод конструктора: Java-класс не должен иметь конструктора.
  - Перекрытие методов: одной службе должен соответствовать только один метод.
  - Наследование классов: Java-класс (группа служб) не должен наследоваться из других классов, за исключением класса **java.lang.Object**. В качестве служб могут использоваться только методы, объявленные в данном Java-классе (а не в его родительских классах).

- Реализация группы служб (Java-класса) не должна порождать *подпроцессов* и менять *состояние* группы служб.
- Регистрация (или разворачивание) группы служб производится через E-Business Center.

Руководство по службам

Здесь приведен список проблем, которые разработчику приложения нужно решить, прежде чем он займется реализацией отдельных функций этого приложения в виде служб. Следует придерживаться правил:

- Разработчик приложения должен сосредоточиться только на бизнес-логике службы. Он не должен беспокоиться о том, имеет ли вызывающая программа права на вызов данной службы или нет. Проверки на аутентификацию и авторизацию перед вызовом службы будет производить оболочка XML Services.
- В настоящее время инфраструктура XML Services поддерживает только синхронный вызов служб, что следует учитывать при их создании.
- Для параметров и возвращаемых значений службы (метода) разработчикам настоятельно рекомендуется использовать следующие Java-типы:
  - **Java-примитивы** void, int, log, short, byte, float, double, boolean (особо отметим, что примитив char и примитивные объекты типа java.lang.Integer не поддерживаются)
  - Java-массив Только одномерный
  - Другие java.lang.String, java.util.Vector, java.util.Hashtable и org.w3c.dom.Element
  - Специальные типы Любой определенный пользователем Java-тип, чей класс является модулем Javabean, в котором должны быть:
    - Методы установки для всех его полей
    - Методы сбора для всех его полей
    - Конструктор, не принимающий параметров

При исполнении службы или при ее вызове нужно производить преобразования Java-типов в XML (с помощью SOAP-кодирования) и обратно. Инфраструктура XML Services по умолчанию поддерживает трансляцию Java-типов в примитивы, массивы и другие описанные выше типы. В случае использования специальных типов

(описанных выше) инфраструктура XML Services позволяет определить информацию о трансляции произвольных типов данных путем регистрации Java-классов как методов преобразования параллельного кода в последовательный и последовательного кода в параллельный. Если класс является модулем Javabean, используются значения по умолчанию из класса BeanSerializer. Поэтому разработчику приложения не придется иметь дело с различными тонкостями преобразования Java-типов в XML и обратно. Настоятельно рекомендуем использовать для специальных тиров значения по умолчанию, предоставляемые оболочкой XML Services.

Внимание

Для типов данных, вложенных в такие сложные структуры, как массивы, Vector, Hashtable и JavaBean, применимы те же правила, за исключением объектов типа org.w3c.dom.Element, которые не могут быть вложены в какую-то другую структуру.

Если разработчик приложения собирается использовать типы данных, отсутствующие в вышеприведенном списке, или при наличии специальных Java-типов он хочет определить данные для трансляции, отличные от значений по умолчанию, предоставляемых оболочкой XML Services, для регистрации службы ему потребуется следующая информация:

- **Класс параллельно-последовательного преобразователя** Java-класс, реализующий трансляцию Java-типа данных в SOAP.
- **Класс последовательно-параллельного преобразователя** Java-класс, реализующий трансляцию SOAP в Java-тип данных.
- **Стиль кодирования** Строка, которая используется в качестве имени для идентификации совокупности параллельно-последовательного и последовательно-параллельного преобразователей, используемой для выбранного типа данных. Если у разработчика есть несколько наборов упомянутых преобразователей для одного и того же типа данных, для их идентификации используются разные стили кодирования.
- **Пространство имен URI** Это строка, которая используется в качестве XML-пространства имен в SOAP-сообщении. В отправляемом или принимаемом SOAP-сообщении каждый тип данных будет транслирован в пару XML-тегов, и для каждой такой пары тегов в качестве пространства имен будет использоваться пространство имен URI.

Внимание

Напомним: во избежание всяких сложностей настоятельно рекомендуем разработчикам выбирать типы данных из приведенного выше списка либо использовать для трансляции значения, предлагаемые по умолчанию.

Модель системы безопасности

Инфраструктура XML Services использует текущую модель системы безопасности из E-Business Foundation Services.

Другими словами, в XML Services используется аутентификация по имени пользователя и паролю, а авторизация основана на разрешениях и ролевых именах, присвоенных каждому пользователю. У каждой службы есть свое разрешение, которое создается автоматически, когда группа служб, содержащая данную службу, разворачивается в инфраструктуре. Имя разрешения должно иметь формат:

<Сокращенное название приложения>_<Имя группы служб>_
<Имя службы>_EXE — все заглавными буквами.

Новым будет следующее:

- Новый определяемый пользователем тип называется "XML_Users". Администраторы должны создавать пользователей, исходя из типа пользователя для доступа к службам инфраструктуры.
- Значимые ролевые имена по умолчанию, в которые группируются разрешения для имеющихся служб.
- Для каждого ролевого имени по умолчанию создается регистрационная запись.

Администраторы могут изменять пользовательский тип "XML_Users", добавляя в него новые ролевые имена и регистрационные записи. Новые пользователи этого типа получат дополнительные ролевые имена и выбранные регистрационные записи. Обычно они являются внешними пользователями (из другой компании), которые получают доступ в информационное хранилище через Интернет.

Подробности исполнения службы

Для того чтобы перед исполнением службы была проведена надлежащая аутентификация и авторизация, необходимо выполнить следующие действия:

- Сразу после получения SOAP-запроса проводится аутентификация пользователя. Она производится по имени пользователя и паролю, которые содержатся в запросе.
- Перед выполнением службы создается JTF-сеанс пользователя, а после завершения исполнения службы он закрывается.
- После создания сеанса пользователя и перед исполнением службы производится авторизация пользователя.

- Все службы исполняются синхронно.
- Для защиты транспортного канала, по которому передается ответ, генерируемый службой, используется протокол SSL (HTTPS).

Руководство по вызову служб

API-интерфейс клиента XML Services предназначен для отделения вызова службы от всех остальных деталей, определяющих эту службу. Для вызова службы используется один Java-метод. Разработчик приложения должен действовать следующим образом:

- Определить, какие входные параметры нужно отправить службе для ее запуска.
- Определить возвращаемое службой значение и все возможные ошибки.
- Определить логическое имя каждого вызова службы. Оно должно быть уникальным для всей системы, чтобы идентифицировать конкретное обращение к службе. Задать логическое имя нужно для того, чтобы разработчик при написании кода мог абстрагироваться от конкретных характеристик конечной точки службы. Эти характеристики можно сконфигурировать на консоли управления CRM Administrative Console. Поэтому логическое имя динамически связывается с характеристиками конечной точки во время исполнения программы.
  Это следующие характеристики:
  - URL-адрес целевого приложения
  - Имя группы служб целевой службы
  - Имя службы
  - Данные, удостоверяющие личность и полномочия пользователя, запрашиваемые целевым приложением
  - Имя, тип данных и стиль кодирования для входных параметров и возвращаемого значения (имена должны быть уникальными)
  - Информация о типе трансляции параметров и возвращаемого значения целевой службы
- Для вызова системной службы нужно определить имя группы служб в виде 'urn:soap-service-manager', имя службы как имя метода системной службы, например 'get_tagret_objects', и данные системы безопасности в виде NO_AUTHENTICATION (так как системные службы являются открытыми).

- Есть два вида служб вызова: LOCALHOST и REMOTE.
  Если целевой URL-адрес определен как 'LOCALHOST', будет применен режим LOCALHOST, в противном случае используется режим REMOTE (даже если сетевые адреса клиента и сервера совпадают).
  - Режим LOCALHOST Если службы работают в том же экземпляре системы, что и клиент, последний может работать в режиме LOCALHOST. Это означает, что запрос к службе не проходит по сети, а непосредственно транслируется в вызов Java API-интерфейса. В этом режиме не требуется определять данные для системы безопасности. Также не нужно вводить информацию о типе трансляции.
  - Режим REMOTE Второй способ запуска клиента — работа в режиме удаленного хоста. Запрос к службе будет направлен по сети на тот URL-адрес, где созданы XML-службы. Для вызова службы в этом режиме перед регистрацией вызова нужно создать *профиль аутентификации*. Кроме того, для этого режима нужно создать данные системы безопасности. Также нужно ввести информацию о типе трансляции параметров, поскольку запрос будет транслироваться в XML/SOAP-формат.
- Для защиты транспортного канала при передаче запроса о вызове службы будет использоваться протокол SSL (HTTPS).

Пример вызова службы

Нижеследующий код представляет собой автономную Java-программу для вызова службы, предоставляющей информацию о курсе акций:

```
import java.util.*;
import oracle.apps.jtf.services.*;                    // has to be imported
import oracle.apps.jtf.services.invocation.*;         // has to be imported
import oracle.apps.jtf.base.session.*;

public class ServiceCallExample {
   /**
    * this example program call a service to get Oracle's stock price
    */
   public static void main(String [] args) {
      try {
         //for stand alone program, a stand alone session has to be started
         // before call services.
```

```
    //for jsp pages or servlets, startRequest ( . . . ) method is used to
    //start session
    ServlerSessionManager.startStandAloneSession("JTF", false,
                                                 "SYSADMIN", "SYSADMIN");
    System.out.println("==================================");
  } catch (Exception e) {
    e.printStackTrace();
  }
  // logical invocation name
  String IN = "oracle.crm.jtf.GET_STOCK_QUOTE";
  //construct all the parameters passed to the service
  Vector params = new Vector();
  try {
    //When using this Param constructor, it is recommended to use
    //"variable.getClass()" instead of "type.class".
    //( in the case of array, java.lang.reflect.Array
    //will not return the correct class of the variable)
    Param parameter = new Param("symbol", symbol.getClass(), "ORCL");
    params.addElement(parameter);
  } catch (ClassNotMatchException e) {
    //This exception is thrown when the parameter type and the value
    //don't match.
    //For example, Param("example", String.class, 3) will cause
    //ClassNotMatchException/
  }
  try{
    //call the service
    ServiceResult result = Client.callService(IN, params);
    if (result==null || result.getFloat()<=0)
       { //service specificerror
         System.out.println ( "ORCL is not a valid stock symbol");
       }
    else { //print out result
           System.out.println ("The current stock price of ORCL is"
                               +result.getFloat());
    }
  }
  catch (InvocationException e) {
    //If a service call fails, an InvocationException is thrown. We can
    //tell what happened from the exception's error code,
    //locate the error and then fix it by either
    // changing code or changing the service
    //invocation configuration in the administration/
```

```
            int errorCode = e.getErrorCode();
            System.out.println(errorCode);
               System.out.println(e.getMessage());
         }
         try{
            System.out.println("======================");
            //close the session
            ServletSessionManager.endStandAloneSession();
         }
         catch (Exception e) {
            e.printStackTrace();
         }
      }
   }
```

Что нужно знать о событиях

Правильное использование событий XML Services позволяет создавать приложения, обладающие самыми современными функциональными возможностями, которые можно использовать для интеграции или настройки работы приложения.

Вот что разработчику следует знать о событиях:

- Объекты событий представляют собой Java-объекты.
- Следует определить объекты событий, чтобы инкапсулировать все данные, которые нужно передать подписчикам событий, когда произойдет данное событие.
- Преобразование типов данных при определении объектов событий производится по тем же правилам, которые используются для параметров службы.
- Сигнал о событии представляет собой асинхронный вызов, а ответы от подписчиков не ожидаются.
- После передачи сигнала о событии приложение сразу же получает функции управления. Система XML Services будет вызывать все абонированные службы в отдельных подпроцессах.

Пример сигнальных событий

Следующий пример иллюстрирует использование API-интерфейса XML Services клиента для сигнала о событии приложения.

```
import java.util.*;
import oracle.apps.jtf.services.*;                    //has to be imported
import oracle.apps.jtf.services.invocation.*;         //has to be imported
import oracle.apps.jtf.base.session.*;

public class EventSignalExample {
   /**
    * This example program signaloracle.crm.oso.NEW_SALES_LEAD_EVENT
    * when a new sales lead comes up.
    * The data type of the event object is LeadInfo which contains
    * information about the new lead like name, address and phone number.
    */
   public static void main(String [] args) {
      try {
         //for stand alone program, a stand alone session has to be
         // started before signal events.
         //for jsp pages or servlets, startRequest(...) method is used to
         //start the session
         ServlerSessionManager.startStandAloneSession("JTF", false,
                                                  "SYSADMIN","SYSADMIN");
         System.out.println("==================================");
      } catch (Exception e) {
         e.printStackTrace();
      }
      String eventName = "oracle.crm.oso.NEW_SALES_LEAD_EVENT";
      //construct the event object
      LeadInfo leadinfo = new LeadInfo("John", "500 Oracle parkway",
                                       "650-506-6789");
      Param param = null;
      try {
         //When using this Param constructor, it is recommended to use
         //"variable.getClass()" instead of "type.class".
         //(in the case of array, java.lang.reflect.Array
         //will not return the correct class of the variable)
         param = new Param("lead", LeadInfo.getClass(), leadInfo);
         params.addElement(parameter);
      } catch (ClassNotMatchException e) {
         //This exception is thrown when the parameter type and the value
         //don't match.
         //For example, Param("example", String.class, 3) wiil cause
         //ClassNotMatchException.
      }
```

```
        try{
            //the event is triggered and the event object is sent to all
            //subscribed listeners
            Client.signalEvent(eventName, param);
        } catch (InvocationException e) {
            //if the event name is not registered, or not configured properly,
            //or the system can not launch threads to call the listeners, an
            //InvocationException is thrown.
            //Because the invocation of a listener is done asynchronously
            //at a spawned new thread, no InvocationException
            //will be thrown even if errors happen in the invocation.
            int errorCode = e.getErrorCode();
            System.out.println(errcode);
            System.out.println(e.getMessage());
        }
        try{
            System.out.println("======================");
            //close the session
            ServletSessionManager.endStandAloneSession();
        }
        catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Пример сигнального события с фильтрацией подписчиков

Следующий пример демонстрирует программу, которая предназначается для информирования подписчиков о каком-то событии. Исходя из характеристик события, программа может сообщить о нем только некоторому подмножеству подписчиков. Например, если приложение занимается распределением объявлений о продаже, уведомления будут рассылаться подписчикам этой службы в соответствии с тем, какие продаваемые товары их интересуют.

```
import java.util.*;
import oracle.apps.jtf.services.*; //has to be imported
import oracle.apps.jtf.services.invocation.*; //has to be imported
import oracle.apps.jtf.base.session.*;
```

```
public class EventSignalFilterExample {
   /**
    * This example program signals an oracle.crm.oso.NEW_SALES_LEAD_EVENT
    * when a new sales lead comes up.
    * The data type of the event object is LeadInfo which contains
    * information about the new lead like name, address and phone number.
    * This event associated with a set of listeners which are configured
    * by admin. In this example, we only want to envoke a subset of the
    * listeners. So we first get all the associated listeners based on the
    * event name, then filter out some listeners based on application-
    * specific logic, then signal the event with the selected list of
    * listeners.
    */
   public static void main(String [] args) {
      try {
         //for stand-alone program, a stand-alone session has to be
         // started
         ServlerSessionManager.startStandAloneSession("JTF", false,
                                                      "SYSADMIN","SYSADMIN");
         System. out.println("==================================");
      } catch (Exception e) {
         e.printStackTrace();
      }
      String eventName = "oracle.crm.oso.NEW_SALES_LEAD_EVENT";
      //construct the event object
      LeadInfo leadInfo = new LeadInfo("John", "500 Oracle parkway",
                                       "650-506-6789");
      Param param = null;
      try {
         //When using this Param constructor, it is recommended to use
         //"variable.getClass()" instead of "type.class".
         //(in the case of array, java.lang.reflect.Array
         //will not return the correct class of the variable)
         param = new Param("lead", LeadInfo.getClass(), leadInfo);
      } catch (ClassNotMatchException e) {
         //This exception is thrown when the parameter type and the value
         //don't match.
      //For example, Param("example", String.class, 3) wiil cause
         //ClassNotMatchException.
      }
      try{
         // get all the listener logical names of this event
         String [] listeners = Client.getListeners (eventName);
         String [] selectedListeners = new String [5];
```

```
            // filter out some listeners based on application-specific logic.
            // put all selected listener logical names in variable
            // "selectedListeners"
            //...
            //the event is triggered and the event object is sent to all
            //selected listeners
            Client.signalEvent(eventName, selectedListeners, param) ;
         } catch (InvocationException e) {
            //if the event name is not registered, or not configured properly,
            //or the system can not launch threads to call the listeners, an
            //InvocationException is thrown.
            //Because the invocation of a listener is done asynchronously
            //at a spawned new thread, no InvocationException
            //will be thrown even if errors happen in the invocation.
            int errorCode = e.getErrorCode();
            System.out.println(errcode);
            System.out.println(e.getMessage());
         }
         try{
            System.out.println("======================");
            //close the session
            ServletSessionManager.endStandAloneSession();
         } catch (Exception e) {
            e.printStackTrace();
         }
      }
   }
```

Службы, являющиеся подписчиками событий

Поскольку подписчик события представляет собой обычную XML-службу и вызывается таким же способом, как любая другая служба, все руководящие указания относительно создания служб применимы и к подписчикам событий.

Разворачивание новой службы

В состав XML Services входит несколько примеров служб. Для иллюстрации всех шагов, которые нужно выполнить для разворачивания и исполнения новой службы, рассмотрим пример адресной книги Address Book.

Вот Java-код для этого примера:

```
package samples.server.addressbook;
import.samples.client.addressbook.*;
import java.util.*;
import java.io.*;
importorg.w3c.dom.*;
import org.apache.soap.util.xml.*;
import oracle.soap.util.xml.XmlUtils;

/**
* See \samples\addressbook\readme for info.
* Samples assume that variable JTFLogFile is set
*
*/

public class Addressbook
{  private static String XML_HOME_VAR = "XML_SERVICE_HOME";
   private static String hashFileName = "hashFile.txt";
   private Hashtable name2AddressTable = new Hashtable();

   private File hashFile = null;

   public AddressBook()
   {  // Load Hashtable file
   hashFile = new File(hashFileName);
   addEntry("John B. Good",
            new Address(123, "Main Street", "Anytown", "NY", 12345,
                        new PhoneNumber(123, "456", "7890")));
   addEntry ("Bob Q. Public",
            new Address(456, "North Whatever", "Notown", "ME", 12424,
                        new PhoneNumber(987, "444", "5566")));
   try {
      FileInputStream istream = new FileInputStream(hashFile);
      ObjectInputStream p = new ObjectInputStream (istream);
      name2AddressTable = (Hashtable) p.readObject();
      istream.close();
   } catch (EOFException e) {
      name2AddressTable = new Hashtable();
   } catch (Exception e) {
      e.printStackTrace();
   }
}
```

```
private void flushHashtable() {
    try {
        FileOutputStream ostream = new FileOutputStream(hashFile);
        ObjectOutputStream p2 = new ObjectOutputStream(ostream);
        p2.writeObject(name2AddressTable);
        p2.flush();
        ostream.close();
    } catch (Exception e) {
        e.printStackTrace();
    }
}

public void addEntry(String name, Address address)
{ // Put it in hashtable in memory
    name2AddressTable.put(name, address);
    flushHashtable();
}

public Address getAddressFromName(String name)
throws IllegalArgumentException
{ if (name == null)
    { throw new
        IllegalArgumentException("The name argument must not be "+"null");
    }
    return (Address)name2AddressTable.get.(name);
}

public Element getAllListings 0
{ Document doc = null;
    try {
        doc = XmlUtils.createDocument();
    } catch (Exception e) {
        e.printStackTrace();
        System.err.println("Error in creating xml document.");
        return null;
    }

    Element bookEl = doc.createElement("AddressBook");
    bookEl.appendChild(doc.createTextNode("\n"));
    for (Enumeration keys = name2AddressTable.keys();
         keyshasMoreElements();)
    { String name = (String)keys.nextElement();
      Address address = (Address)name2AddressTable.get(name);
      Element listingEl = doc.createElement("Listing");
      Element nameEl=doc.createElement("Name");
```

```
nameEl.appendChild(doc.createTextNode(name));
listingEl.appendChild(doc.createTextNode("\n    "));
listingEl.appendChild(nameEl);
listingEl.appendChild(doc.createTextNode("\n    • "));
Element addressEl = doc.createElement("Address");
Element sreetNumEl = doc.createElement("StreetNum");
sreetNumEl.appendChild(doc.createTextNode(address.getStreamNum() + ""));
addressEl.appendChild(doc.createTextNode("\n    "));
addressEl.appendChild(sreetNumEl);
addressEl.appendChild(doc.createTextNode("\n    "));
Element sreetNameEl = doc.createElement("StreetName");
sreetNameEl.appendChild(doc.createTextNode(address.getStreamName()));
addressEl.appendChild(sreetNameEl);
addressEl.appendChild(doc.createTextNode("\n    "));
Element cityEl = doc.createElement("City");
cityEl.appendChild(doc.createTextNode(address.getCity()));
addressEl.appendChild(cityEl);
addressEl.appendChild(doc.createTextNode("\n    "));
Element stateEl = doc.createElement("State");
stateEl.appendChild(doc.createTextNode(address.getState()));
addressEl.appendChild(StateEl);
addressEl.appendChild(doc.createTextNode("\n        "));
Element zipEl = doc.createElement("Zip");
zipEl.appendchild (doc. createTextNode (address.getZip() + " " ));
addressEl.appendChild(zipEl);
addressEl.appendChild(doc.createTextNode("\n        "));
PhoneNumber phone = address.get.PhoneNumber();
Element phoneEl = doc.createElement("PhoneNumber");
phoneEl.appendChild(doc.createTextNode("\n        "));
Element areaCodeEl = doc.createElement("AreaCode");
areaCodeEl.appendChild(doc.createTextNode(phone.getAreaCode() + " " ));
phoneEl.appendChild(areaCodeEl);
phoneEl.appendChild(doc.createTextNode("\n        "));
Element exchangeEl = doc.createElement("Exchange");
exchangeEl.appendChild(doc.createTextNode(phone.getExchange()));
phoneEl.appendChild(exchangeEl);
phoneEl.appendChild(doc.createTextNode("\n      "));
Element numberEl = doc.createElement("Number");
numberEl.appendChild(doc.createTextNode(phone.getNumber()));
phoneEl.appendChild(numberEl);
phoneEl.appendChild(doc.createTextNode("\n    "));
addressEl.appendChild(phoneEl);
addressEl.appendChild(doc.createTextNode("\n    "));
```

```
        listingEl.appendChild(addressEl);
        listingEl.appendChild(doc.createTextNode("\n"));
        bookEl.appendChild(doc.createTextNode(" "));
        bookEl.appendChild(listingEl);
        bookEl.appendChild(doc.createTextNode("\n"));
    }
    return bookEl;
}

public int putlistings (Element el)
{  Element listingEl = DOMUtils.getFirstChildElement(el);
   int count = 0;

   while (listingEl !=null)
   {  String name = null;
      int    streetNum = 0;
      String streetName = "";
      String city = "";
      String state = "";
      int    zip = 0;
      int    areaCode = 0;
      String exchange = "";
      String number = "";

      Element tempEl = DOMUtils.getFirstChildElement(listingEl);
      while (tempEl !=null)
      { String tagName = tempEl.getTagName();
        if(tagName.equals("Name"))
          { name = DOMUtils.getCharacterData(tempEl); }
        else if (tagName.equals("Address"))
          {  Element tempEl2 = DOMUtils.getFirstChildElement (tempEl);
             while (tempEl2 !=null)
             { String tagName2 = tempEl2.getTagName();
               String content2 = DOMUtils.getChildCharacterData(tempEl2);
               if(tagName2.equals("StreetNum"))
                 {  streetNum = Integer.parseInt(content2); }
               else if(tagName2.equals("StreetName"))
                 {  streetName = content2; }
               else if (tagName2.equals("City"))
                 {  city = content2; }
               else if (tagName2.equals("State"))
                 {  state = content2; }
               else if(tagName2.equals("Zip"))
                 {  zip = Integer.parseInt(content2); }
```

```
                else if (tagName2.equals("City"))
                    {  city = content2;  }
                else if(tagName2.equals("PhoneNumber"))
                    { Element tempEl3 = DOMUtils.getFirstChildElement(tempEl2);
                      while (tempEl3 !=null)
                        { String tagName3 = tempEl3.getTagName();
                          String content3 =
                                    DOMUtils.getChildCharacterData(tempEl3);
                          if(tagName3.equals("AreaCode"))
                            {   areaCode = Integer.parseInt(content3);  ,}
                          else if (tagName3.equals("Exchange"))
                            {   exchange = content3;  }
                          else if (tagName3.equals("Number"))
                            {   number = content3;  }
                        tempEl3 = DOMUtils.getNextSiblingElement(tempEl3);
                      }
                }
                tempEl2 = DOMUtils.getNextSiblingElement(tempEl2);
              }
            }
            tempEl = DOMUtils.getNextSiblingElement (tempEl);
        }
        if (name ! = null)
            {  Address address = new Address (streetNum, streetName, city, state,
                                              zip, new PhoneNumber(areaCode,
                                                                   exchange,
                                                                   number));
               addEntry(name, address);
               count++;
            }
            listingEl = DOMUtils.getNextSiblingElement (listingEl);
        }
        return count;
    }
}
```

А вот Java-классы для объекта Address, используемого в качестве параметра для методов AddressBook, которые будут выполнять функции XML-служб.

```
public class Address {
    private int            streetNum;
    private String         streetName;
    private String         city;
    private String         state;
```

```
    private int             zip;
    private PhoneNumber     phoneNumber;
    // constructors and get and set methods not shown
}

public class PhoneNumber {
    private int     areaCode;
    private String  exchange;
    private String  number;
    // constructors and get and set methods not shown '
}
```

Для того чтобы использовать методы AddressBook в качестве XML-служб, нужно зарегистрировать новую группу служб в XML-инфраструктуре. Во-первых, следует убедиться в том, что каталог примеров с исходным кодом AddressBook находится в jserv classpath. После компиляции кода AddressBook можно приступать к разворачиванию служб. Предположим, что есть экземпляр Oracle-приложения, а компонент XML Services уже установлен и готов к использованию. Для разворачивания новых служб нужно отправиться в Oracle E-Business Center, как показано на рис. 7.1, и зарегистрироваться в качестве системного администратора.

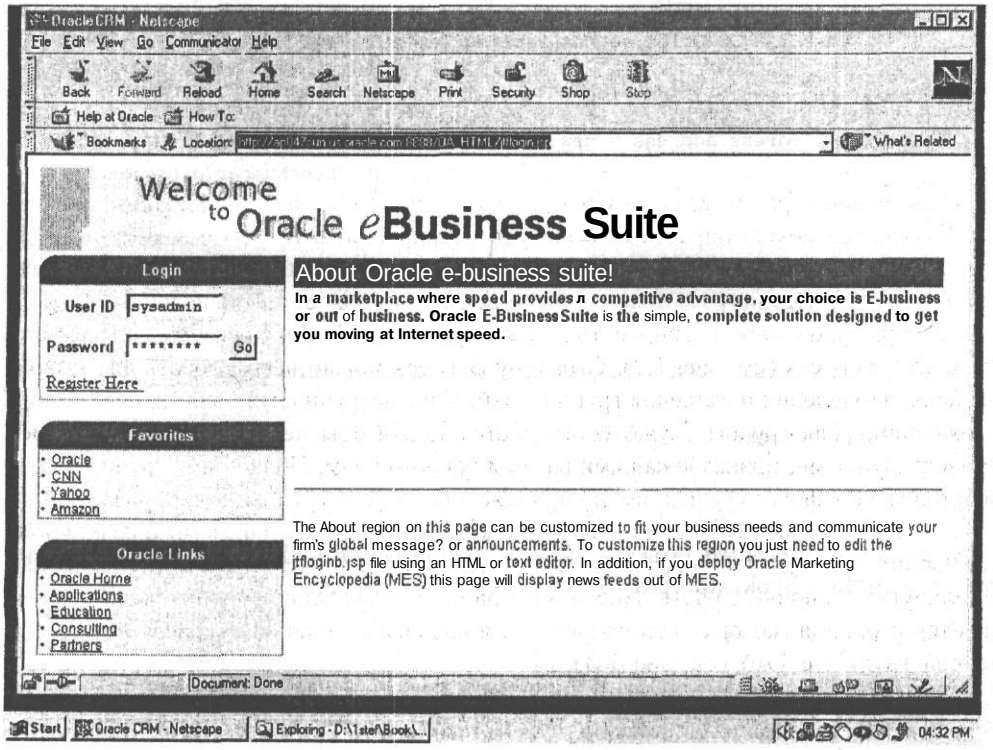


Рис. 7.1. *Страница регистрации Oracle E-Business Suite*

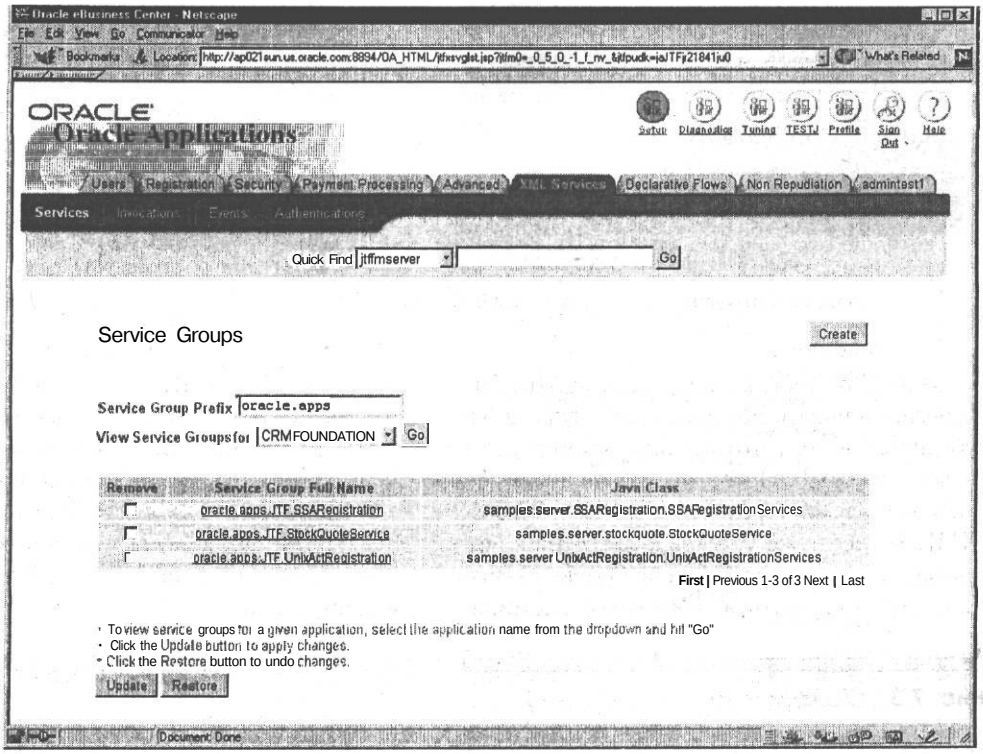


Рис. 7.2. *Страница XML Services*

После регистрации появляется страница, показанная на рис. 7.2. По закладке XML Services можно попасть на страницы управления XML Services. Как видно на рис. 7.2, с этой страницы можно перейти по четырем закладкам в подразделы для работы с службами, вызовами, событиями и службой аутентификации.

Теперь нужно зарегистрировать новую группу служб. Для этого следует выбрать закладку Services (см. рис. 7.2). Страницу Services можно использовать для вывода списка, добавления и удаления групп служб. Список групп служб составляется приложением. Имя группы служб составляется так, чтобы не было конфликта имен между службами, принадлежащими разным приложениям. Полное имя группы служб состоит из префикса группы, имени приложения и имени группы служб, разделенных точками. Для разворачивания новой группы служб нужно щелкнуть мышью по кнопке Create. Для удаления службы или изменения префикса группы служб нужно щелкнуть по кнопке Update. После выбора кнопки создания группы служб открывается страница, которая используется для разворачивания Java-класса в качестве новой группы служб (см. рис. 7.3).

Здесь можно ввести в качестве имени группы служб AddressBook. Тогда с учетом префикса полное имя группы служб будет **oracle.apps.JTF.AddressBook** (JTF — это сокращенное название CRM Foundation). Имя группы используется при создании

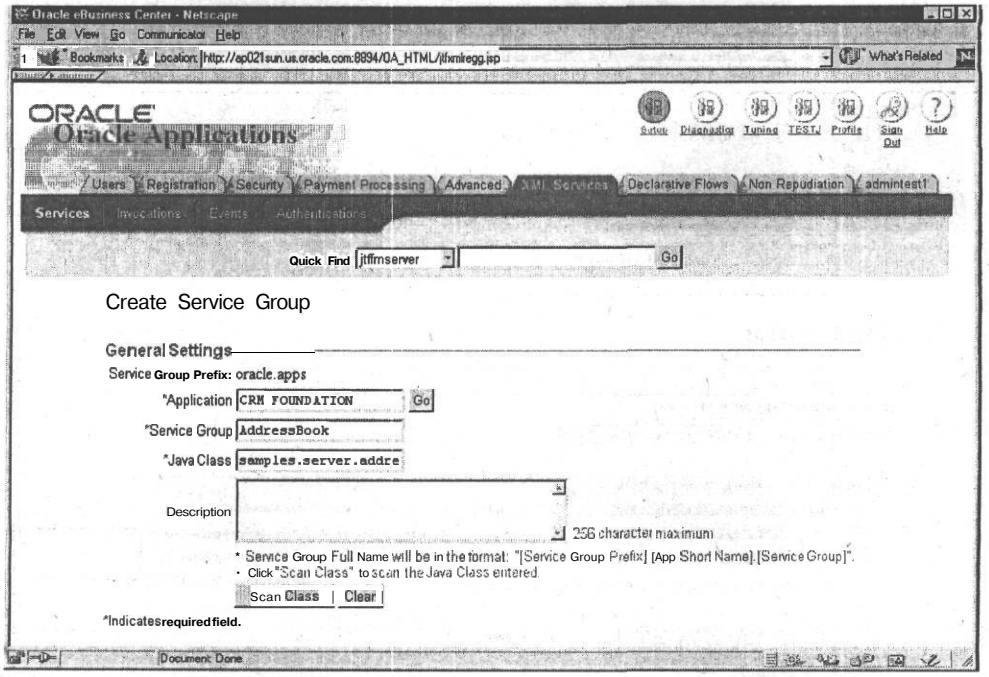


Рис. 7.3. *Создание новой группы служб*

записей о вызове служб из этой группы. Эту группу служб реализует Java-класс **samples.server.addressBook.AddressBook**, и ее название следует ввести в поле Java Class. Можно ввести краткое описание вновь определенной группы служб в поле Description. Теперь можно перейти к следующему этапу разворачивания группы служб. Для этого нужно щелкнуть по кнопке Scan Class, после чего открывается страница, показанная на рис. 7.4 (вернее, здесь показана ее первая половина). На странице Create Service Group нужно ввести всю информацию, необходимую для полного описания новой группы служб.

Как показано на рис. 7.4, в качестве возможных кандидатов службы в новой группе служб предлагаются четыре открытых метода из **AddressBook.java.** Эти службы активизируются щелчком мышью в колонке Enable в соответствующей строке. Методы, которые не активизированы в качестве служб, не будут доступны, и их нельзя будет запускать на исполнение. В любой момент можно использовать страницу, показанную на рис. 7.2, для вывода списка всех групп служб в CRM-приложении. Щелкнув по имени группы служб, можно попасть на страницу Service Update, очень похожую на рис. 7.4. Здесь дезактивируются ненужные службы. Выбрав методы, которые станут службами, нужно сделать следующий шаг, чтобы зарегистрировать преобразования типов для всех параметров, которые будут использоваться в новых службах. Метод **addEntry** использует в качестве входного параметра класс **Address,** класс **Address** использует класс **PhoneNumber**. Надо определить преобразования

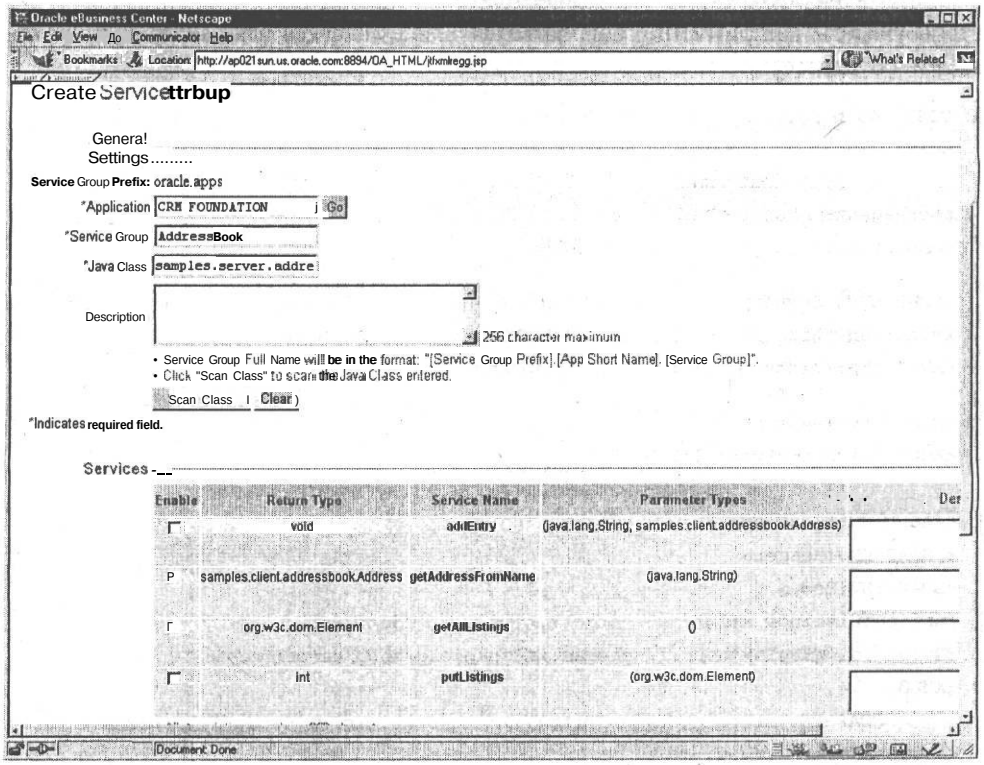


Рис. 7.4. *Выбор входящих в группу служб*

типов для всех не примитивных Java-классов, которые играют роль параметров. Поэтому надо определить преобразования типов для двух классов **Address и Phone-Number.** В этом примере есть еще два параметра: String — входной параметр для **getAddressFromName и org.w3c.doc.Element** — входной параметр для **putListing** и возвращаемый параметр для **getAllListings**. XML-службы для всех примитивных Java-типов по умолчанию используют SOAP-кодирование. Теперь рассмотрим использование класса **org.w3c.doc.Element** в качестве параметра для XML-служб. Чтобы разрешить XML-службам получать доступ к произвольным XML-данным, которые нужно обрабатывать без изменения службы (Java-метод), существует специальный тип кодирования — XML Encoding. Использование XML Encoding для параметров, являющихся XML-элементами в теле SOAP-запроса, гарантирует, что эти параметры будут переданы в Java-методы как объект **org.w3c.doc.Element**. Например, метод **putListing** берет в качестве своего единственного параметра **org.w3c.doc.Element**. Рассмотрим пример iSOAP, приведенный в этой главе для иллюстрации того, как XML-данные пользователя передаются в службу **putListing.** Вернемся к исходному коду **AddressBook.java** и обратимся к методу **putListing,**

чтобы увидеть, как **putListing** обрабатывает список адресов для извлечения информации об адресе.

```
POST /soap/servlet/soaprouter HTTP/1.0
Host: 130.35.57.119
content-type: text/xml
user-agent: Oracle-Soap-Client/1.0 HTTP/1.0
soapaction:"" Content-Length: 1362

<SOAP-ENV: Envelope xmlns:xsi="http://www.w3c.org/1999/XMLSchema-instance"
xmlns:xsd="http://www.w3c.org/1999/XMLSchema" xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/">

<SOAP-ENV:Header>
<AuthenticationType="1">
        <UserName>sysadmin</UserName><PassWD>sysadmin</PassWD>
</Authentification>
</SOAP-ENV:Header>
<SOAP-ENV:Body>
<nsl:putListings xmlns:ns1="oracle.apps.JTF.AddressBook" SOAP-
ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
<XMLDoc
     SOAP-ENV:encodingStyle="http://xml.apache.org/xml-soap/literalxml">
<AddressBook>
   <Listing>
      <Name>Mary Smith</Name>
      <Address>
         <StreetNum>888</StreetNum>
         <StreetName>Broadway</StreetName>
         <City>Somewhere</City>
         <State>FL</State>
         <Zip>87654</Zip>
         <PhoneNumber>
            <AreaCode>222</AreaCode>
            <Exchange>333</Exchange>
            <Number>4444</Number>
         </PhoneNumber>
      </Address>
   </Listing>
   <Listing>
      <Name>Dave Davis</Name>
      <Address>
         <StreetNum>919</StreetNum>
         <StreetName>Baker Lane</StreetName>
```

```
            <City>Sunnytown</City>
            <State>UT</State>
            <Zip>43434</Zip>
            <PhoneNumber>
               <AreaCode>789</AreaCode>
               <Exchange>654</Exchange>
               <Number>3210</Number>
            </PhoneNumber>
         </Address>
      </Listing>
</AddressBook>
</XMLDoc>
</ns1:putListings>
</SOAP-ENV:Body>
<SOAP-ENV: Envelope>

HTTP/1.1 2000 OK
Date: Thu, 24 May 2001 21:43:18 GMT
Server: Apache/1.3.9 (Unix) ApacheJServ/1.1
Set-Cookie: JServSessionIdsoap=t53nsgi63c.j1; path=/
Set-Cookie: crmdev07_etftech=E6E42E5131C13A1f; domain=.oracle.com; path=/
Set-Cookie: crmdev07_etftech=-1; domain=.oracle.com; path=/
Set-Cookie:
ses=202EDDECC14477F632A41448902C07038EC21642401E2F42E194B705147FOCFA286639FFD
3C5DF18E06C88901701CF2F55CC755C73CCEB85F68T8950CBF3B;
expires=Mon, 23-Jul-2001 21:43:19 GMT; domain=.oracle.com; path=/
Content-Length: 429
Connection: close
Content-Type: text/xml; charset=UTF-8

<SOAP-ENV: Envelope xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3c.org/1999/XMLSchema-instance"
xmlns:xsd="http://www.w3c:org/1999/XMLSchema">
<SOAP-ENV:Body>
<ns1:putListingsResponse xmlns:ns1="oracle.apps.JTF.AddressBook" SOAP-
ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
<return xsi:type="xsd:int">2</return>
</ns1:putListingsResponse>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

По умолчанию XML Services использует XML-кодирование для Java-типов **org.w3c.doc.Element**, поэтому не нужно определять кодирование для параметра

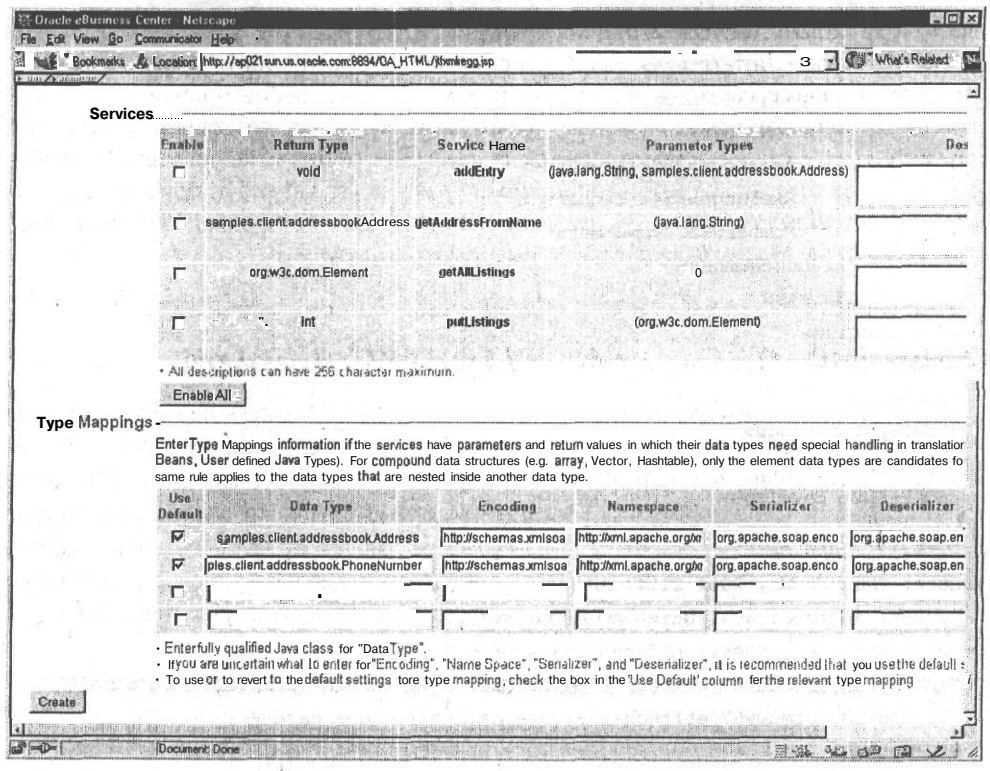


Рис. 7.5. *Выбор преобразования типа для параметров*

org.w3c.doc.Element. На рис. 7.5 показан раздел кодирования страницы Add New Service для параметров, используемых в группе служб Address Book.

После введения преобразований типа (или выбора их по умолчание, как это было сделано в примере выше) нужно щелкнуть по кнопке Create, чтобы развернуть вновь созданную группу служб в репозитории XML Services. Только что зарегистрированные службы можно использовать в двух режимах: локальном и удаленном. Для вызова локальной службы нужно создать запись вызова и выбрать в качестве профиля аутентификации localhost. Если же нужно обратиться к удаленной службе по протоколу SOAP и использовать для ее вызова XML-сообщения, следует создать профиль аутентификации и запись вызова. Локальный режим можно выбрать, если Java-клиент работает на том же экземпляре пакета E-Business Suite. В этом случае обращение к службе не использует протокол SOAP, и вызов службы выполняется как локальный Java-вызов, т. е. обращение происходит очень эффективно. Локальный режим — очень важная функция среды XML Services. Названная среда может использоваться для разворачивания интерфейсных служб. Все приложения строятся независимо и взаимодействуют с другими приложениями с помощью интерфейсных служб (или ключевых точек интеграции), предоставляемых приложениями. Если приложения работают на том же локальном экземпляре, что и E-Business Suite,

производительность будет очень хорошей, поскольку каждый вызов службы эквивалентен вызову Java-метода. В то же время зависимости внутри приложений реализуются в форме служб и обращений к ним, поэтому любое приложение можно заменить приложением стороннего разработчика путем повторной реализации служб, которые старое приложение предлагало другим приложениям для использования. Более того, новое приложение можно развернуть через Интернет по другому URL-адресу. Для этого нужно переписать записи о вызове службы с указанием нового URL-адреса, где развернуты новые службы.

Создание профиля аутентификации

Без ссылки на профиль аутентификации из пакета E-Business нельзя вызвать ни одной внешней службы с использованием API-интерфейсов клиента XML Services. Профиль аутентификации определяет внешний URL-адрес и список действительных идентификаторов пользователей и паролей, которые можно применять для доступа к службам, развернутым по этому URL-адресу. Создадим профиль аутентификации, который клиенты AddressBook смогут использовать для доступа к одноименным службам. На рис. 7.6 показана страница, где создается профиль аутентификации.

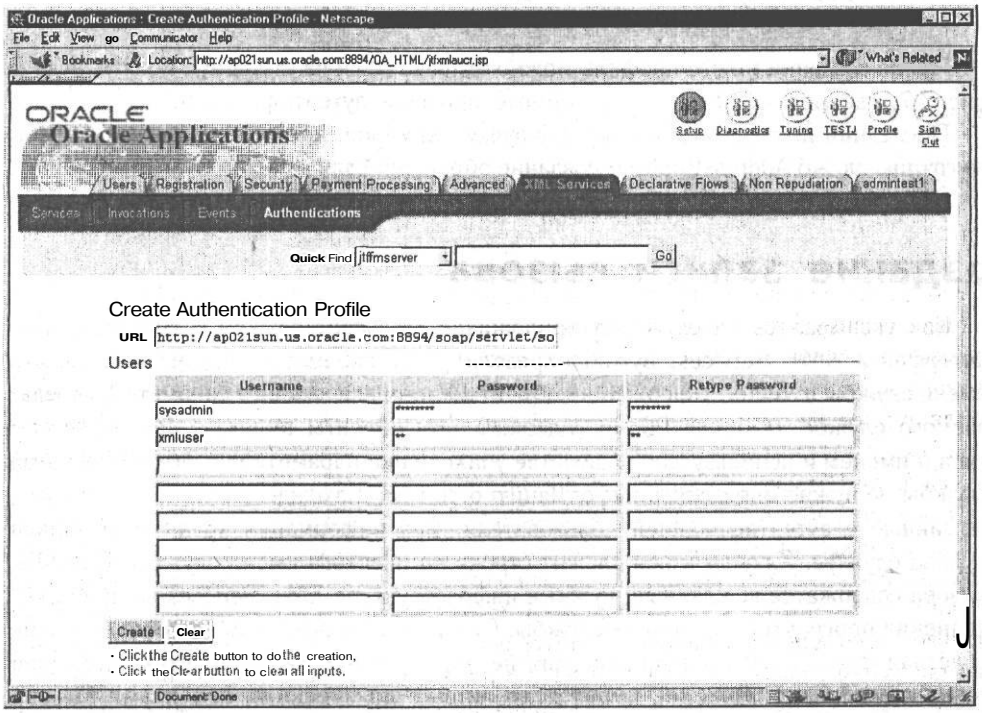


Рис. 7.6. *Создание профиля аутентификации*

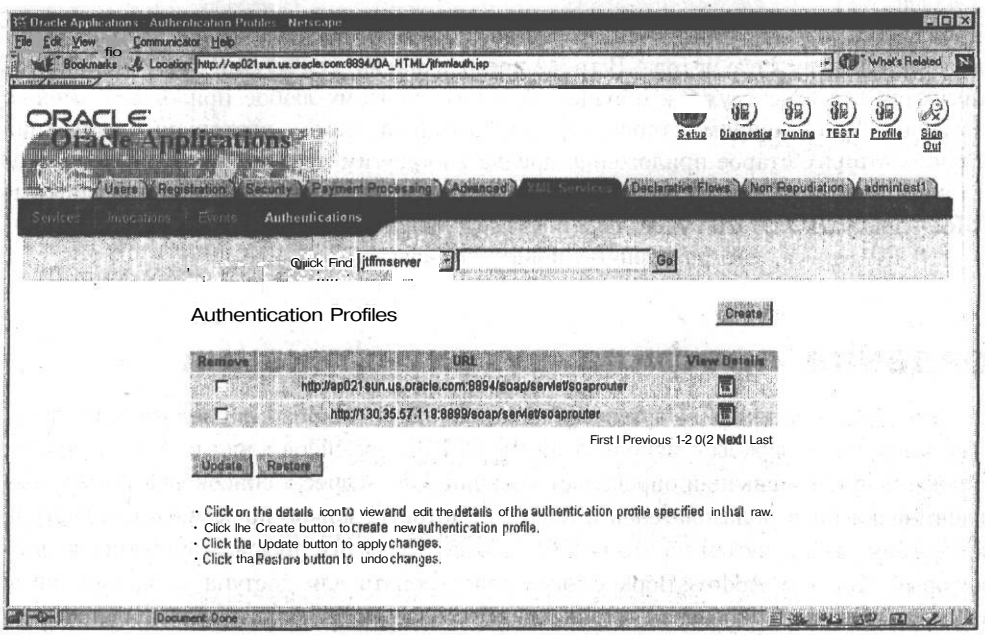


Рис. 7.7. *Профили аутентификации*

После создания такого профиля можно отправляться на страницу, показанную на рис. 7.7, где перечислены все доступные профили аутентификации.

Последний шаг для подготовки к запуску примеров, которые вызывают службы из группы служб AddressBook, — создание обращений для каждой службы в группе.

Создание записи вызова

Как указывалось ранее, когда приложение вызывает службу с помощью API-интерфейса XML Services, нужно определить два параметра: строку логического имени службы и Vector, содержащий объекты Param для каждого параметра, передаваемого службе. (Объект Param содержит Java-объекты вместе с Java-объектом типа и именем и используется в качестве упаковщика параметров.) Для вызова этой службы не нужно определять информацию о ее местонахождении, ее имя и данные, связанные с аутентификацией пользователя. Вся информация хранится в записи вызова службы. Ключом к ней служит строка логического имени службы. В записи вызова содержатся все данные, нужные инфраструктуре XML Services во время исполнения программы для вызова службы. Связывание в реальном времени — очень полезная функция. В коде приложения не определяются характеристики конечной точки службы, приложение имеет дело лишь с логическими аспектами вызова службы. Еще важнее, что службы можно легко заменять на эквивалентные (службы

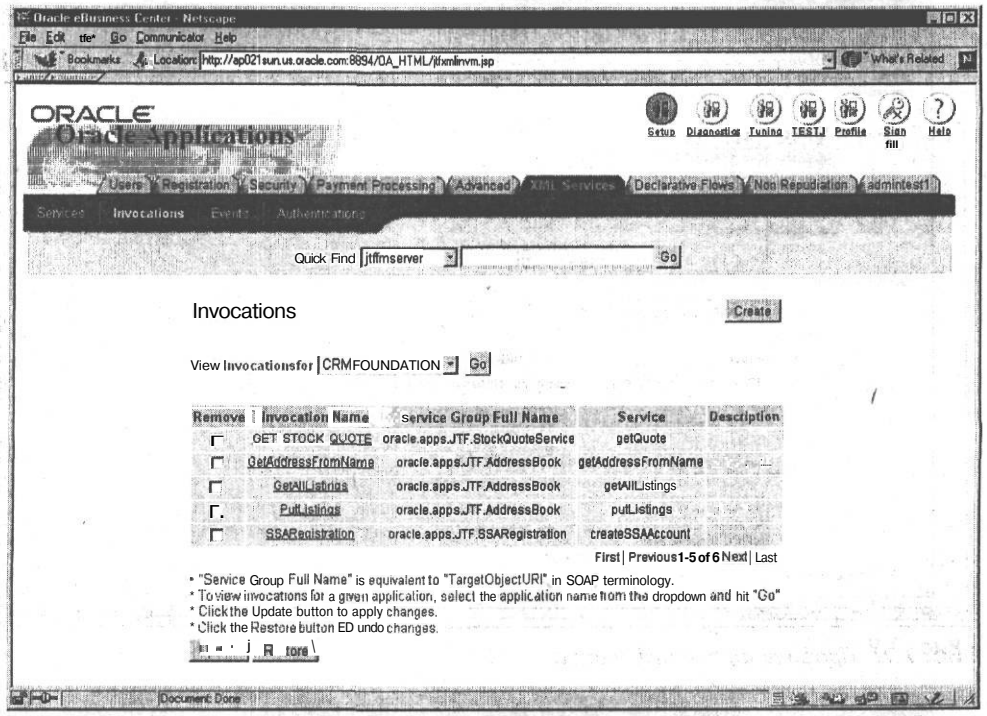


Рис. 7.8. *Список всех вызовов для CRM Foundation*

с такой же сигнатурой вызова) путем такой замены записи вызова, чтобы в этой записи указывался другой провайдер служб. Зная точки вызова для данного приложения, можно определить все внешние интерфейсы, от которых зависит приложение. На рис. 7.8 показана страница, которая используется для управления записями вызова службы. Здесь представлен список всех записей вызова в системе. Такие списки существуют для каждого приложения.

Группу служб AddressBook составляют четыре службы: GetAddressFromName, GetAllListings, PutListing и GetName. Чтобы создать запись вызова для службы GetName, щелкнем по кнопке Create. На экране появится страница, показанная на рис. 7.9 (здесь видна только ее первая часть, используемая для создания новых записей вызова).

На этой странице определим имя группы служб, имя службы, URL-адрес местонахождения службы и идентификатор пользователя, который понадобится для доступа к службе. Пароль для этого идентификатора пользователя хранится в профиле аутентификации.

На рис. 7.10 показана вторая часть страницы с записью вызова.

. Здесь нужно ввести имена всех параметров и Java-типов, которые планируется передавать в службу addEntry в таблице Target Service Signature. Кроме того, нужно определить тип возвращаемых данных и кодировку возвращаемого объекта. Таблица

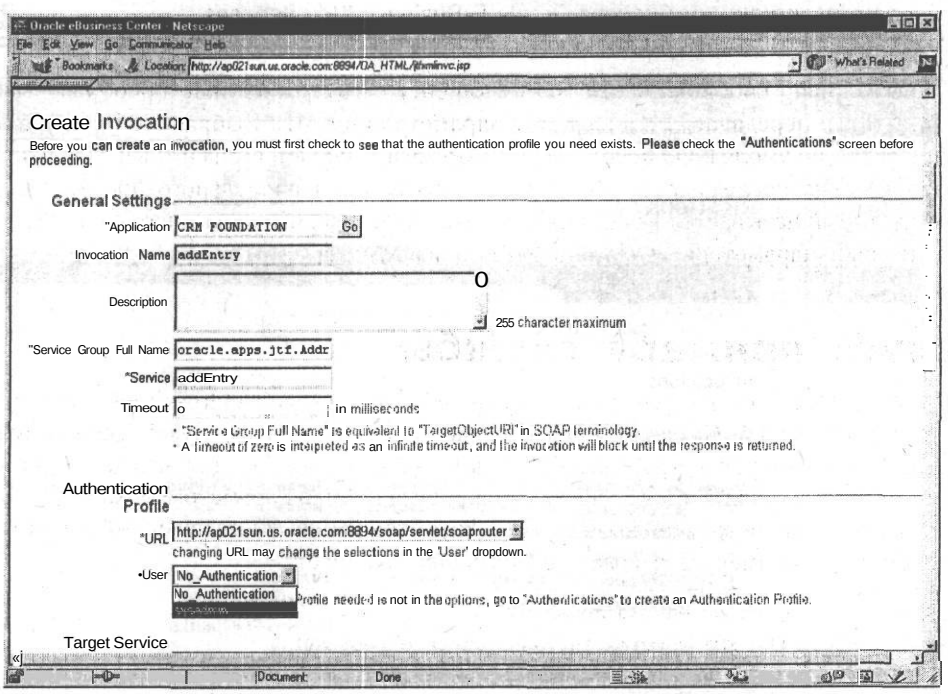


Рис. 7.9. *Создание записи вызова — часть 1*

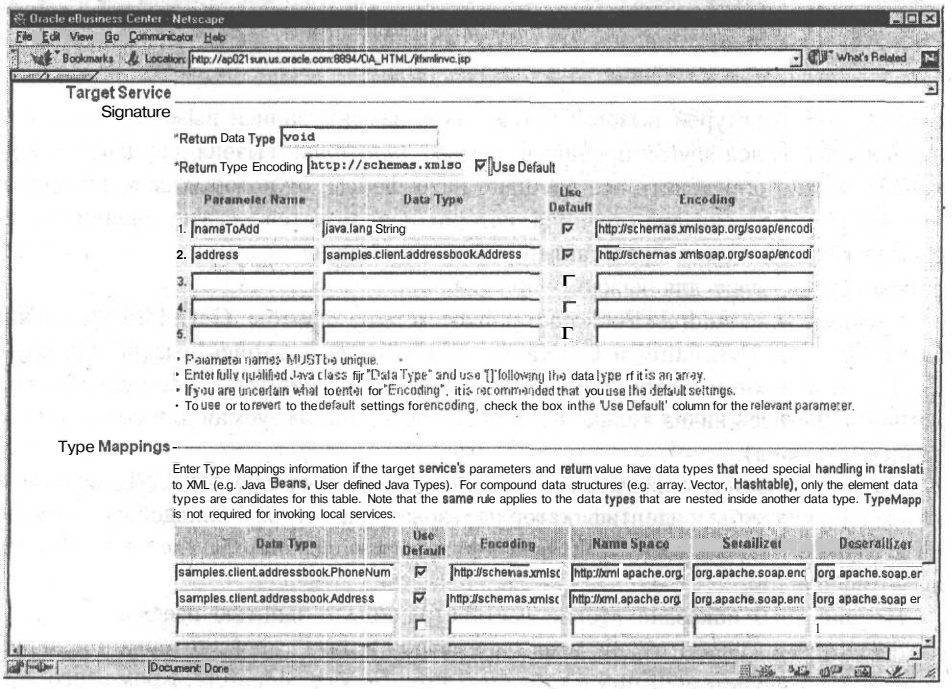


Рис. 7.10. *Создание записи вызова — часть 2*

Type Mapping сообщает CRM SOAP-клиенту о том, как нужно преобразовывать Java-типы, передаваемые в качестве параметров в XML. Выбранное преобразование типов по умолчанию использует SOAP-кодирование и стандартный SOAP-класс параллельно-последовательного и последовательно-параллельного преобразователя, который называется BeanSerializer. Этот класс будет использоваться для преобразования параметров из Java в XML и из XML в Java.

Запуск примера службы

Мы только что провели разворачивание группы служб AddressBook, состоящей из четырех служб. Создана запись вызова для одной из этих служб. Теперь, перед запуском примеров служб, нужно создать записи вызова. Для каждого примера имеется файл readme, где рассказывается, какие именно параметры нужно использовать при создании записей вызова. Итак, подготовившись к вызову службы addEntry, рассмотрим пример Java-кода, который она вызывает:

```
package samples.client.addressbook;
// Include the XML Service Infrastructure client API
import oracle.apps.jtf.services.invocation.*;
import oracle.apps.jtf.base.session.*;
import java.io.*;
import java.util.*;
import java.net.*;
import org.w3c.dom.*;
import org.apache.soap.util.xml.*;
import org.apache.soap.*;
import org.apache.soap.encoding.*;
import org.apache.soap.encoding.soapenc.*;
import org.apache.soap.rpc.*;

/**
 * Put a new address into the address book.
 *
 */
public class PutAddress
{  public static void main(Srting[]  args) throws Exception
   {  if (args.length !=9)
      {  System.err.println("Usage:");
         System.err.println(" java " + PutAddress.class.getName( +
                            " name streetNumber * +
                            "streetName city state zip areaCode"+
                            "exchange number");
         System.exit (1);
      }
```

```
        // Start JTF session to access registered invocation
        ServletSessionManager.startStandAloneSession("JTF", false, "SYSADMIN"
                                                     "SYSADMIN");
        // process the arguments
        String nameToRegister = args[0];
        PhoneNumber phoneNumber = new PhoneNumber (
        Integer.parseInt (args[6], args[7], args[8]);
        Address address = new Address (Integer.parseInt (args[1]),
                                       args [2], args[3], args [4],
                                       Integer.parseInt(args[5]),
                                       phoneNumber);
        // Build argument vector
        Vector params = new Vector ( ) ;
        params.addElement(new Param( "nameToRegister", String.class,
                                     nameToRegister) ) ;
        params.addElement (newParam("address", Address . class , address));
        ServiceResult result = null;
        try {
           // Execute the call
           result = Client.callService("AddEntry", params);
        }
        catch (Exception e) {
           System.out.println( "Error occur in invocation AddEntry");
           e.printStackTrace ( ) ;
           return;
        }
        System.out .println("Entry is added successfully");
        // End JTF session
        ServletSessionManager.endStandAloneSession ( ) ;
    }
}
```

На рис. 7.11 показана команда для запуска этого примера.

```
(skiritzo) addressbook- java ${JAXP} ${JTFDBCFILE} ${FWLogFile}
samples.client.addressbook.PutAddress "John Doe" 123 "Main Street"
AnyTown SS 12 345 800 555 1212
Entry is added successfully
(skiritzo) addressbook- I
```

Рис. 7.11. *Запуск примера PutAddress*

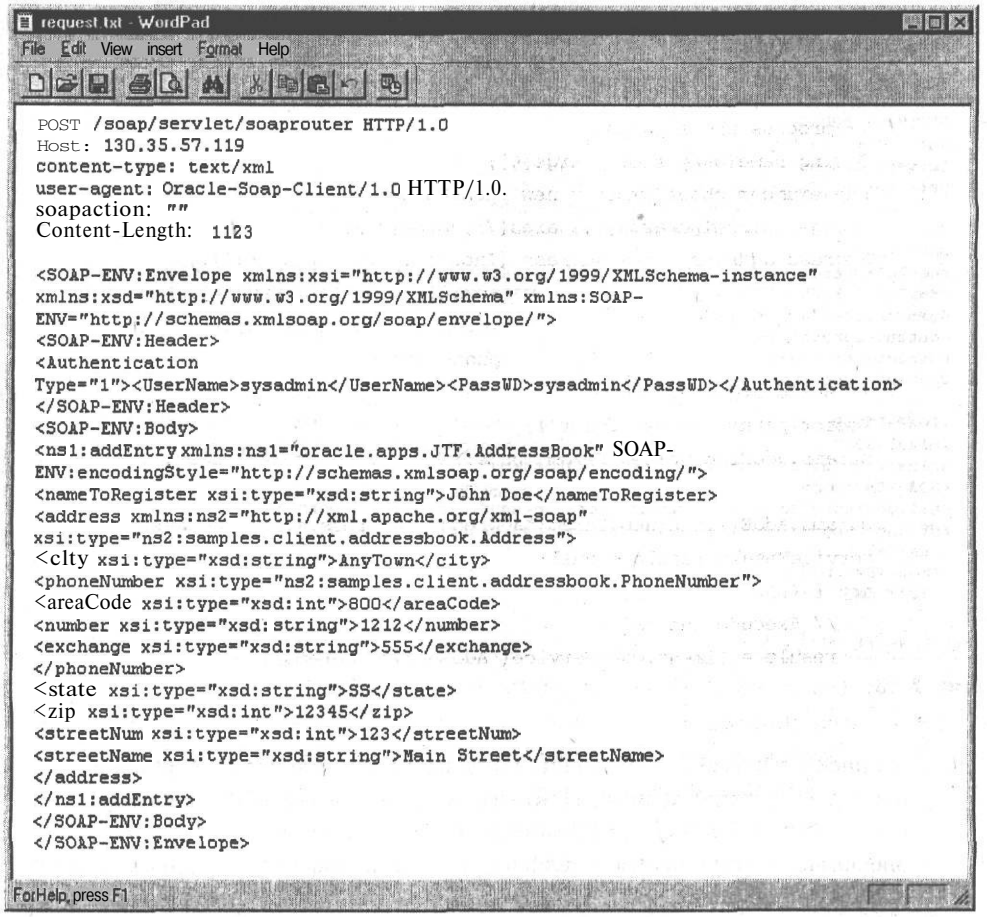


Рис. 7.12. *Сообщение с запросом addEntry SOAP-службы*

Tunnel — это пример приложения для трассировки порта, поставляемого вместе с Oracle SOAP. Правда, использовалась немного модифицированная версия этого приложения для мониторинга и захвата данных, которыми обмениваются SOAP-клиент и SOAP-сервер. Исходный код приложения Tunnel можно скопировать вместе с исходным кодом SOAP с сайта http://xml.apache.org. На рис. 7.12 показано сообщение с запросом, которое отправляет пример PutAddress на SOAP-сервер XML Services. На рис. 7.13 приведен ответ на это сообщение.

После вызова примера addEntry выполняются следующие действия:

1. Приложение вызывает службу с помощью API-интерфейса XML Services. Приложение выдает имя вызова и список параметров для вызова службы.

2. XML Services ищет метаданные вызова и вызывает API-интефейс SOAP-сервера, информация о результирующих данных

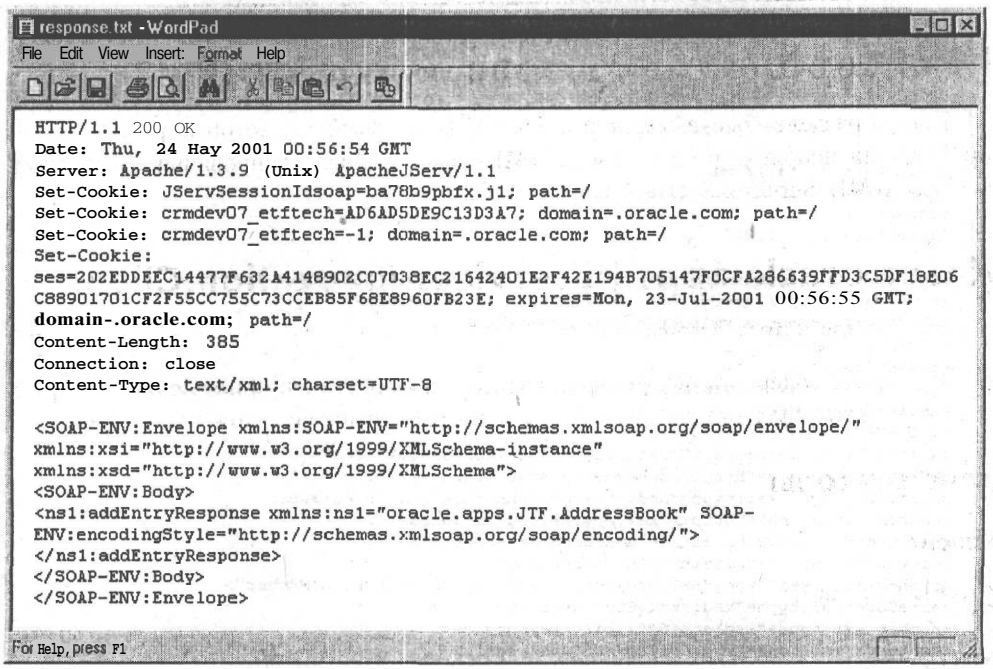


Рис. 7.13. *Ответное сообщение addEntry SOAP-службы*

и список параметров передается приложением. Кроме информации о результирующих данных (URL-адрес, имя [класс] группы служб, имя [метод] службы) в метаданные также входит аутентификационная информация (имя пользователя/пароль). Эта информация также входит в SOAP-сообщение.

3. SOAP-сервер размещает Java-параметры в запрашивающем SOAP-сообщении.

4. SOAP-сервер посылает SOAP-сообщение через транспортный адаптер.

5. Принимающий SOAP-сервер получает SOAP-сообщение.

6. **Принимающий SOAP-сервер распаковывает сообщение** и обращается к провайдеру служб XML Services, передавая ему аутентификационную информацию, имя (класс) группы служб, имя (класс) службы и список параметров.

7. Провайдер служб XML Services ищет метаданные служб (в том числе данные пользователя) и вызывает службу, если пользователь имеет на это право.

8. Процесс получения ответа происходит по точно такому же плану.

API-интерфейсы вызова и события

В этом разделе рассказывается о Java API-интерфейсах, которые разработчики приложений используют для вызова XML-служб и сигнализирования о событиях в среде XML Services.

Класс oracle.apps.jtf.services.invocation.Client

```
public class Client extends Object
```

Этот класс содержит API-интерфейсы для разработчиков приложений, которые предназначаются для вызова службы или сигнализирования о каком-то событии.

Конструкторы

Клиент

```
Public Client O
```

Методы

callService

```
public static ServiceResult callService(String SLN, Vector params)
throws InvocationException
```

Вызов службы по ее логическому имени.

Параметры:

String SLN: логическое имя службы. Это логическое имя во время исполнения программы динамически связывается с метаданными (URL-адрес сервера, идентификатор группы служб, идентификатор службы, идентификатор пользователя, пароль и т. д.), необходимыми для вызова службы. Все метаданные хранятся в виде записи вызова. Логическое имя службы используется для получения записи вызова, содержащей всю информацию для вызова данной службы.

Vector params: Vector объектов Param. Каждый такой объект содержит имя, тип данных и значение одного из параметров, передаваемых службе.

Возвращаемые данные:

ServiceResult: возвращаемый объект после исполнения службы.

Исключения:

InvocationException

signalEvent

```
public static void signalEvent(String eventName, Param eventObj) throws
InvocationException
```

Сигнализирует о событии с использованием имени события. Все связанные с событием слушающие программы (службы) вызываются асинхронно.

Параметры:

String eventName: имя события. Исходя из данного имени, производится динамическое распределение всех связанных с этим событием слушающих программ.

Param eventObject: параметр, который передается в слушающие программы этого события.

Возвращаемые данные:

нет

Исключения:

InvocationException, если имя события не зарегистрировано или инфраструктура XML Services не может породить подпроцессы для вызова слушающих программ.

getListeners

```
public static String[] getListeners(String eventName)
```

Получает список логических имен слушающих программ, связанных с данным именем события.

Параметры:

String eventName: имя события

Возвращаемые данные:

String[]: Логические имена слушающих программ. Если имя события не зарегистрировано, возвращается пустая ячейка. Если в связи с данным событием не зарегистрировано ни одной слушающей программы, возвращается пустой массив.

signalEvent

```
public static void signalEvent(String eventName, String selectedListeners[],
Param eventObj) throws InvocationException
```

Сигнализирует о событии, используя его имя. Связанная с событием слушающая программа (служба) вызывается асинхронно, если содержится в списке слушающих программ.

Параметры:

String eventName: имя события. Исходя из этого имени, производится динамическое распределение всех связанных с событием слушающих программ.

String selectedListeners[]: список логических имен слушающих программ.

Param eventObject: параметр, который передается в слушающие программы этого события.

Возвращаемые данные:

нет

Исключения:

InvocationException, если имя события не зарегистрировано или при вызове не удалось породить подпроцесс для вызова слушающей программы.

Класс oracle.apps.jtf.services.invocation.Param

```
public class Param extends Object
```

Чтобы вызвать службу или сигнализировать о событии, для каждого передаваемого параметра создается объект Parat. В классе Parat хранится имя параметра, тип данных и значение этого параметра.

Конструкторы

Param

```
public Param(String name, Class type, Object value) throws
ClassNotMatchException
```

Параметры:

String name: имя параметра

Class type: тип данных

Object value: значение параметра

Исключения:

ClassNotMatchException, если объект значения параметра не присвоен и не совместим с типом параметра.

Остальные конструкторы для объекта Parat описаны в документации по XML Services Java.

Методы

getName

public String `getName()`

Получает имя параметра.

Возвращаемые данные:

String: имя параметра

setName

public void `setName(String` name)

Устанавливает имя параметра.

Параметры:

String name: имя параметра

getType

`public Class getType()`

Получает тип данных.

Возвращаемые данные:

Class: тип данных

getObject

public Object `getObject()`

Получает значение параметра.

Возвращаемые данные:

нет

getBoolean

public boolean `getBoolean()`

Получает значение параметра.

Возвращаемые данные:

boolean: значение параметра

getInt

public int `getInt()`

Получает значение параметра.

Возвращаемые данные:

int: значение параметра

gefByte

public byte `getByte()`

Получает значение параметра.

Возвращаемые данные:
byte: значение параметра

getChar

public char `getChar()`

Получает значение параметра.

Возвращаемые данные:
char: значение параметра

getDouble

public double `getDouble()`

Получает значение параметра.

Возвращаемые данные:
double: значение параметра

getFloat

public float `getFloat()`

Получает значение параметра.

Возвращаемые данные:
float: значение параметра

getLong

public long `getLong()`

Получает значение параметра.

Возвращаемые данные:
long: значение параметра

getShort

public short `getShort()`

Получает значение параметра.

Возвращаемые данные:
short: значение параметра

setObject

public void `setObject(Object` value) throws ClassNotMatchException

Устанавливает значение параметра.

Параметры:

Object value: значение параметра

Исключения:

ClassNotMatchException, если объект значения параметра не присвоен и не совместим с типом параметра.

setBoolean

public void `setBoolean(boolean` value) throws ClassNotMatchException

Устанавливает значение параметра.

Параметры:

boolean value: значение параметра

Исключения:

ClassNotMatchException, если тип параметра не является логическим.

setInt

public void `setInt(int` value) throws ClassNotMatchException

Устанавливает значение параметра.

Параметры:

int value: значение параметра

Исключения:

ClassNotMatchException, если тип параметра не является целым.

setByte

public void `setByte(byte` value) throws ClassNotMatchException

Устанавливает значение параметра.

Параметры:

byte value: значение параметра

Исключения:

ClassNotMatchException, если тип параметра не является байтом.

setChar

public void `setChar(char` value) throws ClassNotMatchException

Устанавливает значение параметра.

Параметры:

char value: значение параметра

Исключения:

ClassNotMatchException, если тип параметра не является символьным.

setDouble

```
public void setDouble(double value) throws ClassNotMatchException
```

Устанавливает значение параметра.

Параметры:

double value: значение параметра

Исключения:

ClassNotMatchException, если параметр не является числом с удвоенной точностью.

setFloat

```
public void setFloat(float value) throws ClassNotMatchException
```

Устанавливает значение параметра.

Параметры:

float value: значение параметра

Исключения:

ClassNotMatchException, если параметр не является числом с плавающей точкой.

setLong

```
public void setLong(long value) throws ClassNotMatchException
```

Устанавливает значение параметра.

Параметры:

long value: значение параметра

Исключения:

ClassNotMatchException, если параметр не является длинным числом с двойным количеством разрядов.

setShort

```
public void setShort(short value) throws ClassNotMatchException
```

Устанавливает значение параметра.

Параметры:

short value: значение параметра

Исключения:

ClassNotMatchException, если параметр не является коротким числом.

Класс oracle.apps.jtf.services.invocation.ServiceResult

Конструкторы

ServiceResult

```
public ServiceResult(Class type, Object returnValue)
```

Методы

getBoolean

```
public boolean getBoolean()
```

Получает значение параметра.

Возвращаемые данные:

boolean: возвращаемое значение

getByte

```
public byte getByte()
```

Получает значение параметра.

Возвращаемые данные:

byte: возвращаемое значение

getChar

```
public char getChar()
```

Получает значение параметра.

Возвращаемые данные:

char: возвращаемое значение

getDouble

```
public double getDouble()
```

Получает значение параметра.

Возвращаемые данные:

double: возвращаемое значение

getFloat

```
public float getFloat()
```

Получает значение параметра.

Возвращаемые данные:

float: возвращаемое значение

getInt

```
public int getInt()
```

Получает значение параметра.

Возвращаемые данные:

int: возвращаемое значение

getLong

```
public long getLong()
```

Получает значение параметра.

Возвращаемые данные:

long: возвращаемое значение

getObject

```
public Object getObject()
```

Получает значение параметра.

Возвращаемые данные:

Object: возвращаемое значение

getShort

```
public short getShort()
```

Получает значение параметра.

Возвращаемые данные:

short: возвращаемое значение

getType

```
public Class getType()
```

Получает тип возвращаемого значения.

Возвращаемые данные:

Class: тип данных

Извлечение данных в формате XML

Так как в реляционной БД данные организованы иерархическим образом, их можно смоделировать в XML-формате. Предположим, что есть база данных со списком книг, в которой таблица BookList состоит из следующих столбцов: BookID (идентификатор книги), Title (заглавие), Author (автор), Publisher (издательство), Year (год издания), ISBN и Description (описание). Типичный запрос к такой базе данных будет выглядеть следующим образом:

```
SELECT Title, Author, Publisher, Year, ISBN FROM BookList WHERE BookID = 1234;
```

ехнология XML представляет собой некую базу для самых разных приложений, однако следует помнить, что это лишь технология, позволяющая создавать приложения, сама она приложением не является. До тех пор, пока нет определенной схемы, или DTD, приложения не могут использовать XML для надежного обмена или визуализации данных. В этой главе рассматриваются две прикладные области, для которых существуют соглашения о DTD, и обсуждается их конкретная реализация.

Технология XML тесно связана с Интернетом по целому ряду очень важных причин. Так как содержимое XML-документов представляет собой простой текст, обмен документами по существующим Интернет-протоколам, через операционные системы и сетевые экраны осуществляется легко и просто. Наличие такой возможности породило две основные области применения этой технологии: доставка контента на самые разные устройства с поддержкой работы в Интернете и обмен данными в приложениях электронного бизнеса.

Прежде чем приступать к подробному рассмотрению этих областей, нужно познакомиться с двумя Oracle Java-компонентами, которые были представлены в главе 3, представляющими собой компоненты инфраструктуры XML: Oracle XML SQL Utility и Oracle XSQL Servlet.

Oracle XML SQL Utility

Существует довольно много приложений, в которых информация хранится в базе данных и при необходимости извлекается по запросам. Такая организация работы удобна. Использование же базы данных, поддерживающей технологию XML, удобно еще и потому, что ответы на запросы она выдает уже в XML-формате. XML SQL Utility — это набор Java-классов, который принимает такие запросы от приложения, направляет их посредством интерфейса JDBC в базу данных и возвращает результирующие данные в XML-формате в соответствии со схемой базы данных этого запроса. Кроме того, XML SQL Utility может принимать XML-документы, совместимые со схемой базы данных, и сохранять в этой базе непомеченные данные (без тегов).

Кроме чтения и записи XML-данных *в* базы данных, поддерживающие интерфейс JDBC, XML SQL Utility может создавать описание DTD, которое представляет собой схему базы данных. Затем DTD можно использовать в разработке приложений с помощью генераторов классов Oracle Class Generators, как это было показано в главе 2. Допустим, есть две версии интерфейса JDBC — 1.1 и 1.2, и ПО XML SQL Utility для связи с базой данных должно поддерживать хотя бы одну из них.


```
public class read_sample
{  //   ---  ----     --      --    ----------
   // main() - public static void
   public static void main(String args[]) throws SQLException
   {  String tabName = "Booklist";
      String user = "scott/tiger";
      DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());

      // init a JDBC connection by passing in the user
      Connection conn =
```

Извлечение данных в формате XML

Так как в реляционной БД данные организованы иерархическим образом, их можно смоделировать в XML-формате. Предположим, что есть база данных со списком книг, в которой таблица **BookList** состоит из следующих столбцов: BookID (идентификатор книги), Title (заглавие), Author (автор), Publisher (издательство), Year (год издания), ISBN и Description (описание). Типичный запрос к такой базе данных будет выглядеть следующим образом:

```
SELECT Title, Author, Publisher, Year, ISBN FROM BookList WHERE BookID = 1234;
```

Если подобный запрос направить через XML SQL Utility, будет получен следующий ответ:

```
<?xml version="1.0"?>
<ROWSET>
   <ROW id="1">
      <TITLE>The Difference Between God and Larry Ellison: Inside Oracle
            Corporation</TITLE>
      <AUTHOR>Mike Wilson</AUTHOR>
      <PUBLISHER>William Morrow and Co.</PUBLISHER>
      <YEAR>1997</YEAR>
      <ISBN>0688149251</ISBN>
   </ROW>
</ROWSET>
```

Этот ответ может быть представлен в виде строки, если приложение просто дает команду на запись его в файл, или в виде DOM-объекта, который можно направить непосредственно в XML-анализатор синтаксиса Oracle для преобразования XSLT-процессором. Если ответ на запрос выводится в виде DOM-объекта, операция* синтаксического анализа опускается, а если нет, ее нужно проводить перед любым XSL-преобразованием.

Как показано в следующем фрагменте кода, запросы можно обрабатывать, направляя их в класс **oracle.xml.sql.query.OracleXMLQuery**:

```
/** Simple example on using Oracle XMLSQL API; this class queries the
 database with "select * from Booklisting* in scott/tiger schema; then
 from the results of query it generates an XML document */

import java.sql.*;
import java.math.*;
import oracle.xml.sql.query.*;
import oracle.jdbc.*;
import oracle.jdbc.driver.*;
```

```
public class read_sample
{  //=======================================
   // main() - public static void
   public static void main(String args[]) throws SQLException
   {  String tabName = "Booklist";
      String user = "scott/tiger";
      DriverManager.registerDriver(neworacle.jdbc.driver.OracleDriver());

      // init a JDBC connection by passing in the user
      Connection conn =
         DriverManager.getConnection("jdbc:oracle:oci8:"+user+"@");

      // init the OracleXMLQuery by using the initialized JDBC connectioon
      // and passing in "Booklist" as tabName
      OracleXMLQuery qry = new OracleXMLQuery(conn,"select * from "+tabName);

      // get the XML document in the string format which allows us
      // to print it
      String xmlString = qry.getXMLString();

      // print out the result to the screen
      System.out.println("OUPUT IS:\n"+xmlString);

      // Close the JDBC connection
      conn.close();
   }
}
```

В XML SQL Utility есть возможность работать с интерфейсом командной строки, который может оказаться полезным при генерации DTD, связанного с определенной схемой базы данных. Допустим, что все необходмое установлено, как полагается. Тогда полный список доступных опций командной строки можно получить с помощью команды:

```
java oraclexml
```

В следующей командной строке показано, как создать DTD, связанный с заданной схемой базы данных, куда направляются запросы:

```
java oraclexml getxml -user "scott/tiger" -withDTD "SELECT *
   FROM Booklist"
```

Для таблицы Booklist, о которой говорилось выше, XML SQL Utility может выдать следующий DTD, встроенный в XML-документ, полученный в ответ на запрос:

```
<!ELEMENT BOOKLIST (BookID, Title, Author, Publisher, Year, ISBN, Description)>
<!ELEMENT BookID (#PCDATA)>
<!ELEMENT Author (#PCDATA)>
```

```
<! ELEMENT Publisher (#PCDATA)>
<! ELEMENT Year (#PCDATA)>
<! ELEMENT ISBN (#PCDATA)>
<!ELEMENT Description (#PCDATA)>
```

Сохранение данных в формате XML

После того *как* в базе данных создана схема, XML SQL Utility может сохранять в ней данные, имеющие формат XML, если они соответствуют DTD, сгенерированному из созданной схемы. XML SQL Utility позволяет отображать XML-документы на строки таблицы. Имена тегов элементов отображаются на столбцы с XML-строками, по умолчанию преобразованными к соответствующим типам данных. Если XML-элемент имеет дочерние элементы, производится преобразование к объекту SQL-типа.

Для сохранения данных в формате XML программа XML SQL Utility инициирует вставку оператора, связывающего все значения элементов в выражении VALUE. Содержимое каждого элемента строки отображается на отдельный набор значений.

Если вернуться к предыдущему примеру BookList, сгенерированный SQL-оператор, дающий команду на сохранение введенных данных, имеющих XML-формат, будет выглядеть так:

```
INSERT INTO BOOKLIST (BookID, TITLE, AUTHOR, PUBLISHER, YEAR, ISBN,
   DESCRIPTION) VALUES (?,?,?,?,?,?,?) and BIND the values,
BOOKID -> 1234
TITLE -> The Difference Between God and Larry Ellison: Inside Oracle
         Corporation
AUTHOR -> Mike Wilson
PUBLISHER -> William Morrow & Co.
YEAR -> 1997
ISBN -> 0688149251
DESCRIPTION -> Account of Larry Ellison;
```

Следующий пример кода показывает, как это можно сделать с помощью Java-программы:

```
/** Simple example on using Oracle XMLSQL API; this class inserts the
data from an XML document into the database */

import oracle.xml.sql.query.*;
import java.sql.*;
import oracle.jdbc.driver.*;
import oracle.jdbc.*;
import java.net.*;
```

```
public class save_sample
{  //=======================================
   // main() - public static void
   public static void main (String args[]) throws SQLException
   {  String tabName = "Booklist";            // table into which to insert
      String fileName = "sampdoc.xml";        // file containing the xml doc
      DriverManager.registerDriver(neworacle.jdbc.driver.OracleDriver());

      // init a JDBC connection by passing in the user and password
      Connection conn =
         DriverManager.getConnection("jdbc:oracle:oci8:scott/tiger@");

      // Init the OracleXMLSave class by passing in the JDBC connection
      // and the "BOOKLIST" table name.
      OracleXMLSave sav = new OracleXMLSave (conn, tabName);

      // Point to the XML file, sampdoc.xml to save using a URL
      URL url = sav. createURL(fileName);

      // Save the data populating rows in the BOOKLIST table and
      // return the number of rows written
      int rowCount = sav.insertXML(url) ;

      // Print out the confirmation of the successful insertion
      System.out.println(" successfully inserted "+rowCount+
                         " rows into "+ tabName) ;

      // Close the JDBC connection
      conn.close();
   }
}
```

Как и в случае с функцией **getXML**, существует версия функции сохранения данных **putXML** в режиме командной строки. Ее полезно использовать при больших объемах загрузки XML-данных. XML-документ, содержащий список книг, соответствующий нашему DTD, можно загрузить *с* помощью следующей командной строки:

```
java oraclexml putXML -user "scott/tiger" sampdoc.xml Booklist
```

Выполнение обновлений с помощью XML SQL Utility

Обновления отличаются от ввода данных тем, что могут применяться сразу к нескольким строкам таблицы. Обновляемый XML-документ может соответствовать более чем одной строке таблицы, если соответствующие ему столбцы не являются ее ключевыми столбцами. Поэтому обновления требуют наличия списка ключевых

столбцов, которые XML SQL Utility использует для идентификации обновляемой строки. Это показано в следующем примере обновления списка книг:

```
<ROWSET>
   <ROW id="1">
      <BOOKID>1234</BOOKID>
      <TITLE> The Difference Between God and Larry Ellison: Inside Oracle
              Corporation </TITLE>
      <AUTHOR>Mike Wilson</AUTHOR>
      <PUBLISHER>William Morrow and Co.</PUBLISHER>
      <YEAR>1997</YEAR>
      <ISBN>0688149251</ISBN>
   </ROW>
</ROWSET>
```

Это XML-обновление приводит к следующему SQL-оператору, который передается в BookID и ключевые столбцы:

```
UPDATE BOOKLIST SET TITLE = ?, AUTHOR = ?, WHERE BOOKID = ?
      and bind the values,
      BOOKID -> 1234
      TITLE -> The Difference Between God and Larry Ellison: Inside
               Oracle Corporation
      AUTHOR -> Mike Wilson;
```

Отметим, что не нужно обновлять все столбцы исходного XML-документа. Вот код, показывающий, как это можно сделать из Java-программы:

```
/** Simple example on using Oracle XMLSQL API; this class updates the
database data from an XML document submitted into the database */

import oracle.xml.sql.dml.*;
import java.sql.*;
import oracle.jdbc.driver.*;
import oracle.jdbc.*;

public class ListUpdate
{  //=======================================
   // main() - public static void
   public static void main (String args[]) throws SQLException
   {  String tabName = "BOOKLIST";          // table into which to insert
      DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());

      // init a JDBC connection by passing in the user and password
      Connection conn =
         DriverManager.getConnection("jdbc:oracle:oci8:scott/tiger@");
```

```
        // Init the OracleXMLSave class by passing in the JDBC connection
        //and the "BOOKLIST" table name.
        OracleXMLSave sav = new OracleXMLSave (conn, tabName) ;

        // init the key column used for the update
        String [] keyColNames = new String [1];
        keyColNames [0] = "BOOKID";
        sav.keyColumnNames (keyColNames);

        // Assume that the user passess in this document as the first argument!
        sav.updateXML(argv[0]);
        sav.close();
    }
}
```

Удаление документов с помощью XML SQL Utility

XML SQL Utility также поддерживает удаление XML-документов. Эта программа идентифицирует строки, которые нужно удалить, с помощью ключевых столбцов. Все происходит точно так же, как при обновлении данных. Если не задан один или несколько ключевых столбцов, то XML SQL Utility попытается согласовать столбцы в документе. Ниже показан XML-документ и соответствующий ему эквивалент, сгенерированный SQL-оператором удаления:

```
<ROWSET>
<ROW num="1">
<BOOKID>1234</BOOKID>
<TITLE > The Difference Between God and Larry Ellison: Inside Oracle
Corporation</TITLE>
<AUTHOR>Mike Wilson</AUTHOR>
<PUBLISHER>William Morrow and Co.</PUBLISHER>
<YEAR>1997</YEAR>
<ISBN>0688149251</ISBN>
</ROW>
</ROWSET>

DELETE FROM BOOKLIST WHERE TITLE=? AND AUTHOR=? AND PUBLISHER=? AND YEAR=?
AND ISBN=? AND BOOKID=?
binding,
BOOKID <- 1234
TITLE <- The Difference Between God and Larry Ellison: Inside
Oracle Corporation
AUTHOR <- Mike Wilson
```

```
PUBLISHER <- William Morrow & Co.
YEAR <- 1997
ISBN <- 0688149251;
```

В следующем примере показано, как это удаление можно произвести с помощью Java-программы, используя в качестве ключевого столбца BookID:

```
/** Simple example on using Oracle XMLSQL API; this class deletes the
database data from an XML document submitted into the database */

import oracle.xml.sql.dml.*; i mport java.sql.*;
import oracle.jdbc.driver.*;
import oracle.jdbc.*;

public class ListDelete
{  //=======================================
   // main() - public static void
   public static void main(String args[]) throws SQLException
   {   String tabName = "BOOKLIST";          // table into which to delete data
      DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());

      // init a JDBC connection by passing in the user and password
      Connection conn =
         DriverManager.getConnection("jdbc:oracle:oci8:scott/tiger@");

      // Init the OracleXMLSave class by passing in the JDBC connection
      // and the "BOOKLIST" table name.
      OracleXMLSave sav = new OracleXMLSave(conn, tabName);

      // init the key column used for the update
      String [] keyColNames = new String [1];
      keyColNames [0] = "BookID";
      sav.keyColumnNames(keyColNames);

      // Assume that the user passess in this document as the first argument!
      sav.deleteXML(argv[0]);
      sav.close();
   }
}
```

Установка XML SQL Utility

Процедура инсталляции XML SQL Utility на стороне клиента или на стороне сервера ничем не отличается от установки любой другой Java-библиотеки. Так как эта утилита запакована в виде архива формата **.jar**, при использовании файлов **xsu111.jar** или **xsu12.jar** в сеансе связи с БД нужно явно описать их имена и путь

в CLASSPATH. Для организации среды сеанса связи с базой данных в комплект поставки входят также файлы **env.bat** для ОС Windows и **env.csh** для ОС Unix. Наконец, для подключения к базе данных нужен соответствующий набор JDBC-классов версии 1.1 или 1.2.

В приложениях, где утилита XML SQL Utility взаимодействует с базой данных Oracle8i, должна быть дополнительная опция инсталляции. Так как в СУБД Oracle8i уже включена виртуальная Java-машина JServer Java Virtual Machine, jar-файл с XML SQL Utility можно загрузить прямо в базу данных, где к ней можно будет обращаться из внутренних или внешних процедур. СУБД Oracle8i версии 8.1.6 поставляется

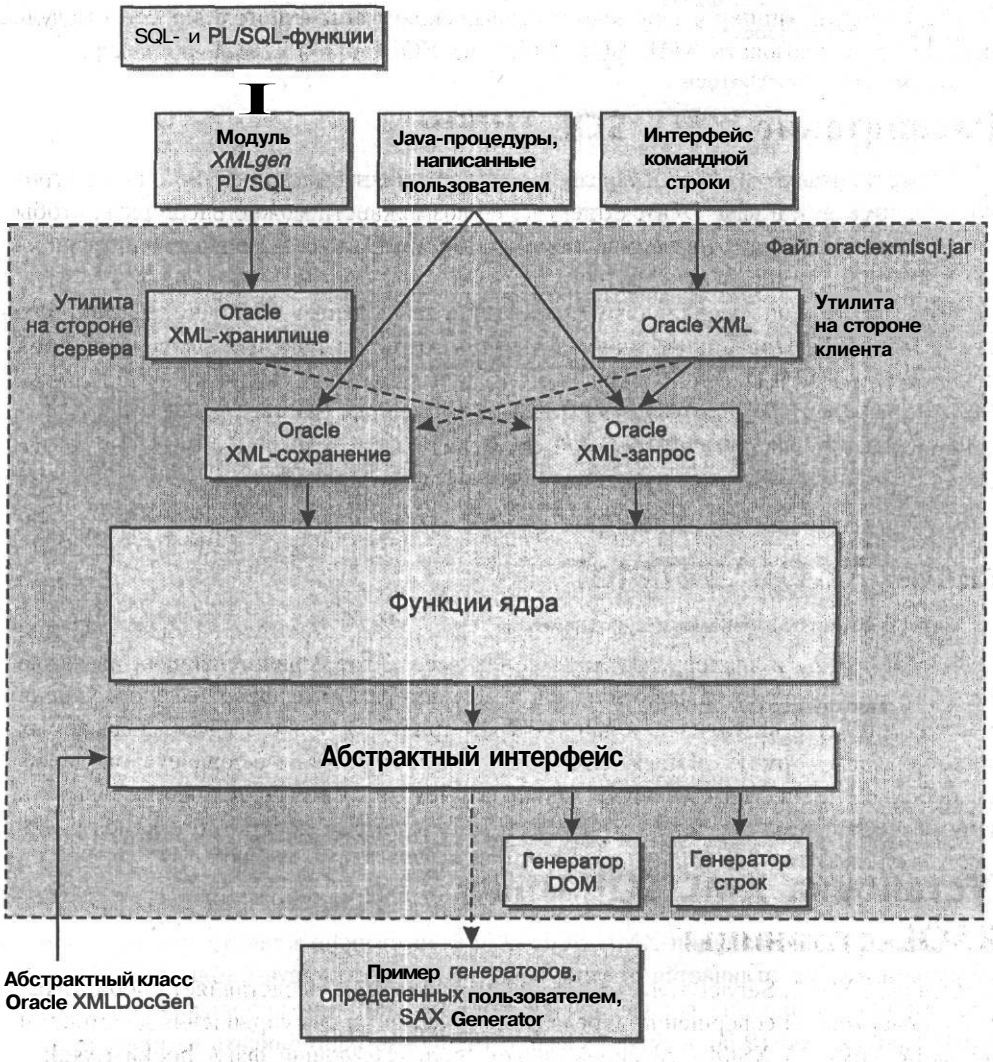


Рис. 8.1. *Архитектура XML SQL Utility*

с виртуальной Java-машиной JServer стандарта Java 1.1, и для нее можно загрузить файл xsu111.jar, а для версии 8.1.7, которая поддерживает стандарт Java 1.2, нужен файл **xsu12.jar**. В этой конфигурации JDBC-интерфейс работает в памяти, что существенно сокращает время извлечения из базы больших XML-документов.

Чтобы помочь пользователю с установкой XML SQL Utility с СУБД Oracle8i, в комплект поставки входят файлы **oraclexmlsqlload.bat** для компьютеров, работающих на базе Windows, и **oraclexmlsqlload.csh** для ОС Unix. После запуска эти скрипты загружают все классы из **oraclexmlsql.jar** за исключением класса командной строки OracleXML. Названные скрипты также загрузят анализатор XML-синтаксиса для Java v2 и запустят на исполнение скрипт xmlgensql.sql, который создает PL/SQL-модуль **xmlgen** и запускает тестовый скрипт **oraclexmltest.sql**. Этот модуль позволяет использовать XML SQL Utility из SQL- или PL/SQL-процедур.

Расширение XML SQL Utility

Хотя утилита XML SQL Utility сейчас поддерживает вывод информации и в строковом формате, и в виде DOM-объектов, ее возможности можно расширить, чтобы она поддерживала и другие формы, в том числе SAX. На рис. 8.1 показана архитектура XML SQL Utility в виде функциональной блок-схемы.

Функции ядра по генерации XML-документа заключены в оболочку абстрактного слоя **OracleXMLDocGen**. В текущей версии этот абстрактный класс расширен классом **OracleXMLDocGenDOM** для генерации выходных XML-документов в виде DOM-объектов и классом **OracleXMLDocGenString** для генерации выходных XML-документов в строковом формате. Для поддержки других представлений класс OracleXMLDocGen можно расширить дополнительными классами.

Oracle XSQL Servlet

XSQL Servlet, впервые представленный в главе 3, строится на базе функций XML SQL Utility и анализатора XML-синтаксиса. Этот сервлет написан на языке Java, он предоставляет разработчикам и Web-мастерам интерфейс высокого уровня для динамической визуализации данных в Интернете, а также в других задаваемых пользователем форматах. Oracle XSQL Servlet может работать с сервлетными движками большинства Web-серверов. Он позволяет автоматически преобразовывать данные в тот формат, который лучше всего подходит для конкретного клиентского браузера и конкретной платформы.

XSQL-страницы

Основой XSQL Servlet является XSQL-страница. Она представляет собой XML-файл, содержащий совершенно определенные элементы для управления действиями сервлета. Простой XSQL-страницей может быть следующий файл **booklist.xsql**:

```
<?xml version="1.0"?>
<xsql-stylesheet type="text/xsl" href="booklist.xsl"?>
<xsql:query connection="demo" xmlns:xsql="urn:oracle-xsql">
   select * from Booklist
</xsql:query>
```

На рис. 8.2 показана последовательность технологических операций, которые выполняются, когда Интернет-браузер запрашивает страницу, а Web-сервер передает этот запрос на XSQL Servlet после регистрации расширения **xsql** на сервере. Затем сервлет направляет страницу в анализатор XML-синтаксиса для возвращения его команд. В этом случае сначала отправляется команда открыть JDBC-соединение с псевдонимом **demo,** а потом — запрос "Select * from Booklist". Сервлет выполняет

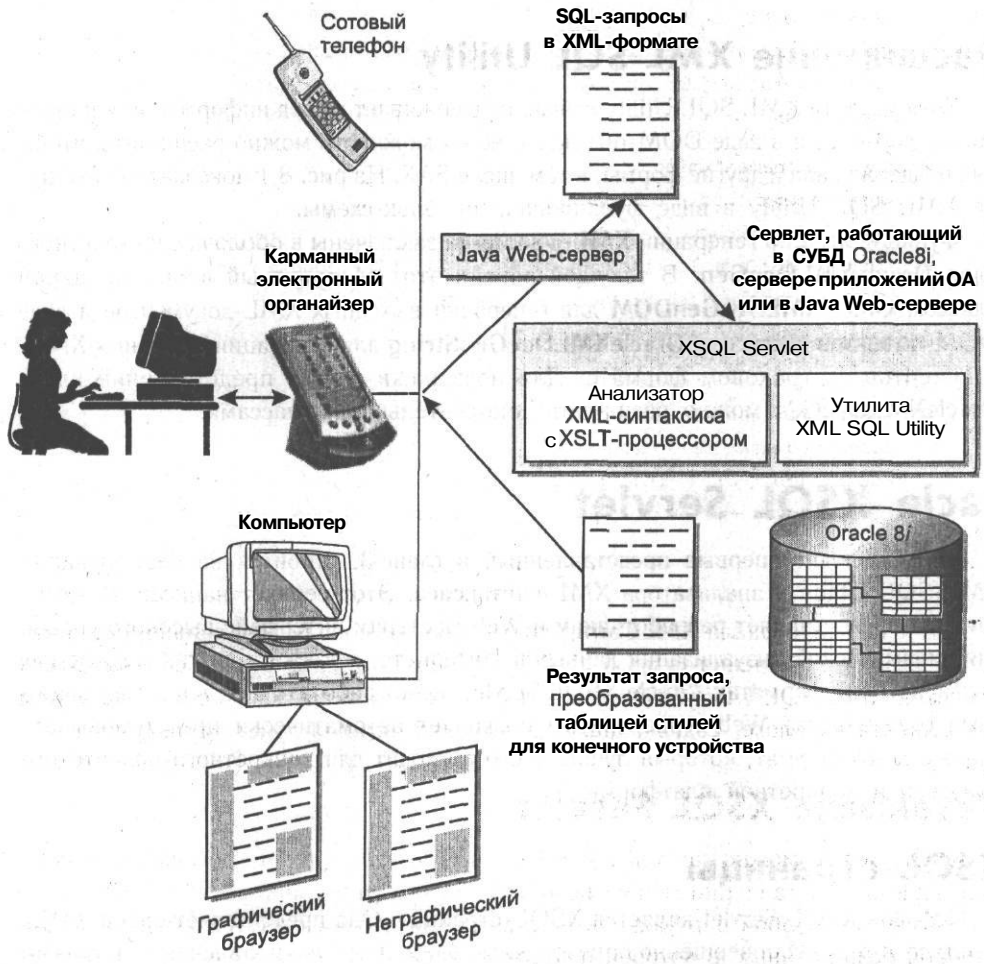


Рис. 8.2. *Процесс XSQL-страницы*

эти команды, передавая данные в утилиту XML SQL Utility, которая выполняет запрос, как было описано выше, и возвращает результат в виде объекта XML DOM. После чего сервлет передает ссылку на таблицу стилей и объект DOM в XSLT-процессор анализатора XML-синтаксиса, чтобы преобразовать его в HTML-формат для вывода на экран клиентского браузера.

Самым важным элементом вышеописанного файла является элемент **<xsql:query>**, содержащий информацию о связи с базой данных в своем атрибуте connection и SQL-запрос в его теле. Значение атрибута connection — это псевдоним, который находится в секции <connectiondefs> файла **XMLConfig.xml**:

```
<connectiondefs>
   <connection name="demo">
      <username>scott</username>
      <password>tiger</password>
      <dburl>jdbc:oracle:thin:@localhost:1521:ORCL</dburl>
      <driver>oracle.jdbc.driver.OracleDriver</driver>
   </connection>

   <connection name="xmlbook">
      <username>xmlbook</username>
      <password>xmlbook</password>
      <dburl>jdbc:oracle:thin:@localhost:1521:ORCL</dburl>
      <driver>oracle.jdbc.driver.OracleDriver</driver>
   </connection>

   <connection name="lite">
      <username>system</username>
      <password>manager</password>
      <dburl>jdbc:Polite:POlite</dburl>
      <driver>oracle.lite.poljdbc.POLJDBCDriver</driver>
   </connection>
</connectiondefs>
```

В этом фрагменте файла XMLConfig.xml, используемого по умолчанию, есть описание строки соединения с БД и JDBC-драйвера, которые будут использоваться утилитой XML SQL Utility. Так как этот файл хранится на сервере в недоступном для клиента каталоге, содержащаяся в нем информация будет защищенной.

Установка XSQL Servlet

XSQL Servlet можно загрузить с сайта OTN по адресу **technet.oracle.com/tech/xml**. При его инсталляции можно использовать целый ряд настроек. XSQL Servlet используется с любой виртуальной Java-машиной версий от 1.1 и выше, а также с любой базой данных, в которой поддерживается JDBC-интерфейс. XSQL Servlet прошел тестирование для работы с инструментарием разработчика JDK версий 1.1.8

и 1.2.2 для Windows и некоторых Unix-платформ, в том числе Solaris, Linux и HP-UX. Вот список поддерживаемых и протестированных сервлетных движков:

- Allaire JRun 2.3.3
- Apache 1.3.9 с JServ 1.0 и 1.1
- Apache 1.3.9 с Tomcat 3.1 Beta Servlet Engine
- Apache Tomcat 3.1 Beta 1 Web Server + Servlet Engine
- Caucho Resin 1.1
- NewAtlanta ServletExec 2,2 for IIS/PWS 4.0
- Oracle8i Lite Web-to-Go Server
- Oracle Application Server 4.0.8.1 (с заплаткой "JSP Patch")
- Oracle8i 8.1.7 Beta "Aurora" Servlet Engine
- Sun JavaServer Web Development Kit (JSWDK) 1.0.1 Web Server

Подробные инструкции по установке XSQL Servlet и демонстрационные примеры можно найти в главе 9.

Составление запросов к XSQL Servlet

XSQL Servlet предназначается для создания динамических Web-страниц из запросов к базе данных. XSQL-страницы можно связывать с любым Web-сайтом. Они могут содержать один и более запросов, результаты которых будут заменять соответствующую секцию **<xsql:query>** на странице. Затем полученные результаты можно определенным образом отсортировать, используя для этого атрибуты тега **<xsql:query>**. В таблице 8.1 представлены все доступные варианты настройки ответов на запросы.

Таблица 8.1

Опции атрибутов элемента <xsql:query>

Имя атрибута	Значение по умолчанию	Описание
rowset-element	<ROWSET>	Имя элемента для результатов запроса. Чтобы отключить распечатку элемента документа, значение этого атрибута устанавливается равным пустой строке.
row-element	<ROW>	Имя элемента для каждой строки в результатах запроса. Чтобы отключить распечатку элемента строки, значение этого атрибута устанавливается равным пустой строке.

Таблица 8.1 (*продолжение*)

Опции атрибутов элемента <xsql:query>

Имя атрибута	Значение по умолчанию	Описание
max-rows	Выбрать все строки	Максимальное число строк, выбираемых из ответа на запрос. Удобно использовать для выборки из результата запроса *N* верхних строк или (в комбинации с атрибутом skip-rows) *N* следующих строк.
skip-rows	Не пропускать ни одной строки	Количество строк, которые нужно пропустить перед возвращением результатов запроса.
id-attribute	id	Имя атрибута для идентификационного атрибута для каждой строки в ответе на запрос.
id-attribute-column	Номер строки	Имя столбца для передачи значения идентификационного атрибута для каждой строки в ответе на запрос.
null-indicator	Пропустить элементы, имеющие значение NULL	Если установить значение этого атрибута равным "y" или "yes", то атрибут null-indicator будет использоваться только в тех элементах любого столбца, которые имеют значение NULL
tag-case	Использовать регистр, в котором записано имя столбца или псевдоним из запроса.	Если значение этого атрибута установить равным upper, то в результатах запроса имена элементов для столбцов будут записываться прописными буквами.

При передаче запроса по протоколу HTTP в него также можно передать параметры. Если перед именем параметра поставить значок @, то XSQL Servlet будет искать соответствующие ему параметры в HTTP-запросе, а затем в атрибутах **<xsql:query>**. При обнаружении соответствия выполняется лексическая замена. Вот пример XSQL-страницы, использующей упомянутую функцию, когда строка HTTP-запроса имеет вид **http://localhost/xsql/demo/booksearch.xsql?year=2001**:

```
<?xml version="1.0"?>
<xsql:query xmlns:xsql="urn:oracle-xsql" connection="demo"
   SELECT TITLE, AUTHOR, DESCRIPTION FROM BOOKLIST
      WHERE YEAR = {@year}
</xsql:query>
```

Запросы, в ответ на которые не приходит ни одной строки, также можно обрабатывать, добавляя необязательный элемент **<xsql:no-rows-query>** внутрь тегов

<xsql:query>. Тогда пользователь вместо сообщения об ошибке увидит сформатированную страницу. Ниже представлен пример запроса, по которому сначала производится попытка извлечения списка книг с заданным именем автора, а если это не удается, делается попытка найти неточное соответствие введенному имени:

```
<?xml version="1.0"?>
<xsql:query xmlns:xsql="urn:oracle-xsql" connection="demo"
      SELECT TITLE, AUTHOR, DESCRIPTION FROM BOOKLIST
         WHERE AUTHOR = UPPER ('{@author}')

   <xsql:no-rows-query>
      SELECT TITLE, AUTHOR, DESCRIPTION FROM BOOKLIST
         WHERE AUTHOR LIKE UPPER ('%{@author}%')
         ORDER BY AUTHOR
   </xsql:no-rows-query>

</xsql:query>
```

Преобразование результатов XSQL-запроса с помощью таблиц стилей

Реальная мощь XSQL Servlet состоит в его способности динамически преобразовывать результаты запросов с помощью XSL-таблиц стилей. Описание таблицы стилей включается в XSQL-файл. Оно применяется только один раз — когда в ответ на запрос приходит ответ в формате XML. Обычно, как это показано в следующем примере, результаты запросов преобразуются в HTML-формат, но вообще-то таблица стилей может выполнять любые текстовые преобразования. На рис. 8.3 показан результат запроса и последующее преобразование.

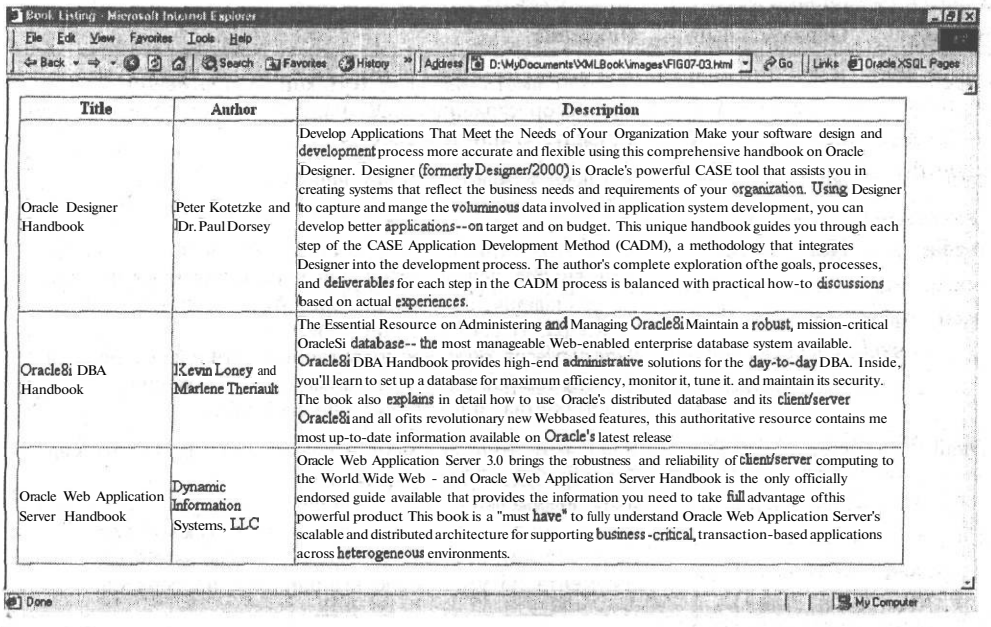
Приведенная ниже XSQL-страница и связанная с ней таблица стилей вернут результат запроса в виде HTML-таблицы:

Searchauthor.xsql

```
<?xml version="1.0"?>
<xsql-stylesheet type="text/xsl" href="totable.xsl"?>
<xsql:query xmlns:xsql="urn:oracle-xsql" connection="demo"
      SELECT Title, Author, Description FROM Booklist
         WHERE Author = UPPER ('@author')

   <xsql:no-rows-query>
      SELECT Title, Author, Description FROM Booklist
         WHERE Author LIKE UPPER ('%@author%')
         ORDER BY Author
   </xsql:no-rows-query>

</xsql:query>
```

Totable.xsl

```
<html xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
   <head>
      <title>Book Listing</title>
   </head>
   <body>
      <table border="1" cellspacing="0">
            <tr>
               <th><b>Author</b></th>
               <th><b>Title</b></th>
               <th><b>Description</b></th>
            </tr>
         <xsl:for-each select="ROWSET/ROW">
            <tr>
               <td><xsl:value-of select="Title"/></td>
               <td><xsl:value-of select="Author"/></td>
               <td><xsl:value-of select="Description"/></td>
            </tr>
         </xsl:for-each>
      </table>
   </body>
</html>
```



Title	Author	Description
Oracle Designer Handbook	Peter Kotetzke and Dr. Paul Dorsey	Develop Applications That Meet the Needs of Your Organization Make your software design and development process more accurate and flexible using this comprehensive handbook on Oracle Designer. Designer (formerly Designer/2000) is Oracle's powerful CASE tool that assists you in creating systems that reflect the business needs and requirements of your organization. Using Designer to capture and mange the voluminous data involved in application system development, you can develop better applications--on target and on budget. This unique handbook guides you through each step of the CASE Application Development Method (CADM), a methodology that integrates Designer into the development process. The author's complete exploration ofthe goals, processes, and deliverables for each step in the CADM process is balanced with practical how-to discussions based on actual experiences.
Oracle8i DBA Handbook	Kevin Loney and Marlene Theriault	The Essential Resource on Administering and Managing Oracle8i Maintain a robust, mission-critical OracleSi database-- the most manageable Web-enabled enterprise database system available. Oracle8i DBA Handbook provides high-end administrative solutions for the day-to-day DBA. Inside, you'll learn to set up a database for maximum efficiency, monitor it, tune it. and maintain its security. The book also explains in detail how to use Oracle's distributed database and its client/server Oracle8i and all ofits revolutionary new Webbased features, this authoritative resource contains me most up-to-date information available on Oracle's latest release
Oracle Web Application Server Handbook	Dynamic Information Systems, LLC	Oracle Web Application Server 3.0 brings the robustness and reliability of client/server computing to the World Wide Web - and Oracle Web Application Server Handbook is the only officially endorsed guide available that provides the information you need to take full advantage ofthis powerful product This book is a "must have" to fully understand Oracle Web Application Server's scalable and distributed architecture for supporting business-critical, transaction-based applications across heterogeneous environments.

Рис. 8.3. *Сформатированныерезультаты запроса страницы SearchAuthor.xsql*

Поддерживается также описание нескольких таблиц стилей. В этом случае XSQL Servlet выбирает метод преобразования путем поиска соответствия между строкой **user-agent** в HTTP-заголовке и значением необязательного атрибута **media** в элементе **<xml-stylesheet>**. При проверке соответствия регистр букв не имеет значения. В итоге метод преобразования выбирается на основании первого попавшегося при разборе файла соответствия строки **user-agent** и атрибута **media.** В примере ниже показано, как реализуется поддержка нескольких браузеров.

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" media="lynx" href="booklist-lynx.xsl" ?>
<?xml-stylesheet type="text/xsl" media="msie" href="booklist-ie.xsl" ?>
<?xml-stylesheet type="text/xsl" href="booklist.xsl"?>

<xsql:query connection="demo" xmlns:xsql="urn:oracle-xsql"
   select * from BOOKLIST
</xsql:query>
```

Отметим, что последнее описание таблицы стилей не имеет атрибута **media.** Его можно применять к любому HTTP-запросу, который не соответствует остальным таблицам стилей, т. е. оно применяется по умолчанию. В таблице 8.2 представлены разрешенные к употреблению атрибуты, которые можно добавлять к элементу **<xml-stylesheet>**, и их функции.

Таблица 8.2

Варианты значений атрибутов для элемента <xml:stylesheet>

Атрибут	Обязательный	Описание
type	Да	Должен иметь значение **text/xml**, в противном случае процессор страниц XSQL Page Processor проигнорирует команду **<?xml-stylesheet?>**.
href	Да	Указывает, какой следует использовать URL-адрес таблицы стилей, относительный или абсолютный.
media	Нет	Если этот атрибут есть, его значение используется для выполнения нечувствительной к регистру букв проверки на совпадение со строкой User-Agent запрашиваемого браузера. В соответствии с результатами этой проверки используется определенная таблица стилей, зависящая от запрашивающего программного приложения и устройства, на котором оно работает.
client	Нет	Если этот атрибут имеет значение yes, на клиентский браузер будет загружаться необработанный XML-документ, а для его обработки браузером будет использоваться команда **<?xml-stylesheet?>**. Если этот атрибут не определен, преобразование документа по умолчанию выполняется на сервере.

Последний метод применения таблицы стилей состоит в передаче ее URL-адреса в качестве HTTP-параметра, например:

```
http://localhost/yourdatapage.xsql?paraml=value&xml-stylesheet=yourstyle.xsl
```

Этот метод особенно полезен при создании прототипов и разработке приложений. Замена URL-адреса таблицы стилей значением **none** дает гарантию передачи необработанного XML-документа без применения к нему таблицы стилей.

Вставка XML-документов с помощью XSQL Servlet

Используя все возможности утилиты XML SQL Utility, можно создать XSQL-страницу, которая будет вставлять XML-документы в базу данных. XML-документ можно направить в класс **OracleXMLSave** утилиты XML SQL Utility, используя для этого элемент Action Element **<xsql:insert-request>**. Как уже говорилось, для сохранения документа в базе данных должна существовать схема. На первый взгляд это может показаться каким-то ограничением, однако способность сервлета XSQL Servlet применять таблицу стилей к представленному XML-документу обеспечивает необходимые функциональные возможности для фильтрации и преобразования документов.

Возвращаясь к примеру со списком книг, приведенному в этой главе выше, можно создать XSQL-страницу, которая будет принимать списки книг не только в заранее заданном формате базы данных, но и почти в любом текстовом формате. Например, продавец книг хочет вывести на экран или на печать список имеющихся в его магазине книг, но в этих списках используется набор тегов, отличный от тегов в схеме базы данных. Это можно сделать, создав соответствующую XSL-таблицу стилей и применив ее к имеющимся спискам. Следующая XSQL-страница может принимать информацию о книгах из хранилища Джо (Joe's Book) через HTTP и преобразовывать эту информацию в схему базы данных Booklist, применив к ней таблицу стилей **joebooks.xsl**, после чего результирующий XML-файл вставляется в класс **OracleXMLSave:**

```
<?xml version="1.0">
<xsql:insert-request xmlns:xsql="urn:oracle-xsql"
            connection="demo"
            table="BOOKLIST"
            transform="joesbooks.xsl"/>
```

Однако для того, чтобы этот фрагмент кода работал должным образом, нужно создать еще один элемент. База данных генерирует столбец BookID, который нужно создать для каждого вставляемого в базу данных документа. Это можно сделать, установив для таблицы Booklist триггер, генерирующий идентификатор ID всякий раз при вставке данных в таблицу. При добавлении каждого нового списка операцию

создания нового идентификатора BookID будет выполнять следующий SQL-скрипт (при условии, что уже создана последовательность **bookid_seq**):

```
CREATE TRIGGER booklist_autoid
BEFORE INSERT IN BOOKLIST FOR EACH ROW
BEGIN
   SELECT bookid_seq.nextval
   INTO :new.BookID
   FROM dual;
END;
```

Другие элементы Action Element, поддерживаемые сервлетом XSQL Servlet, перечислены в таблице 8.3.

Таблица 8.3

Элементы Action Element и их функции при создании XSQL-страниц

Action Element	Описание
<xsql:query>	Исполняет произвольный SQL-оператор и приводит результат этого действия к каноническому XML-формату.
<xsql:dml>	Исполняет SQL DML-оператор или анонимный PL/SQL-блок.
<xsql:set-stylesheet-param>	Устанавливает значение параметра XSQL-таблицы стилей верхнего уровня. Это значение параметра можно установить, задав необязательный параметр **value** или включив в содержимое элемента SQL-оператор.
<xsql:insert-request>	Вставляет полученный в результате запроса XML-документ (возможно, преобразованный) в таблицу базы данных. Если запрос был составлен в HTML-формате, результирующий XML-документ составляется на основе полученных по HTTP-запросу параметров, cookies и переменных сеанса связи с базой данных.
<xsql:inlude-xml>	Включает произвольные XML-ресурсы в любую точку страницы с использованием относительного или абсолютного URL-адреса.
<xsql:inlude-request-params>	Включает в XSQL-страницу такую важную информацию, как HTTP-параметры, переменные значения сеанса связи и cookies для ее последующего использования в таблице стилей.
<xsql:include-xsql>	Включает результаты одной. XSQL-страницы в любую точку внутри другой XSQL-страницы.
<xsql:include-owa>	Включает результаты исполняемой хранимой процедуры, использующей модули *Oracle Web Agent* (*OWA*), в базу данных для генерации XML-документа.

Таблица 8.3 (*продолжение*)

Элементы Action Element и их функции при создании XSQL-страниц

Action Element	Описание
<xsql:action>	Вызывает определенный пользователем Java-обработчик для исполнения заданной логической схемы и включения заданной XML-информации в XSQL-страницу.
<xsql:ref-cursor-function>	Включает каноническое XML-представление набора результатов указателя, возвращаемого хранимой процедурой PL/SQL.
<xsql:set-page-param>	Устанавливает параметр уровня страницы (локальный), который можно включить в последующие SQL-операторы на станице. Это значение можно установить, используя статическое значение, значение другого параметра или результаты исполнения SQL-оператора.
<xsql:include-param>	Включает параметр и его значение как элемент в XSQL-страницу.
<xsql:set-session-param>	Устанавливает параметр HTTP-сеанса связи с базой данных. Это значение можно установить, используя статическое значение, значение другого параметра или результаты исполнения SQL-оператора.
<xsql:set-cookies>	Устанавливает HTTP cookies. Это значение можно установить, используя статическое значение, значение другого параметра или результаты исполнения SQL-оператора.
<xsql:insert-param>	Вставляет значение одного параметра, содержащего XML. Его также можно использовать для приведения преобразования к каноническому формату.

Обновление данных с помощью XSQL Servlet

Во многих приложениях требуется, чтобы данные или документы не заменялись полностью, а обновлялись лишь в тех частях, где происходят какие-то изменения. Для выполнения этой операции используется триггер, как это делается для автоматической генерации BookID.

В состав Oracle8i включен триггер INSTEAD OF, позволяющий вызывать хранимые PL/SQL- и Java-процедуры всякий раз, когда встречается оператор INSERT. Эти триггеры используют представления Oracle8i Object View, связанные с оператором INSERT.

Например, если таблица Booklist должна была обновляемой, сначала можно организовать поиск уникального сочетания названия книги и ее автора, и в случае обнаружения такового выполнить операцию UPDATE вместо INSERT. Для этого

нужно создать представление Object View, соответствующее таблице Booklist, что можно сделать с помощью следующих SQL-команд:

```
CREATE VIEW Booklistview AS
SELECT * FROM Booklist
```

Затем нужно создать триггер и связать его с представлением. В примере, приведенном ниже, для этого используется PL/SQL, но ту же самую операцию можно произвести с помощью Java хранимой процедуры.

```
CREATE OR REPLACE TRIGGER insteadOfIns_booklistview
INSTEAD OF INSERT ON booklistview FOR EACH ROW
DECLARE
   notThere BOOLEAN := TRUE;
   tmp      VARCHAR2(1);
   CURSOR chk IS SELECT 'x'
                 FROM Booklist
                 WHERE Title = :new.title
                 AND Author = :new.author;
BEGIN
   OPEN chk;
   FETCH chk INTO tmp;
   notThere := chk%NOTFOUND;
   CLOSE chk;
   IF notThere THEN
      UPDATE INTO Booklist (Title,
                            Author,
                            Publisher,
                            Year,
                            ISBN,
                            Description)
      VALUES (:new.title,
              :new.author,
              :new.Publisher,
              :new.Year,
              :new.ISBN,
              :new.Description);
   END IF;
END;
```

Наконец, нужно изменить XSQL-файл таким образом, чтобы обновлять не таблицу Booklist, а представление Booklistview.

```
<?xml version="1.0">
<xsql:insert-request xmlns:xsql="urn:oracle-xsql"
                     connection="demo"
```

```
table="Booklistview"
transform="joebooks.xsl"/>
```

Последнее замечание. Так как на уникальность проверяется комбинация названия книги и имени ее автора, то для ускорения проверки и повышения общей производительности можно создать специальный индекс с помощью следующего *SQL-оператора:*

```
CREATE UNIQUE INDEX booklist_index ON booklist (Title, Author);
```

Предыдущий пример объясняет, как провести обновление с помощью триггерной функции представления объекта, но для этого можно использовать функцию обновления утилиты XML SQL Utility, сервлет XSQL Servlet и следующий очень простой **.xsql** файл:

```
<?xml version="1.0"?>
<xsql:dml connection="demo" xmlns:xsql="urn:oracle-xsql">
   update Booklist set status='S' where BookID = '1';
</xsql:dml>
```

Этот пример иллюстрирует простоту и мощь XML-интерфейса языка XSQL.

Web-сайт, усиленный поддержкой технологии XML

На традиционных Web-сайтах приходится производить переформатирование всего содержимого для ожидаемого размера экрана, и практически нет никаких функций автоматического детектирования клиента и соответствующей настройки представляемой информации. Некоторые разработчики пытаются решить проблему настройки браузеров, предоставив пользователю выбор одного из вариантов представления сайта: с фреймами (frames), без фреймов (non-frames), только *в* виде текста (text) или текста с картинками (text+images). Такие страницы обычно хранятся в статическом виде и обновляются вручную, что чревато ошибками или пропусками данных. Любые изменения в форматировании нужно повторять на каждой странице.

Появление каскадных таблиц стилей существенно улучшило ситуацию, так как они позволяют выделить большую часть HTML-форматирования в отдельный файл. Однако остается зависимость от HTML-форматирования статического исходного файла. А многим компаниям, занимающимся электронной коммерцией, нужна высокая скорость доставки клиенту текущих данных, поэтому им необходимо реализовать доставку истинной на данный момент информации и ее динамическую обработку с помощью таблиц стилей для представления в окне браузера. Для решения задачи к Web-сайтам были добавлены базы данных масштаба предприятия, одним из примеров которых и является СУБД Oracle.

Решение с поддержкой технологии XML

В главе 3 демонстрировалось использование возможностей технологии XML для описания данных в структурированном виде, а также возможности технологии XSL для выполнения практически любого текстового преобразования данных. Реализуется это с помощью единого репозитория данных. Этот репозиторий может выполнять функции процессора базы данных для системы представления, будь то браузер, карманный компьютер, сотовый телефон или пейджер. Генерировать запросы к БД, в ответ на которые возвращается самая свежая информация, можно с помощью встроенных ссылок, а не путем обращения к статическим Web-страницам.

В базы данных можно помещать самую разную информацию, в том числе и сформатированную в виде XML-файлов, Здесь можно также хранить информацию, готовую для непосредственного использования. Допускается создание репозиториев, содержащих повторно используемые фрагменты документов или модули, которые можно по запросу собирать с помощью XSL-таблиц стилей, существенно упрощая онлайновую публикацию документов. Разделение содержимого и его представления также упрощает производственный процесс, поскольку разработчики представлений могут работать с таблицами стилей, не рискуя внести ошибки в хранящуюся информацию.

Реальным примером такого Web-сайта является сайт по торговле недвижимостью, где собрана информация от нескольких *MLS-служб* (multiple listing services), в которых списки домов на продажу от отдельных агентов объединены в один большой список. Такой Web-сайт принимает MLS-листинги от нескольких источников, каждый из которых может иметь свой собственный XML-формат. К каждому такому XML-источнику применяется своя XSL-таблица стилей, чтобы преобразовать его к единому формату для размещения в базе данных. Сводка этих данных представляется в запрашивающие браузеры, а из них можно вести поиск в предлагаемой информации или запрашивать расширенные листинги.

Конструирование требований

Для того чтобы правильно сконструировать Web-сайт, нужно выполнить два общих требования. Во-первых, БД должна принимать данные в XML-формате и выполнять их соответствующее XSL-преобразование. Кроме того, она должна динамически визуализировать листинги разными способами и в любых объемах без заметной потери производительности.

Во-вторых, система должна уметь обрабатывать несколько форматов изображений и выбирать соответствующий тип для вывода на дисплей запрашивающей системы или выводить информацию без графики в случае, если запрашивающей системой является сотовый телефон или пейджер. В идеале система должна не только отвечать на прямые запросы, но и генерировать сообщения в ответ на получение листингов в соответствии с первым требованием. Это особенно полезно, например, агентам по недвижимости, которые ищут новые листинги с предложениями о покупке или продаже дома, чтобы представить их клиентам.

Архитектура

Вышеизложенным требованиям идеально соответствует система, в которой в качестве репозитория используется база данных. Для одновременного хранения и обычных данных, и изображений можно взять СУБД Oracle8i. Для связи данных с их иллюстрациями создаются объектные представления. Такие полные листинги после XSL-преобразования с помощью соответствующей таблицы стилей направляются на Интернет-браузеры.

На рис. 8.4 показана общая архитектура сайта агентства по недвижимости. XML-данные из разных источников поступают в XSQL Servlet, где преобразуются к нормализованному формату и сохраняются в базе данных.

Клиенты с помощью браузера могут вести поиск в листингах или выводить на экран листинги, отобранные в соответствии с заранее заданными критериями. Формат этого представления автоматически приводится в соответствие с возможностями запрашивающего браузера. Кроме того, в случае изменения состояния листинга или его обновления на клиентские браузеры могут рассылаться соответствующие уведомления.

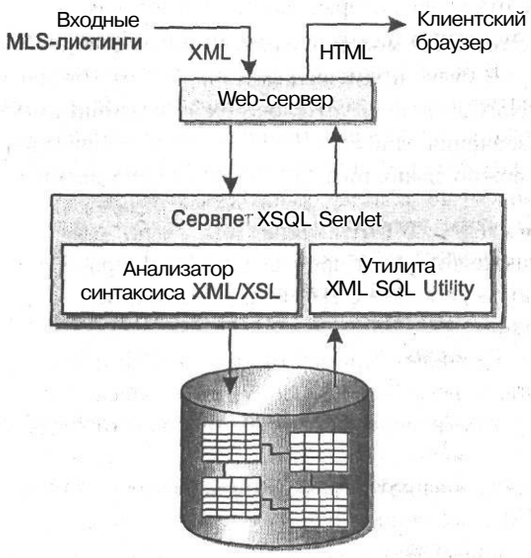


Рис. 8.4. *Функциональная диаграмма сайта Real Estate по торговле недвижимостью*

Пример реализации

Чтобы реализовать описанную выше систему, нужно создать нормализованную схему MLS-данных. Такую модель данных можно описать с помощью следующего DTD:

```
<!ELEMENT LISTING (ListingID, MLS_Number, Address, Area, City, State,
                   Zipcode, Description, AskPrice, Agent, ImageURI,
                   Category, Status)>
<!ELEMENT ListingID (#PCDATA)>
<!ELEMENT MLS_Number (#PCDATA)>
<!ELEMENT Address (#PCDATA)>
<!ELEMENT Area (#PCDATA)>
<!ELEMENT City (#PCDATA)>
```

```
<: ELEMENT State (#PCDATA)>
<! ELEMENT Zipcode (#PCDATA)>
<!ELEMENT Description (#PCDATA)>
<! ELEMENT AskPrice (#PCDATA)>
<! ELEMENT Agent (#PCDATA)>
<! ELEMENT ImageURI (#PCDATA)>
<! ELEMENT Category (SFH I CONDO I APTBLDG I DUP I THM I COM I LOT) >
<!ELEMENT Status (A I P I S I C)>
```

Этот DTD можно сохранить в качестве шаблона для создания таблиц стилей, которые будут применяться к листингам каждой MLS-службы. Например, если служба NorCal использует слово **Neighborhood** вместо Area и Zip вместо **Zipcode** для обозначения одних и тех же вещей, преобразование этих данных к единому знаменателю можно выполнить с помощью следующей таблицы стилей:

```
<?xml version="1.0"?>
<ROWSET xmlns:xsl=http: //www.w3.org/1999/XSL/Transform
        xsl :version="1.0">
  <xsl: for-each select="NorCal/LISTING">
  <ROW>
  ...
  <Area><xsl:value-of select="Neighborhood"/></Area>
  ...
  <Zipcode><<xsl:value-of select="Zip"/></Zipcode>
  ...
  </ROW>
 </xsl : for-each>
</ROWSET>
```

После того как созданы все таблицы стилей, можно подготовить XSQL-страницу, которая будет принимать информацию от каждой службы и производить соответствующее преобразование, как показано на следующей XSQL-странице, где используется Action Element **<xsql:insert-request>**, описанный выше в разделе "Вставка XML-документов с помощью XSQL Servlet".

```
<?xml version="1.0"?>
<xsql : insert-requestxmlns :xsql="urn:oracle-xsql"
            connection="demo"
            listingview="LISTING"
            transform="norcal .xsl"/>
```

XSQL-страницы также можно использовать на стороне клиента для составления запросов и создания различных страниц, например страницы с выборкой информации по полю **Area** или **Zipcode,** а также для детализации поиска с учетом приемлемого диапазона цен или типа недвижимости, как показано на рис. 8.5.

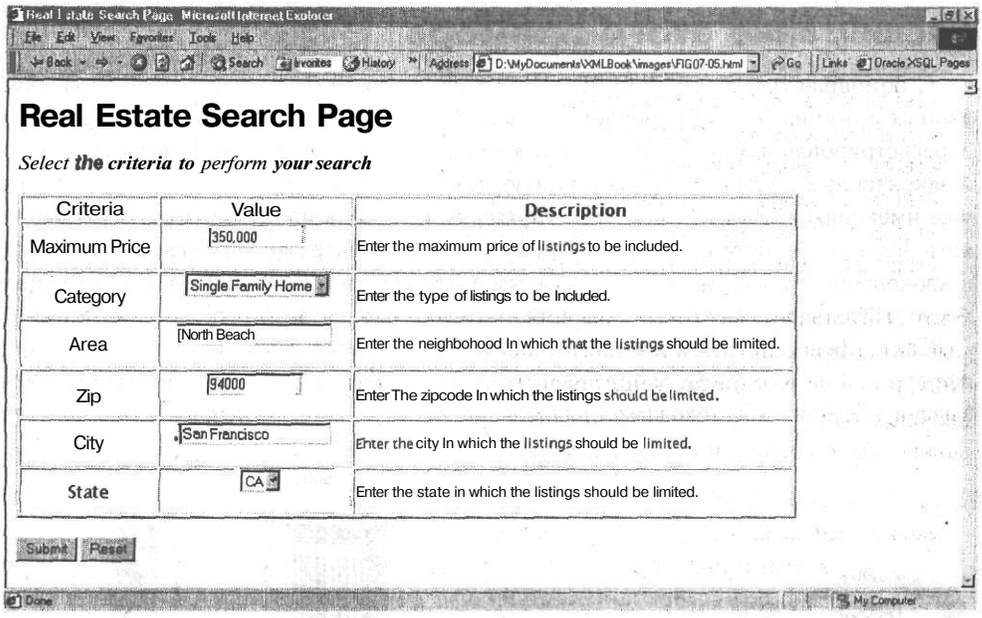


Рис. 8.5. *Пример страницы поиска в MLS-листинге*

Для этих операций поиска создаются ссылки на XSQL-страницы, выполняющие конкретные запросы, или используется метод **@parameter** для передачи критериев поиска в главную XSQL-страницу.

В последнее время заметно выросла популярность сотовых телефонов, карманных компьютеров и пейджеров, имеющих средства доступа в Web, поэтому наш сайт также должен удовлетворять специфичным требованиям вывода информации на небольшие экраны указанных устройств. Для этого в HTTP-заголовке используется строка **user-agent** и атрибут **media** в описании таблицы стилей. Например, в следующей XSQL-странице демонстрируется реализация поддержки карманного компьютера Palm Pilot, сотового телефона Unwired Planet и стандартных браузеров:

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" media="Palm" href="ListingPP.xsl"?>
<?xml-stylesheet type="text/xsl" media="Mozilla" href="ListingWMLMZ.xsl"?>
<?xml-stylesheet type="text/xsl" media="UP" href="ListingMZ.xsl"?>
<?xml-stylesheet type="text/xsl" href="Listing.xsl"?>

<xsql:query connection="demo" xmlns:xsql="urn:oracle-xsql">
   select * from Listingview WHERE
            Zipcode=94065
</xsql:query>
```

Расширение этого примера

С помощью триггера INSTEAD OF INSERT, описанного в разделе "Обновление данных с помощью XSQL Servlet", можно сделать так, чтобы Web-сайт уведомлял зарегистрированных пользователей о появлении новых листингов, о незавершенных и законченных сделках с недвижимостью, т. е. о вставке новых данных и обновлении уже имеющихся. Вызываемая этим триггером хранимая процедура может запрашивать нужную регистрационную таблицу адресов электронной почты и выполнять широковещательную рассылку по адресам пользователей, имеющих соответствующее место жительства, почтовый код или интересующихся конкретными листингами.

Сайт, предоставляющий полный спектр услуг, может также иметь финансовую информационную службу, чтобы предупреждать клиентов и агентов о подтверждении выдачи кредита, о необходимых для совершения сделки документах и прочем, что

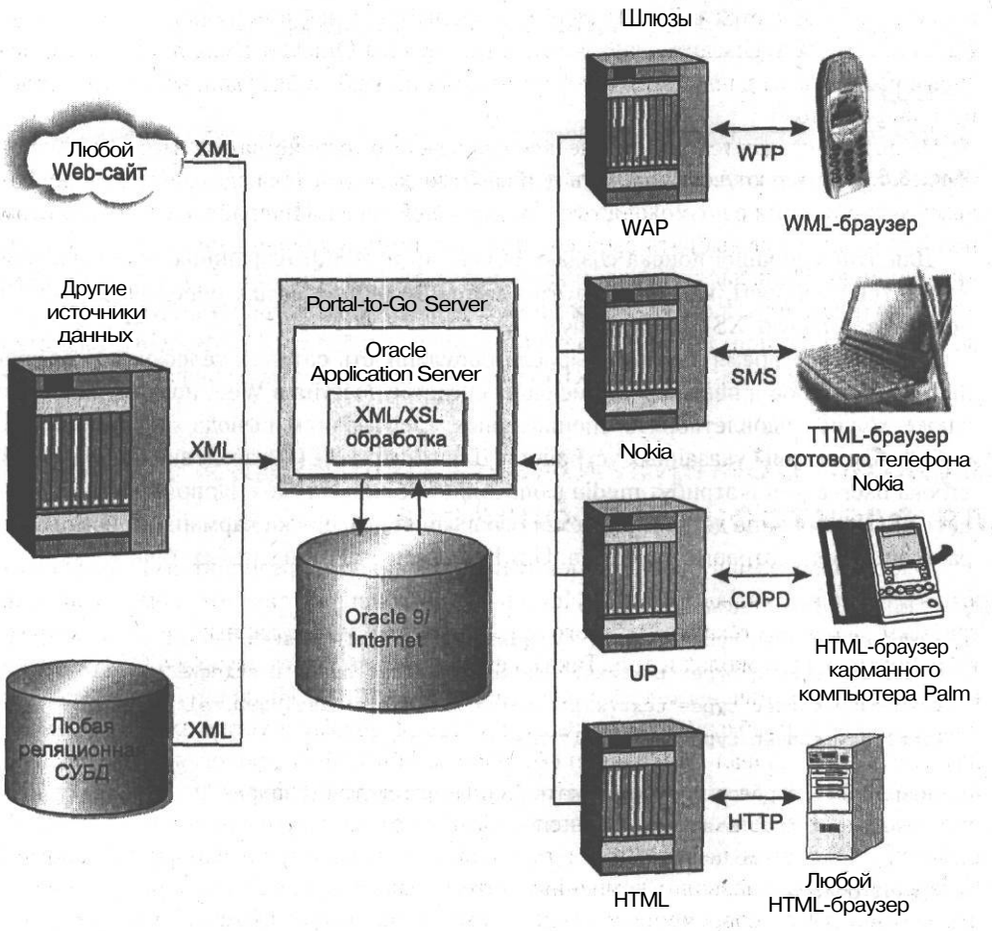


Рис. 8.6. *Архитектура Oracle Portal-to-Go*

может понадобиться при операциях с недвижимостью. Наконец, на этом сайте можно организовать службу, занимающуюся аукционными продажами домов, а такая служба предполагает прием ставок и уведомление участников торгов о сделанных ставках, причем должна поддерживаться возможность рассылки этих уведомлений на самые разные коммуникационные устройства.

Oracle Portal-to-Go

Почти все функции, описанные выше, уже реализованы в программном продукте Portal-to-Go производства Oracle. Изначально он был создан для провайдеров служб проводной и беспроводной связи. Portal-to-Go позволяет одному репозиторию обслуживать устройства нескольких типов, а для форматирования информации в соответствии с возможностями каждого устройства используются сетевые шлюзы.

На рис. 8.6 показана архитектура Portal-to-Go. Поступающая информация XML-формата направляется в БД Oracle8i и сохраняется в ней в нормализованном виде. Работающие на Web-сайте многочисленные службы Oracle9i Dynamic Services, которые возвращают данные в XML-формате, поставляют в базу данных дополнительную информацию.

На выходе создаются специфические для каждого устройства сетевые шлюзы, которые осуществляют взаимодействие с самыми разными проводными и беспроводными устройствами с возможностью двусторонней связи. При работе с таким сайтом пользователи могут выбрать свои собственные представления данных и нужную им информацию. Пользователи могут даже работать в отключенном режиме, который идеально подходит для обладателей многих беспроводных устройств. Для этого можно использовать портлеты — четко определенные области HTML/XML-разметки в Web-странице, предоставляющей базовые услуги доступа к информации.

Службы обмена XML-сообщениями для электронного бизнеса

Требования к обмену данными между приложениями и предприятиями увеличиваются буквально с каждым днем. Исторически сложилось так, что обмен данными требовал наличия технологического соглашения о типах данных, их структурах, транспортных протоколах и т. п. Такими функциями обладает служба CORBA, но ее возможности ограничены тем, что она используется в сети Интернет, и для передачи информации ей необходимо преодолевать сетевые шлюзы и устанавливать временные соединения. Идеальным решением проблемы была бы технология, позволяющая приложениям отправлять и принимать данные в текстовой форме по самым разным протоколам и самоописывающимся способом. Поэтому приложения для взаимодействия друг с другом не должны быть привязаны к какому-то набору технологий. В перспективе это позволит компаниям интегрировать приложения через Интернет, что соответствует современным тенденциям консолидации промышленности и открывает новые возможности для хостинга приложений.

Решение с поддержкой технологии XML

Рост популярности языка XML и связанных с ним технологий XSL, XML Schema, XML Query и XPath позволяет произвести реальную интеграцию приложений нескольких предприятий. Этим приложениям так или иначе приходится обмениваться структурированными данными, будь то заказ на покупку товаров или сообщение об обновлении информации в каталоге. При использовании данных в формате XML отпадает необходимость в метаданных, так как их структура четко определяется логикой приложения. Для подтверждения правильности сообщений эти данные можно включить в определения XML Schema или DTD. XSL-таблицы стилей могут преобразовывать сообщения в форматы, соответствующие конкретному приложению. И, наконец, вся эта работа может выполняться общими компонентами, созданными на базе открытых стандартов. Компании, разрабатывающие продукты на базе таких стандартов, могут быть уверены в том, что они будут широко использоваться и поддерживаться в индустрии.

Вернемся к примеру Booklist, который подходит на все случаи жизни, и создадим онлайновый книжный магазин, в котором будут использоваться разные XML-технологии. Исходить будем из того, что книжный магазин должен принимать листинги продаваемых книг и позволять клиентам покупать эти книги через Интернет. Кроме того, на этом же сайте должны быть собраны листинги с книгами, продаваемыми в других магазинах, чтобы привлечь клиентуру в национальном масштабе.

Требования к конструкции Web-сайта

Для того чтобы правильно сконструировать этот Web-сайт и приложение, нужно составить модели данных для тех объектов, которые предполагается использовать. Так как основным хранилищем информации будет БД, и именно здесь больше всего ограничений при моделировании, то конструирование сайта начнем с создания схемы базы данных. Сначала для генерации DTD используем утилиту XML SQL Utility. Затем генератор классов Class Generator преобразует эти DTD в DOM-классы, используемые на внешнем интерфейсе приложения для создания XML-документов, которые затем будут вставляться в базу данных с помощью XML SQL Utility.

Такая система должна принимать списки книг от индивидуальных продавцов и большие листинги от других книжных магазинов. Кроме того, нужно представлять книжные каталоги предполагаемым покупателям, обрабатывать транзакции продаж и уведомлять об этом продавцов. Наконец, нужно обрабатывать платежи, поступающие от покупателей и направляемые продавцам.

В требованиях к разработке нужно учесть общую производительность системы. Процесс извлечения данных в XML-формате масштабируется довольно хорошо, зато узкими местами системы могут стать сложные преобразования и требование, чтобы каждому экземпляру таблицы стилей соответствовал только один подпроцесс. Расшить эти узкие места можно путем минимизации обработки данных при использовании таблиц стилей и с помощью кэш-буферов промежуточного слоя, например Oracle iCache.

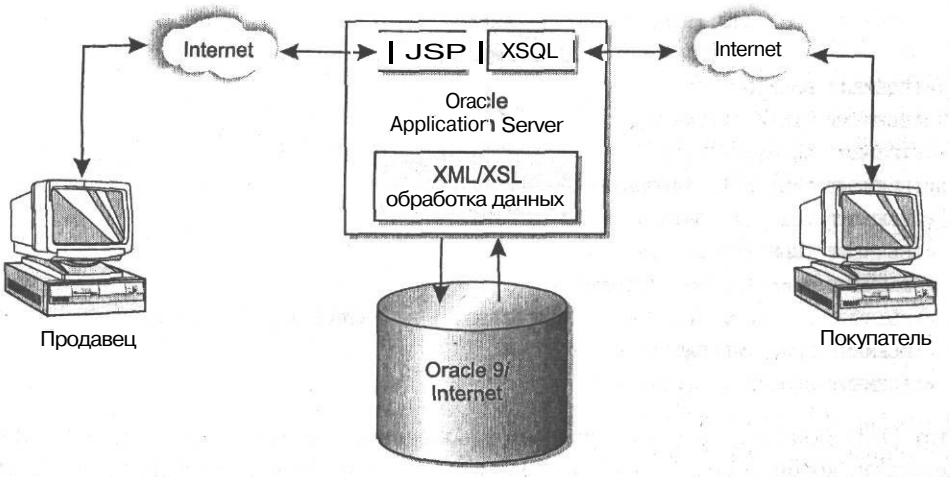


Рис. 8.7. *Функциональная схема онлайнового книжного магазина*

Архитектура

Вышеописанную систему можно реализовать с помощью СУБД Oracle8i, которая будет выполнять функции репозитория данных, и сервера приложений Oracle Application Server, который будет работать и как Web-сервер, и как ПО промежуточного слоя. Клиенты будут взаимодействовать с сайтом с помощью стандартных браузеров. Продавцы книг смогут отправлять на сайт листинги XML-формата и принимать уведомления о продажах. Функциональная схема такой системы показана на рис. 8.7.

Для того чтобы разместить каталог книг в этом магазине, продавец вводит данные в страницу Java Server Page (JSP), которая вызывает DOM-классы для генерации XML-документа. Затем производится синтаксический разбор этого документа, и его данные вставляются в схему Booklist. Кроме того, в схему Account помещаются бухгалтерские реквизиты продавца. Клиенты могут получить на экранах своих мониторов списки книг по категориям или в виде результатов поиска по каким-то критериям. Для этого им приходится иметь дело с HTML-страницами, генерируемыми сервлетом XSQL Servlet. При покупке книг клиентом транзакция фиксируется и сохраняется в БД, после чего продавцу по электронной почте отправляется уведомление.

Все это не является полным описанием приложения и его архитектуры, однако вышесказанное и примеры реализации разных компонентов этой системы могут дать хорошее представление об использовании технологии XML и стать основой создания аналогичных приложений.

Пример реализации

Рассмотрим особенности реализации онлайнового книжного магазина, расширив схему Booklist, представленную выше, и добавив к ней схему Client. Вот, например, XML DTD, который представляет расширенный каталог книг:

```
<!ELEMENT BOOKLIST (BookID, Title, Author, Publisher, Year, ISBN,
                    Description, Category. Cost, Status)>
<!ELEMENT BookID (#PCDATA)>
<!ELEMENT Title (#PCDATA)>
<!ELEMENT Author (#PCDATA)>
<!ELEMENT Publisher (#PCDATA)>
<!ELEMENT Year (#PCDATA)>
<!ELEMENT ISBN (#PCDATA)>
<!ELEMENT Description (#PCDATA)>
<!ELEMENT Category (FICTION I NONFICTION I REFERENCE I TECHNICAL)>
<!ELEMENT Cost (#PCDATA)>
<!ELEMENT Status (A I S)>
```

Этот DTD можно сохранить в виде входных данных для генератора классов Class Generator, который создаст набор DOM-классов, а те, в свою очередь, — должным образом сформатированные XML-документы. Вот пример командной строки:

```
Java samplemain booklist.dtd
```

Для всех элементов будут созданы исходные Java-файлы, которые затем можно скомпилировать в Java-классы с помощью компилятора Javac. Затем с помощью этих классов создается страница JSP; эту операцию выполняет программа:

```
<HTML>
   <HEAD>
      <TITLE>Book Submission</TITLE>
   </HEAD>
<BODY bgcolor = "#ffffff">
<%@ page import="java.io.*" %>
<%
String Title = request.getParameter("title");
String Author = request.getParameter("author");
String Publisher = request.getParameter("publisher");
String Year = request.getParameter("year")
String ISBN = request.getParameter("isbn");
String Description = request.getParameter("description")
String Category = request.getParameter("category");
String Cost = request.getParameter("cost")
String status = new String ("A");                    // A for available c
try {
   out.println(author + " " + title + * " + category + " " '+ isbn
               + « « + publisher + - " + year + * " + description
               + " " + cost +" "+ status);
   Booklist entry = new Booklist();
   oracle.xml.parser.v2.DTD dtd = entry.getDTDNode();
```

```
        Title t = new Title (title);
        Author a = new Author (author);
        Publisher p = new Publisher (publisher);
        Year y - new Year (year);
        ISBN i - new ISBN(isbn);
        Category c = new Category (category);
        Cost ct = new Cost (cost);
        Status st = new Status (status);

        entry.addNode(t);
        entry.addNode(a);
        entry.addNode(p);
        entry.addNode(y);
        entry.addNode(i);
        entry.addNode(c);
        entry.addNode(ct);
        entry.addNode(st);
        entry.print(newFileOutputStream("result.xml"));
    }
    catch (Exception e) {
        out.println(" ---- » + e + "..e.");
        e.printStackTrace(newPrintWriter(out));
    }
    %>
    </BODY>
    </HTML>
```

Эту JSP-страницу можно вызвать из HTML-страницы. Для этого используются команды языка JavaScript и интерфейс заполняемой на экране браузера формы. Пример кода такой HTML-страницы показан ниже, а вид на экране браузера продемонстрирован на рис. 8.8.

```
<html>
<head>
   <title>Book Submission Form</title>
</head>
<body bgcolor="#ffffff">
<script language="JavaScript">
   functioncheckRequiredFields () {
      var frmR = window.document.frmReport;
      if (frmR.title.value == "") (
         alert("Please enter a value for the title");
         return false;
      }
```

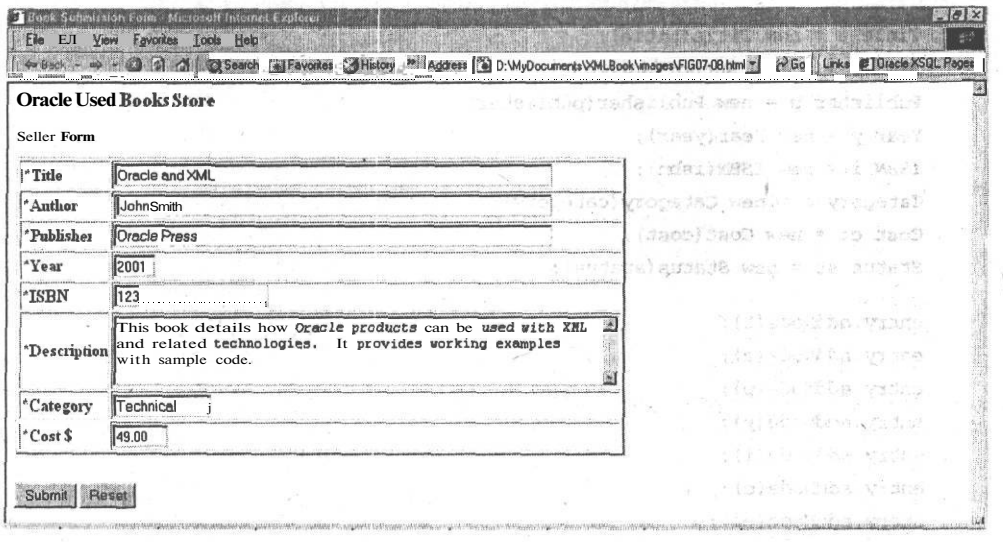


Рис. 8.8. *Пример формы с информацией о книге*

```
if (frmR.author.value == •*) {
   alert ("Please enter the author");
   return false;
}
if (frmR.publisher.value == "") {
   alert ("Please enter the publisher");
   return false;
}
if (frmR.year.value == "") {
   alert ("Please enter the Year");
   return false;
}
if (frmR.isbn.value == "") {
   alert ("Please enter the ISBN number");
   return false;
}
if (frmR.description.value == "") {
   alert ("Please enter the description");
   return false;
}
if (frmR.category.value == "") {
   alert("Please enter the category - Fiction, Nonfiction,
         Reference, or Technical");
   return false;
}
```

```
        if (frmR.isbn.cost == "") {
           alert ("Please enter the cost");
           return false;
        }
    frmR.action = "procSell.jsp";
    frmR.submit();
    return true;
}
/script>
</p>
<h3><img src="openbook.gif"> Oracle Used Books Store  
<a href=""><font size="-1"><i>Main</i></font></a></h3>
<br>
<br>
<br>
<h4>Seller Form</h4>
<form name="frmReport">
   <table width="90%" align="center">
      <tr>
         <tdxdiv align="left"><p><font color="red">*</font>Title</td>
         <tdxinput type="text" name="title" value size="60"></td>
      </tr>
      <tr>
         <tdxdiv align="left"><p><font color="red">*</font>Author</td>
         <tdxinput type="text" name="author" value size="60"></td>
      </tr>
      <tr>
         <tdxdiv align="left"><p><font color="red">*</font>Publisher</td>
         <tdxinput type="text" name="publisher" value size="60"></td>
      </tr>
      <tr>.
         <tdxdiv align="left"><p><font color="red">*</font>Year</td>
         <td><input type="text" name="year" value size="4"></td>
      </tr>
      <tr>
         <tdxdiv align="left"><p><font color="red">*</font>ISBN</td>
         <tdxinput type="text" name="isbn" value size="20"></td>
      </tr>
      <tr>
         <td<div align="left"><p><font color="red">*</font>
            Description</td>
         <tdxinput type="text" name="description" value size="200"></td>
      </tr>
```

```
        <tr>
          <td><div align="left"><p><font color="red">*</font>Category</td>
          <td><input type="text" name="category" value size="12"></td>
        </tr>
        <tr>
          <td><div align="left"><p><font color="red">*</font>Cost $</td>
          <td><input type="text" name="Cost" value size="6"></td>
        </tr>
      </table>
      <pxbr>
        <input type="button" value="Submit" onClick=
                "checkRequiredFields();">
        <input type="reset" value="Reset">
      </p>
    </form>
  </body>
  </html>
```

Далее приведен пример XML-документа, получаемого с помощью JSP-страницы и передаваемого в утилиту XML SQL Utility для вставки в базу данных:

```
<?xml version="1.0"?>
<Booklist>
    <BookID>001234</BookID>
  <Title>The Difference Between God and Larry Ellison: Inside Oracle
      Corporation</Title>
  <Author>Mike Wilson</Author>
  <Publisher>William Morrow and Co.</Publisher>
  <Year>1997</Year>
  <ISBN>0688144251</ISBN>
  <Description>Account of Larry Ellison</Description>
  <Category>Computer</Category>
  <Cost>30.00</Cost>
  <Status>A</Status>
</Booklist>
```

Для конструирования Web-страницы покупателя в начальном представлении запроса на книги определенной категории используется XSQL-страница, как это показано на рис. 8.9. Щелчок мышью по любой ссылке приведет к вызову конкретной страницы, т. е. сервлет, используя введенные параметры, может передать категорию книги в заглавную страницу.

```
<?xml version="1.0"?>
<xml-stylesheet type="text/xsl" href="categorydetail.xsl"?>
```

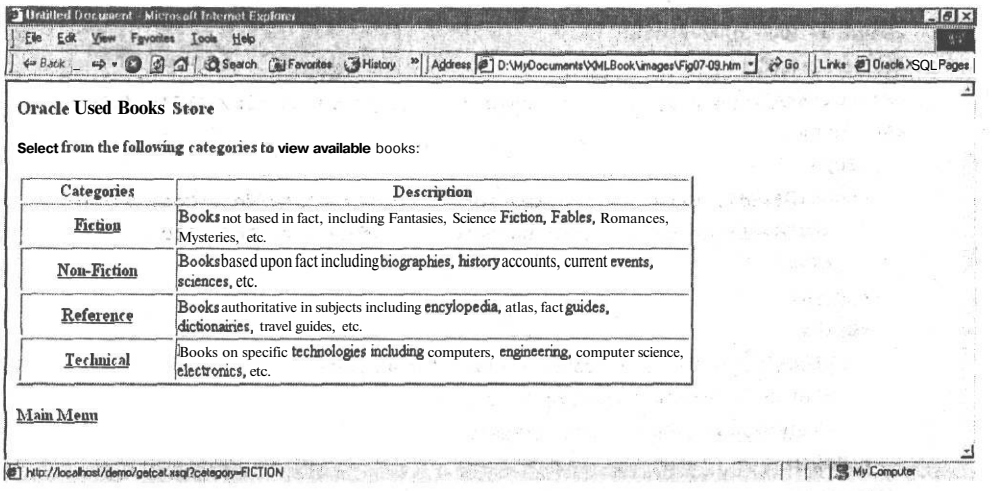


Рис. 8.9. *Страница представления каталога книжного магазина Bookstore по категориям книг*

```
<xsql:query xmlns:xsql="urn:oracle-xsql" connection="demo"
   SELECT Title, Author, Description, Cost FROM Booklist
      WHERE Category = FICTION
</xsql:query>
```

Затем таблица стилей **categorydetail.xsl** может сформатировать полученные листинги и добавить форму для оформления покупки. Благодаря поддержке функции обновления, подробно описанной выше, содержимое столбца Status (Состояние) изменится с *"A"* (available — доступно) на "*S*" (sold — продано), а функции триггерной процедуры можно расширить таким образом, чтобы она отправляла по электронной почте уведомление о продаже (для этого используется модуль **utl_smtp**).

Остается лишь создать форму для генерации счета и записать всю информацию о прохождении заказа и совершении покупки. Это можно сделать с помощью утилиты XML SQL Utility, генератора классов Class Generator и архитектуры JSP-страницы заказа книги. Реализацию этой задачи оставляем читателю, но начать ее можно со следующего XML-файла:

```
<Client>
   <ClientID>123456</ClientID>
   <Name>Mike Wilson</Name>
   <Address>123 Main St.</Address>
   <City>San Francisco</City>
   <State>CA</State>
   <Country>USA</Country>
   <Zipcode>94000</Zipcode>
   <Email>mwilson@anyhere.com</Email>
```

```
    <Date>03-MAY-2002</Date>
    <Time>22:00</Time>
    <Status>A</Status>
    <Account>
       <Buy>
       <BookID>103454</BookID>
          <Date>04-JUN-2003</Date>
          <Time>11:05</Time>
       </Buy>
       <Sell>
          <BookID>001234</BookID>
          <Date>03-MAY-2002</Date>
          <Time>22:00</Time>
       </Sell>
    </Account>
</Client>
```

Расширение этого примера

Вышеприведенный простой пример с книжным магазином можно легко распространить на самые разные области. На стороне клиента можно добавить функцию поиска по неточному соответствию, описанную выше. Хранимые процедуры Java и PL/SQL можно вызывать в службу авторизации для проверки кредитоспособности клиента и подтверждения номера кредитной карточки. При приеме листингов книг в XML-формате от других поставщиков сайт может действовать просто как сборщик информации, оставляя продавцам хлопоты по обработке поставок товара.

Хотя этот пример демонстрирует взаимоотношения между рядовым потребителем и компанией, ту же самую технологию и архитектуру можно использовать для B2B-сайта (business-to-business), на котором общаться между собой будут уже компании. Оптовики и производители могут представлять свои товары компаниям с предложениями о продаже, или наоборот, компании могут публиковать заказы на покупку, инициирующие отправку сообщений поставщикам с запросами о ценах.

В случае обмена сообщениями между приложениями для чтения и записи данных можно вызывать классы из утилиты XML SQL Utility. Для рассылки сообщений в нужном порядке можно использовать логику выбора маршрута, поддерживаемую в компонентах *Advanced Queueing* и Oracle Workflow СУБД Oracle8i. Ниже довольно подробно рассмотрена работа сервера интеграции Oracle Integration Server.

Oracle Integration Server

Oracle Integration Server (*OIS*) — это пакет программных продуктов, предназначенный для создания инфраструктуры обмена XML-сообщениями, позволяющей без проблем провести интеграцию приложения в корпоративных интрасетях или

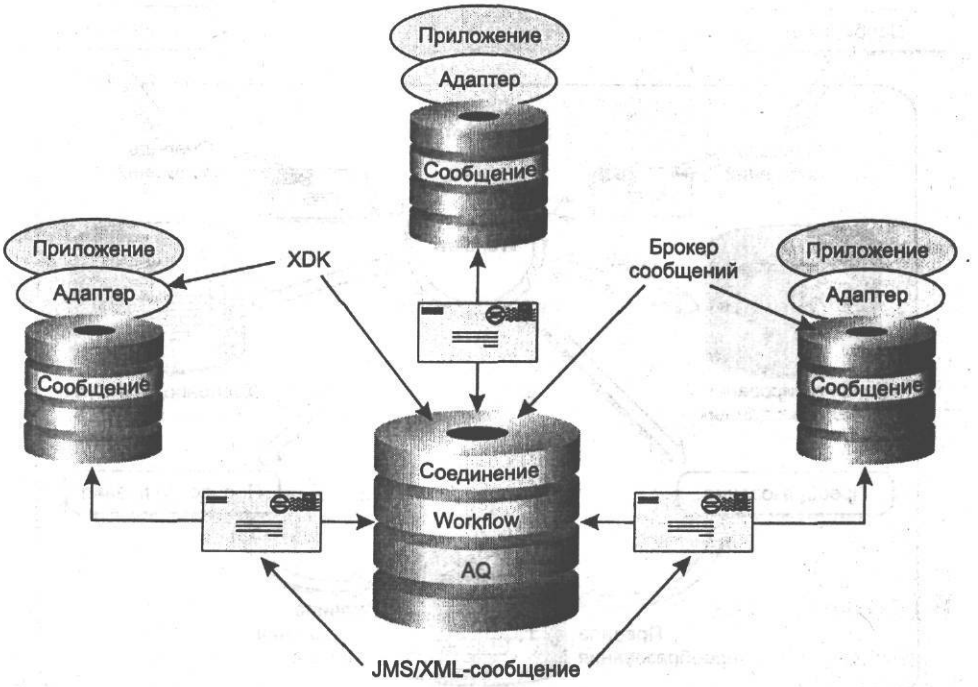


Рис. 8.10. *Архитектура сервера Oracle Integration Server*

в Интернете. На рис. 8.10 показана архитектура сервера OIS. Отметим, что она построена на базе архитектуры соединений-сообщений (hub-and-spoke), которая упрощает администрирование и управление за счет сокращения процессов маршрутизации. Для генерации содержимого XML-сообщения, которое может быть воспринято *брокером сообщений* Oracle Message Broker *(OMB),* строятся или берутся готовые адаптеры обмена данными. Затем брокер OMB упаковывает сообщение в конверт службы Java Messaging Service и направляет его в концентратор. Такой тип транспортировки сообщений хорош тем, что он прозрачен для сетевых шлюзов и позволяет контролировать качество службы и использование ресурсов.

Когда сообщение приходит в концентратор (см. рис. 8.11), оно ставится в очередь и обрабатывается процессором правил в Oracle Workflow для отправки его в нужное место. Чтобы преобразовать сообщение в формат, приемлемый для целевого приложения, к нему применяется XSL-таблица стилей, после чего сообщение запрашивается и отправляется через брокер OMB в одно или несколько приложений.

Этот тип сообщений, разрешенный открытыми XML-стандартами, позволяет организовать слабую связь между приложениями и их асинхронное взаимодействие друг с другом. При этом логика интегрирования приложений выносится на концентратор, что избавляет разработчика от написания сложных программ интеграции в самих приложениях.

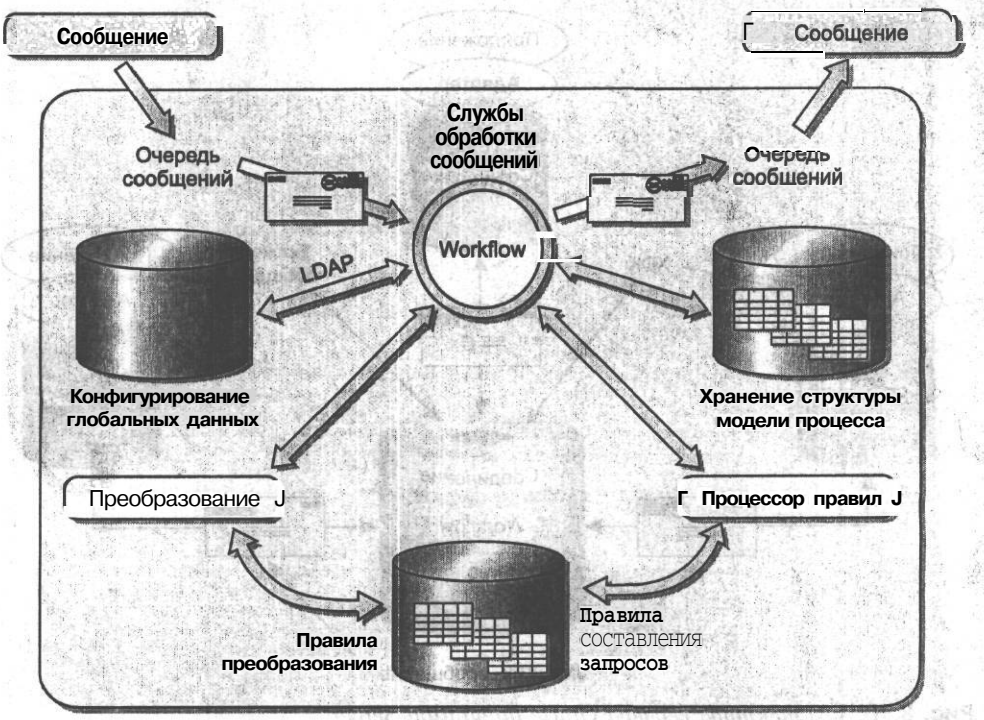


Рис. 8.11. *Архитектура концентратора сервера Oracle Integration Server*

Как можно видеть из приведенных примеров, технология XML и базы данных тесно связаны на нескольких уровнях интеграции приложений. Единственное, чего не хватает для реализации гладкой интеграции,— это стандарт для XML Schema. Однако некоторые независимые группы разработчиков, например XML.org и Open Applications Group, уже начали создавать репозитории для отдельных типов DTD и схем, чтобы упростить обмен данными между компаниями, работающими в одном сегменте рынка. А создание открытого стандарта позволит производить точные отображения данных путем добавления в стек XML-технологий простых и сложных типов данных. Это ускорит внедрение технологии XML в качестве формата обмена данными.

Глава 9

Разработка приложения с использованием XML-технологий Oracle

Мы уже представили целый ряд продуктов производства Oracle, поддерживающих технологию XML, а также многие XML-стандарты. В этой главе проанализирован процесс разработки и разворачивания реального приложения, которое использует эти стандарты, поддерживаемые в инструментариях разработчика Oracle XDK, и новые XML-функции СУБД Oracle9i. В этом приложении подробно рассмотрены важные возможности технологии XSLT, таких программных продуктов, как XML Class Generator, XSQL Servlet, XML SQL Utility, Oracle Text, и новых типов данных и операторов XML, появившихся в Oracle9i.

Web-сайт с ответами на часто встречающиеся вопросы (FAQ), поддерживающий технологию XML

Очень часто компании открывают на своих Web-сайтах специальные разделы поддержки выпускаемых ими продуктов. Эти разделы могут выглядеть по-разному. Это может быть просто опубликованный на сайте сборник электронных версий руководств по эксплуатации, а может быть и моделируемый сотрудниками сервисной службы дискуссионный форум. Нередко в этих разделах поддержки клиентов есть специальный раздел FAQ (frequently asked questions), т. е. там опубликованы ответы

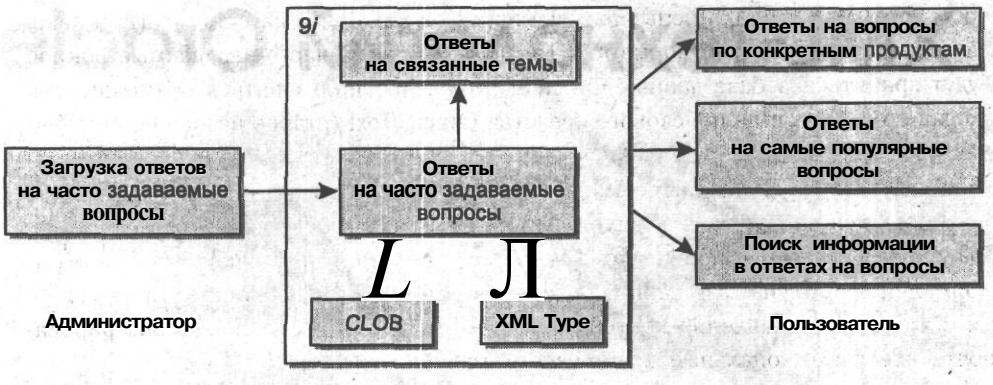


Рис. 9.1. *Функциональная схема приложения*

на часто задаваемые пользователями вопросы. Создание такого приложения и является предметом исследования в данной главе. Его функциональная схема показана на рис. 9.1.

Посетители сайта могут просмотреть все содержимое раздела FAQ, провести поиск интересующей их информации, а могут просмотреть только наиболее часто запрашиваемые ответы на вопросы. При появлении на экране списка ответов пользователь может увидеть и ответы на вопросы, и тематически связанные с ними ответы (для этого на странице размещены соответствующие ссылки). Внутри каждого ответа имеются ссылки на онлайновый глоссарий технических или специальных терминов. Все функциональные возможности данного раздела подробно описаны в этой главе ниже. Администраторы Web-сайта могут загружать в базу данных новые ответы на вопросы и получать статистику по каждому ответу, включая количество обращений и дату последнего обновления.

Требования к приложению

Известно, что в арсенале Oracle есть множество продуктов и компонентов, поддерживающих технологию XML, в том числе и инструментарии разработки ПО на разных языках программирования — C, C++, PL/SQL и Java. Но во всех этих инструментариях разработчика поддерживается одинаковый набор XML-функций. Такая ситуация усложняет работу службы технической поддержки, поскольку многие вопросы, относящиеся к технологии XML, равным образом можно отнести ко всем инструментариям XDK. Много встречается и специфичных для каждого языка вопросов. Поэтому основным требованием к FAQ-сайту Oracle9i XML будет следующее: все хранящиеся на нем вопросы и ответы должны быть помечены в соответствии со своей принадлежностью к одному или нескольким инструментариям разработчика XDK либо к XML-функциям самой СУБД Oracle9i.

Кроме простого списка ответов на вопросы для каждого инструментария XDK и каждой XML-функции пользователь вправе ожидать, что на сайте будет и какое-либо средство поиска информации. Так как ответы на часто задаваемые вопросы будут храниться в базе данных Oracle9i, можно воспользоваться возможностями входящего в ее состав поискового средства Oracle Text (старое название interMedia Text), которое поддерживает технологию XML. Кроме того, нужен механизм составления запросов к базе данных FAQ, который можно реализовать в виде Web-формы.

В большинстве случаев содержимое для Web-сайтов берется из HTML-страниц, хранящихся в файловых системах на серверах. В нашем приложении почти вся информация берется из базы данных. Благодаря этому можно воспользоваться мощными средствами управления контентом и в то же самое время продемонстрировать почти все функциональные возможности технологии Oracle9i XML.

Проектирование приложения

Так как в этой главе описано приложение, создаваемое как часть более крупного Web-сайта, сосредоточимся на тех Web-страницах, которые будут частью раздела FAQ. На рис. 9.2 показана схема разработки и функционирования приложения для этого раздела.

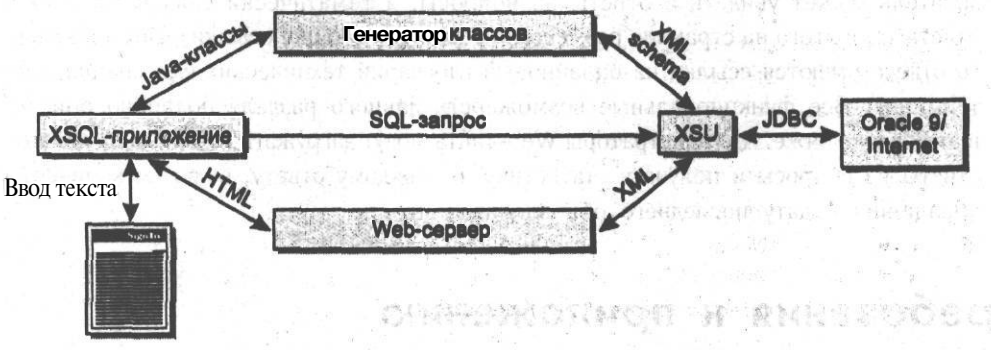


Рис. 9.2. *Схема разработки и функционирования приложения FAQ*

Отметим, что на этом рисунке показаны два типа работ с базой данных — создание приложения и его исполнение. Процесс разработки приложения идет от схемы базы данных через утилиту XML SQL Utility к созданию XML Schema, которая затем обрабатывается генератором классов для создания классов приложения. Затем эти классы используются для построения приложения на базе XSQL Servlet. В процессе своей работы приложение принимает на входе текст в форме вопросов и ответов и создает XML-документ, который направляется через Web-сервер на драйвер XSU и размещается в базе данных. В следующих разделах рассматриваются все этапы создания приложения с использованием большинства функций базы данных Oracle9i и инструментария XDK.

Конструирование схемы базы данных

Для создания этого приложения максимально используем базу данных и ее возможности. Поэтому нужно создать такую схему базы данных, в которой будут храниться не только вопросы и ответы на них, но также и метаданные. Это позволит связать FAQ с XML-функциями и компонентами, а также проследить отправление запросов и отбор информации. Сохраним в базе ключевые слова и их определения, чтобы создать глоссарий технических терминов. Наконец, необходима поддержка системы ссылок на связанные ответы на вопросы, что позволит добиться повышения производительности работы базы данных за счёт использования больших символьных объектов (CLOB) в таблицах в кэше.

Схема БД будет очень простой. Она состоит из таблицы **xmlfaq_xmltype** со следующими столбцами: категория (category), вопрос (question), подробное описание вопроса (questionde), ответ (answer), язык (language), отправитель (sendby) и дата создания (createtime). Столбцы "ответ" и "подробное описание вопроса" будут создаваться как данные типа XMLType, что позволит сохранять и индексировать ответы в виде XML-документов. Эта схема создается с помощью следующего скрипта:

```
Rem Script to create XML Faq Schema
CREATE TABLE xdkfaq_xmltype(
   question     VARCHAR2(4000) NOT NULL,
   questionde   sys.xmltype NOT NULL,
   answer       sys.xmltype NOT NULL,
   category     VARCHAR2(30) ,
   language     VARCHAR2(4000),
   id           NUMBER,
   createtime   VARCHAR2(30),
   CONSTRAINT   xdkfaq_xmltype_pk PRIMARY KEY (id)
);
create sequence xdkfaqid_seq start with 1;
```

Следующая таблица в этой схеме, очень похожая на предыдущую, используется для вывода вопросов и ответов на экран браузера. Для хранения ответов и вопросов FAQ создается таблица **xdkfaq:**

```
CREATE TABLE xdkfaq(
   question     VARCHAR2(4000) NOT NULL,
   questionde   VARCHAR2(4000) NOT NULL,
   answer       VARCHAR2(4000) NOT NULL,
   category     VARCHAR2(30) NOT NULL,
   language     VARCHAR2(10) NOT NULL,
   createtime   VARCHAR2(20) NOT NULL,
   diff         NUMBER,
   id           NUMBER NOT NULL,
   hits         NUMBER,
   sendby       VARCHAR2(10),
   CONSTRAINT xdkf aq_pk PRIMARY KEY (id)
);
create sequence xdkfaqgl_seq start with 1;
```

Для хранения FAQ используются столбцы: вопрос (question), подробное описание вопроса (questionde) и ответ (answer). Столбцы категория (category), язык (language), время создания (createtime), число обращений (hits) и отправитель (sendby) будут использоваться администратором для управления разделом FAQ, а также для поиска информации.

В этой таблице есть также столбец ID, в котором содержатся уникальные идентификаторы ID для каждого ответа на вопрос. ID будет генерироваться в последней строке при создании с помощью SQL-оператора последовательности xdkfaqid_seq. Затем идентификатор ID будет генерироваться для каждой входной записи FAQ с использованием той же последовательности в триггере:

```
select xdkfaqid_seq.nextval
into :new.ID
from dual;
```

Для создания глоссария нужно создать таблицу, которую можно будет связать с отдельными строками с помощью DBURIType. В столбце ссылок будет находиться ссылка DBUri-ref, указывающая на содержание данного пункта глоссария. Эта таблица создается с помощью следующего скрипта:

```
CREATE TABLE xdkfaq_glossary
   id        NUMBER NOT NULL,
   name      VARCHAR2 (30),
   link      VARCHAR2(100) ,
   content   VARCHAR2(4000) ,
   CONSTRAINT xdkfaq_gloss PRIMARY KEY  (id)
);
```

Наконец, можно запустить приложение на Web-сайте. Для этого создается кэш, используемый при выводе на экран фиксированного набора данных, например первых двадцати самых популярных вопросов и ответов. Для кэша нужно создать две таблицы с помощью следующего скрипта:

```
CREATE TABLE xdkfaq_ccache (
   category VARCHAR2(32) NOT NULL,
   link      VARCHAR2(100),
   content   CLOB NOT NULL,
   CONSTRAINT xdkfaqccache_pk PRIMARY KEY  (category)
);
```

Отметим, что сначала была сконструирована схема базы данных, а затем из нее была создана нужная XML Schema. Порядок должен быть именно таким, поскольку, скорее всего, из двух этих объектов именно база данных будет иметь больше ограничений. Кроме того, когда XML-документы хранятся как данные без тегов, в большинстве случаев возникают альтернативные обращения к названным данным и их обновления с помощью SQL-операторов; нужно учитывать описанную схему базы данных при таком хранении XML-документов.

Генерация XML Schema

После проектирования и создания схемы БД можно использовать утилиту XML SQL Utility для генерации связанной XML Schema. Для этого нужно просто отправить через драйвер XSU следующий запрос:

```
select category, question, answer, language, sendby, createtime from xdkfaq
```

Далее идет листинг XML Schema, сгенерированной по предыдущему запросу:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2000/10/XMLSchema">
 <xsd:element name="ROWSET">
  <xsd:complexType>
   <xsd:sequence>
    <xsd:element name="ROW" minOccurs="0" maxOccurs="unbounded">
     <xsd:complexType>
      <xsd:sequence>
       <xsd:element name="CATEGORY" type="xsd:string minOccurs="0"/>
       <xsd:element name="QUESTION" type="xsd:string minOccurs="0"/>
       <xsd:element name="ANSWER" type="xsd:string minOccurs="0"/>
       <xsd:element name="LANGUAGE" type="xsd:string minOccurs="0"/>
       <xsd:element name="SENDBY" type="xsd:string nulllable="true"
         minOccurs="0"/>
       <xsd:element name="CREATETIME" type="xsd:string minOccurs="0"/>
      </xsd:sequence>
      <xsd:attribute name="num" type="xsd:integer"/>
     </xsd:complexType>
    </xsd:element>
   </xsd:sequence>
  </xsd:complexType>
 </xsd:sequence>
</xsd:schema>
```

Отметим, что отображение из базы данных на XML Schema создает не атрибуты, а элементы из столбцов таблицы. Поскольку XML Schema может иметь только атрибуты типа как простые XML-типы, это отображение на элементы дает намного более богатые возможности контроля типов данных, что очень важно при разработке сложных схем, определяющих вложенные таблицы, области и т. д.

Генерация Java-классов

, Теперь есть XML Schema, соответствующая схеме БД. Поэтому можно использовать Java Class Generator для генерации классов, которые затем применяются в Java-приложениях, апплетах или страницах Java Server Page (JSP) для вставки

XML-данных обратно в базу данных. Это довольно просто сделать с помощью файла **oracg.bat,** запускающегося из командной строки. Вот как будет выглядеть эта строка для файла **xmlfaq.xsd:**

```
java oracg -c -s xmlfaq.xsd -p rowset
```

По этой команде будет создан файл с исходным кодом класса **rowset.java.** Весь листинг программы слишком велик для воспроизведения на страницах книги, поэтому здесь представлен только его фрагмент, чтобы показать тип генерируемого класса:

```
/**
 * Adds the local element
 * @param QUESTION the element node
 * @exception InvalidContentException if the element is not valid.
 */
public void addQUESTION(ROWSET.ROWSET_type.ROW.ROW_Type.QUESTION theQUESTION)
            throws InvalidContentException
               { super.addElement(theQUESTION); }
/**
 * Adds the local element
 * @param ANSWER the element node
 * @exception InvalidContentException if the element is not valid.
 */
public void addANSWER(ROWSET.ROWSET_type.ROW.ROW_Type.ANSWER theANSWER)
            throws InvalidContentException
               { super.addElement(theANSWER); }
```

Заметим, что для простоты использования код сопровождается комментариями.

Хранение XML-документов как данных типа XMLType

Перед тем как непосредственно браться за создание приложения, нужно рассказать об Oracle9i XMLType. В Oracle9i появился новый определяемый системой тип данных, который называется XMLType. Он предназначается для обработки XML-документов. XMLType можно использовать *внутри сервера* в PL/SQL- и SQL-операторах. В нашем приложении этот новый тип данных используется для того, чтобы:

- Хранить XML-документы как данные типа XMLType
- Генерировать данные типа XMLType с помощью функций SYS_XMLGEN и SYS_XMLAGG

- Извлекать данные из XMLType с помощью функций **Extract()** и **ExistsNone()**
- Вести поиск в XML-документах, сохраненных как данные типа XMLType, создавая для этого функциональные индексы

Для сохранения XML-документа в виде XMLType достаточно просто определить столбец XMLType, как это делалось раньше. Затем, прежде чем вставить в этот столбец документ, использовав для этого функции SYS_XMLGEN и SYS_XMLAGG, можно с помощью конструктора XMLType создать экземпляр XMLType.

Генерация XML с помощью SYS_XMLGEN и SYS_XMLAGG

SYS_XMLGEN и SYS_XMLAGG — это встроенные SQL-функции, которые можно использовать не только для обработки XMLType, но и для генерации или группировки XML-документов внутри SQL-запросов. Например, следующий запрос использует функцию SYS_XMLGEN для извлечения всех ключевых слов и возвращения их в набор XML-документов, где каждое ключевое слово соответствует одному ответу с корневым элементом ROW:

```
select sys_xmlgen(f.answer.extract('//keywords')).getStringVal()
  from xdkfaq_xmltype f;
```

В следующем запросе функция SYS_XMLGEN используется для извлечения тех же ключевых слов, но здесь они возвращаются в сгруппированном виде в XML-документ с корневым элементом ROWSET, содержащим все элементы ROW.

```
select sys_xmlagg( sys_xmlgen(f.answer.extract('//keywords'))).getStringVal()
  from xdkfaq_xmltype f;
```

Извлечение данных из XMLType с помощью функций ExtractQ и ExistsNone()

В двух предыдущих запросах уже использовалась функция **Extract()**. Вообще же эта функция используется для извлечения фрагментов XML-документов, которые хранятся как данные типа XMLType. По следующему запросу эта функция должна возвратить узлы XML-документа:

```
select extract(e.answer, 'keywords').getStringVal()
  from xdkfaq_xmltype e;
 where existsNode(e.answer, 'code')>0;
```

А вот фрагмент результата выполнения этого запроса:

```
<keywords>importNode</keywords>
<keywords>adoptNode</keywords>
<keywords>document fragment</keywords>
<keywords>owner document</keywords>
```

Использование XMLType для связанных ответов на часто задаваемые вопросы FAQ

Чтобы создать Web-сайт с ответами на часто задаваемые вопросы FAQ, используются возможности XMLType для построения системы связей между разными FAQ. Эта система связей строится на основе элементов "keywords" (ключевые слова), которые содержатся в ответах. Есть два способа создания такой системы. Первый состоит в том, чтобы для хранения связей между FAQ построить специальную таблицу, как в следующем фрагменте кода:

```
CREATE TABLE xmlfaq_urimanage(
   keyword  VARCHAR2(300) NOT NULL,
   faqid    NUMBER NOT NULL,
   CONSTRAINT faquri_pk PRIMARY KEY (keyword, faqid)
);
```

Данные вводятся через триггер в SQL-операторе INSERT, как показано в следующем фрагменте кода:

```
key_typea sys.xmlType;
...
key_typea := sys.xmlType.createxml(:new.answer);
   if key_typea.existsNode('//keywords') =1 THEN
      LOOP
         v_temp:= key_typea.extract('//keywords'||'['||i||']/text()');
         EXIT WHEN v_temp IS NULL;

         key := v_temp.getStringVal();
         i := i+1;
         -- Insert new content to xdkfaq_query
         INSERT INTO xdkfaq_query_urimanage(keyword,faqid) VALUES(key,:new.id);
      END LOOP;
   END IF;
```

Таким образом, XSQL-страница будет выглядеть следующим образом:

```
<?xml version="1.0"?>
<xml-stylesheet type="text/xsl" media="mozilla" href="xsl/faqrelative.xsl"?>
<page connection="xdkdemo" xmlns:xsql="urn:oracle-xsql">
<xsql:set-page-param name="SIG" value="{@sig}"/>
<xsql:query>
   <![CDATA[
     select faq.id,
            faq.question,
      cursor(select question, id,
         sys_dburigen(faq1.ID,faq1.answer).getUrl() as quri,
         sys_dburigen(faq1.ID,faq1.answer) .getUrl() as auri,
         from xdkfaq faq1
         where faq1.id in (select distinct(uri.faqid)
            from xdkfaq_urimanage
         where uri.keyword in (select keyword
                  from xdkfaq_urimanage
                  where faqid = {@id})
            and uri.faqid !={@id}))
        as relatives
      from xdkfaq faq
      where faq.id = {@id}
      ORDER BY question
    ]]>
</xsql:query>
<xsql:include-param name="SIG" />
</page>
```

Другой способ предполагает использование преимуществ увеличения производительности за счет функциональных индексов в XMLType. В этом случае связанные FAQ извлекаются с помощью следующей XSQL-страницы:

```
<?xml version="1.0"?>
<xml-stylesheet type="text/xsl" href="xdkfaqrelative.xsl"?
<page connection="xdkdemo" xmlns:xsql="urn:oracle-xsql">
<xsql:dml>
 alter session set query_rewrite_enabled=true
</xsql:dml>
<xsql:dml>
 alter session set query_rewrite_integrity=trusted
</xsql:dml>
```

```
<xsql:include-owa>
   show_related({@id});
</xsql:include-owa>
</page>
```

PL/SQL-процедура **show_related** выглядит так:

```
CREATE OR REPLACE PROCEDURE show_related (p_id in NUMBER) IS
   I NUMBER :=1; — relative FAQ variables
   key_typea sys.xmlType;
   key VARCHAR2(300);
   v_temp sys.xmltype;
   v_temp1 sys.xmltype;
   v_result VARCHAR2(4000);
   v_question VARCHAR2(4000);
BEGIN
      htp.p('<ROWSET>');
      htp.p('<ROW>');
      htp.p('<ID>');
      htp.print(p_id);
      htp.p('</ID>');
      htp.p('<QUESTION>');
      select f.questionde.extract('/question/text()').getClobVal()
         into v_question from xdkfaq_xmltype f where id = p_id;
      htp.print(v_question);
      htp.p('</QUESTION>');
      htp.p('<RELATIVES>');
      ----------------------------------------------------------
      — Get relative faq for each keyword
      ----------------------------------------------------------
      select answer into key_typea from xdkfaq_xmltype f where id = p_id;
      if key_typea.existNode('//keywords') =1 THEN
   LOOP
         v_temp:= key_typea.extract ('//keywords'[']||i||']/text()');
         EXIT WHEN v_temp IS NULL;
         key := v_temp.getStringVal();
      i := i+1;
         select sys_xmlagg(sys_xmlgen(relative_type(e.question,
            e.id),sys.xmlgenformattype.createformat('RELATIVES_ROW'))
               ).extract('/ROWSET//RELATIVES_ROW').getClobVal() into v_result
         from xdkfaq_xmltype e
         where e.id != p_id and e.id in (
```

```
        select f.id from xdkfaq_xmltype f
        where f.answer.extract('/answer/keywords/text()')getClobVal()
                                    like '%'||key||'%');
        htp.print(v_result);
      END LOOP;
    END IF;
      htp.p('</RELATIVES>');
      htp.p('</ROW>');
      htp.p('</ROWSET>');
EXCEPTION
  when others then
    BEGIN
      htp.p('</RELATIVES>');
      htp.p('<error>exeption</error'>;
      htp.p('</ROW>');
      htp.p('</ROWSET>');
    END;
  END show_related;
```

Создание Web-приложения

Web-приложение FAQ можно разрабатывать, используя в качестве основного движка XSQL Servlet. Этот сервлет, впервые появившийся в Oracle8i и доработанный в Oracle9i, представляет собой надежную и гибкую платформу для приложений такого типа. XSQL Servlet используется:

- Для генерации статического контента из страниц, хранящихся в базе данных
- Для приема вопросов и ответов FAQ для последующего их включения в базу данных
- Для создания интерфейса поиска по ключевым словам
- Для вывода на экран браузера результатов запросов
- Для управления соединениями с базой данных

Возможности XSQL Servlet по выводу на экран браузера статического контента уже рассматривались раньше. На рис. 9.3 представлена начальная страница нашего приложения, где текущее содержимое, представленное в формате xHTML, и его таблица стилей были получены из таблицы кэша БД путем передачи имени этой страницы в сервлет XSQL Servlet.

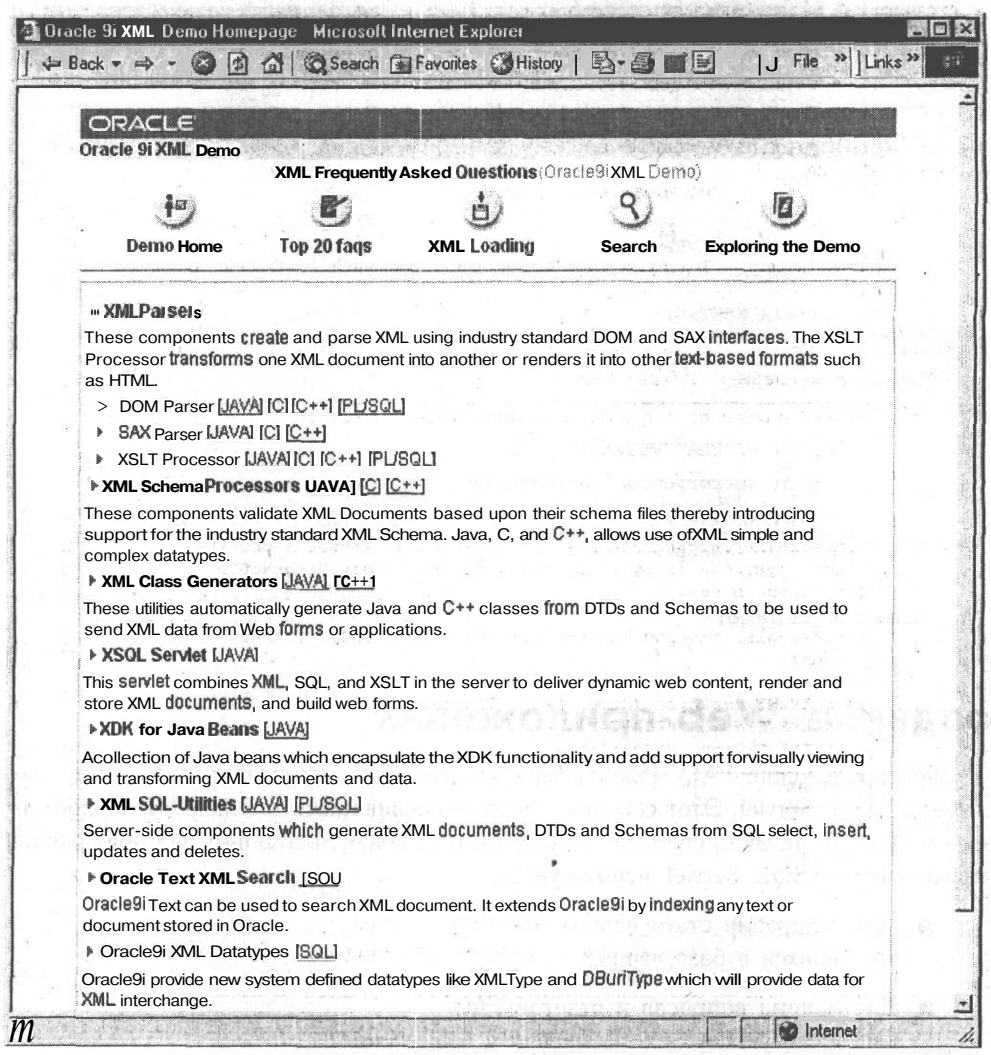


Рис. 9.3. *Начальная страница FAQ*

Составление запросов на получение FAQ

Для того чтобы получить ответы на часто задаваемые вопросы, т. е. FAQ, используются классы, сгенерированные с помощью Class Generator в XSQL-обработчике действий. Прежде чем сделать это, следует создать форму, в которую пользователь будет вводить данные для FAQ. Эта форма, созданная путем возвращения файла **submit.xml** из БД и преобразования его с помощью таблицы стилей **submit.xsl**, показана на рис. 9.4.

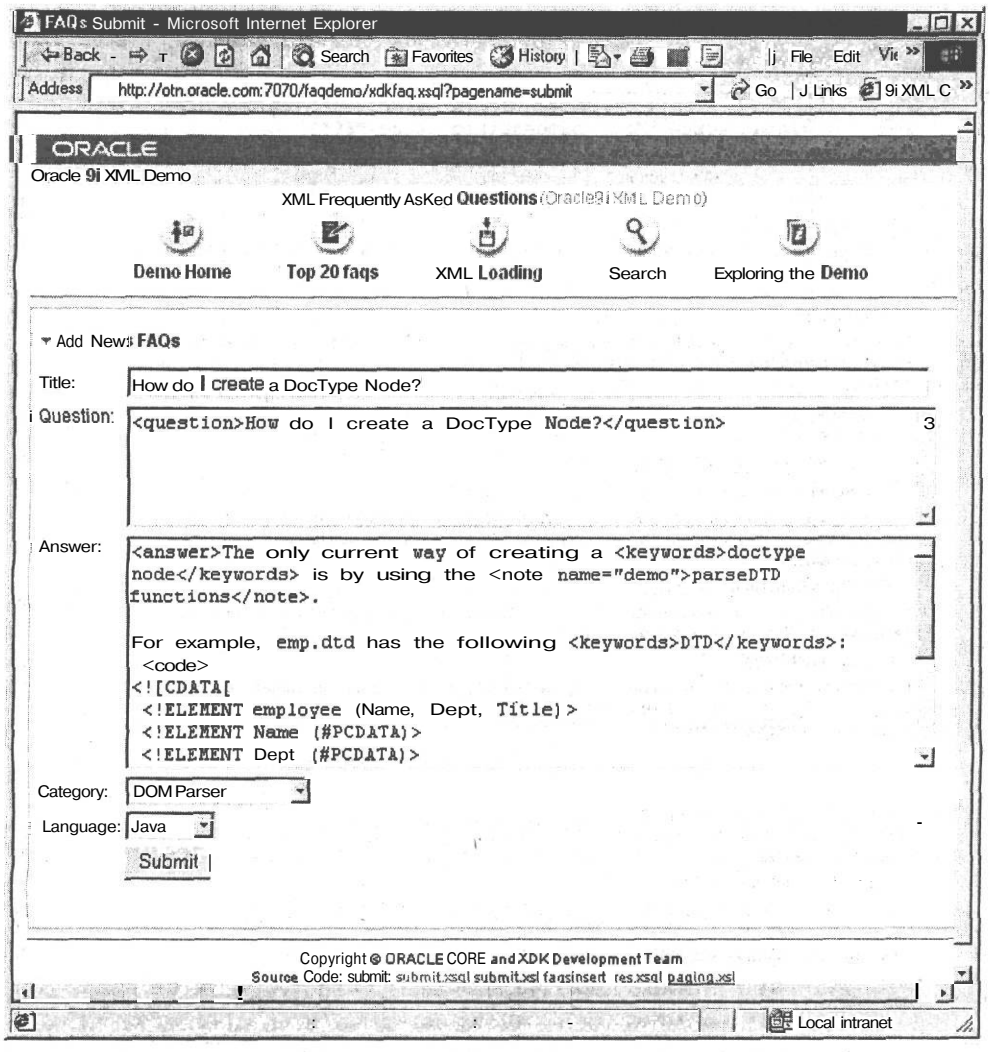


Рис. 9.4. *Форма для ввода данных в FAQ*

Следующий фрагмент кода — это активная часть страницы, показанной на рис. 9.4. Он показывает, как данные передаются во вторую XSQL-страницу **faqinsert_res**, которая обращается к обработчику действий.

```
<tr>
 <td colspan="2" height="366">
   <form method="post" action="xdkfaq.xsql?pagename=faqinsert_res"
    name="addfaqs">
      <table width="60%" border="0">
```

```
<tr>
 <td width="5%">Title</td>
 <td width="95%">
   <input type="text" name="title" size="80"
    value="How do I create a DocType Node?">
 </td>
</tr>
<tr>
 <td width="5%" valign="top">Question:</td>
 <td width="95%">
   <textarea name="question" cols="70"
   rows="5">&lt;question&gt;&lt;/question&gt;</textarea>
 </td>
</tr>
<tr>
 <td width="5%" valign="top">Answer:</td>
 <td width="95%">
   <textarea name="answer" cols="70" rows="10">
    &lt;
    answer&gt;
    The only current way of creating a &lt;keywords&gt;
    Doctype node&lt;/keywords&gt;
    is by using the &lt;note name="demo"&gt;
    parseDTD functions&lt;/note&gt;
    For example, emp.dtd has the following
    &lt;keywords&gt;DTD&lt;/keywords&gt;:&lt;code&gt;&lt;
    ![CDATA[
      &lt; !ELEMENT employee (Name, Dept, Title)&gt;
      &lt; !ELEMENT Name (#PCDATA)&gt;
      &lt; !ELEMENT Dept (#PCDATA)&gt;
      &lt; !ELEMENT Title (#PCDATA)&gt;
    ]]&gt;
    &lt;/code&gt;
    You can use the following code to create a doctype node:
    &lt;code&gt;
    parser.parseDTD(new FileInputStream(emp.dtd), "employee");
    doc = parser.getDocument();
    /* doc has the dtd as its child or dtd = parser.getDoctype();*/
    &lt;/code&gt;&lt;/answer&gt;</textarea>
 </td>
</tr>
<tr>
```

```
            <td width="5%">Category:</td>
            <td width="95%">
              <select name="comp">
                 <option value="domparser">DOM Parser</option>
                 <option value="saxparser">SAX Parser</option>
                 <option value="xslt">XSLT Processor</option>
                 <option value="schema">Schema Processor</option>
                 <option value="xsu">XML SQL Utility</option>
                 <option value="xsql">XSQL Servlet</option>
                 <option value="soap">SOAP</option>
                 <option value="jbeans">Java Beans</option>
              </select>
            </td>
          </tr>
        <tr>
          <td width="5%">Language:</td>
          <td width="95%">
             <select name="lang">
                <option value="java">Java</option>
                <option value="c">C</option>
                <option value="cpp">C++</option>
                <option value="plsql">PL/SQL</option>
                <option value="xml">XML</option>
             </select>
          </td>
          </tr>
          <tr>
           <td width="5%" valign="top" height="8"> </td>
           <td width="95%" height="8">
             <input type="submit" name="Submit" value="Submit">
           </td>
          </tr>
        <table>
    </form>
```

Далее идет довольно простой исходный код для файла **faqinsert_res.xsql**:

```
<?xml version="1.0"?>
<!-- <?xml-stylesheet type="text/xsl" href="xsl/faqinsert_res.xsl"?>
-->
<page connection="xdkdemo" xmlns:xsql="urn:oracle-xsql">
<webpage title="Insert FAQs Items">
   <xsql:action handler="FAQsInsertDemo" />
   </webpage>
   </page>
```

Обработчик действий **FAQsInsertDemo** — это Java-класс, использующий сгенерированные классы и JDBC-соединение, чтобы вставить FAQ в базу данных. Ниже представлен источник для этого класса:

```
import java.sql.*;
import java.net.*;
import oracle.xml.xsql*;
import oracle.xml.sql.dml.OracleXMLSave;
import javax.servlet.http.*;
import oracle.xml.parser.v2.*;
import org.w3c.dom.*;
import java.io.*;
import org.xml.sax.SAXParseException;
import java.io.StringReader;
import oracle.xml.parser.v2.XMLOutputStream;
import oracle.xml.classgen.CGXSDElement;
import oracle.xml.classgen.SchemaClassGenerator;
import oracle.xml.classgen.InvalidContentException;
import ROWSET;
public class FAQsInsertDemo extends XSQLActionHandlerImpl
{  ByteArrayInputStream bInput= null;
   String m_title=null;
   String m_question=null;
   String m_answer=null;
   String m_status=null;
   String m_comp=null;
   String m_lang=null;
   public void handleAction(Node root) throws SQLException
   {  if (getPageRequest().getRequestType().equals("Servlet"))
      {  /*
         ** get request parameters
         */
         XSQLServletPageRequest xspr =
                              (XSQLServletPageRequest)getPageRequest();
         HttpServletRequest req = xspr.getHttpServletRequest();
         m_title = req.getParameter("title");
         m_question = req.getParameter("question");
         m_answer = req.getParameter("answer");
         m_status = req.getParameter("status");
         m_comp = req.getParameter("comp");
         m_lang = req.getParameter("lang");
```

```
/*
 ** XML well-formedness check and validation of question and answer
 */
DOMParser dp = new DOMParser();
try {
   dp.parse(new StringReader(m_question));
}
catch (SAXParseException spe){
   this.reportError(root, "Please check the question"+
                    * Input, it needs to be well-formed.\n Error;"
                    +spe.getMessage());
   return;
}
catch(Exception e){}
try{
   dp.parse(new StringReader(m_answer));
}
catch (SAXParseException spe){
   this.reportError(root, "Please check the answer"+
                           • Input, it needs to be well-formed. \n
                           Error;"+spe.getMessage());
   return;
}
catch(Exception e){}
/*
 ** User Generated Classes ti build XML document
 */
ByteArrayOutputStream baos = new ByteArrayOutputStream();
DataOutputStream output = new DataOutputStream(baos);
XMLOutputStream out = new XMLOutputStream(output);
try {
   ROWSET.ROWSET_Type rstype = new ROWSET.ROWSET_Type ();
   ROWSET.ROWSET_Type.ROW.ROW_Type rtype =
                          new ROWSET.ROWSET_Type.ROW.ROW_Type();
   Integer num_in = new Integer(1);
   rtype.setNum(num_in);
   /*
    * Set data for the ROW
    */
   ROWSET.ROWSET_Type.ROW.ROW_Type.CATEGORY category =
                       new ROWSET.ROWSET_Type.ROW.ROW_Type.CATEGORY();
   category.setType(m_comp);
   rtype.addCATEGORY(category);
```

```
ROWSET.ROWSET_Type.ROW.ROW_Type.QUESTION question =
                    new ROWSET.ROWSET_Type.ROW.ROW_Type.QUESTION();
question.setType(m_title);
rtype.addQUESTION(question);
ROWSET.ROWSET_Type.ROW.ROW_Type.QUESTIONDE questionde =
              new ROWSET.ROWSET_Type.ROW.ROW_Type.QUESTIONDE();
questionde.setType(filter(m_question));
rtype.addQUESTIONDE(questionde);
ROWSET.ROWSET_Type.ROW.ROW_Type.ANSWER answer =
                    new ROWSET.ROWSET_Type.ROW.ROW_Type.ANSWER();
answer.setType(filter(m_answer)); rtype.addANSWER(answer);
ROWSET.ROWSET_Type.ROW.ROW_Type.LANGUAGE language =
                    newROWSET.ROWSET_Type.ROW.ROW_Type.LANGUAGE();
language.setType(m_lang);
rtype.addLANGUAGE(language);

ROWSET.ROWSET_Type.ROW.ROW_Type.DIFF diff =
                    new ROWSET.ROWSET_Type.ROW.ROW_Type.DIFF();
diff.setType(new Integer(1));
rtype.addDIFF(diff);
ROWSET.ROWSET_Type.ROW row = new ROWSET.ROWSET_Type.ROW();
row.setType(rtype);
rtype.addROW(row);
ROWSET rowset = new ROWSET();
rowset.setType(rstype);

rowset.ptint(out)
out.flush();
/*
 ** Print out to ByteArrayOutputStream
 */
byte[] data = baos.toByteArray();
out.close();
bInput = new ByteArrayOutputStream(data);
}catch(InvalidConnectException ex)
{  this reportError(root,ex.getMessage());
   //e.printStackTrace();
   return;
}catch(IOException e)
{  this reportError(root,e.getMessage());
   //e.printStackTrace();
   return;
}
```

```
            /*
             ** Connect to Database and send the data
             */
            try {
               Connection conn = xspr.getJDBCConnection() ;
               OracleXMLSave sav = new OracleXMLSave(conn, "xdkdemo.xdkfaq") ;
               sav.insertXML(bUnput) ;
               sav.close() ;
               /*
                ** Output generated xml file to the screen
                */
               String outtemp=baos.toString() ;

               Element e =
                  getActionElement ( ) . getOwnerDocument ( ) . createElement ( "InputDoc") ;
               addResultElement(e,"CONTENT",outtemp);
               root.appendChild(e);
               bInput.close() ;
               conn.close();
            }catch (Exception e) {
               this reportError(root,"Exception caught "+e.getMessage()) ;
               return;
            }
         }
      }
      public static String filter(String input)
      {  if (input == null) return "";
         StringBuffer filtered = new StringBuffer(input.length());
         char c;
         for ( int i=0; i<input . length(); i++)
            {  c = input.charAt(i);
               if (c == '<')
                  { filtered.append("&lt;");  }
               else if (c == '>')
                  { filtered.append("&gt; ");  }
               else if (c == '"')
                  { filtered.append("&quot;");  }
               else if (c == '&')
                  {fi1tered. append("&amp; " ) ; }
               else
                  { filtered.append(c);  }
            }
         return (filtered. toString());
      }
   }
```

Поиск информации в FAQ

Механизм вставки информации в FAQ понятен. Теперь нужно определить способ ее извлечения. Это можно сделать с помощью XSQL-страницы, используя возможности Oracle Text и XMLType. На рис. 9.5 показана форма поиска информации, которую можно использовать. Следует отметить, что поиск ведется и по категории, и по ключевому слову. Категории можно найти в поле со списком разных XML-компонентов, а ключевые слова вводятся в свободной форме.

Следующий листинг представляет собой активную часть страницы поиска. Для передачи параметров поиска в XSQL-страницу, которая отправляет запрос в базу данных, в нем используется метод Form.

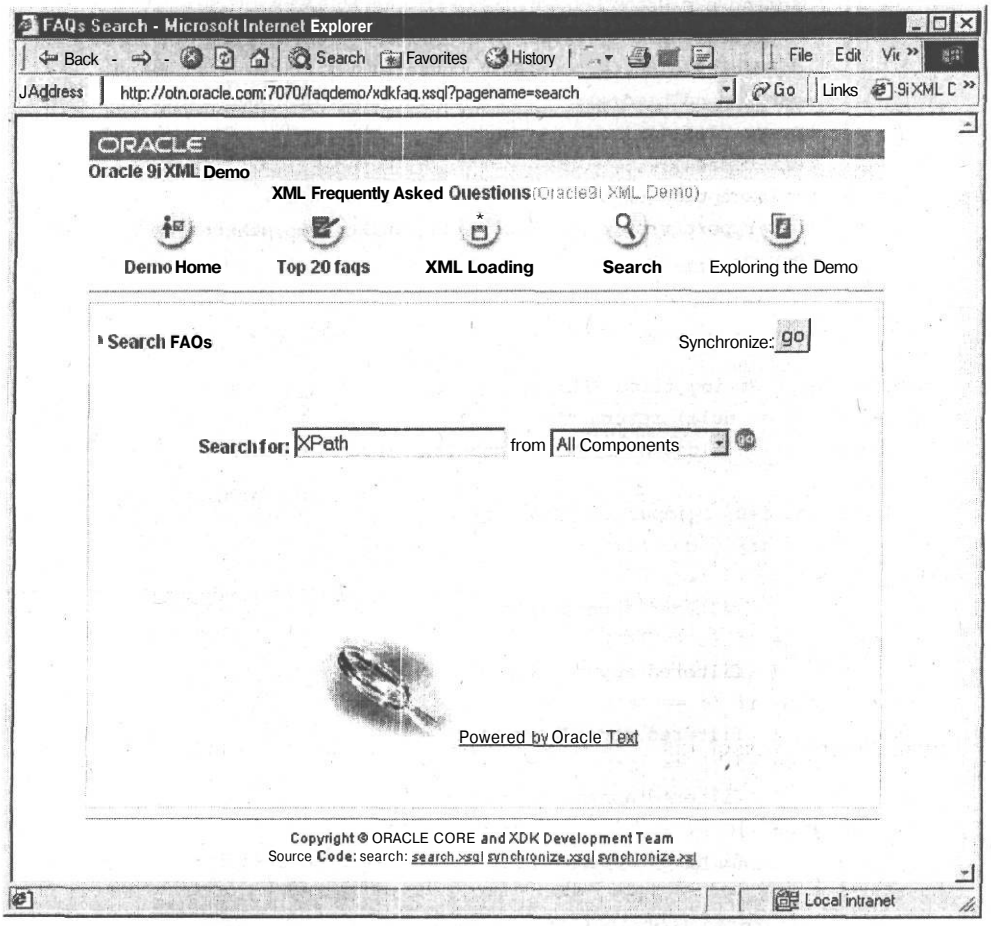


Рис. 9.5. *Форма поиска информации в FAQ*

```
<form method="post" name="find">
   <b>Search for:</b>
   <input type="text" name="search"> from
    <SELECT name="category" size="1">
         <OPTION value="all" selected>All Component</OPTION>
         <OPTION value="domparser">DOM Parser</OPTION>
         <OPTION value="saxparser">SAX Parser</OPTION>
         <OPTION value="xslt">XSLT Processor</OPTION>
         <OPTION value="schema">Schema Processor</OPTION>
         <OPTION value="classgen">Class Generator</OPTION>
         <OPTION value="xsql">XSQL Servlet</OPTION>
         <OPTION value="jbeans">Java Beans</OPTION>
         <OPTION value="xsu">XML SQL Utility</OPTION>
         <OPTION value="xmltype">XMLType</OPTION>
         <OPTION value="dburi">DBUri</OPTION>
         <OPTION value="oracletext">Oracle Text</OPTION>
    </SELECT>
    <A href="xdkfaq.xsql?pagename=faqsearch"
     onClick="if(document.find.category.value=='all')
     search_loc='xdkfaq.xsql?pagename=faqsearch&amp;search='
     +document.find.search.value;
     else search_loc =
     'xdkfaq.xsql?pagename=faqsearch_cat&amp;search='
     +document.find.search.value+'&amp;cat=
     '+document.find.category.value;
     document.location.href = search_loc;
     return false;"
     onMouseOver="self.status='xdkfaq.xsql?pagename=faqsearch';
     return true;"
     onMouseOut="self.status='';returntrue;">
     <img src="images/go.gif" width="16" height="17" border="0"
      alt='Click to search.".
    </A>
</form>
```

Файл **faqsearch.xsql,** представленный в следующем листинге, передает два запроса в базу данных через драйверы XSU и JDBC:

```
<?xml version="1.0"?>
<page id="1" connection="xdkdemo" xmlns:xsql="urn:oracle-xsql">
   <xsql:set-page-param name="query" value={@search} />
<xsql:action handler="Paging" rows-per-page="10"
 url-params="xdkfaq.xsql?pagename=faqsearch&".
```

```
<![CDATA[
      SELECT count(question) FROM xdkfaq
      WHERE contains(answer,'{@search}')>0
   ]]>
  </xsql:action>
<xsql:query skip-rows="{@paging-skip}" max-rows="{@paging-max}">
<![CDATA[
     SELECT * FROM xdkfaq WHERE contains(answer,'{@search}')>0
     ORDER BY hits DESC
   ]]>
   </xsql:query>
   <xsql:include-param name="query" />
   </page>
```

Первый запрос извлекает номера ответов, удовлетворяющих запросу, а по второму запросу извлекаются сами ответы, соответствующие этим номерам.

Использование операторов HASPATH и INPATH для поиска в данных типа XMLType

Напомним, что в предыдущем запросе использовался оператор CONTAINS. Если же нужно провести специализированный поиск в каком-то специфичном разделе ответов, хранящихся как данные типа XMLType, то для упрощения этой процедуры можно использовать новые операторы HASPATH и INPATH.

Предположим, например, что в одном из ответов раздела FAQ фигурирует следующее предложение:

```
<answer>
The parser validates a document against its <keyword>DTD</keyword>.
</answer>
```

Для поиска только тех ответов, где фигурирует ключевое слово "DTD", можно использовать следующий SQL-запрос:

```
select * from xdkfaq where contains(answer,'DTD inpath(/answer/keywords')>0;
```

Оператор HASPATH работает точно так же, как работает оператор **ExistsNode()** в XMLType, и его можно использовать для ограничения поиска. Это показано в следующем примере:

```
select * from xdkfaq
      where contains(answer,'haspath(/answer/keywords)'>0;
```

Использование функциональных индексов для повышения производительности поиска

Если параметры поиска известны, процедура поиска существенно ускоряется за счет использования *функционального индекса*. Такой прием позволяет заранее вычислить результаты поиска. В нашем случае ответы хранятся в XML-файле, в котором могут встречаться ключевые слова, помеченные следующим образом: <keyword>термин</keyword>. Преимущества функциональных индексов можно оценить, пользуясь такими функциями XMLType, как **ExistsNode(), isFragment() и getNumberVal().** Создание функционального индекса иллюстрирует следующий пример:

```
create index quef_idx on xdkfaq_xmltype
       (substr(sys.xmltype.getStringVal(
       sys.xmltype.extract(answer,'/answer/keywords/text()')),1,255));
       select sys_xmlagg(
       sys_xmlgen(relative_type(e.question.getStringVal(),e.id),
       sys.xmlgenformattype.createformat('RELATIVES_ROW'))
       ).extract('ROWSET//RELATIVES_ROW').getStringVal()
  into v_result
  from xdkfaq_xmltype e
 where e.id in
     (select f.id
      from xdkfaq_xmltype f
      where f.answer.extract('/answer/keywords/text()').getStringVal()
                              like '%'||key||'%');
```

Этот функциональный индекс позволяет вести поиск связанных ответов на вопросы одновременно с основным поиском. Без функционального индекса ответ на запрос может занять несколько минут, а при использовании индекса — меньше секунды.

Прямая связь с содержимым БД с использованием URI-Refs

Когда еще не было СУБД Oracle9i, навигация по содержимому баз данных Oracle осуществлялась только с помощью операторов SQL. С появлением Oracle9i появилась возможность использовать для навигации и извлечения данных URI-ссылки. DBUri-Ref — это локальные ссылки (URL), действующие только в базе данных.

После расстановки ссылок DBUri помеченную ими информацию можно извлекать из базы данных с помощью SQL Query. Чтобы построить ссылки на информацию, хранящуюся в базе данных, можно генерировать ссылку DBUri-Ref из SQL Query. В следующем далее примере показано, как в запросе SQL Query для создания ссылок на ответы FAQ с идентификационным номером 5 используется новая функция **sys_dburigen**.

```
select sys_xmlagg(
   sys_xmlgen(sys_dburigen(id,answer))).extract('//URL/text()').getStringVal()
   from xdkfaq where id = 5;
```

В нашем FAQ-приложении для создания индексов глоссария FAQ используются ссылки DBUri-Ref.

Построение глоссария

В этой главе уже приводился скрипт, строящий таблицу **xdkfaq_glossary** для ключевых слов, которые нужно связать с онлайновыми определениями. Теперь надо сгенерировать XML-файлы с использованием имени глоссария и DBUri-ссылок. Для этого можно создать в таблице FAQ триггер, добавляющий ссылки по всем элементам </keyword> внутри файла Glossary. Это можно сделать с помощью следующей XSL-таблицы стилей и SQL-скрипта:

```
<?xml version="1.0"?>
<xml:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
version="1.0">
<xsl:output method="xml" indent="yes" />
<xsl:template match="node()|@*">
- <!--
 Copy the current node
-->
<xsl:copy>
- <!--
 Including any attributes it has and any child nodes
-->
<xsl:apply-templates select"@*|node()" />
</xsl:copy>
</xsl:template>
<xsl:template match="keywords">
<keywords>
<xsl:if test="document('glos.xml')/glossarys/link[@name=current()/text()]">
<xsl:attribute name="link">
```

```
<xsl:value-of
   select="document('glos.xml')/glossarys/link[@name=current()/text()]"/>
</xsl:attribute>
</xsl:if>
<xsl:apply-templates/>
</keywords>
</xsl:template>
</xml:stylesheet>

-----------------------------------------------------------------
FAQDBGlossary.sql
 Create Glossary for the input XML document
 grant javauserpriv to xdkdemo
 grant javasyspriv to xdkdemo
 modify the uri:/private/tomcat/webapps/xdkdemo/source/faq/sql
-----------------------------------------------------------------

CREATE OR REPLACE PROCEDURE MakeGlossary (xmlbuf IN VARCHAR2,
                                          resbuf IN OUT NOCOPY VARCHAR2) AS

   p xmlparser.Parser
   dir varchar2(100) :='/private/public_html/xdk/source/faq/glossary';

   xmldoc xmldom.DOMDocument;

   proc xslprocessor.Processor;
   ss xslprocessor.Stylesheet;
   xsldoc xmldom.DOMDocument;
Begin
 new parser
   p := xmlparser.newParser;
   xmlparser.SetBaseDir(p,dir);

 -- parse xml buffer
    xmlparser.parseBuffer(p, xmlbuf);

 -- get document
    xmldoc := xmlparser.getDocument(p);

 -- parse xsl buffer
    xmlparser.parse(p, dir ||'/'||'glos.xsl');

 -- get document
    xsldoc := xmlparser.getDocument(p);
```

```
  — make stylesheet
     ss := xslprocessor.newStylesheet(xsldoc, dir);

  — process xsl from doc
     proc := xslprocessor.newProcessor;

    xslprocessor.processXSL(proc,ss,xmldoc,resbuf);

    xmldom.freeDocument(xmldoc);
    xmldom.freeDocument(xsldoc);
    xslprocessor.freeStylesheet(ss);
    xslprocessor.freeProcessor(proc);

    xmlparser.freeParser(p);
end;
/
show errors;
```

На рис. 9.6 показан ответ FAQ с термином "XPath", который оформлен как ссылка на определение глоссария. Если щелкнуть по этому слову мышью, оно будет передано в качестве параметра в другую XSQL-страницу **faqdburi_show.xsql**, присоединенную к этой ссылке DBURI_Ref — **/PUBLIC/XDKFAQ_GLOSSARY/ROW**.

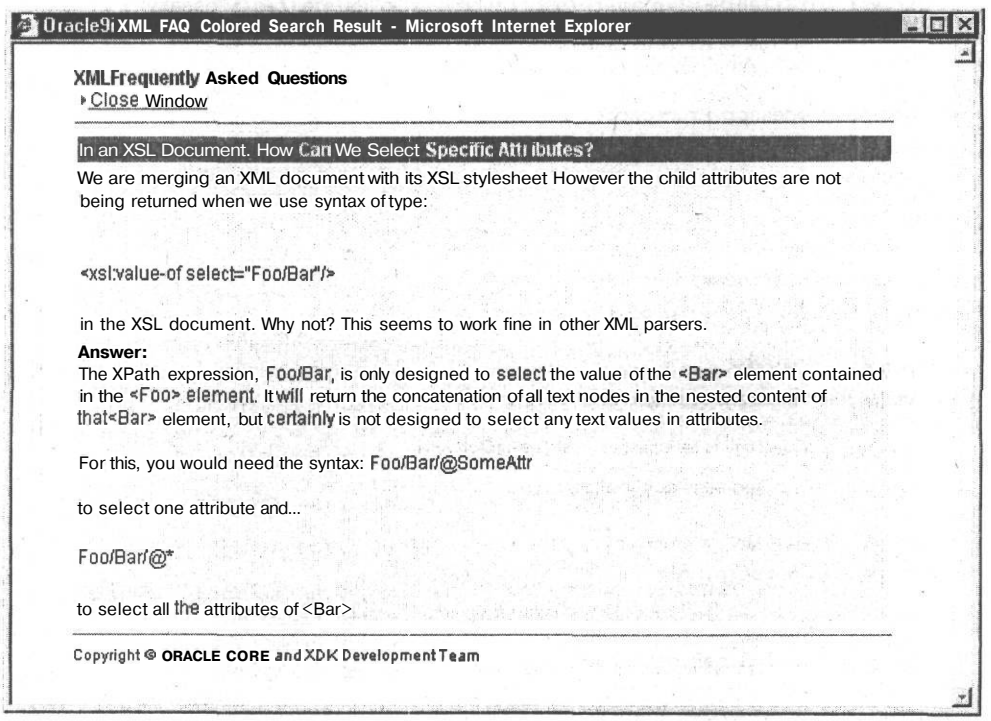


Рис. 9.6. *Ответ FAQ с DBURI-ссылкой*

Расширение приложения

В предыдущем разделе проделана основная работа по созданию реального приложения с поддержкой технологии XML, с использованием XML функциональных возможностей СУБД Oracle9i и компонентов инструментария разработчика XDK. Эту заготовку можно расширять и модифицировать. Одно такое расширение обсуждается в последнем разделе этой главы. Оно представляет собой выдачу списка "Top 20" FAQ-ответов, составленного на основании количества обращений пользователей. Такое Web-приложение можно назвать очень распространенным. На рис. 9.7 показана целиком вся страница, которая генерируется из обновляемого каждый час

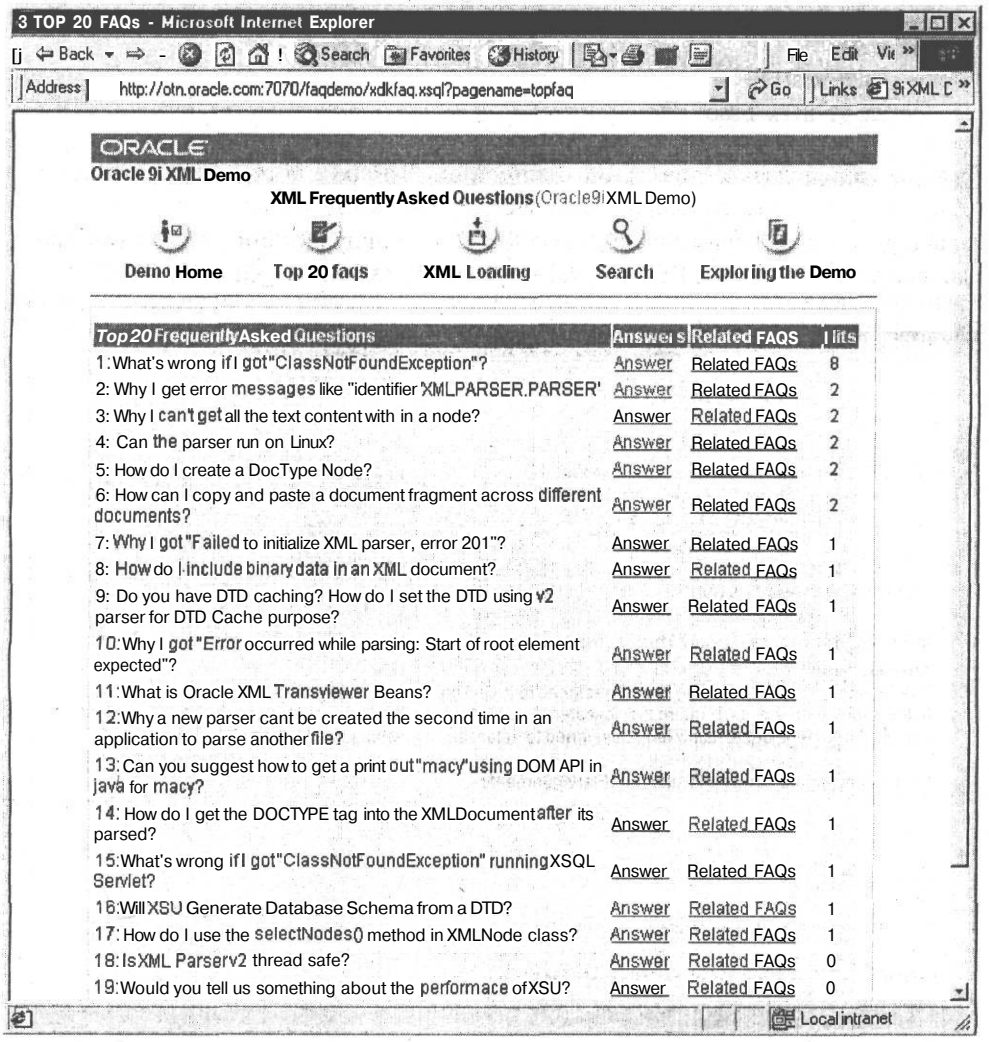


Рис. 9.7. *Страница Top 20 FAQ*

CLOB-объекта, а этот объект, в свою очередь, обновляется в соответствии с данными о количестве обращений пользователей к каждому вопросу и ответу.

На этой странице представлены также "Relative FAQ", т. е. вопросы и ответы, связанные с каждым вопросом из списка "Top 20". Они извлекаются из кэшей своих собственных CLOB-объектов, как это было показано ранее. Страница на рис. 9.7 сгенерирована с помощью следующего кода:

```
<?xml version="1.0"?>
<page connection="xdkdemo" xmlns:xsql="urn:oracle-xsql">
<webpage title="Top 20 Frequently Asked Question">
<xsql:query max-rows="20">
<![CDATA[
   SELECT *
     FROM XDKFAQ
   ORDER BY hits DESC
]]>
</xsql:query>
</webpage>
</page>
```

Полный исходный код этой демонстрационной программы доступен на сайте издательства "Лори".

Глава 10

XML-приложения, предлагаемые на сайте OTN

На сайте *Oracle Technology Network (OTN)* есть несколько приложений, в которых используется технология XML. Здесь разработчики могут протестировать XML-технологии компании Oracle и ознакомиться с последними примерами использования XML, которые могут оказаться полезными для решения реальных проблем. Сайт OTN находится по адресу **http://otn.us.oracle.com.** На его главной странице слева есть вертикальная навигационная панель, на которой располагаются ссылки на сайты, имеющие отношение к OTN. Здесь можно найти ссылку на домашнюю страницу XML, но прямого доступа на нее с главной страницы OTN нет. На нее можно попасть, щелкнув по ссылке Technologies на навигационной панели или набрав в адресном окне браузера её URL:
http://otn.oracle.com/tech/xml/xdkhome.html.

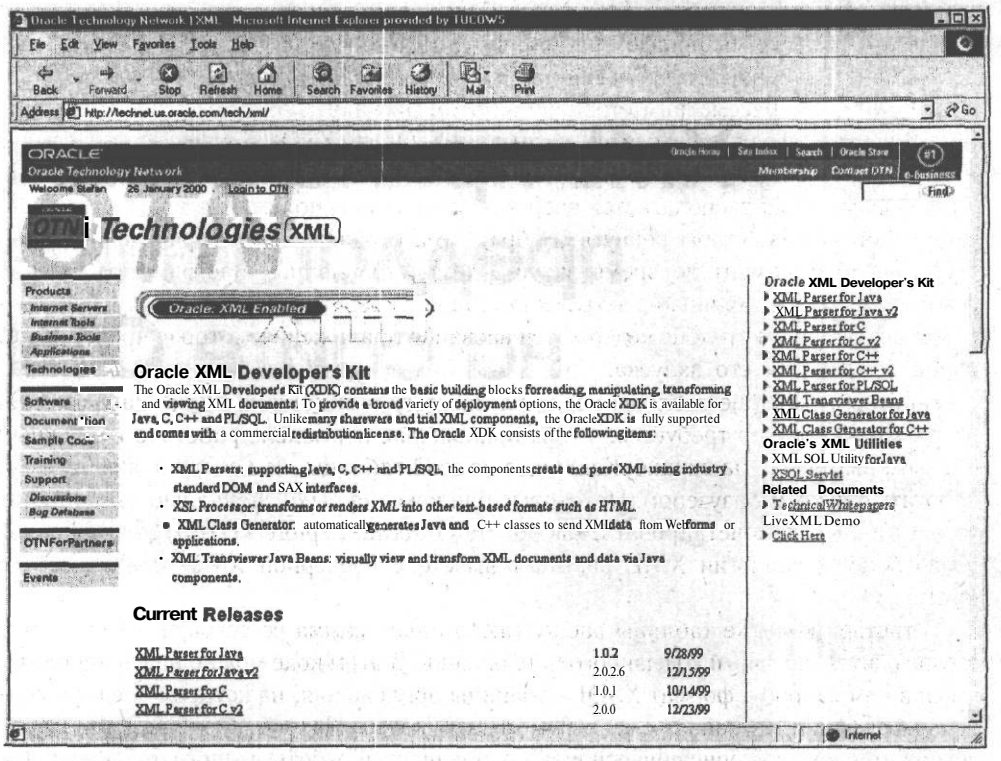


Рис. 10.1. *Страница XML Home и ссылка Live XML Demo*

На домашней странице XML можно найти все доступные XML-продукты Oracle, они входят в комплекты разработчиков Oracle XML Developer's Kits (XDK). В настоящее время на сайте есть пять инструментальных комплектов XDK для разных языков программирования: Java, JavaBeans, C, C++ и PL/SQL. Чтобы помочь пользователям разобраться с тем, как работать с каждым продуктом, в пакеты XDK включены демонстрационные приложения, разработанные Стивом Мюнчем (Steve Muench). Их также можно использовать для того, чтобы убедиться в правильности установки и функционирования соответствующих XML-продуктов.

Кроме этих демонстрационных примеров есть также несколько примеров приложений, которые размещены непосредственно на домашней странице XML. Они запускаются на исполнение прямо через Web и представляют собой интерактивную демонстрацию возможностей технологии XML. Эти демонстрационные приложения призваны проиллюстрировать использование технологий Oracle XML для решения реальных проблем. В этой главе дается обзор приложений, опубликованных на сайте OTN, и их работы. В конце главы объясняется установка и запуск этих приложений.

Доступ к XML-приложениям

Чтобы получить доступ к XML-приложениям, опубликованным на сайте OTN, нужно щелкнуть по ссылке Live XML Demo, как показано на рис. 10.1.

По этой ссылке мы попадем на страницу, показанную на рис. 10.2, где есть список демонстрационных приложений, в которых используются XML и Oracle9i.

Внимательно рассмотрим страницу на этом рисунке. Она состоит из двух фреймов. В верхнем фрейме находится заголовок — Oracle XSQL Servlet Demos, ссылки на OTN и страницы демонстрационных программ, использующих технологию XML. Этот фрейм всегда располагается вверху и его можно использовать в качестве навигационной панели, чтобы вернуться обратно на страницу или направиться на сайт OTN в любой момент во время запуска демонстрационных программ. В нижнем фрейме находится таблица из четырех столбцов, содержащая список доступных приложений. В первом столбце содержится название приложения, которое представляет собой ссылку для его запуска.

Во втором столбце таблицы находится пиктограмма ie5 или any. Она указывает на то, какой браузер требуется, чтобы приложение нормально работало и должным образом выглядело на экране. Как видно из этой таблицы, не все приложения могут работать с любым браузером. Некоторые приложения были специально созданы для того, чтобы продемонстрировать, как браузер Internet Explorer 5 (IE5) со встроенной поддержкой технологии XML работает вместе с серверной XML-технологией от Oracle.

В третьей колонке таблицы расположена пиктограмма со ссылкой на страницу, где находится исходный кодданного приложения. В этом коде можно найти используемый в нем запрос и формат XSQL-страницы приложения, на которой представлена вся информация, связанная сданным исходным кодом. Наконец, в четвертой колонке содержится краткое описание основных особенностей работы данного приложения.

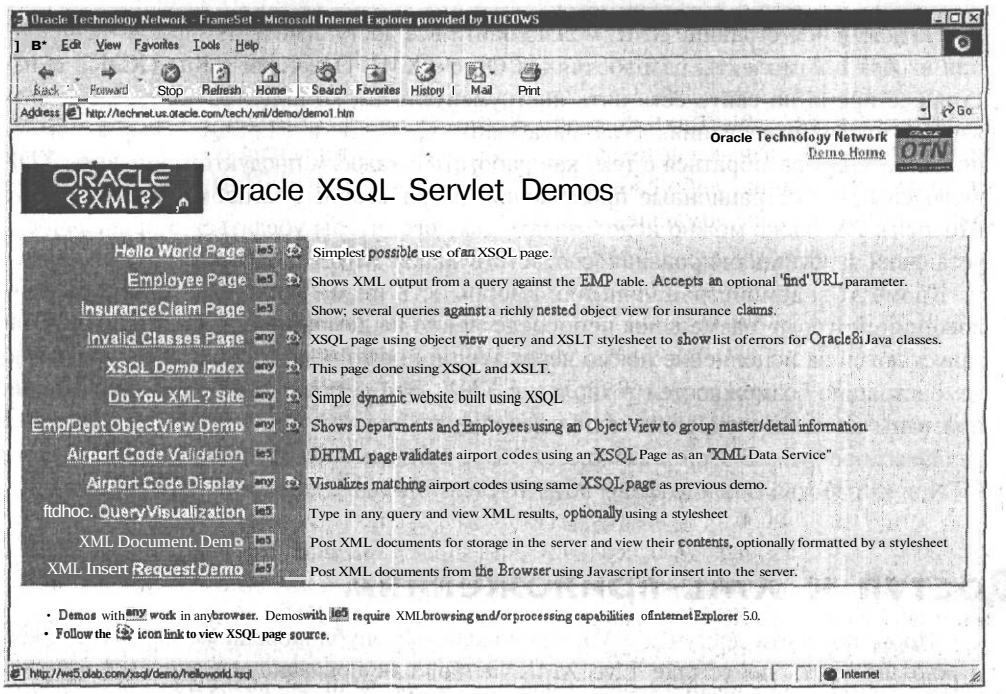


Рис. 10.2. *Демонстрационные приложения Oracle XML Servlet*

Щелчком мышью по ссылке любого приложения из первого столбца открывается страница этого приложения в нижнем фрейме экрана. Чтобы увидеть приложения в новом окне браузера без фреймов, можно запустить их, щелкнув правой кнопкой мыши. В браузерах, поддерживающих эту функцию, появится список опций. Если выбрать опцию "Open in New Window" (открыть в новом окне), откроется новое окно для этого же приложения, но без фреймов. Таким способом можно запустить на исполнение сразу несколько приложений. Кроме того, в адресном окне нового окна браузера будет располагаться URL-адрес вызова выбранного приложения. Это может оказаться полезным, если нужно поэкспериментировать с добавлением или изменением URL-параметров, которые используются при запуске приложения.

Что демонстрируют XML-приложения

СУБД Oracle8i/9i и инструментальные пакеты Oracle XML Developer's Kits поддерживают все основные технологии, необходимые для комбинирования на сервере технологий SQL, XML и XSL. Компоненты каждого пакета XML Developer's Kit позволяют быстро и легко разрабатывать Java, G, C++ и PL/SQL-приложения, использующие основные технологии.

В пакетах XDK уже есть многие типичные классы приложений, и их можно строить вообще без всякого программирования. Это осуществляется с помощью простого, но мощного языка сценариев, построенного на базе шаблонов. Этот язык позволяет довольно быстро и недорого разрабатывать настраиваемые программные модули с самыми разными функциями. Такой подход, который либо вообще не требует программирования, либо сводит его к минимуму, демонстрируют XML-приложения, находящиеся на домашней странице XML.

Эти XML-приложения построены на базе стандартных шаблонов, называемых *Oracle XSQL-страницы* и поддерживаемых сервлетом XSQL Servlet. Сервлет устанавливается на любом Web-сервере, который поддерживает сервлеты, и используется с любой базой данных с интерфейсом JDBC. Работая *с* базой данных Oracle8i или 9i, можно использовать все функции Oracle XSQL-страниц для создания мощных, но простых приложений.

Те, кто знаком с программированием на языке SQL, могут использовать XSQL-страницы вместе с файлами XSL-преобразования для генерации HTML-, XML- и практически любых других текстовых файлов, т. е. для генерации ответов на запросы к базе данных. Для создания XSQL-страниц и соответствующих файлов XSL-преобразования, которые, по сути, являются XML-документами, нужен только текстовый редактор. XSQL-страница — это правильный XML-документ с расширением **.xsql**. Она содержит запрос и необязательную ссылку на одну или несколько XSL-таблиц стилей, которые применяются к XML-данным, извлекаемым из базы данных.

После того как XSQL-страница построена и загружена на Web-сервер, на нее можно отправлять запросы из браузера, используя для этого URL-адрес:

```
http://yourserver.com/EmpXSQLPage.xsql?dept=30
```

На сервере этот запрос обрабатывает сервлет XSQL Servlet, и процедура эта состоит из следующих шагов:

1. Синтаксический разбор XSQL-страницы и извлечение всей релевантной информации, относящейся к данному запросу
2. Установка соединения с базой данных Oracle8i/9i с использованием псевдонима
3. Отправка запроса(ов), заданных на XSQL-странице, в утилиту XML SQL Utility (XSU)
4. Результат запроса направляется в XML-документ, который отправляется обратно в сервлет
5. Применение XSL-преобразования к XML-документу, если определена XSL-таблица стилей
6. Возвращение результирующего документа в браузер

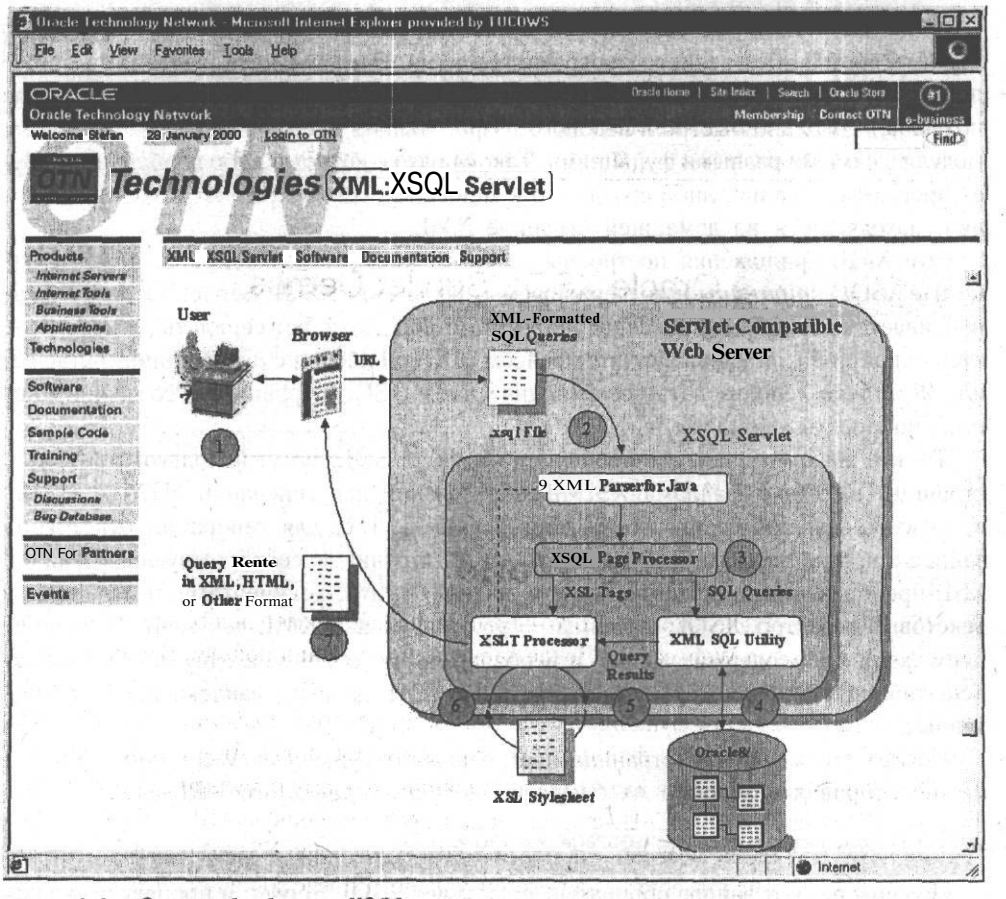


Рис. 10.3. *Схема обработки XSQL-страниц*

Если таблица стилей преобразует XML-данные из запроса к базе данных в HTML-формат, этот HTML-файл можно увидеть непосредственно в окне любого браузера. Процесс обработки XSQL-страниц на сервере показан на рис. 10.3.

Сервлет XSQL Serviet и утилиту XML SQL Utility можно загрузить с сервера OTN, они входят в комплект разработчика XDK for Java. Подробные инструкции по установке XSQL Serviet даны в конце главы в разделе "Установка и запуск XML-приложений". ,

XML-приложения

В этом разделе более подробно рассмотрим каждое приложение, предлагаемое для загрузки на сайте OTN.

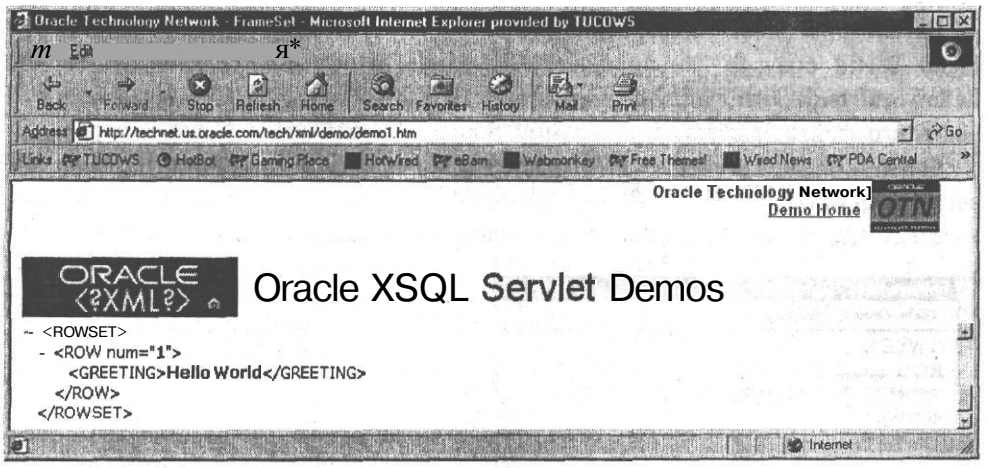


Рис. 10.4. *Выходной экран приложения Hello World*

Приложение Hello World

Приложение Hello World — это самый простой пример использования Oracle XSQL-страницы на Web-сайте. После запуска приложения появляется экран, показанный на рис. 10.4.

В браузере IE5 есть функция Direct Browsing, которая позволяет автоматически выводить на экран XML-файлы, как это делается с HTML- и GIF-файлами. В XML-файлах нет средств для их представлений, поэтому браузер IE5 для преобразования XML-файлов в HTML-формат использует встроенную XSL-таблицу стилей. Если в XML-файле нет ссылки на какую-то определенную XSL-таблицу стилей, которую IE5 может использовать для его визуального представления, то для создания древовидного представления XML-контента IE5 использует таблицу стилей по умолчанию (см. рис. 10.4). На выходе этого приложения находится XML-документ, древовидная структура которого выведена в окне браузера IE5.

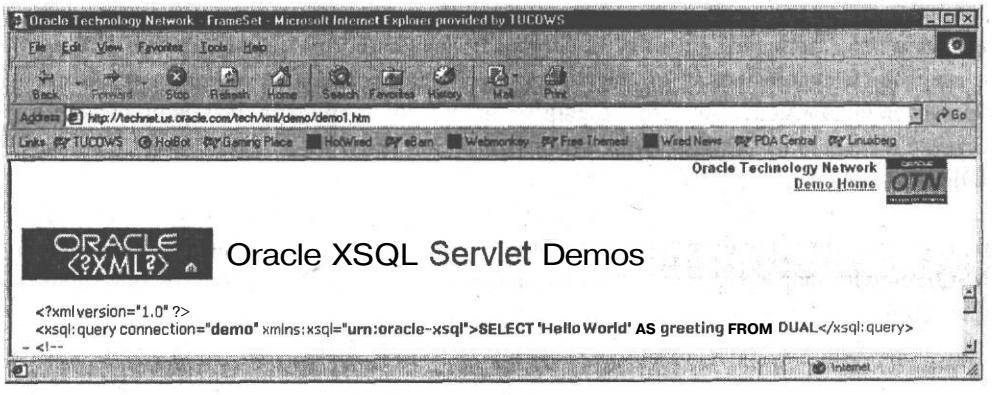


Рис. 10.5. *Исходный код XSQL-страницы приложения Hello World*

Рассмотрим работу приложения Hello World. После щелчка мышью по ссылке Hello World браузер запрашивает URL-адрес http://ws5.olab.com/xsql/demo/**helloworld.xsql**. Этот файл показан на рис. 10.5. Можно вывести на экран исходный код этого файла, выбрав пиктограмму Source (на ней изображена камера).

Файл, показанный на рис. 10.5, представляет собой XSQL-страницу. Его расширение **.xsql** указывает Web-браузеру на то, что для обработки запроса нужно использовать XSQL Servlet. Теперь вернемся к выходному экрану приложения Hello World, показанному на рис. 10.4. Чтобы увидеть данные, которые возвращаются в браузер как результат работы приложения Hello World, нужно щелкнуть правой кнопкой мыши по нижней части экрана, где показан XML-документ, и в выпадающем меню выбрать опцию View Source (просмотр источника). Страница с исходным кодом появится в окне текстового редактора Notepad (см. рис. 10.6).

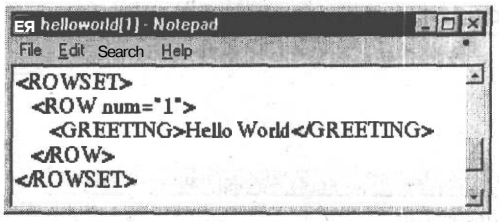


Рис. 10.6. *Приложение Hello World — данные, возвращаемые с сервера*

Вернемся к рис. 10.5. Понятно, что страница в действительности имеет только один элемент.

```
<xsql:query connection="demo" xmlns:xsql="urn:oracle-xsql">
 SELECT 'Hello World' AS greeting FROM DUAL
</xsql:query>
```

После синтаксического разбора этой XSQL-страницы XSQL Servlet ищет элементы с тегом **<xsql:query>**. Затем XSQL Servlet рассматривает текст в теге **<xsql:query>** как SQL-оператор и направляет его в базу данных для исполнения, используя для доступа в БД информацию, содержащуюся в атрибуте connection. Результат запроса автоматически форматируется как XML-файл и отправляется обратно в браузер.

Структура этого выходного XML-документа соответствует внутренней структуре схемы базы данных, которая вернула результаты запроса. Столбцы отображаются на элементы верхнего уровня. Скалярные значения отображаются на элементы с текстовым содержимым; объектные типы — на элементы с атрибутами, которые являются подэлементами; а коллекции — на списки элементов. Наконец, объектные ссылки и ссылочные ограничения отображаются на ссылки на идентификаторы XML IDREF.

Для приложения Hello World результирующий набор данных содержит один столбец — greeting (приветствие) — и одну строку с одним элементом — Hello World. XSQL Servlet форматирует названный набор данных в XML-файл, как показано на рис. 10.6.

Еще один простой пример создания XML-документа из запроса, отправленного на XSQL Servlet:

```
<xsql:query connection="demo" xmlns:xsql="urn:oracle-xsql">
SELECT DEPTNO, ENAME FROM EMP WHERE DEPTNO = 7654;
</xsql:query>
```

Он генерирует следующий XML-документ:

```
<?xml version="1.0"?>
<ROWSET>
   <ROW id="1">
      <DEPTNO>7654</DEPTNO>
      <ENAME>STEVE</ENAME>
   </ROW>
   <ROW id="2">
      <DEPTNO>7654</DEPTNO>
      <ENAME>JOHN</ENAME>
   </ROW>
   <ROW id="3">
      <DEPTNO>7654</DEPTNO>
      <ENAME>MARRY</ENAME>
   </ROW>
</ROWSET>
```

Рассмотрим формат выходного XML-документа. По умолчанию считается, что ROWSET — это имя элемента XML-документа. ROW — это имя элемента каждой строки в ответе на запрос. Данные, которые находятся, например, в DEPTNO и ENAME, представлены в виде элементов, вложенных в узел ROW.

Таким образом, данные здесь представлены в виде элементов, а атрибуты используются там, где это необходимо для ввода ограничений на данные. Если приложению для форматирования требуется другой набор тегов, XSL-таблица стилей может выполнить преобразование динамически (см. следующий раздел).

Теперь еще раз посмотрим на тег **<query>** в файле **helloworld.xsql**:

```
<xsql:query connection="demo" xmlns:xsql="urn:oracle-xsql">
```

Атрибут **connection="demo"** определяет, где нужно искать информацию, необходимую для подключения к базе данных. Сервлет XSQL Servlet использует для этого файл конфигурации **XSQLConfig.xml.** У этого файла может быть несколько входных элементов, каждый из которых представляет собой отдельную связь с базой данных. Каждый такой элемент содержит всю необходимую информацию для подключения и доступа в нужную базу данных. Файл **XSQLConfig.xml** представляет собой XML-файл следующего формата:

```
<?xml version="1.0"?>
<connectiondefs dumpallowed="no">
   <connection name="demo">
      <username>scott</username>
      <password>tiger</password>
      <dburl>jdbc:oracle:thin:@localhost:1521:ORCL</dburl>
      <driver>oracle.jdbc.driver.OracleDriver</driver>
   </connection>
   <connection name="lite">
      <username>system</username>
      <password>manager</password>
      <dburl>jdbc:Polite:POlite</dburl>
      <driver>oracle.lite.poljdbc.POLJDBCDriver</driver>
   </connection>
</connectiondefs>
```

При установке XSQL Servlet можно отредактировать этот файл, чтобы создать новые входные элементы для специальных соединений с базой данных. Каждый такой входной элемент должен для успешной установки соединения содержать элементы для имени пользователя, пароля, URL-адреса базы данных и JDBC-драйвера.

Приложение Hello World — это один XML-документ с расширением **.xsql**, содержащий запрос к базе данных. Когда XSQL-страница запрашивается браузером, XSQL Servlet выдает результат в виде XML-файла. Процедуру установки соединения с БД выполняет программа XSQL Connection Pool. XSQL Servlet отправляет запрос через XML SQL Utility и возвращает результаты в XML-формате. Так как XSQL-страница Hello World не обращается к таблице стилей для преобразования XSL-файла в выходной файл XML-формата, браузер получает ответ на запрошенную XSQL-страницу в XML-формате.

Приложение Employee Page

Приложение Employee Page (страница сотрудника) демонстрирует, как можно отправить запрос в базу данных, а сервер форматирует результаты этого запроса в формат HTML. При такой технологии каждый браузер будет выводить на экран результат в нужном формате. На рис. 10.7 показан экран браузера после выполнения приложения Employee Page.

Это приложение использует одну важную функцию Oracle XSQL-страниц: способность запрашивать преобразование в HTML-формат результирующего XML-файла. Преобразование производится на сервере с помощью XSQL Servlet в отличие от форматирования, которое производится на стороне клиента браузер IE5 при визуализации XML-файлов. В обоих случаях форматирование выполняется с применением к XML-файлу XSL-таблицы стилей. Когда на странице браузера, показанной

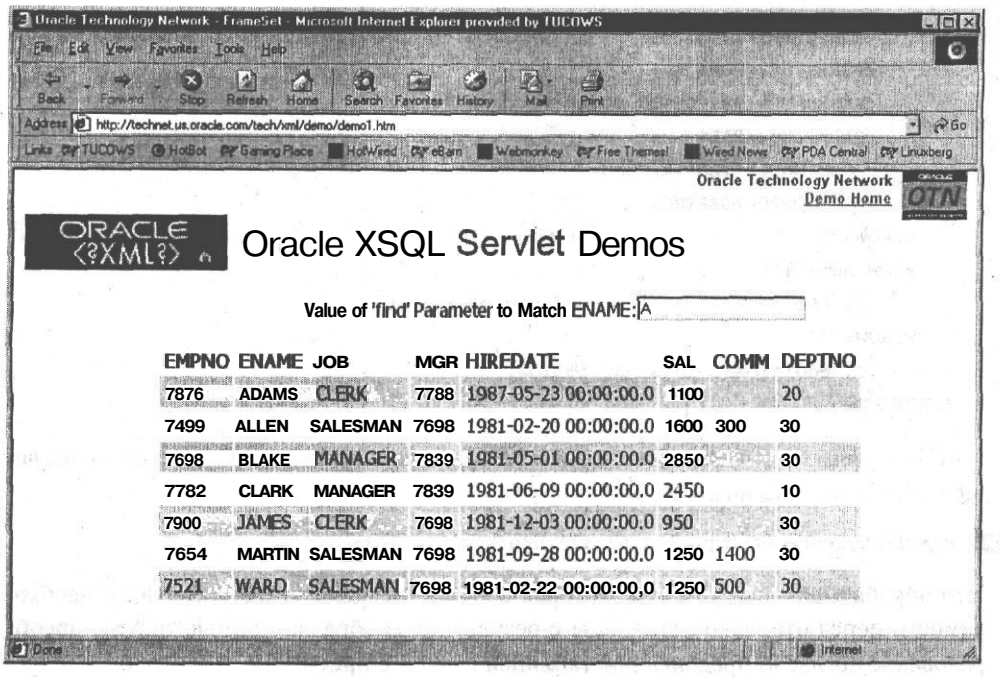


Рис. 10.7. *Результаты выполнения приложения Employee Page*

на рис. 10.4, выбрана ссылка Employee Page, браузер запрашивает файл XSQl-страницы **emp.xsql,** который содержит следующие XML-элементы:

```
<?xml-stylesheeet type="text/xsl" href="emp.xsl"?>
<xsql:query connection="demo" xmlns:xsql="urn:oracle-xsql"
            find="%"
            sort="ENAME"
            null-indicator="yes" >
      SELECT *. FROM EMP
      WHERE ENAME LIKE '%{@find}%'
      ORDER BY {@sort}
</xsql:query>
```

Если строки, описывающей таблицу стилей, в этом файле нет, процессор страниц XSQL Page вернет следующий XML-документ:

```
<?xml version="1.0"?>
<ROWSET>
   <ROW num="1">
      <EMPNO>7876</EMPNO>
      <ENAME>ADAMS</ENAME>
```

```
        <JOB>CLERK</JOB>
        <MGR>7788</MGR>
        <HIREDATE>1987-05-23 00:00:00.0</HIREDATE>
        <SAL>1100</SAL>
        <COMM NULL="YES"></COMM>
        <DEPTNO>20</DEPTNO>
    </ROW>
    <ROW num="2">
        ... the data for the second row of result ...
    </ROW>
        ... additional rows may follow...
</ROWSET>
```

Посмотрим, что произойдет, если в XSQL-страницу **emp.xsql** включить следующее описание таблицы стилей:

```
<?xml-stylesheeet type="text/xsl" href="emp.xsl"?>
```

Это обрабатывающая команда, которая сообщает сервлету XSQL Servlet о необходимости перед отправкой страницы с результатами обратно применить XSL-преобразование согласно предписанию таблицей стилей **emp.xsl.** Это команда определяет имя XSL-таблицы стилей, которую Oracle XSQL Servlet должен использовать для преобразования результирующего XML-документа в документ формата HTML (или любого другого текстового формата). Когда с помощью вышеуказанной команды запрашивается XSQL-страница, браузер принимает обратно HTML-документ, в котором данные представлены в виде таблицы.

Итак, приложение Employee Page состоит из двух текстовых файлов:

- **emp.xsql** Это XML-файл, содержащий SQL-запрос, ссылку на таблицу стилей **emp.xsl** и ссылку на псевдоним, который предоставляет информацию для связи с базой данных.
- **emp.xsl** Это файл XSL-таблицы стилей, определяющий XSL-преобразование, которое должен выполнить XSL Processor.

XSQL Servlet формирует окончательный результат, обрабатывая результирующий XML-файл и описанный XSL-файл, который XSL Processor включает в анализатор синтаксиса Oracle XML Parser. XSQL Servlet получает результирующие данные в XML-формате, направляя SQL-запрос в утилиту Oracle XML SQL Utility. Важно хорошо разобраться в том, как сформатировать результирующие данные в XML-файл, чтобы создать такую XSL-таблицу стилей, которая преобразует промежуточный XML-формат в окончательный формат, нужный клиенту.

Приложение Insurance Claim

Приложение Insurance Claim (заявление о выплате страхового возмещения) демонстрирует несколько разных способов использования XSQL-таблиц в реальной ситуации работы любой страховой компании. В этих примерах данные выводятся в виде XML-файлов, полученных в результате запросов к таблицам, содержащим определяемые пользователем типы данных. Модель заявления о выплате страхового возмещения, построенная с помощью универсального языка моделирования *Unified Modeling Language* (*UML*), показана на рис. 10.8.

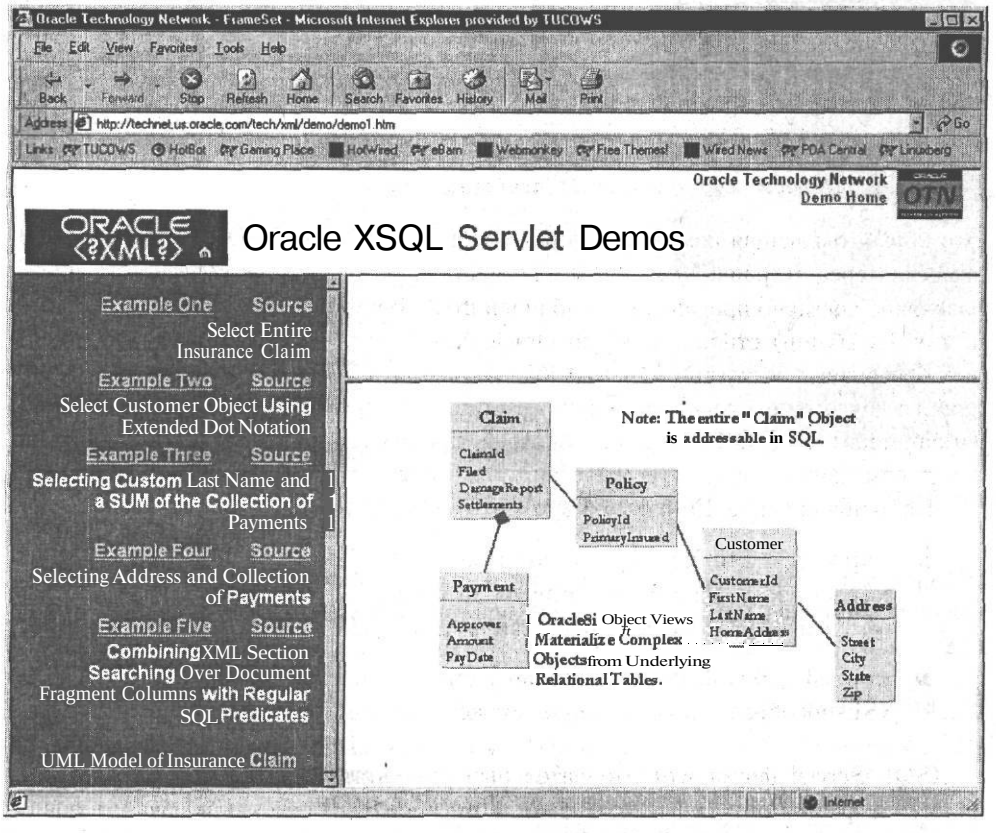


Рис. 10.8. *UML-модель заявления о выплате страхового возмещения в Приложении Insurance Claim*

Со страницы, показанной на рис. 10.8, можно запустить на исполнение пять разных примеров. Правая нижняя часть экрана состоит из двух фреймов: если щелкнуть мышью по Ссылке Source, которая есть в каждом примере, в верхнем фрейме появится исходная XSQL-страница; в нижнем фрейме показан результирующий

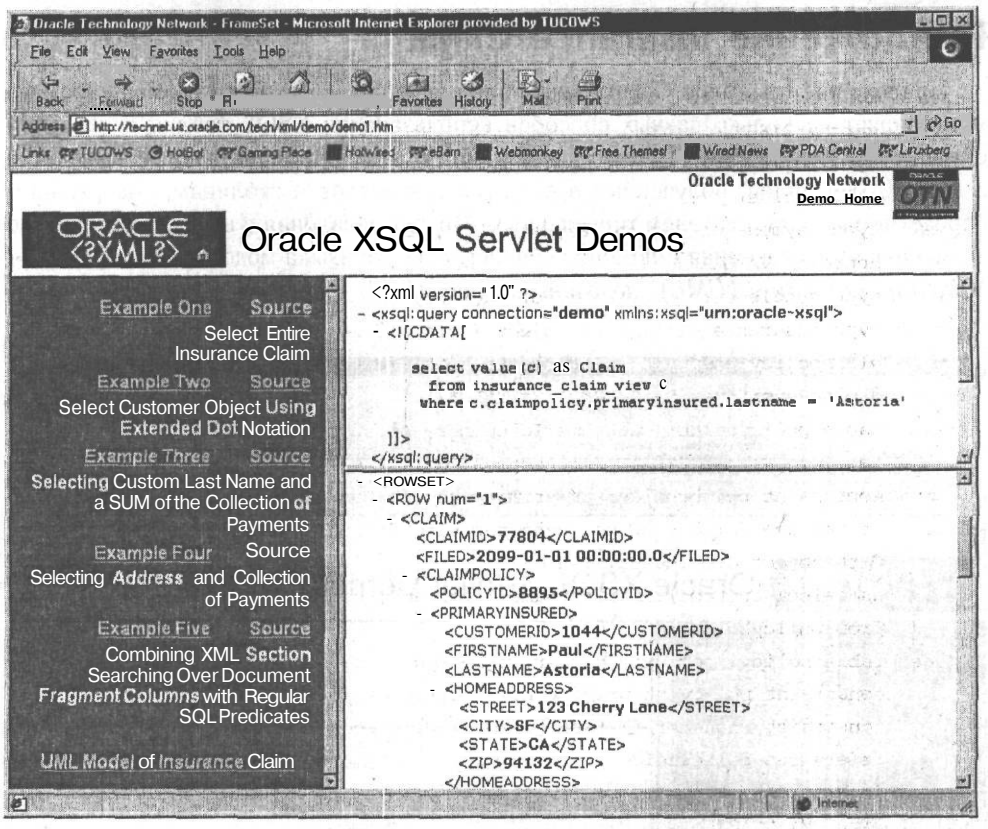


Рис. 10.9. *Результат работы первого примера*

XML-документ, возвращаемый браузеру после запуска соответствующей XSQL-страницы. Исходная XSQL-страница и результирующие данные для первого примера показаны на рис. 10.9.

Внимание

Отметим использование секции CDATA при выполнении запроса. Эта секция должна использоваться, если запрос содержит текст или символы, которые в противном случае могут быть восприняты как разметка.

Следующий листинг представляет собой SQL-запрос для создания таблиц, представлений и определяемых пользователем типов, которые используются во всех примерах приложения Insurance Claim. Например, можно использовать эту DDL-библиотеку вместе с запросом в XSQL-странице данного примера, чтобы понять, как XSQL Servlet генерирует XML-файл из наборов данных, содержащих определяемые пользователем типы.

```
create type address_t as object( Street varchar2(80),
       City Varchar2(80), State VARCHAR2(80), Zip NUMBER );
create type policyholder_t as object( CustomerId number, FirstName
       varchar2(80), LastName varchar2(80), HomeAddress address_t);
create type policy_t as object( policyID number, primaryinsured
       policyholder_t);
create type payment as object( PayDate DATE, Amount NUMBER,
       Approver VARCHAR2(8));
create type settlements_t as table of payment;
create type insurance_claim_t as object ( claimid number, filed date,
       claimpolicy policy_t, settlements settlements_t,
       damageReport varchar2(4000) /* XML */);
create table policyholder( Customerld number, FirstName varchar2(80),
       LastName varchar2(80), HomeAddress address_t,
       constraint policyholder_pk primary key (customerid));
create or replace force view policyholder_view of policyholder_t
       with object OID (customerid)
       as select customerid, firstname, lastname, homeaddress
       from policyholder;
create table policy( policyid number, primarycustomerid number,
       constraint policy_pk primary key (policyid),
       constraint customer_fk foreign key (primarycustomerid)
       references policyholder);
create or replace force view policy_view of policy_t
       with object OID (policyid)
       as select p.policyid, (SELECT value(phv) from policyholder_view phv
       WHERE phv.customerid = p.primarycustomerid) as primaryinsured
       from policy p;
create table insurance_claim( claimid number, filed date,
       claimpolicy number, damageReport varchar2(4000) /* XML */,
       constraint insurance_claim_pk primary key (claimid),
       constraint policy_fk foreign key (claimpolicy) references policy);
create table settlement_payments( claimid number, PayDate DATE,
       Amount NUMBER, Approver VARCHAR2(8),
       constraint claim_fk foreign key (claimid) references
       insurance_claim);
create or replace force view insurance_claim_view of insurance_claim_t
       with object OID (claimid) as select c.claimid,c.filed,
       (SELECT value(pv) from policy_view pv
       WHERE pv.policyid = c.claimpolicy),
       CAST(MULTISET(SELECT PAYMENT(sp.paydate,sp.amount,sp.approver)
       as Payment from settlement_payments sp
```

```
        WHERE sp.claimid = c.claimid) AS settlements_t), c.damagereport
        from insurance_claim c;
create  index ctx_xml_i on insurance_claim(damagereport)
        indextype is ctxsys.context
        parameters ('LEXER Demo SECTION GROUP demo_xml');
```

В первом примере в XSQL-странице используется следующий SQL-запрос:

```
select value(c) as Claim
        from insurance_claim_view c where
        c.claimpolicy.primaryinsured.lastname = 'Astoria'
```

В ответ XSQL Servlet генерирует следующий XML-файл:

```
<?xml version="1.0"?>
<ROWSET>
<ROW num="1">
   <CLAIM>
      <CLAIMID>77804</CLAIMID>
      <FILED>1999-01-01 00:00:00.0</FILED>
      <CLAIMPOLICY>
         <POLICYID>8895</POLICYID>
            <PRIMARYINSURED>
               <CUSTOMERID>1044</CUSTOMERID>
               <FIRSTNAME>Paul</FIRSTNAME>
               <LASTNAME>Astoria</LASTNAME>
               <HOMEADDRESS>
                     <STREET>123 Cherry Lane</STREET>
                     <CITY>SF</CITY>
               <STATE>CA</STATE>
               <ZIP>94132</ZIP>
                  </HOMEADDRESS>
            </PRIMARYINSURED>
         </CLAIMPOLICY>
         <SETTLEMENTS>
            <SETTLEMENTS_ITEM>
               <PAYDATE>1999-01-05 00:00:00.0</PAYDATE>
               <AMOUNT>7600</AMOUNT>
               <APPROVER>JCOX</APPROVER>
            </SETTLEMENTS_ITEM>
         </SETTLEMENTS>
         <DAMAGEREPORT>The insured's <;VEHICLE>car<;/VEHICLE>broke
through the guard rail and plumetted into a ravine. The cause
```

```
was determined to be <;CAUSE>faulty brakes<;/CAUSE> Amazingly
there were no casualities.</DAMAGEREPORT>
      </CLAIM>
   </ROW>
   <ROW num="2">
      . . . the data for row number 2 follows . . .
   </ROW>
</ROWSET>
```

В остальных примерах используются другие запросы, но все они построены тем же самым способом. Выделяется только пример 5, так как в нем используется сразу несколько функций базы данных. Возьмем, например, такой запрос:

```
select sum(n.amount) as TotalApproveAmount
  from insurance_claim_view v, TABLE(v.settlements) n
 where n.approver =. 'JCOX'
   and contains (damageReport, 'Brakes whithin Cause') > 0
```

Этот запрос имеет оператор where, который ищет заявления с определенной XML-разметкой в разделе damageReport. Можно запустить этот запрос на исполнение, используя поддержку технологии XML в программе Intermedia Text. Следующие операции выполняются уже с теми данными, которые удовлетворяют названному условию и в которых в качестве сотрудника, подписавшего заявление на выплату, значится JCOX.

Приложение Invalid Classes

XSQL-страница для приложения Invalid Classes использует для вывода списка ошибок в Java-классах Oracle9i запрос объектного представления и XSL-таблицу стилей. Результирующий экран этого приложения показан на рис. 10.10.

А вот SQL-программа, которая создает пользовательские типы и представление, используемое в этом приложении:

```
create type "XSQLJavaClassError" as object ( "Message" varchar2(4000) );
create type "XSQLJavaClassErrors" as table of "XSQLJavaClassError";
create view "XSQLJavaClassErrorView" as
        select replace(dbms_java.longname(uo.object_name),'/','.')
        "ClassName",
        CAST(MULTISET(SELECT "XSQLJavaClassError"(ue.text)
        FROM user_errors ue WHERE name = uo.object_name
        ORDER BY ue."SEQUENCE") AS "XSQLJavaClassErrors") AS "Errors"
        from user_objects uo where object_type = 'JAVA CLASS'
        and status = 'INVALID'
        order by replace(dbms java.longname(object_name),'/','.');
```

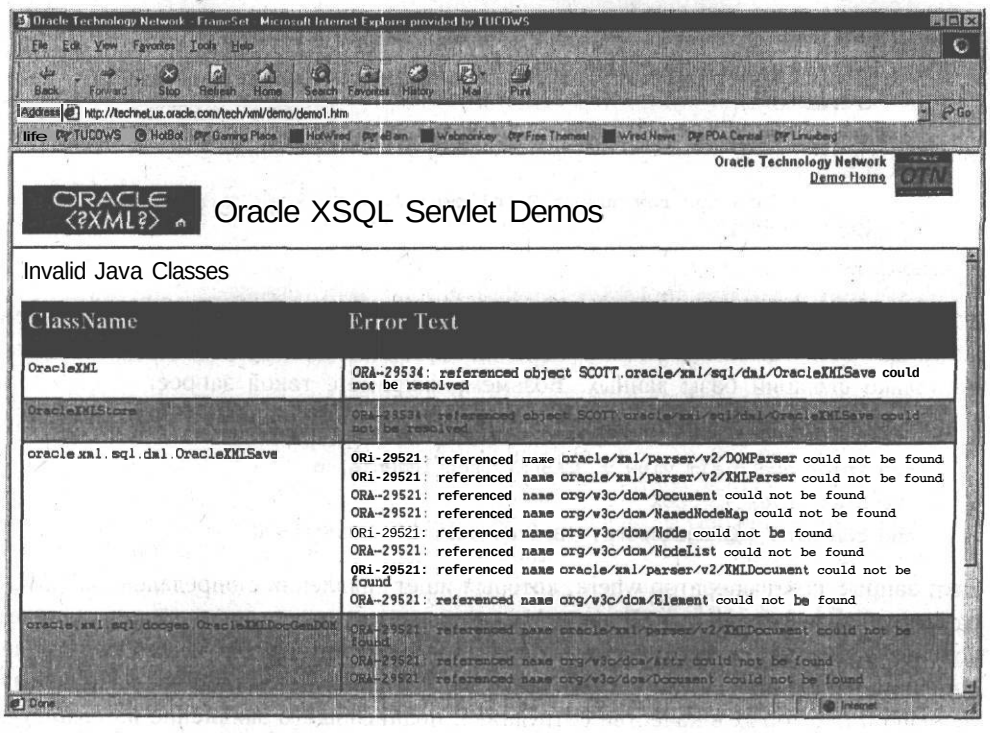


Рис. 10.10. *Приложение Invalid Classes*

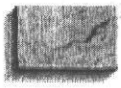
Чтобы понять работу оператора CREATE_VIEW, нужно иметь определенную информацию о модуле **DBMS_JAVA**, который берется из Oracle8i и 9i. При инициализации JServer, т. е. виртуальной Java-машины, работающей на сервере Oracle, скрипт **initjvm.sql** создает PL/SQL-модуль **DBMS_JAVA**. Некоторые из API-интерфейсов **DBMS_JAVA** предназначаются для пользователей, другие существуют только для внутреннего применения. Из всех этих API-интерфейсов рассмотрим только один, который использовался в предыдущем SQL-запросе:

```
FUNCTION longname (shortname VARCHAR2) RETURN VARCHAR2
```

Функция **longname** возвращает значение longname из объекта Java-схемы.

Так как Java-классы и методы могут иметь имена, длина которых превышает максимальную длину SQL-идентификатора, JServer использует для внутреннего доступа в БД сокращенные имена. Эта функция просто возвращает оригинальное Java-имя для любого (потенциально) сокращенного имени. Примером использования такой функции является печать полных составных имен классов, которые по тем или иным причинам оказались неправильными.

```
select dbms_java.longname(object_name) from user_objects
        where object_type = 'JAVA CLASS' and status = 'INVALID';
```



Внимание

Подробно разобранный пример использования этой функции и методы определения того, какой из объектов Java-схемы находится на сервере, можно найти в руководстве "Oracle8i or 9i Java Stored Procedures Developer's Guide".

XSQL-страница, выполняющая этот запрос к базе данных, имеет два элемента:

```
<?xml-stylesheeet type="text/xsl" href="invalidclasses.xsl"?>
<xsql:query connection="demo" tag-case="lower" xmlns:xsql="urn:oracle-xsql">
        select * from "XSQLJavaClassErrorView"
</xsql:query>
```

Описанная XSL-таблица стилей **invalidclasses.xsl** применяется сервлетом XSQL Servlet к XML-документу, представляющему собой набор данных, полученный в результате запроса. Полученный после этого HTML-документ отправляется обратно в браузер. Это приложение аналогично приложению Employee Page, но в нем используется представление, созданное из таблиц USER_ERROR и USER_OBJECT. Это приложение показывает, как легко можно создавать отчеты для пользовательских схем и публиковать их в Web в задаваемом пользователем формате.

Индекс XSQL Demo Index

Рассмотренные выше приложения наглядно демонстрируют, как легко и гибко можно использовать для Web-публикации метод, построенный на базе шаблонов. Это неудивительно: страница Demo Index Page, показанная на рис. 10.2 и использующаяся для навигации по демонстрационным приложениям, также построена с помощью описанной технологии XSQL-страниц. На рис. 10.11 показана примененная для этого XSQL-страница и XSL-таблица стилей, которая строит HTML-файл для страницы, изображенной на рис. 10.2.

Ниже приведена SQL-программа, которая создает и загружает таблицы, использованные этой XSQL-страницей:

```
create table xsqldemolist(title varchar2(200), description varchar2(4000),
       url varchar2(200), sourceurl varchar2(200), ie5only varchar2(1) );
insert into xsqldemolist values ('HelloWorld Page',
       'Simplest possible use of an XSQL page.', 'helloworld.xsql',
       'helloworld-xsql.xml', 'Y');
insert into xsqldemolist values ('Employee Page',
       'Shows XML output from a query against the EMP table. Accepts an
       optional'
       || '''find'' URL parameter.', 'emp.xsql?find=A',
       'empsource.html', 'Y');
```

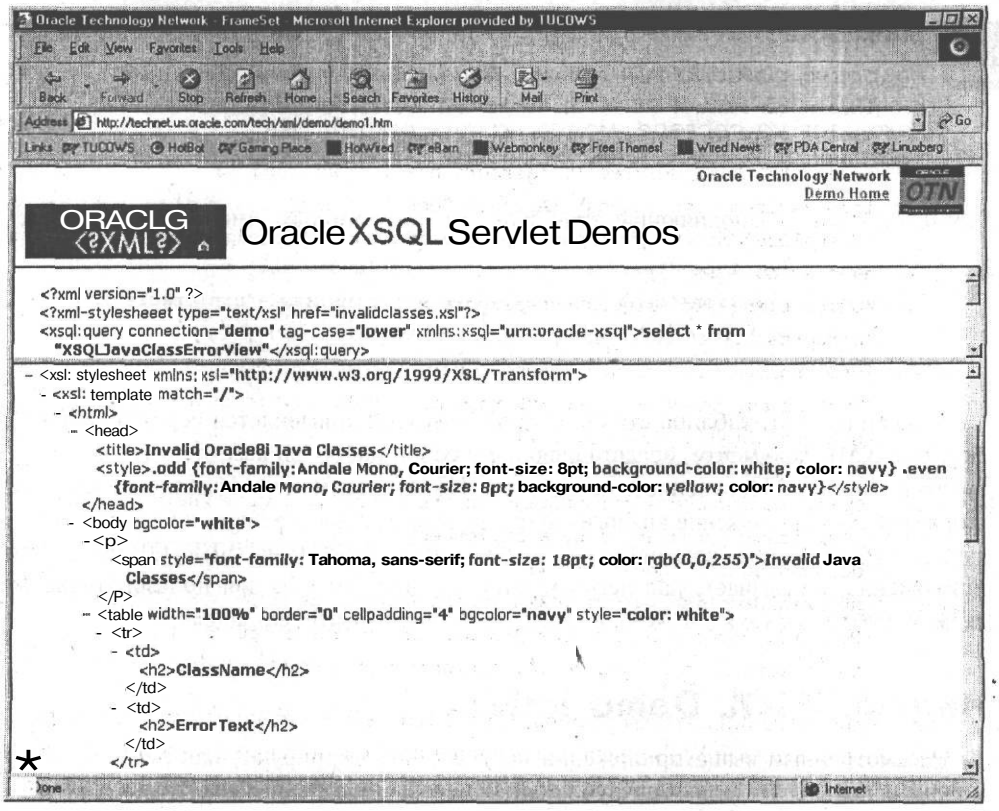


Рис. 10.11. *XSQL-страница и XSL-преобразования, используемые в XSQL Demo Index*

```
insert into xsqldemolist values ('Insurance Claim Page',
        'Shows several queries against a richly nested object view
        for insurance claims.',
        'xsqlov.htm', 'None', 'Y');
insert into xsqldemolist values ('Invalid Classes Page',
        'XSQL page using object view query and XSLT stylesheet to show list of
        || 'errors for Oracle8i Java classes.', 'invalidclasses.xsql',
        'invalidclassessource.html', 'N');
insert into xsqldemolist values ('XSQL Demo Index',
        'This page done using XSQL and XSLT.', 'index.xsql',
        ' indexsource.html', 'N');
insert into xsqldemolist values ('Do You XML? Site',
        'Simple dynamic website built using XSQL', 'doyouxml.xsql',
        'doyouxmlsource.html', 'N');
insert into xsqldemolist values ('Emp/Dept ObjectView Demo',
        'Shows Departments and Employees using an Object View to
```

```
        group master/detail information.',
        'empdept.xsql','empdeptsource.html', ' N');
insert into xsqldemolist values ('Airport Code Validation',
        'DHTML page validates airport codes using an XSQL Page
        as an "XML Data Service"', 'airport.htm', 'None', 'Y');
insert into xsqldemolist values ('Airport Code Display',
        'Visualizes matching airport codes using same XSQL page
        as previousdemo.',
        'airport.xsql?xml-stylesheet=airport.xsl',
        'airportsource.html','N');
insert into xsqldemolist values ('Adhoc Query Visualization',
        'Type in any query and view XML results, optionally using a stylesheet.',
        'sqltoxml.html', 'None', 'Y');
insert into xsqldemolist values ('XML Document Demo',
        'Post XML documents for storage in the server and view their contents,'
        ||'optionally formatted by a stylesheet',
        'docdemo.html', 'None', 'Y');
insert into xsqldemolist values ('XML Insert Request Demo',
        'Post XML documents from the Browser using JavaScript '||
        'for insert into the server.', 'newsstorydemo.html', 'None', 'Y');
```

Как можно видеть из этого примера, СУБД Oracle9i — отличное место для хранения данных и страниц Web-сайта. Следующее приложение продемонстрирует возможности Oracle9i в создании Web-сайтов.

Сайт Do You XML!

Web-сайт Do You XML? — это внутренний Web-сайт компании Oracle. Он демонстрирует, как можно использовать базу данных для обслуживания Web-сайта, HTML-страницы которого создаются сервлетом XSQL Servlet и XSL-таблицами стилей. Страница сайта Do You XML? показана на рис. 10.12.

Этот сайт иллюстрирует несколько функций сервлета XSQL Servlet, которые хорошо подходят для доставки динамического Web-контента с учетом особенностей клиентского устройства. На этом сайте работает страница **doyouxml.xsql**:

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" media="mozilla" href="doyouxml.xsl" ?>
<?xml-stylesheet type="text/xsl" media="MSIE 5" href="doyouxml.xsl" ?>
<?xml-stylesheet type="text/xsl" href="doyouxml.xsl" ?>
```

Отметим, что здесь имеются три описания таблиц стилей. Благодаря тегу **media** в HTTP-заголовке запрашивающего браузера сервлет XSQL Servlet может применить для преобразования полученного XML-файла соответствующую таблицу стилей. Поэтому обладатель полнофункционального браузера и владелец WAP-телефона

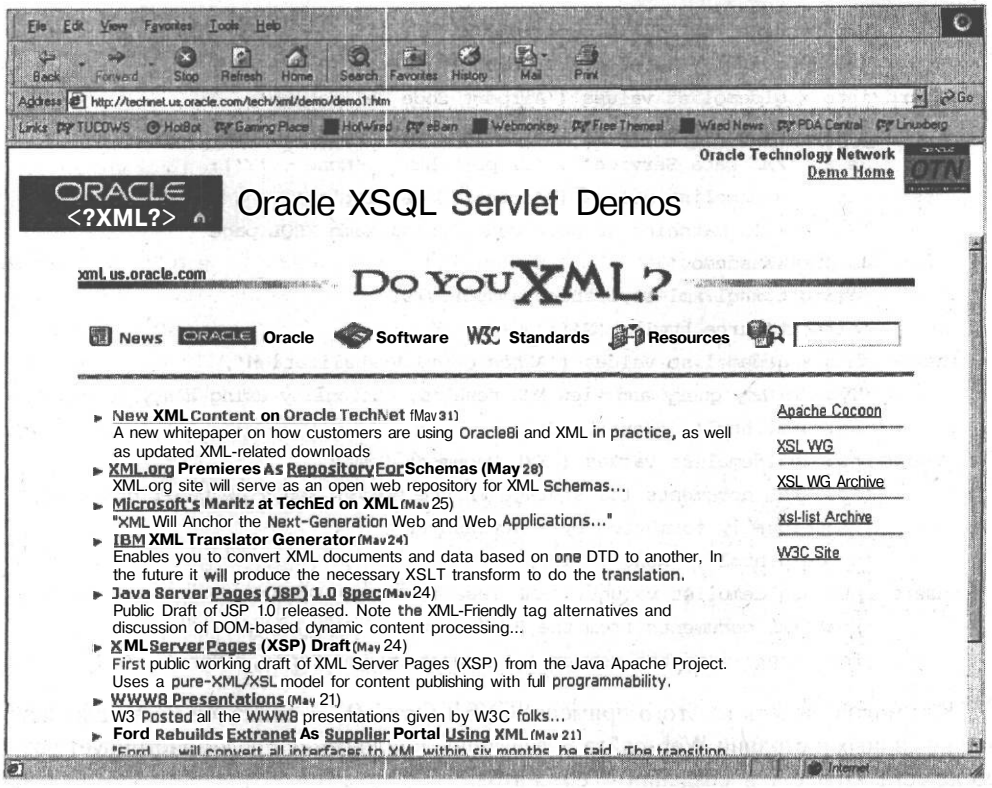


Рис. 10.12. *Do YouXML?*

увидят на экранах своих устройств специально настроенную на них страницу, и им не придется сохранять несколько ее статических версий, а потом заниматься синхронизацией и обновлением информации.

```
<datapage connection="demo" xmlns:xsql="urn:oracle-xsql">
```

Как уже указывалось ранее, элемент datapage определяет псевдонимы для соединения с базой данных, которые находятся в файле **XSQLConfig.xml**. Этот файл должен храниться в безопасном месте, и доступ к нему осуществляется только с сервера. Кроме того, этот элемент определяет пространство имен "xsql", чтобы предотвратить конфликты имен в XSQL-файле, поэтому все ключевые слова будут иметь такой же префикс, как в элементе "xsql:query":

```
<xsql:query rowset-element="" row-element="INFO"><![CDATA[
   select count(*) as "TOTAL", '{@cat}' as cat from site_entry
   where categories like '%'||UPPER('{@cat}')||'%' and permanent is null
]]>
</xsql:query>
```

В этом запросе подсчитывается общее число элементов в листинге. В дальнейшем он будет использоваться для возвращения групп из шести элементов.

```
<xsql:query rowset-element="SiteStructure" row-element="Category"><![CDATA[
    SELECT name "Name", icon "Icon" FROM site_category ORDER BY id
]]>
</xsql:query>
```

В этом запросе создается верхняя навигационная панель, в которой находятся и текст и пиктограммы из базы данных. Если эту Web-страницу будет запрашивать устройство с ограниченными графическими возможностями либо вообще без таких возможностей, применяемая в этом случае таблица стилей просто проигнорирует все графические пиктограммы в этом шаблоне и построит навигационную панель без них.

```
<xsql:query
   category="NEWS" rowset-element="MAINITEMS"
   row-element="ITEM" max-rows="6" skip="0"
   skip-rows="{@skip}" >
<![CDATA[
   select title, id, description,url,to_char(timestamp,'Mon DD') timestamp
   from site_entry where categories like '%'||UPPER('{@cat}')||'%'
   and permanent is null order by id desc
]]>
</xsql:query>
```

Третий запрос несколько сложнее, так как в нем устанавливаются переменные значения, которые в дальнейшем будут увеличиваться, чтобы возвращать последовательные наборы элементов. Отметим, что возвращение элементов происходит по категориям, соответствующим категориям, указанным в навигационной панели, и на экран они выводятся вместе с соответствующими указателями времени создания.

```
<xsql:query
   rowset-element="SIDTEBARITEMS" row-element="ITEM"
   max-rows="5" category="NEWS">
<![CDATA[
   select title,id,description,url,to_char(timestamp,'DD-MON-YYYY') timestamp
   from site_entry where categories like '%'||UPPER('{@cat}')||'%'
   and permanent > 0 order by id desc
]]>
</xsql:query>
</datapage>
```

Этот четвертый по счету и последний запрос создает полосы прокрутки для быстрой навигации по разделам. Это достигается за счет передачи в запрос заранее заданных переменных, чтобы извлечь первые пять элементов из категории news.

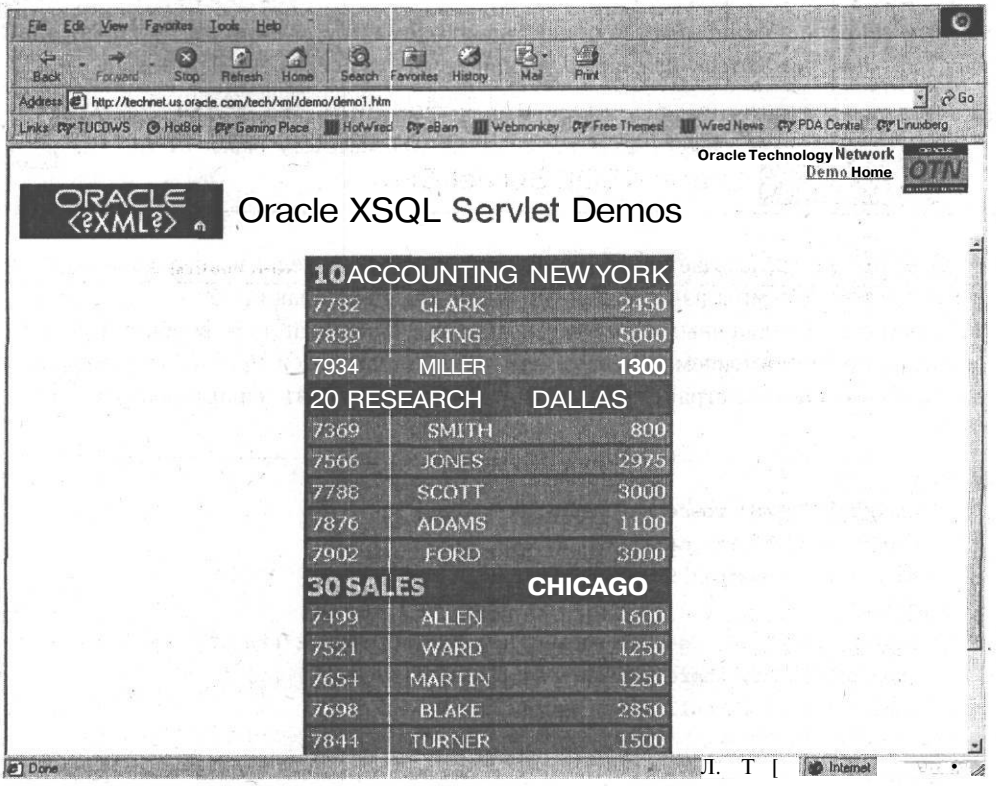


Рис. 10.13. *Employee/Department Object View*

Приложение Employee/Department Object View

В этом приложении скомбинированы элементы из приложения Insurance Claims, в котором результирующие наборы данных, содержащие типы пользователя, преобразовывались в XML-формат, и приложения Employee Page, в котором результаты форматировались с помощью XSL-таблиц стилей. Выходной экран приложения показан на рис. 10.13.

Можно выбрать пиктограмму ссылки на источник этого приложения и просмотреть три исходных файла, использованных для создания приложения. В первом фрейме, как показано на рис. 10.14, находится используемая этим приложением XSQL-страница. Во втором фрейме на этом же рисунке показана XSL-таблица стилей, которая применяется для преобразования выходного XML-файла. При этом преобразовании происходит форматирование результирующего набора данных в ячейки таблицы на HTML-странице. В третьем фрейме находится SQL-запрос и данные, использованные для создания таблиц базы данных, обслуживающей это приложение.

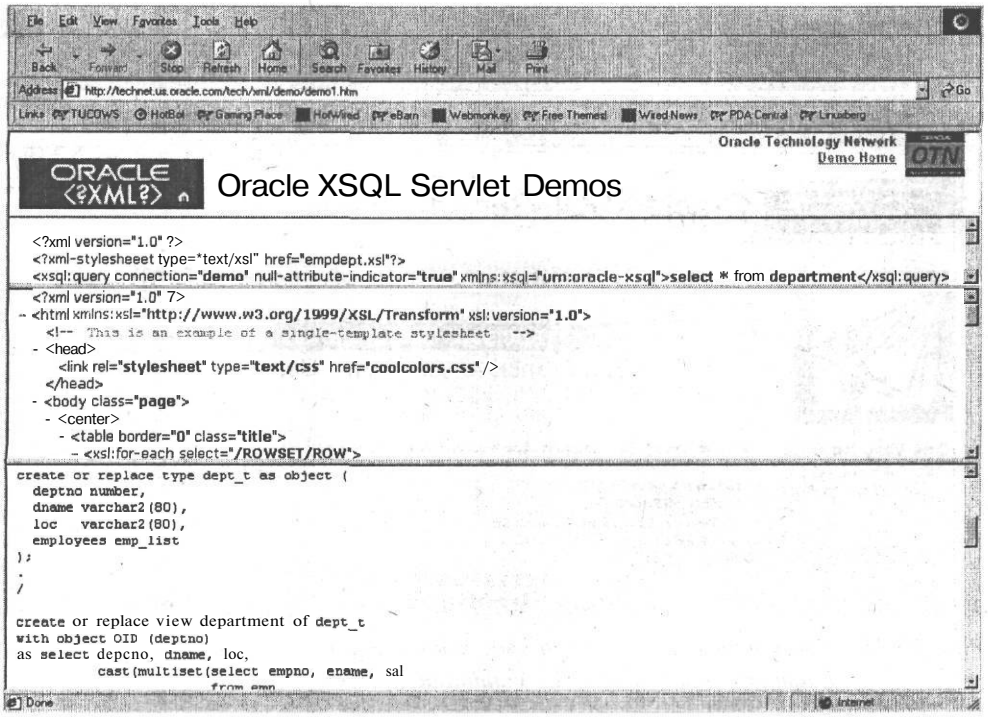


Рис. 10.14. *Исходные файлы для приложения Employee/DepartmentObject View*

Приложение Airport Code Validation

Приложение Airport Code Validation позволяет определить название города, введя трехбуквенный код аэропорта, и наоборот, с его помощью можно определить код аэропорта, введя название города. Для ввода текста в этом приложении используется форма с одним полем.

После набора текста в поле ввода и нажатия TAB приложение разыскивает в таблице AIRPORT трехбуквенный код аэропорта, соответствующий введенным символам. Если найден аэропорт с точно таким же кодом, приложение выводит на экран код аэропорта и ближайшее к нему название города. Если такой код не найден, приложение предполагает, что введенные символы — это название города, и ищет в таблице AIRPORT все записи с названиями городов, которые содержат введенные символы. Затем в поле со списком выводятся десять первых подходящих записей, в которых потом можно выбрать нужный город и код аэропорта. На рис. 10.15 показан вид экрана после ввода кода аэропорта SFO.

На стороне клиента это приложение использует JavaScript, а для взаимодействия с сервером — острова данных (data islands). Для получения ответа после введения кода аэропорта приложение использует в качестве дейтаграмм XML-файлы. Если поиск кода был успешным, возвращается дейтаграмма OK, XML-файл для которой

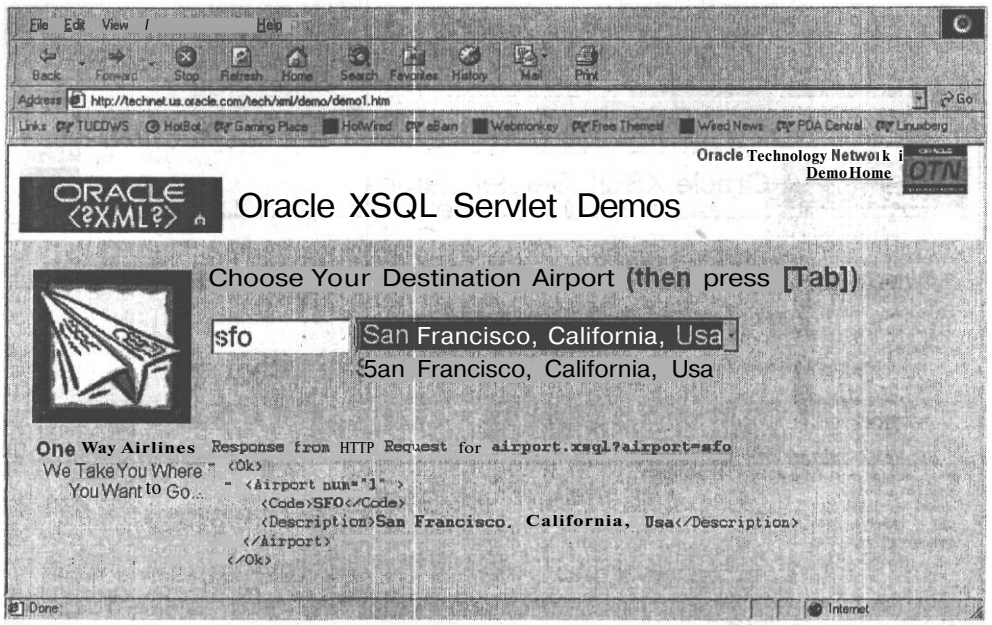


Рис. 10.15. *Поиск названия города по коду аэропорта в приложении Airport Code Validation*

показан на рис. 10.15. Если поиск кода аэропорта закончился неудачей, возвращается XML-дейтаграмма с указателем ошибки Error. Дейтаграмма Error содержит десять совпадений, которые являются ответом на второй запрос. Этот второй запрос запускается на исполнение, только если в ответ на первый запрос придет пустой набор данных. Второй запрос пытается найти записи, в которых название имени может содержать введенные символы. На XSQL-странице этого приложения определены два следующих запроса:

```
<?xml version="1.0"?>
<xsql:query xmlns:xsql="urn:oracle-xsql" connection="demo" airport="SFO"
         rowset-element="OK" max-rows="1" row-element="Airport">
         SELECT tla "Code", description "Description" FROM AIRPORT
         WHERE tla = UPPER ('{@airport}')
<xsql:no-rows-query
         max-rows="10" rowset-element="Error" row-element="Airport">
         SELECT tla "Code", description "Description" FROM AIRPORT ,
         WHERE UPPER(description) LIKE UPPER('%{@airport}%') ORDER BY tla
</xsql:no-rows-query>
</xsql:query>
```

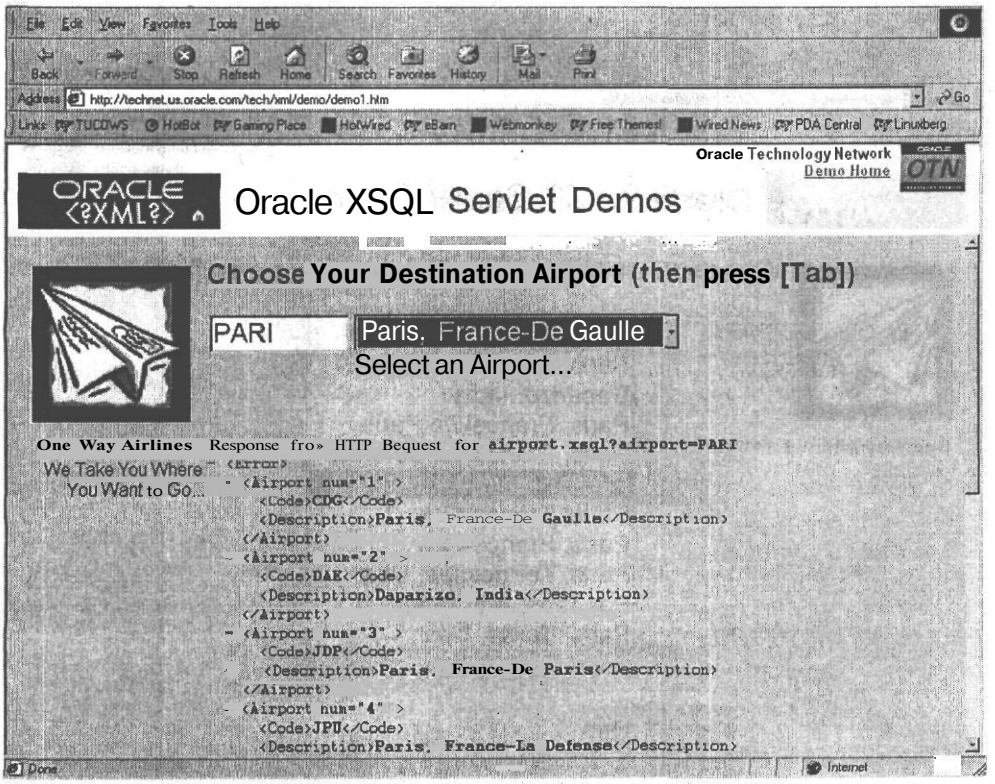


Рис. 10.16. *Запрошенной строке символов соответствует более одного города*

На рис. 10.16 показано, что происходит, когда ведется поиск кода аэропорта города Париж и для этого вводятся символы **PARI.** На экране — возвращенная дейтаграмма Error в XML-формате. Этот файл содержит первые десять записей, полученных по второму запросу, который определен в элементе, начинающемся с тега **<xsql:no-rows-query>** (см. выше).

На рис. 10.17 показано, что данные из дейтаграммы Error были загружены в поле со списком. Затем этот список можно использовать для сужения выбора. Все это произошло благодаря тому, что для сбора информации из входного XML-файла использовался JavaScript.

Рассмотрим несколько фрагментов HTML-файла и JavaScript-код с демонстрационной страницы Airport. HTML-файл создает форму **airport** со следующими XML-элементами: поле ввода текста с именем **code** (оно идентифицируется в JavaScript как **airport.code**), элемент выбора под названием **lookup** (идентифицируется как **airport.lookup**) и содержательный элемент, который загружается, чтобы показать на экране описание аэропорта. Содержательный элемент называется **description,** а его идентификатор в JavaScript — **airport.description.**

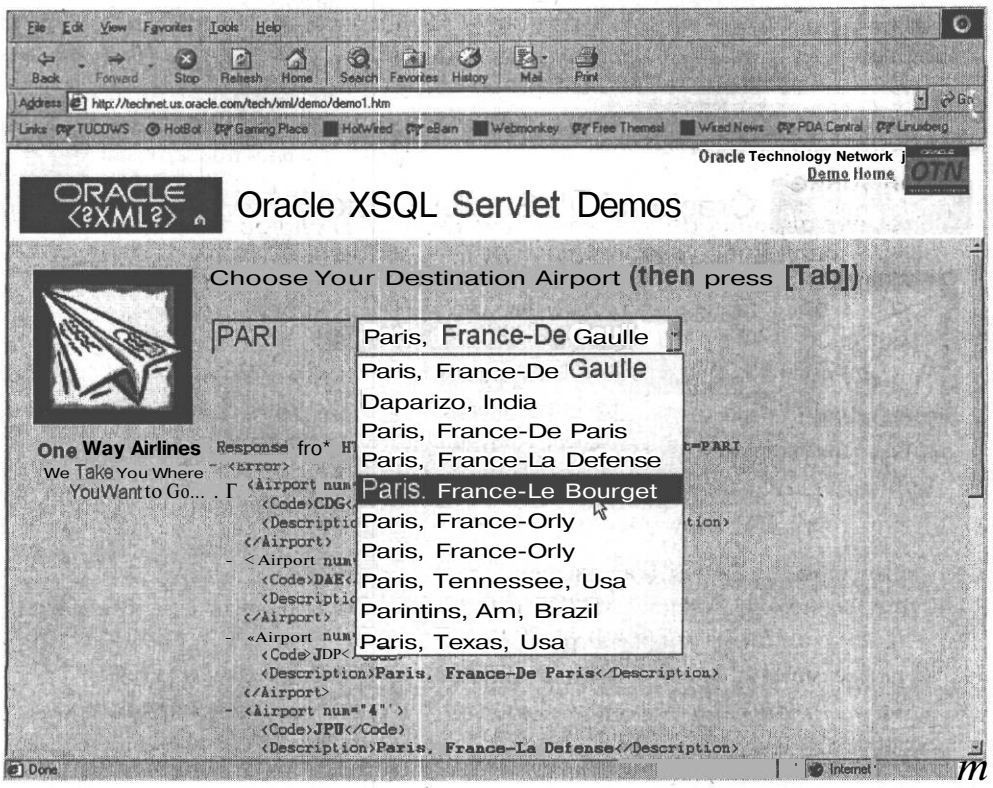
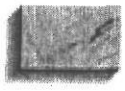


Рис. 10.17. *Использование JavaScript в клиентском браузере для заполнения поля со списком с предположениями о возможных названиях городов в демонстрационной программе Airport*

```
<form id="airport" method="GET" action="">
<div align="left">
<table border="0" height="89">
<tr>
   <td valign="top">
   <input style="font-family:Arial; font-size: 18pt" id="code" name="code"
          type="text" size="8" onchange="validateAirport()"><br>
</td>
<td valign="top">
   <select style="font-family:Arial; font-size: 18pt"
      onchange="setAirportCode(this)" name="lookup" size="1">
   </selectxbr>
   <div class="st" style="font-family:Arial; font-size:18pt" id="description">
   </div>
</td>
```

```
</tr>
</table>
</div>
</form>
```



Внимание

Элемент ***airport.code*** *устанавливает атрибут* ***oncharge*** *в виде* ***oncharge="validateAirport()"****. Это означает, что функция* ***validateAirportO*** *будет исполняться всякий раз, когда в поле кода аэропорта вводится новый набор символов.*

Следующий HTML-код создает XML-фрагмент или остров данных (data island), который можно использовать в качестве шаблона для загрузки. В этом XML-фрагменте содержится результат запроса, полученный с сервера.

```
<xml id="airportCodes">
</xml>
```

Этот пустой XML-фрагмент пока является только заглушкой. Но следует помнить, что во встроенном в страницу JavaScript-коде можно установить ссылку на этот фрагмент по указанному имени **airportCodes**.

Теперь рассмотрим работу функции **validateAirport()**. Как уже говорилось ранее, эта функция запускается браузером IE5 всякий раз, когда пользователь вводит в окно код аэропорта и нажимает клавишу TAB. Вот код этой функции:

```
<script>
function validateAirport() {
   event.returnValue = true;
   removelistoptions();
   if ( airport.code.value != "" ) {
      airportCodes.async = false;
      url = "airport.xsql?airport=" + airport.code.value;
      airportCodes.load( url );
      var DE = airportCodes.documentElement;
      result.innerHTML = DE.transformNode(ie5default.documentElement);
      urlexample.innerHTML = " Response from HTTP Request for <b
      style='color:blue'><pre>"+url+"</prex/b>";
      if ( DE.nodeName == "Ok" ) {
         description.innerText = DE.selectSingleNode( "//Description").text;
         var newOption = new Option;
         newOption.text = DE.selectSingleNode("//Description") .text;
         newOption.value = DE.selectSingleNode("//Code") .text;
         airport.lookup.options.add( newOption, 0 );
         airport.lookup.options[0] .selected= true;
```

```
        } else {
           var matches = DE.selectNodes ("Airport");
           if (matches.length== 0) {
              description.innerText = "Invalid Code, No Suggestions...";
           } else {
              if (matches.length == 1) {
                 airport.code.value =
                 matches.item(0).selectSingleNode("Code");
                 description.innerText =
                 matches.item(0).selectSingleNode("Description");
                 var newOption = new Option;
                 newOption.text =
                    matches.item(0).selectSingleNode("Description").text;
                 newOption.value =
                    matches.item(0).selectSingleNode("Code").text;
                 airport.lookup.options.add(newOption,0);
                 airport.lookup.options[0].selected= true;
                 setAirportCode(airport.lookup.options);
              } else {
                 for ( ctr = 0; ctr < matches.length ; ctr++ ) {
                    var newOption = new Option;
                    newOption.text =
                       matches.item(ctr).selectSingleNode("Description").text;
                    newOption.value=
                       matches.item(ctr).selectSingleNode("Code").text;
                    airport.lookup.options.add( newOption,ctr );
                    airport.lookup.options[0].selected= true;
                    description.innerText = "Select an Airport...";
                 }
              }
           }
        }
     }
  }
  function setAirportCode(s) {
     airport.code.value = s.options[s.selectedIndex].value;
     description.innerText = s.options[s.selectedIndex].text;
  }
  </script>
```

Разберем следующие две строки этого кода:

```
url = "airport.xsql?airport=" + airport.code.value;
airportCodes.load(url);
```

Первая строчка генерирует URL-адрес для XSQL-страницы, используемой приложением. Вторая строка выполняет все остальные магические действия, а именно:

- Отправляет на сервер запрос об исполнении XSQL-страницы, определяемой указанным URL-адресом
- Обрабатывает эту XSQL-страницу сервлетом XSQL Servlet и возвращает дейтаграмму в виде XML-файла
- Присваивает эти XML-данные XML-объекту с именем **airportCodes** (вспомните ранее данное определение острова данных).

JavaScript-код, который идет после **airportCodes.load(url)**, теперь может получить доступ к возвращенной дейтаграмме. Вот как приложение интерпретирует то, что произошло при выполнении этого запроса:

```
var DE = airportCodes.documentElement;
часть кода пропущена...
      if ( DE.nodeName == "Ok") {
теперь мы знаем, что пришла дейтаграмма OK
      } else {
теперь мы знаем, что пришла дейтаграмма Error,
и нужно выбрать название аэропорта...
      var matches = DE.selectNodes("Airport");
далее код опять пропущен
   }
```

Следующие строчки проясняют, как поле со списком поиска заполняется данными из XML-элементов полученной дейтаграммы:

```
newOption.text=matches.item(ctr).selectSingleNode("Description").text;
newOption.value = matches.item(ctr).selectSingleNode("Code").text;
         airport.lookup.options.add(newOption,ctr);
```

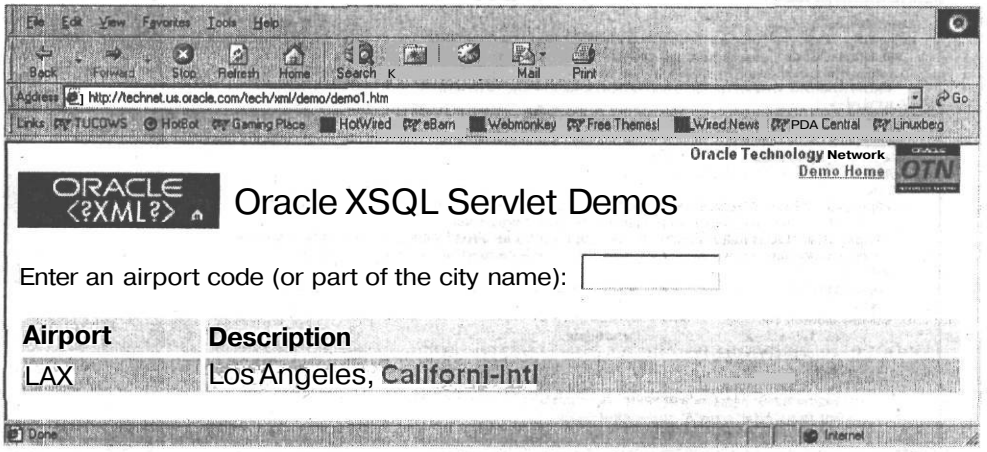


Рис. 10.18. *Приложение Airport Code Display — введены символы LAX*

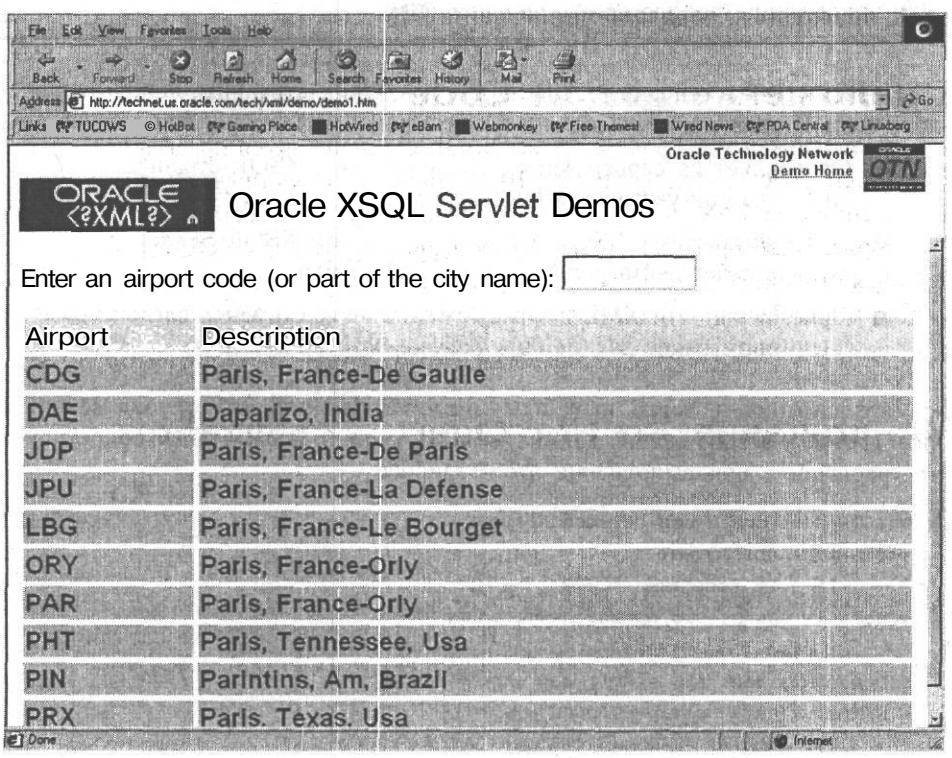


Рис. 10.19. *Приложение Airport Code Display — введены символы PARI*

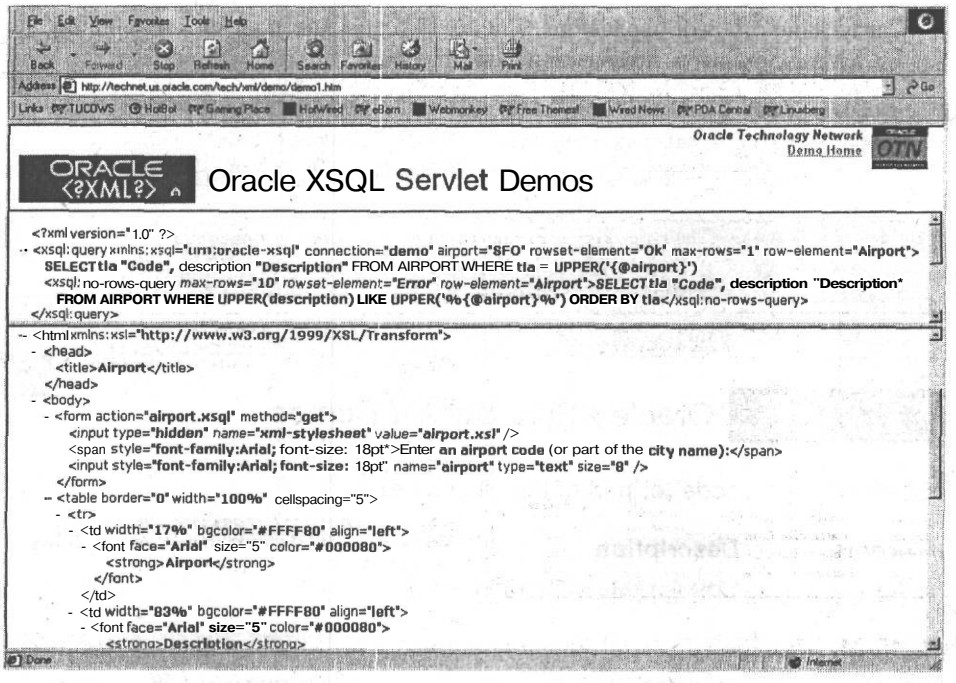


Рис. 10.20. *ПриложениеAirport Code Display — исходные страницы*

Приложение Airport Code Display

Это приложение использует ту же XSQL-страницу, что и предыдущее приложение Airport Code Validation. На рис. 10.18 показана результирующая страница после ввода символов LAX, а на рис. 10.19 — после ввода символов **PARI.**

На рис. 10.20 представлен экран с исходным кодом для XSQL-страницы и XSL-таблицы стилей. В отличие от предыдущего приложения содержимое экрана не зависит от встроенной в браузер IE5 поддержки технологии XML, следовательно, это приложение будет работать с любым браузером.

Приложение Ad Hoc Query Visualization

В этом приложении на одном экране собраны почти все темы, затронутые в предыдущих приложениях. На рис. 10.21 показано, как это визуальное приложение позволяет пользователю выполнять следующие действия:

- Вводить запрос.
- Выбирать таблицу стилей, которую нужно применить к результирующим XML-данным.

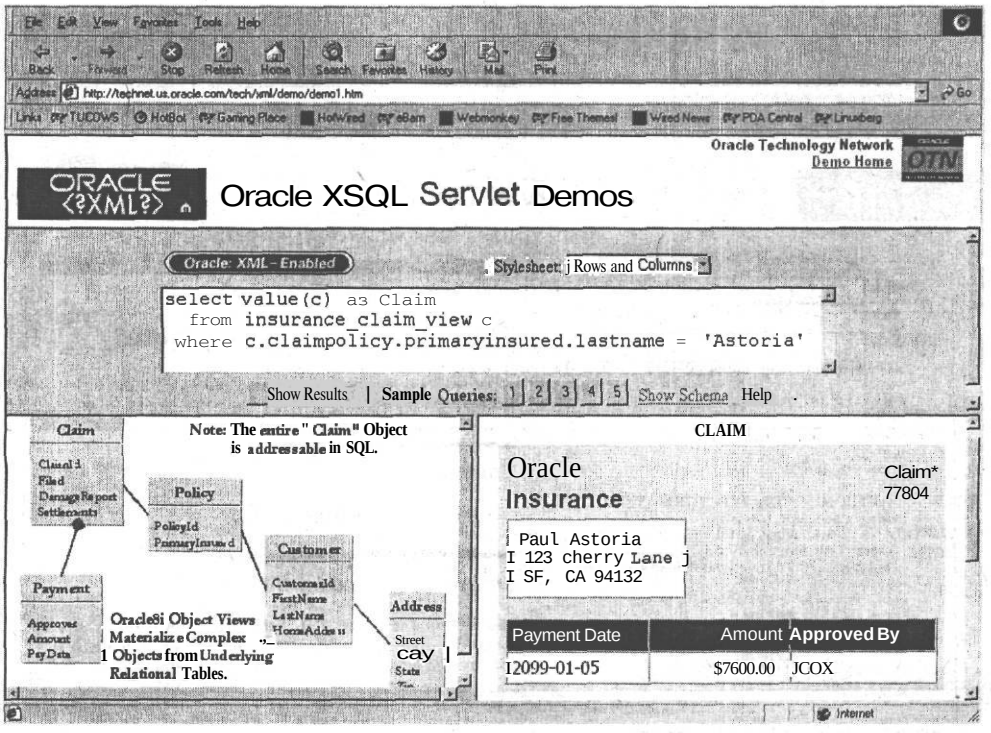


Рис. 10.21. *Приложение Ad Hoc Query Visualization*

Если щелкнуть мышью по кнопке Show Results, SQL-программа обратится к базе данных Oracle9i, и в левом нижнем фрейме экрана появится результат запроса. Если нужна схема, используемая этим приложением, следует обратиться к SQL-листингу приложения Insurance Claim, которое обсуждалось выше в разделе "Приложение Insurance Claim".

Приложение XML Document Demo

Это довольно сложный пример того, как XML-документы сохраняются на сервере. На экран выводится список всех ранее сохраненных документов. Если выбрать какой-то документ, приложение выведет его на экран, используя для преобразования результата таблицу стилей (но этот параметр вводить не обязательно). Экран приложения показан на рис. 10.22.

Список сохраненных документов показан внизу слева, а в правом фрейме выводится содержимое выбранного документа. Если выбрать ссылку на таблицу стилей из перечисленных в столбце FORMATTED, в правом окне появится документ в HTML-формате, полученный с применением заданной XSL-таблицы стилей. Если выбрать ссылку DOCID, на экран будет выведен XML-документ без всякого преобразования на сервере.

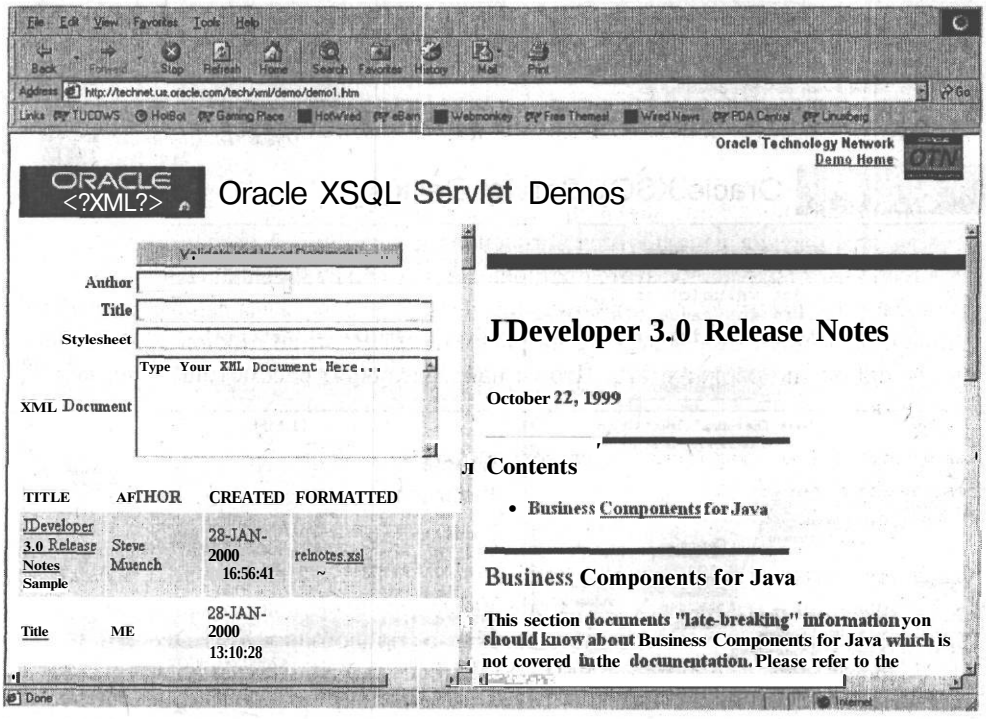


Рис. 10.22. *Приложение XML Document Demo*

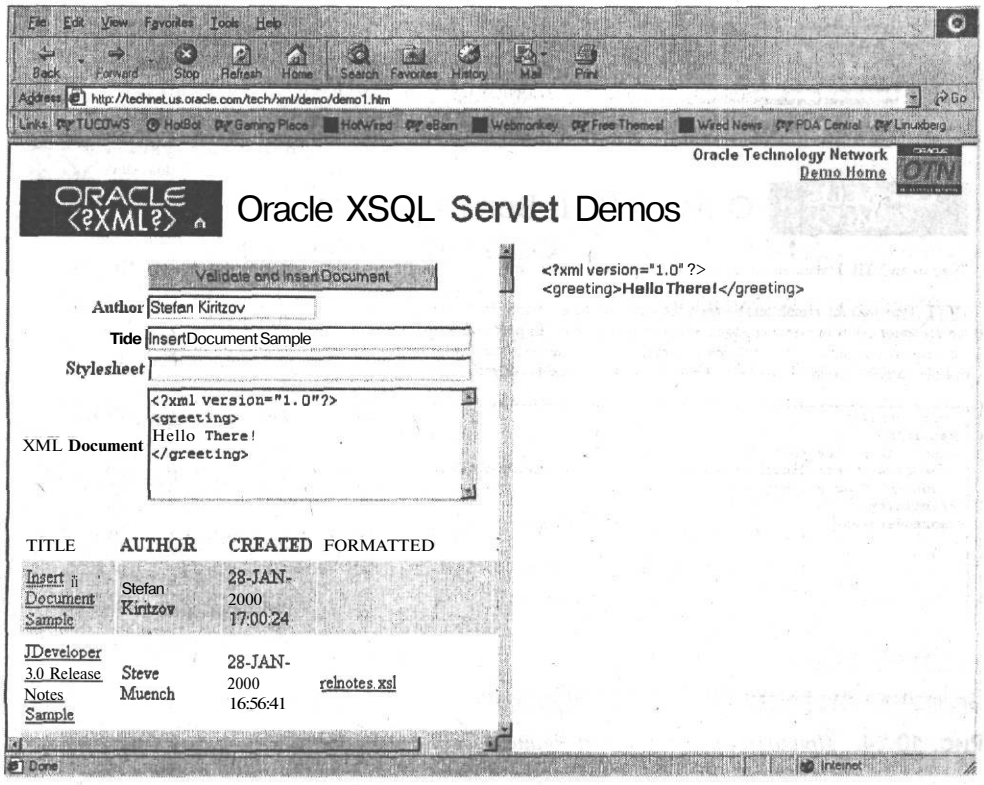


Рис. 10.23. *Добавление нового документа в приложение XML Document Demo*

На рис. 10.23 показано, как в приложение добавляется новый документ.

В этом приложении используется встроенная в браузер IE5 поддержка технологии XML. В нем также используется встроенная поддержка языка JavaScript для переноса основной нагрузки по обработке вводимых пользователем данных с клиента на сервер. Это приложение похоже на рассмотренное выше приложение Airport Code Validation, но сложнее его. Чтобы найти отличия, рассмотрим следующий JavaScript-код.

```
var test = new ActiveXObject("Microsoft.XMLDOM");
test.async = false;
test.loadXML(val);
```

Здесь для загрузки XML используется объект ActiveX, а вместо использования XML-островов данных (как в Airport Code Validation) выполняется синтаксический разбор. Это приложение показывает, как JavaScript, объект XMLDOM ActiveX и HTML-формы объединяются с XML, SQL, Oracle XSQL-страницами и базой данных Oracle8i/9i.

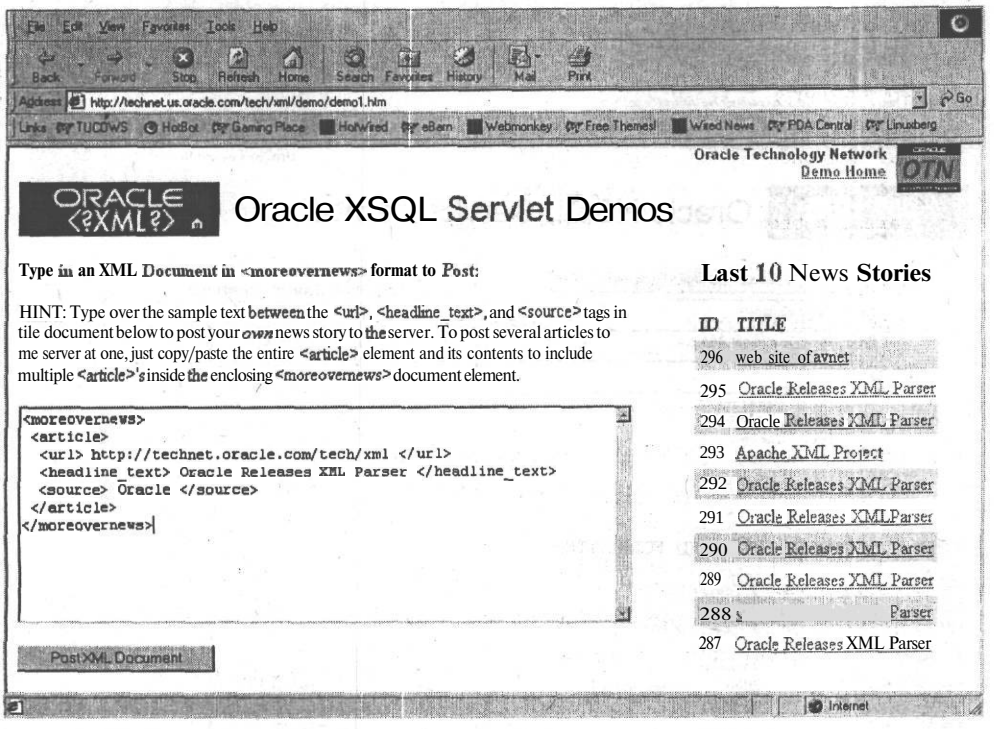


РИС. 10.24. *Приложение XML Insert Request Demo*

Приложение XML Insert Request Demo

С помощью приложения XML Insert Request Demo можно публиковать на сайте текстовые материалы. Каждая такая статья состоит из XML-файла с заголовком и URL-адресом страницы, где опубликован ее полный текст. Это приложение показано на рис. 10.24.

Оно демонстрирует, как отправлять XML-файлы с браузера для сохранения на сервере. При этом используется одна важная функция XSQL Servlet, позволяющая применять технологию XML для обновления таблиц базы данных. Вот HTML-страница приложения с JavaScript-кодом, выполняющим эту процедуру:

```
<HTML>
<BODY>
<SCRIPT>
   function PostOrder (xmldoc)
   { var xmlhttp = new ActiveXObject ("Microsoft.XMLHTTP");
     xmlhttp.open("POST", "insertnewsstory.xsql",false);
     xmlhttp.send(xmldoc);
     return xmlhttp.responseXML;
   }
```

```
function submitInfo(){
   var xmldoc = new ActiveXObject ("Microsoft.XMLDOM");
   xmldoc.async = false;
   xmldoc.loadXML(xmldocText.value);
   var response = PostOrder(xmldoc);
   alert(response.documentElement.xml);
   parent.bottom.location = "newsstorylist.xsql";
}
</SCRIPT>
<b>Type in an XML Document to Post:<b><br>
<TEXTAREA rows="12" style="width:100%" cols="70" NAME="xmldocText"
><moreovernews>
   <article>
      <url> http://otn.oracle.com/tech/xml </url>
      <headline_text> Oracle Releases XML Parser </headline_text>
      <source> Oracle </source>
   </article>
</moreovernews></TEXTAREA>
<P><INPUT TYPE=button Value="Post XML Document"
         onclick="submitInfo()">
</BODY>
</HTML>
```

А вот XSQL-страница, которая запрашивает операцию вставки:

```
<?xml version='1.0'?>
<xsql:insert-request connection="demo" xmlns:xsql="urn:oracle-xsql"
   table="newsstory" transform="moreover-to-newsstory.xsl"/>
```

Наконец, на рис. 10.25 показано, что произойдет, если выбрать первую статью из списка, изображенного справа на экране на рис. 10.24.

Установка и запуск XML-приложений

Приложения, рассмотренные в этой главе, являются частью Oracle XSQL Servlet, который, в свою очередь, входит в состав инструментария разработчика XDK for Java. Теперь надо загрузить названный инструментарий с домашней страницы Oracle Technology Network XML, щелкнув мышью по ссылке XDK for Java на экране, показанном в начале этой главы на рис. 10.1, и далее следовать инструкциям, подробно изложенным в настоящем разделе. Домашняя страница XDK for Java показана на рис. 10.26.

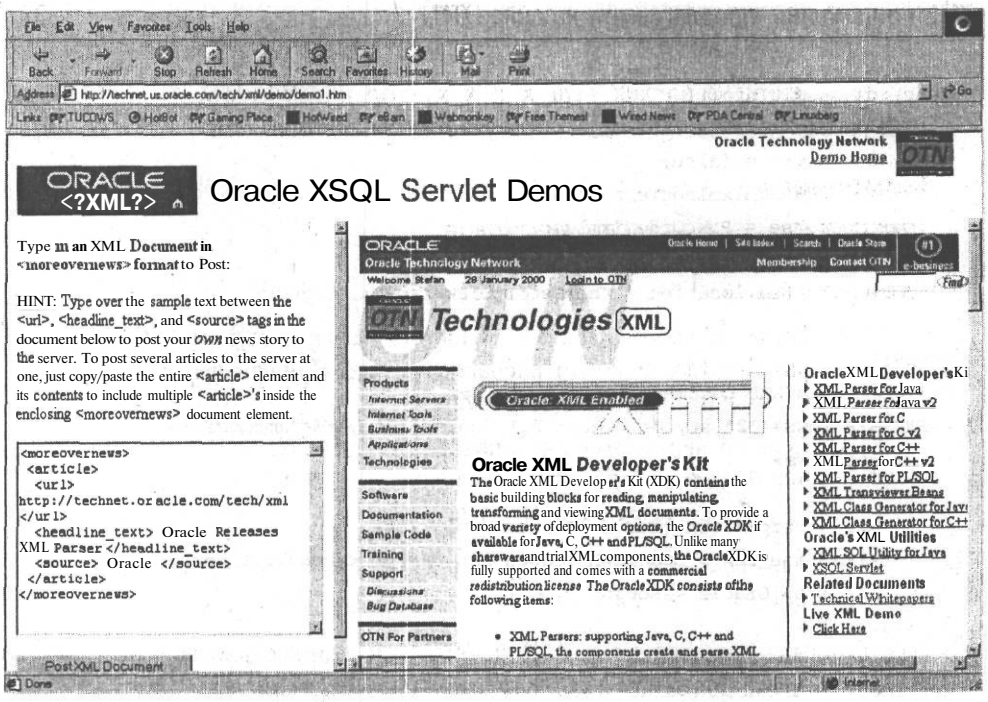


Рис. 10.25. *Приложение XML Insert Request — Oracle делает XML-файл!*

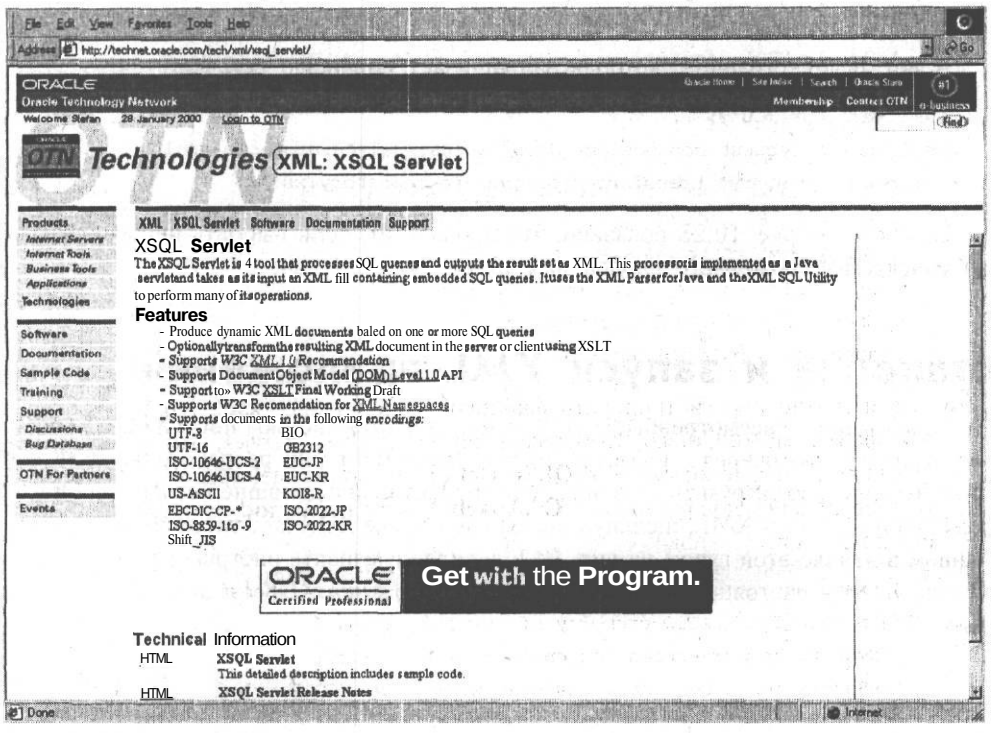


Рис. 10.26. *Домашняя страница XSQL Servlet*

Загрузите с сайта файл **xdk_java_x_x_x_x_x.zip** для **Windows** или файл для UNIX **xdk_java_x_x_x_x_x.tar.gz**, действуя следующим образом:

Внимание

Содержимое этих файлов одинаковое.

1. Выберите каталог, в который будете копировать файл.
   - Если вы работаете с ОС Windows, можете выбрать каталог C:\, а потом распаковать загруженный файл с помощью архиватора WinZip.
   - Если на компьютере установлена ОС UNIX, используйте следующую команду:
     tar `xvfz xdk_java_9_0_1_0_0.tar.gz`
2. Извлеченные из архива файлы будут распределены по подкаталогам **/bin, /lib, /xdk.**
3. Все файлы с расширениями JAR и ZIP, необходимые для запуска XSQL Servlet, находятся в подкаталоге /lib. Убедитесь, что эти файлы есть в CLASSPATH для движка Java Servlet Web-сервера. Например, при использовании Apache JServ добавьте в файл **C:\Apache\Jserv\conf\jserv.properties** следующие строки:

```
# Oracle XSQL Servlet
wrapper.classpath=C:\xdk\lib\oraclexsql.jar
# Oracle JDBC (8.1.5)
wrapper.classpath=C:\xdk\lib\classes12.zip
# Oracle XML Parser V2 (with XSQL Engine)
wrapper.classpath=C:\xdk\lib\xmlparserv2.jar
# Oracle XML SQL Components for Java
wrapper.classpath=C:\xdk\lib\xsu12.jar
# XSQLConfig.xml File location
wrapper.classpath=C:\xdk\lib\
```

4. Затем нужно зарегистрировать файловое расширение **.xsql**, чтобы эти файлы можно было отображать на класс Java Servlet, который называется **oracle.xml.xsql.XSQLServlet**. При установленном Web-сервере Apache в файл конфигурации **C:\Apache\Jserv\conf\mod_jserv.conf** нужно добавить строки:

```
# Executes a servlet passing filename with proper extension
# in PATH_TRANSLATED property of servlet request.
# Syntax: ApJServAction [extension] [servlet-uri]
ApJServAction .xsql /servlets/oracle.xml.xsql.XSQLServlet
```

5. Для удобства запуска демонстрационных программ скопируйте эти файлы в подкаталог **demo.** При работе с ОС Windows и Web-сервером Apache используйте следующую команду:

```
xcopy /s c:\xdk\xdk\demo\java\xsql\demo C:\Apache\htdocs\xsql\demo
```

6. Демонстрационные программы настроены так, чтобы использовать схему SCOTT в базе данных на локальной машине (точнее, на машине, где работает Web-сервер). Запустив на исполнение локальную БД и войдя в нее с именем пользователя SCOTT и паролем TIGER, вы получите все нужные установки. В противном случае придется редактировать файл **.\xdk\lib\XSQLConfig.xml**, чтобы он соответствовал значениям для имени пользователя, пароля, dburl и драйвера для соединения "demo" с базой данных, как показано ниже:

```
<?xml version="1.0"?>
<XSQLConfig>
        :
   <connectiondefs dumpallowed="no">
      <connection name="demo">
         <username>scott</username>
         <password>tiger</password>
         <dburl>jdbc:oracle:thin:@localhost:1521:ORCL</dburl>
         <driver>oracle.jdbc.driver.OracleDriver</driver>
      </connection>
      <connection name="lite">
         <username>system</username>
         <password>manager</password>
         <dburl>jdbc:Polite:POlite</dburl>
         <driver>oracle.lite.poljdbc.POLJDBCDriver</driver>
      </connection>
   </connectiondefs>
        :
</XSQLConfig>
```

7. Чтобы установить таблицы базы данных, используемые в примерах, обсуждавшихся в этой главе, запустите в подкаталоге **demo** все SQL-скрипты (они имеют расширение **.sql**). Например, для установки таблиц, используемых в приложении Airport Demo, введите команду:

```
sglplus scott/tiger @airport.sql
```

8. Перезапустите Web-сервер. После этого можно запускать браузер Internet Explorer 5 и отправляться по адресу **http://yourserver/xsql/demo/helloworld.xsql**. Если все было установлено правильно, появится экран, показанный выше на рис. 10.2, и будут работать все остальные приложения.

Для запуска демонстрационных приложений можно также использовать *Oracle Web-to-Go Web-сервер*. Oracle Web-to-Go — это однопользовательский Web-сервер, предназначенный для мобильных приложений. Так как он полностью поддерживает сервлеты, можно запустить с помощью этого Web-сервера XSQL Servlet. Если используется пакет JDeveloper 3.x, то Web-to-Go у вас уже есть. Предположим, что инструментарий JDeveloper 3.x находится в каталоге **d:\jdev**, тогда приведенный ниже скрипт будет запускать Web-сервер Web-to-Go, используя установки конфигурации, определенные в файле **d:\jdev\lib\webtogo.ora**. Если JDeveloper находится в другом каталоге, придется соответствующим образом подредактировать этот скрипт.

```
@@echo off
setlocal
set JDEV_HOME=d:\jdev3
set PATH=%JDEV_HOME%\bin;%PATH%
set CP=%JDEV_HOME%\java\lib\classes12.zip
set CP=%CP%;%JDEV_HOME%\lib\webtogo.jar
set CP=%CP%;%JDEV_HOME%\lib\xmlparserv2.jar
set CP=%CP%;%JDEV_HOME%\lib\classgen.jar
set CP=%CP%;%JDEV_HOME%\lib\ojsp.jar
set CP=%CP%;%JDEV_HOME%\jswdk-1.0\lib\servlet.jar
set CP=%CP%;C:\xsql\lib
set CP=%CP%;C:\xsql\lib\classes111.zip
set CP=%CP%;C:\xsql\lib\oraclexsql.jar
set CP=%CP%;C:\xsql\lib\oraclexmlsql.jar
set CP=%CP%;C:\xsql\lib\xmlparserv2.jar
jre -classpath %CP% oracle.lite.web.JupServer
endlocal
```

Сохраните этот скрипт в файле **wtg.bat** и используйте для запуска сервера Web-to-Go команду **wtg**. Но перед запуском Web-to-Go следует отредактировать файл **d:\jdev3\lib\webtogo.ora**.

В разделе [FILESYSTEM] добавьте строку:

```
ROOT_DIR=C:\
```

Данная строка устанавливает физический каталог, который будет работать как виртуальный корень документа Web-сервера.

В разделе [MIMES] добавьте строку:

```
xsql=text/html;handler=oracle.xml.xsql.XSQLServlet
```

Эта строка регистрирует файл класса XSQL Servlet в качестве обработчика файлов с расширением ***.xsql.**

В разделе [MIMES] добавьте также следующие строчки:

```
xml=text/xml
xsl=text/xml
```

Эти строки устанавливают тип MIME по умолчанию для XML- и XSL-файлов.

Весь исходный код *отредактированного* файла **webtogo.ora** будет выглядеть так:

```
[WEBTOGO]
PORT = 7070
USE_SYSTEM_CLASSPATH=YES
DEBUG=YES
[FILESYSTEM]
ROOT_DIR=C:\
TYPE=OS
[MIMES]
html=text/html
xml=text/xml
xsl=text/xml

jsp=text/html;handler=oracle.jsp.JspServlet
xsql=text/html;handler=oracle.xml.xsql.XSQLServlet

[APPLICATIONS]
xmlfile=.\wtgapp.xml

[SERVLET_PARAMETERS ]
```

После изменения файла **webtogo.ora** можно запускать Web-сервер Web-to-Go, просто запуская на исполнение файл **wtg.bat.** Затем запустите браузер Internet Explorer 5 и отправляйтесь по адресу **http://yourserver/xsql/demo/helloworld.xsql**. Если Web-to-Go установлен правильно, появится экран приложения, показанный в этой главе выше на рис. 10.2. Итак, теперь все описанные демонстрационные приложения будут работать на вашем компьютере с использованием Web-сервера Web-to-Go и базы данных Oracle8i или 9i.

Глава 11

Тенденции развития

о данным исследовательской компании Forrester Research, оборот электронной коммерции при торговле между компаниями (business-to-business, B2B) только в США к 2003 г. превысит 1 триллион долларов, т. е. будет в десять раз больше, чем для розничной электронной торговли (business-to-consumer, B2C). Как такое возможно? Предположим, что компании и целые отрасли индустрии перевели свои операции по продаже и покупке на электронные биржи в Интернете, так что производимые поставщиками комплектующие всегда можно найти и купить по приемлемой цене в Web на специализированном онлайновом форуме типа Интернет-аукциона eBay. Информация о том, какие поставщики предлагают свой товар и когда они могут поставить конкретные комплектующие, всегда будет доступна в онлайне. На такой бирже поставщики смогут продавать избыточные складские запасы, оформлять сделки на отложенные продажи своих товаров и принимать заказы на их производство. В работе таких бирж уже участвуют автомобилестроительные компании Ford Motor Company, GM и Daimler-Crysler.

Кому еще будет выгодна организация крупных электронных бирж в Интернете, кроме компаний, занимающихся поставками и закупками товаров и комплектующих? Очевидно, компаниям, занимающимся базами данных, так как для работы этих бирж нужно будет обрабатывать огромные объемы данных и массу транзакций, совершаемых через Интернет. А для этого всем игрокам на бирже понадобится соответствующее программное обеспечение. Поэтому заявление Ларри Эллисона "Интернет меняет все" целиком и полностью верно и для компании Oracle. Создание Интернет-бирж выгодно и компаниям-разработчикам стандартизированных приложений для конкретных областей промышленности, а также консалтинговым фирмам, которые будут участвовать в создании таких электронных бирж. Все это заинтересует и производителей накопительных систем, компьютерного и сетевого оборудования. Наконец, эти тенденции будут на руку и компаниям, специализирующимся на технологии XML, так как электронные биржи в качестве стандарта для обмена данными, скорее всего, выберут именно эту технологию. Эти компании произведут интегрирование XML-технологии в конкретные приложения и предложат соответствующие услуги, средства и продукты клиентам, которые будут работать на электронных биржах.

Все эти цели могут оказаться не очень перспективными с точки зрения быстрых прибылей, но технология в своем развитии вынуждена пройти через этап, когда она не воспринимается всерьез. В конце концов приложения этой технологии нужны не только в сфере бизнеса, по такому же принципу можно построить и биржи для

обмена информацией между, например, специалистами по охране окружающей среды, обеспокоенными распространением опасных для здоровья материалов, или между школьниками в классах.

Фундаментальные концепции технологии XML по большей части абсолютно понятны, способы приложения ее идей могут быть просто потрясающими, и от дальнейших перспектив просто захватывает дух. Для тех, кого заинтересует эта технология, открываются бесконечные возможности и блестящие перспективы. Другой вопрос, станет ли технология XML еще одним стандартным обобщенным языком разметки (Standard Generalized Markup Language, SGML). Но пока есть прогресс в создании новаторских приложений и в разработке открытых стандартов, застоя в XML-индустрии и развитии этой технологии не будет. Поэтому XML-технология может плодотворно развиваться еще много лет.

В этой главе рассказывается об организациях, занимающихся разработкой и принятием стандартов, и о той работе, которую они выполняют в области XML-технологии. Затем рассмотрим применение XML в разных отраслях индустрии и влияние этой технологии на Интернет.

Роль стандартизирующих организаций

Открытые стандарты программного обеспечения. Эти простые слова означают очень многое. Например, в идеальной ситуации, когда все стандарты являются открытыми, разработчики смогут писать программные приложения, состоящие из компонентов производства разных поставщиков, и никаких проблем с совместимостью у них не будет. Проблема привязки к какому-то одному поставщику просто исчезнет. Все это пока является идеалом, и на пути к нему должна возрасти роль таких стандартизирующих организаций, как консорциум World Wide Web Consortium (W3C), Organization for the Advancement of Structured Information (OASIS) и др. Они должны разрабатывать такие качественные стандарты, описывающие функциональные возможности программных технологий, чтобы удовлетворить потребности сообщества разработчиков ПО. И так как требования к технологиям у разных подмножеств этого сообщества постоянно меняются, стандартизирующие организации должны постоянно держать нос по ветру и учитывать изменения в индустрии ПО. Им приходится постоянно приспосабливаться к изменяющимся нуждам пользователей и откликаться на эти изменения выпуском спецификаций на новые функции и технологии. В противном случае эти организации рискуют отстать от жизни и стать ненужными. Программные стандарты особенно важны, когда в стандартизации новых технологий, в том числе XML, нуждаются целые отрасли. Если таких стандартов нет либо они неадекватно отражают требования жизни, процесс взаимодействия компаний через Интернет будет чрезвычайно затруднен. Стандарты, выпускаемые организациями W3C и OASIS, являются ключом, который должен открыть двери к новым приложениям и к новым методам ведения бизнеса.

Роль W3C

В настоящее время самой важной стандартизирующей организацией для XML-индустрии является консорциум W3C. Как указывается на Web-сайте этой организации:

> Консорциум World Wide Web Consortium был создан в октябре 1994 г., чтобы способствовать полному раскрытию потенциала системы World Wide Web путем разработки общих протоколов, что должно ускорить ее эволюцию и гарантировать совместимость различных продуктов... Продвигая идею совместимости продуктов от разных поставщиков и создавая условия для открытых дискуссий, W3C берет на себя обязательства по управлению эволюцией системы Web.

По существу W3C — консорциум, состоящий из сотрудников W3C и компаний-участниц, таких как Oracle, IBM, Microsoft и Sun Microsystems, которые платят членские взносы, а за это их представители участвуют в работе целого ряда комитетов, разрабатывающих спецификации открытых стандартов для программного обеспечения. Это очень мощная организация.

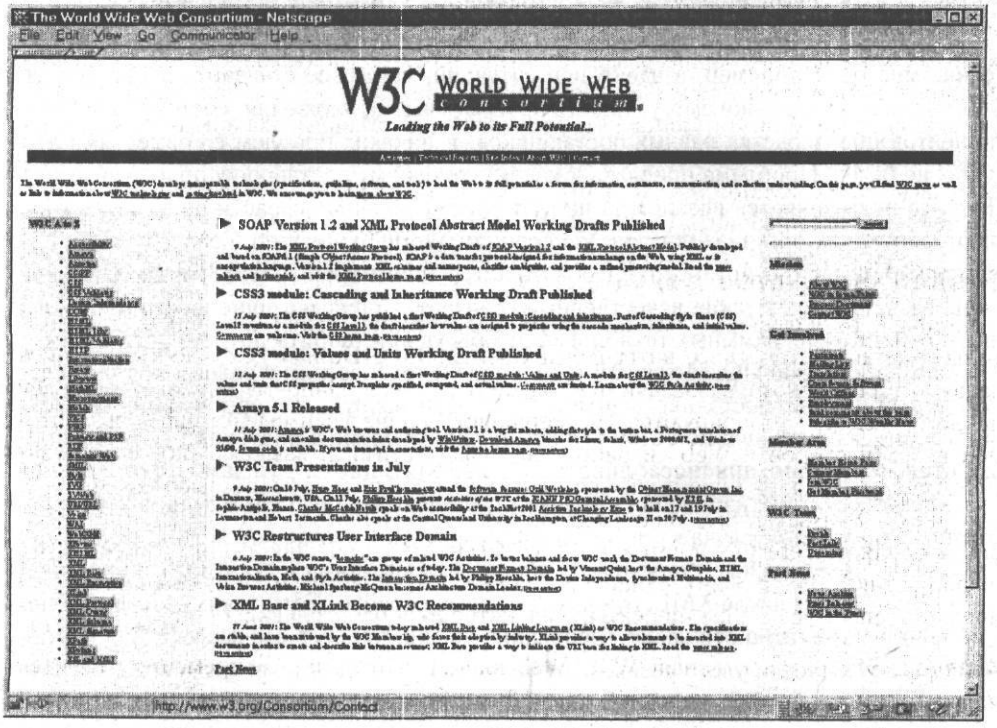


РИС. 11.1. *Web-сайт консорциума World Wide Web Consortium (Copyright 1994-2001, W3C [MIT, INRIA, KEIO]. All rights reserved.)*

Следует, правда, признать, что некоторые компании-члены этого консорциума не дожидаются, пока соответствующий комитет формально примет спецификации, и разрабатывают свои собственные запатентованные API-интерфейсы. Причем некоторые из них, например API-интерфейсы SAX, разработанные Дэйвом Меггинсоном (Dave Megginson), не получают одобрение W3C. Несмотря на это, разработчики ПО знают, что именно открытые стандарты наиболее перспективны, и именно этим путем надо идти. Когда консорциум W3C выпускает какие-то API-спецификации, разработчики тут же на них набрасываются, а если этого не происходит, следовательно, спецификации попросту провалились и никому не нужны. Если же массы разработчиков взяли на вооружение новые спецификации, то созданные ранее патентованные API-интерфейсы часто оказываются на обочине. В процессе выработки стандартов принимает участие много компаний, поэтому в таком деле, как электронный обмен информацией через Интернет, нет компании-монополиста.

Web-страница консорциума W3C находится по адресу http://www.w3c.org (см. рис. 11.1).

На этой странице слева располагаются ссылки на различные комитеты W3C и технологии. С правой стороны находятся ссылки на разделы, где излагаются цели и задачи консорциума, рассказывается о разных формах участия в работе консорциума. Тут же находится раздел, посвященный членам консорциума, где можно ознакомиться с условиями приема в эту организацию, архив новостей, средство поиска информации на сайте и архив почтовой рассылки. Вверху и внизу страницы расположены ссылки на следующие разделы сайта: Activities (деятельность консорциума), Technical Report (техническая документация), Site Index (индекс сайта), About W3C (о консорциуме W3C) и Contact (контактная информация). Наконец, по середине страницы располагаются аннотации текущих новостей и релизов спецификаций со ссылками на их полные тексты. Некоторые из этих ссылок мы рассмотрим, чтобы оценить вклад консорциума W3C в развитие технологии XML.

Раздел форматов документов

Раньше раздел форматов документов Documents Format Domain был частью раздела пользовательского интерфейса User Interface Domain. Сейчас в его список задач входят и некоторые задачи предыдущего раздела. Например, в качестве целей данного раздела декларируются "усовершенствование взаимодействия пользователей и компьютеров в Web" и работа над "форматами и языками, которые будут представлять информацию пользователям более точно и с более высоким уровнем контроля". Указывается и такая задача, как "совершенствование технологии, которая позволяет Web-пользователям более эффективно воспринимать и публиковать информацию". На момент написания данной книги на этой Web-странице были представлены следующие проекты, над которыми работал консорциум:

Amaya Это разработанный W3C Web-клиент, который одновременно является и браузером, и средством разработки. Он должен функционировать как тестовый инструмент и демонстрировать работу некоторых новых спецификаций, Web-форматов и протоколов.

Graphics В октябре 1996 г. консорциум W3C опубликовал формат для растровых изображений (Portable Network Graphics, PWG), а сейчас он разрабатывает открытый формат (Scalable Vector Graphics, SVG) для векторной графики, написанный на языке XML. Так как графика в Интернете сейчас развивается гораздо активнее, чем текст, в ближайшие годы консорциум, скорее всего, выпустит дополнительные рекомендации и спецификации, касающиеся этой сферы.

Hypertext Markup Language (HTML) Язык Hypertext Markup Language, известный как HTML, в настоящее время является стандартом для Web-страниц. Однако консорциум W3C разрабатывает язык XHTML, который является языком HTML следующего поколения. В языке XHTML должны быть учтены преимущества XML-приложений. Кроме того, одна из подгрупп консорциума работает над механизмом XForms для преобразования данных XML-формата. Предполагается, что технология XForms позволит свести к минимуму операции по написанию скриптов.

Internationalization Не ушли от внимания W3C такие проблемы интернационализации Интернета, как работа с языками, имеющими многобайтовые наборы символов, например китайским или японским. В связи с этим консорциум W3C занимается разработкой спецификации для кодировки Unicode, использующейся в XML-файлах. Результатом должна стать технология, позволяющая обрабатывать тексты на разных языках на любом компьютере в любой точке земного шара.

Math Язык Mathematical Markup Language (MathML) был создан консорциумом W3C в 1998 г. Это была рекомендация, позволяющая вставлять в HTML- и XML-документы математические выражения. Затем появилась новая версия этой рекомендации — MathML 2.O.

Style Sheets Консорциум W3C продолжает совершенствовать стандарт *каскадных таблиц стилей* (Cascading Style Sheets, *CSS),* чтобы реализовать более полный контроль над представлением Web-страниц. А с публикацией спецификации расширяемого языка стилей (extensible Style Language, XSL) представление XML-документов также должно перестать быть просто представлением, в него добавятся функции по преобразованию XML-документов.

Раздел взаимодействия

Раздел взаимодействия (Interaction Domain) раньше был частью раздела пользовательского интерфейса User Interface Domain. Сейчас его задачей является "исследование новых способов доступа к Web-информации". В этой области W3C занимается следующими проблемами:

Device Independence (независимость от устройств) Сейчас для доступа в Web используются самые разные устройства: персональные компьютеры, сотовые телефоны, карманные электронные органайзеры, навигационные устройства

системы GPS. В дальнейшем следует ожидать появления новых средств доступа, и все они должны общаться в сети прозрачно для пользователя. Консорциум W3C считает развитие этого рода совместимости очень перспективной задачей.

Synchronized Multimedia (синхронизация мультимедиа-данных)
Для интеграции звука, видео, текста и графики в Web консорциум W3C занимается разработкой языка *Synchronized Multimedia Integration Language (SMIL).* Пока спецификация SMIL является рекомендацией W3C, а не стандартом.

Voice Browser (голосовой браузер) Не так давно в Интернете появились службы загрузки MP3-музыки. Они пользуются большой популярностью, поэтому неудивительно, что консорциум W3C решил заняться и звуковыми Интернет-технологиями, которые работают также в центрах обработки телефонных звонков и в службах голосовой почты.

Сейчас консорциум рассматривает новые голосовые функции для языка XML, технологии оцифровки и тегирования речи и API-интерфейсы для навигации в голосовых данных. Если же говорить о XML-технологии, то внедрение в/ нее голосовых возможностей откроет просто потрясающие перспективы (достаточно представить себе передачу через Интернет телевизионного контента со всеми тегами для быстрого поиска и отбора нужных программ).

Раздел технологии и общества

Раздел технологии и общества (Technology and Society) посвящен "этическим и юридическим проблемам, связанным с внедрением новых технологий". Сюда относятся и проблемы обучения пользователей новым технологиям, и разъяснения им достоинств и недостатков внедрения новых технологий. В этом разделе представлены следующие проекты:

Metadata (метаданные) Этот проект появился в 1997 г. Его цель — создание технологий защиты интеллектуальной собственности в Интернете с использованием *шифрования и цифровых подписей на базе стандарта описания ресурсов* (Resource Description Framework, *RDF).* Вопрос защиты интеллектуальной собственности в Интернете стоит очень остро, поэтому консорциум W3C активно занимается этими проблемами.

XML Signature (xmldsig) XML Signature (XML-подпись) — это один из проектов, запущенных консорциумом W3C для реализации защиты и идентификации XML-документов. В этой сфере консорциум W3C занимается:

- Выработкой требований к XML-подписи
- Разработкой синтаксиса XML-подписей и технологии их обработки

Platform for Privacy Preferences (P3P) — платформа для обеспечения защиты частной информации. Инициатива P3P призвана обеспечить защиту пользователей в Интернете от посягательств на их личную информацию. Но защита эта должна реализовываться так, чтобы Интернет сохранил привлекательность для электронной коммерции. Первым пунктом реализации этого проекта стала рекомендация W3C для Web-сайтов о публикации информации для пользователей о политике сайта в области защиты частной информации. Проблемы защиты частной информации в Интернете с развитием электронной коммерции очень обострились, и консорциум W3C может внести существенный вклад в их решение.

Public Policy Role (Государственная политика) В консорциуме W3C есть специальная рабочая группа, которая занимается инициативами по информированию населения и политиков о возможностях Интернета в деле преодоления международных границ и проблем.

Electronic Commerce (Электронная коммерция) Электронная коммерция — или капитализм в Интернете — это концептуальное изменение методов ведения бизнеса, связанное с использованием Интернета как среды для продажи и покупки товаров и услуг. В консорциуме W3C есть специальная группа, которая рассматривает проблемы и приоритеты электронной коммерции и их возможные решения. Самыми острыми проблемами в этой сфере являются недостаток средств широкополосного доступа в сеть и обеспечение защиты потребителей.

Security (Безопасность) Проблемы безопасности идут рука об руку с проблемами электронной коммерции. Кроме того, обеспечение безопасности является важной проблемой для всей сети Интернет. Если пользователи не верят, что информация, передаваемая через Интернет, надежно защищена от посторонних глаз, они будут неохотно пользоваться такой средой, а это может стать тормозом для развития Интернета.

Очевидно, что основную роль в понимании важности проблем безопасности в Web сыграла электронная коммерция. Сейчас люди все больше задумываются над тем, как далеко может зайти коммерциализация системы Web, которая уже не является чисто информационной средой, как это было когда-то. Проблема защиты интеллектуальной собственности в Интернете сейчас стоит довольно остро, особенно когда дело касается доставки по сети кинофильмов и музыкальных записей.

Раздел архитектуры

Миссия раздела архитектуры (Architecture Domain) состоит в "использовании мощи компьютерной обработки в повседневной жизни человека" в рамках эволюции системы Web. На Web-странице этого раздела перечислены следующие проекты.

Document Object Model (DOM) Document Object Model — это стандартный API-интерфейс, позволяющий отслеживать, получать и обновлять XML-документы.

В перспективе следует ожидать появления все большего числа приложений баз данных, где в XML-схемах будут использоваться DTD, а сами XML-схемы будут отображены на схемы баз данных.

Jigsaw Jigsaw — разработанный W3C объектно-ориентированный Web-браузер, построенный на базе технологии Java.

XML: Structured Document Interchange (XML: обмен структурированными документами) Язык extensible Markup Language (XML) — это язык разметки, созданный на базе языка SGML. Он более мощный, чем язык HTML. Сейчас роль языка XML в электронной коммерции становится все более важной. Это объясняется тем, что он позволяет манипулировать структурированными данными, хранить их и извлекать из Web-сайтов с базами данных, а затем передавать эти данные на другие Web-сайты.

XML Protocol (XML протокол) Эта группа занимается разработкой технологий, позволяющих организовать взаимодействие в распределенных средах посредством технологии XML. В сферу ее интересов входит не только формат XML-сообщений, но и протоколы взаимодействия, например протокол Simple Object Access Protocol (SOAP).

Hypertext Transport Protocol (HTTP) Консорциум W3C является лидером в продвижении протокола Hypertext Transport Protocol (HTTP), который позволяет организовать межплатформный обмен мультимедиа-информацией по сети Интернет. С 1990 г. консорциум W3C активно совершенствует протокол HTTP, добавляя в него новые функциональные возможности и повышая производительность его работы, и будет продолжать это и в дальнейшем, чтобы протокол отвечал новым требованиям, связанным с ростом трафика в сети Интернет.

URI (Uniform Resource Identifiers) Назначение этой группы состоит в "мониторинге и проверке" документов других организаций, касающихся механизмов глобальных идентификаторов для Web. Данная группа занимается постоянными URI-идентификаторами, метаданными, новыми URI-схемами и интернационализированными идентификаторами.

Работа в разделе архитектуры идет очень активно. Это доказывает хотя бы пример создания технологии XML. Но и в дальнейшем в этой области следует ожидать очередных нововведений, о которых обычно объявляется на конференциях W3C (http://www.wwwlO.org). Материалы одной такой конференции представлены на рис. 11.2.

На этой конференции, прошедшей в мае 2001 г., были представлены доклады, темы которых отражают тенденции развития Интернет-технологий. Они посвящены производительности системы Web, проблемам защиты XML-документов, увеличению

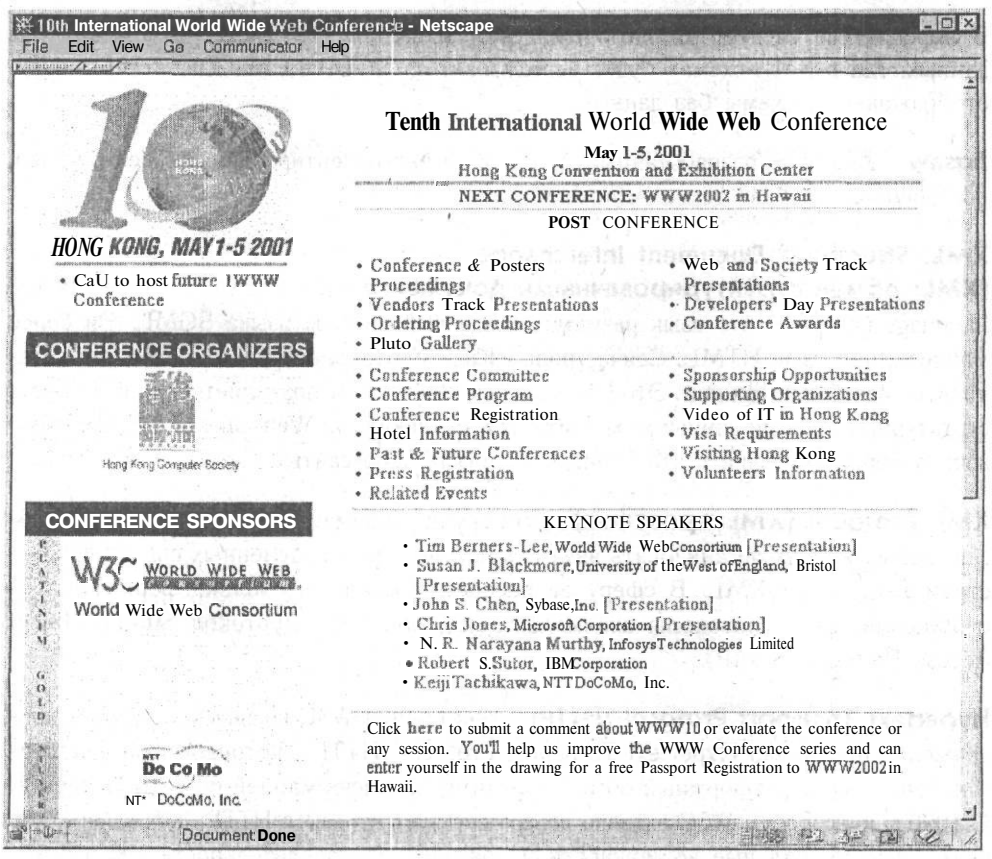


Рис. 11.2. *Web-сайт консорциума World Wide Web Consortium (Copyright 1994-2001, W3C [MIT, INRIA, KEIO]. All rights reserved.)*

информативности запросов в применении к XML-файлам, форматам публикации документов, средствам Web-поиска для карманных компьютеров, хранению Web-страниц в базах данных, использования профилей в Web, использованию сценариев в технологии XML, анализу гиперссылок, более эффективным поисковым механизмам, новым средствам создания и поиска Web-сайтов, голосовым и видео-форматам, Web-порталам, Web-архитекторам нового поколения, Web-кэшированию мультимедиа-данных, анализу информации в Web, рекламе в Интернете и анализу пользователей Web-страниц.

Таким образом, работы по "совершенствованию инфраструктуры Web и повышению уровня ее автоматизации" активно продвигаются и внутри консорциума W3C и за его пределами. Поэтому есть надежда, что новые разработки станут открытыми стандартами, как и те, что поддерживает консорциум W3C.

Инициатива по обеспечению доступности Web

Цель инициативы по обеспечению доступности системы Web (Web Accessibility Initiative, WAI) состоит в том, чтобы добиться доступности системы Web для людей с физическими недостатками через развитие технологий, создание соответствующих нормативов, сервисных программ, программ обучения и помощи, а также за счет проведения исследований и разработок в этой области. В настоящее время на Web-странице, посвященной этой инициативе, перечислены проекты, касающиеся всех перечисленных направлений. Здесь же приведены слова директора консорциума W3C и создателя системы World Wide Web Тима Бернерса-Ли (Tim Berners-Lee): "Мощь системы Web состоит в ее всеобщности. И очень важно, чтобы доступ в Web имел любой человек, независимо от его физического состояния".

Программное обеспечение с открытым исходным кодом от W3C

На сайте консорциума W3C предлагается для загрузки целый ряд программных продуктов, распространяемых с открытым исходным кодом. Конечно, хотелось бы, чтобы их было побольше, но в поисках подобного ПО можно заглянуть еще на сайт Apache Software Foundation **(http://xml.apache.org).** На Web-сайте Apache XML, страница которого показана на рис. 11.3, предлагаются для загрузки Java/C++ XML/XSLT-процессоры, Java-средство Web-публикации Cocoon с поддержкой технологии XML, форматирующие объекты Java XSL и средство быстрой разработки динамических страниц Java-сервера Xang.

В Web также можно найти сайты с открытыми кодами специализированных DTD или XML-схем, которые стандартизируются, например, консорциумом RosettaNet. Страница его сайта находится по адресу **http://www.rosettanet.org** (см. рис. 11.4). Кроме того, многие компании-члены консорциума W3C также предлагают на своих Web-сайтах ПО для бесплатной загрузки. Такое ПО можно найти и на Web-сайте компании Oracle по адресу **http://technet.oracle.com/tech/xml.**

Роль организации OASIS

Есть еще одна организация, которая в будущем будет оказывать очень большое влияние на индустрию XML. Это — Organization for the Advancement of Structured Information Standards (OASIS), из названия которой следует, что она выступает за развитие стандартов в области структурированной информации. Как указывается на ее Web-сайте:

> OASIS... — это некоммерческий международный консорциум, призванный ускорить принятие независящих от продуктов форматов, построенных на базе открытых стандартов, в том числе таких стандартов, как SGML, XML, HTML и CGM ...

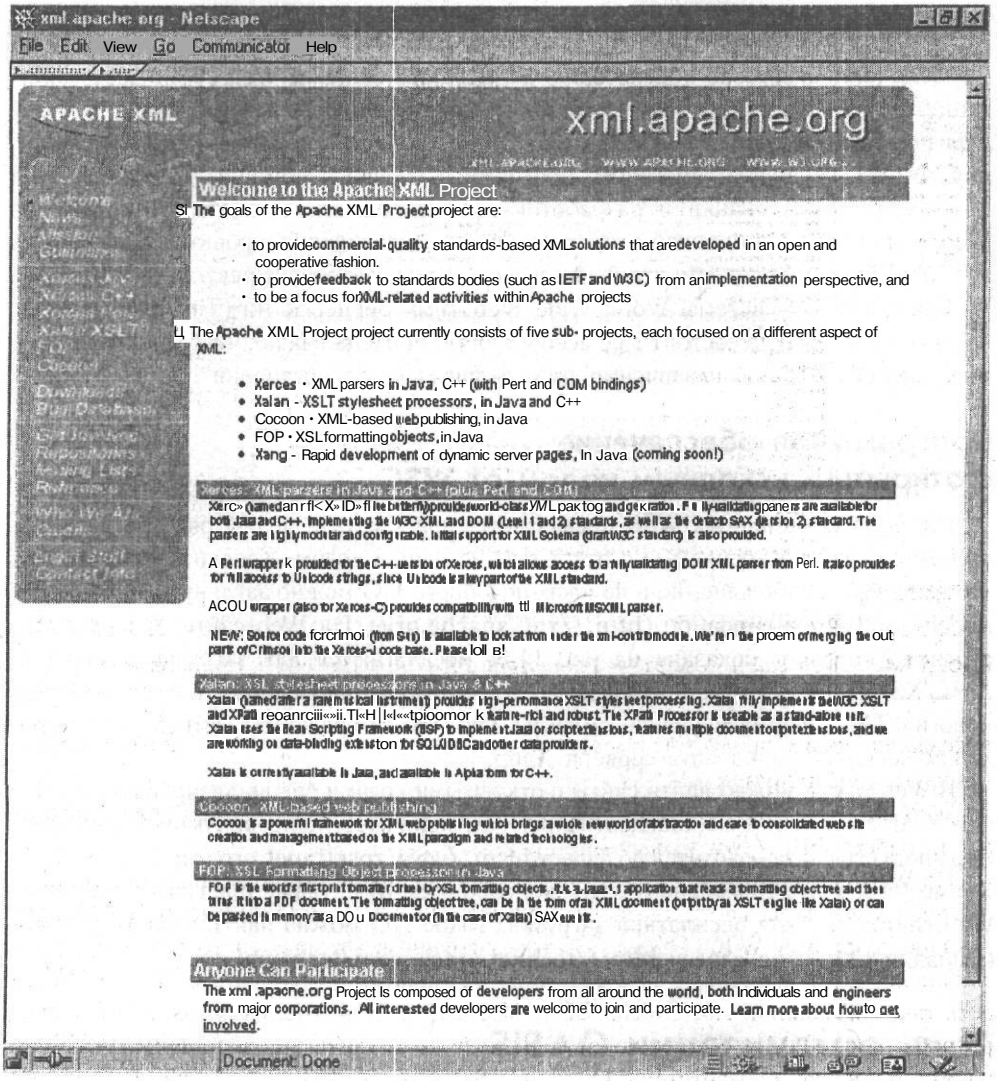


Рис. 11.3. *Web-сайт Apache XML (Copyright 2001, The Apache Software Foundation. All rights reserved.)*

Консорциум OASIS ставит своей целью сделать эти стандарты простыми для внедрения, выпуская для этого рекомендации относительно стратегий развития конкретных приложений. Наконец, этот консорциум предоставляет своим членам возможность "создания, получения, координирования и распространения информации, описывающей методологии, технологии и реализации этих стандартов". Скорее всего, компании, работающие с технологией XML, в дальнейшем будут использовать

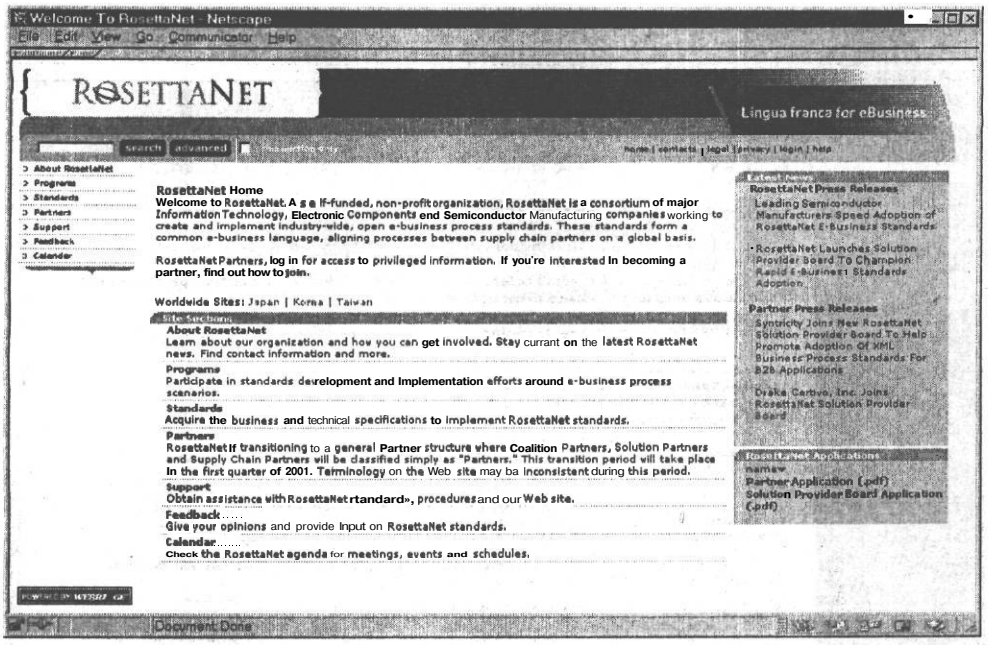


Рис. 11.4. *Web-сайт RosettaNet (Copyright 2001, RosettaNet. All rights reserved.)*

в создании своих продуктов именно те рекомендации и стратегии, которые были приняты консорциумом OASIS.

Web-сайт OASIS, главная страница которого показана на рис. 11.5, располагается по адресу **http://www.oasis-open.org.**

На Web-сайте OASIS есть ссылки на информацию об этой организации, на библиотеку, календарь, карту сайта, список членов OASIS и раздел новостей. Аналогично консорциуму W3C, OASIS — это консорциум, который состоит из компаний, финансирующих его работу за счет своих членских взносов. В числе членов OASIS есть такие компании, как Oracle, IBM, Microsoft и Sun Microsystems. Хотя многие разработчики ПО, работающие в XML-индустрии, знают консорциум OASIS по пакету проверки соответствия OASIS/NIST XML (NIST — National Institute of Standards and Technology, Национальный институт стандартов и технологий США), этот консорциум является также оператором Web-портала XML-индустрии XML.ORG, на котором публикуются последние новости об XML-приложениях и есть архивы справочной документации по XML-спецификациям, в том числе XML-схемам.

Главная страница сайта XML.ORG, которая показана на рис. 11.6, находится по адресу **http://www.xml.org.**

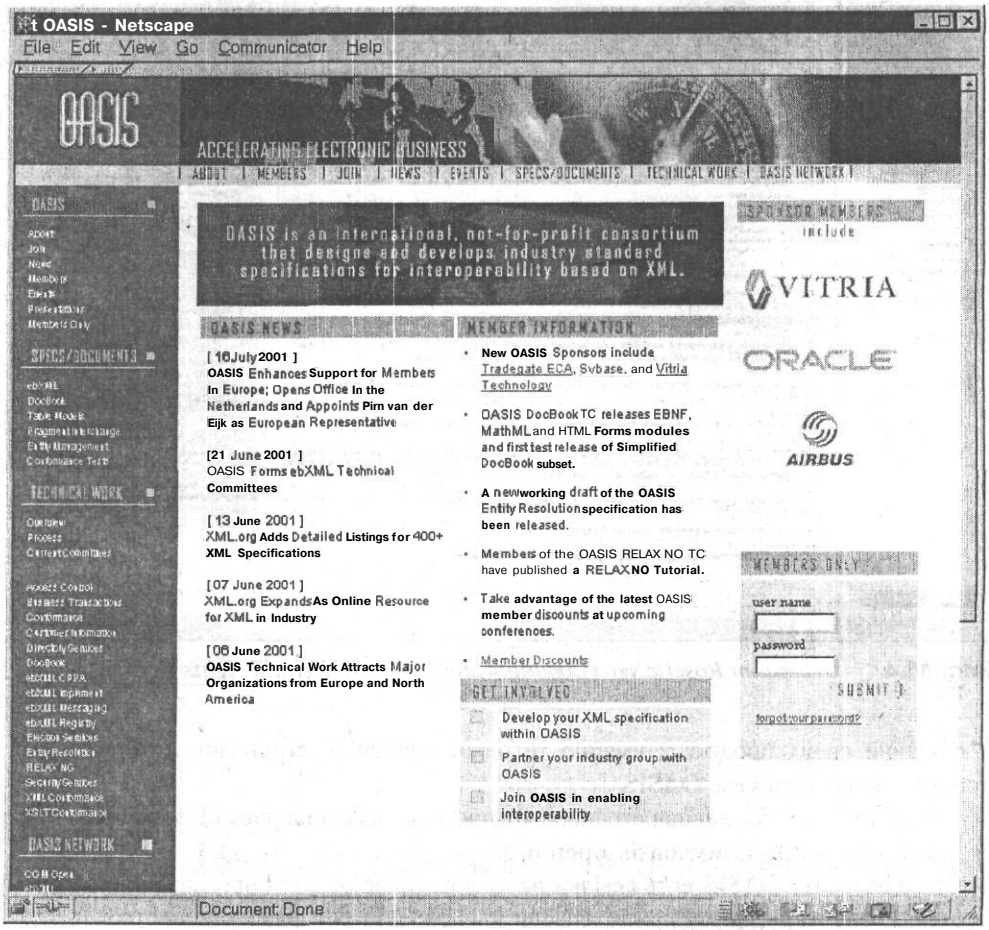


Рис. 11.5. *Web-сайт консорциума OASIS (Copyright 2001, OASIS. All rights reserved.)*

, Консорциум OASIS и его портал XML.ORG работают в сотрудничестве с другими стандартизирующими организациями, в том числе с консорциумом W3C, поэтому можно ожидать, что в дальнейшем в Web будет публиковаться еще больше справочных материалов по XML-технологиям, в том числе тестовые пакеты и XML-схемы. Другой вопрос, пойдет ли развитие этих справочных разделов с документацией по пути расширения публикации новых программ, распространяемых с открытым исходным кодом, но в любом случае появление в открытом доступе спецификаций на новые функции и соответствующей документации будет весьма полезно для XML-индустрии.

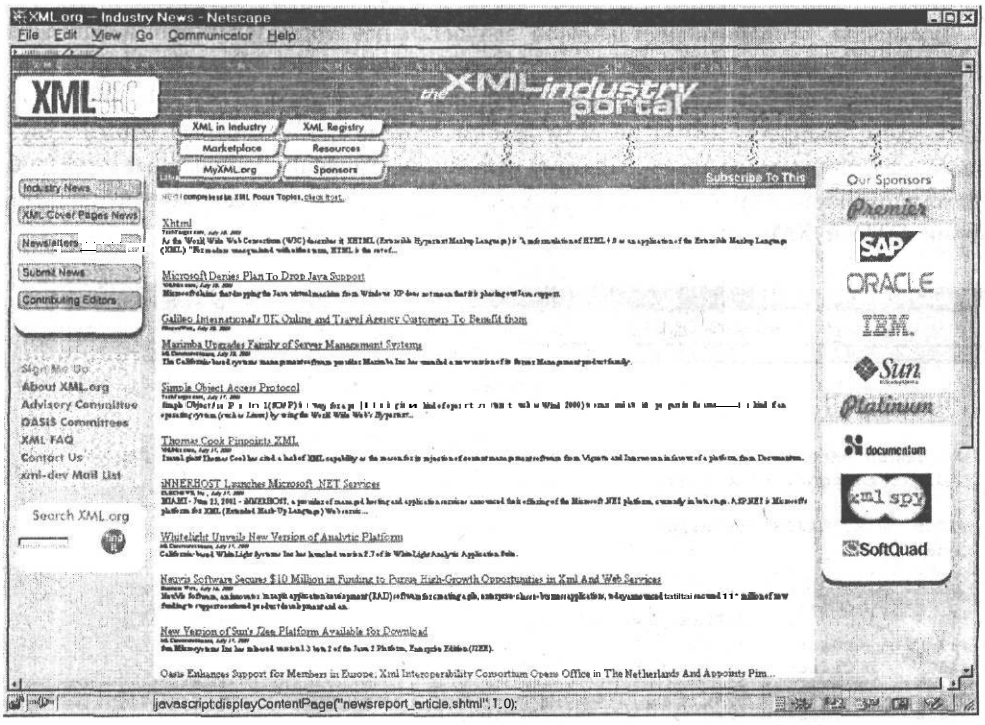


Рис. 11.6. *Верхняя часть главной страницы Web-сайта XML.ORG (Copyright 2001, xml.org. All rights reserved.)*

Схемы внедрения технологии XML в различных отраслях бизнеса

Как уже кратко упоминалось во введении к этой главе, некоторые отрасли индустрии уже создали или будут создавать в Интернете электронные биржи для покупки и продажи товаров, образуя таким образом массовые электронные торговые сообщества. Создание таких бирж заметно упростится, если информация, которую нужно хранить, извлекать из баз данных и посылать по сети, будет записана в формате XML. Не исключено, что некоторые отрасли будут для управления данными в XML-формате стандартизировать свою деятельность на базе DTD и XML-схем (XML Schema), например, DTD или XML Schema будут использоваться для описания деталей тормозной системы для поставщиков. Однако наличие нескольких форматов XML-данных неизбежно приведет к проблемам с производительностью и к появлению преобразователей данных, настроенных на решение каких-то частных задач.

Возможно, уговорить поставщиков запчастей для тормозов принять какие-то стандартизированные форматы данных будет нелегко. Но если это произойдет, и стандарты, построенные на базе специфичных для данной отрасли DTD или XML-схем, распространятся по всем другим отраслям, электронные биржи будут расти, как грибы после дождя. Поэтому целый ряд компаний, организаций, отраслей промышленности и правительство США решили создать для каждой отрасли репозитории DTD и XML-схем.

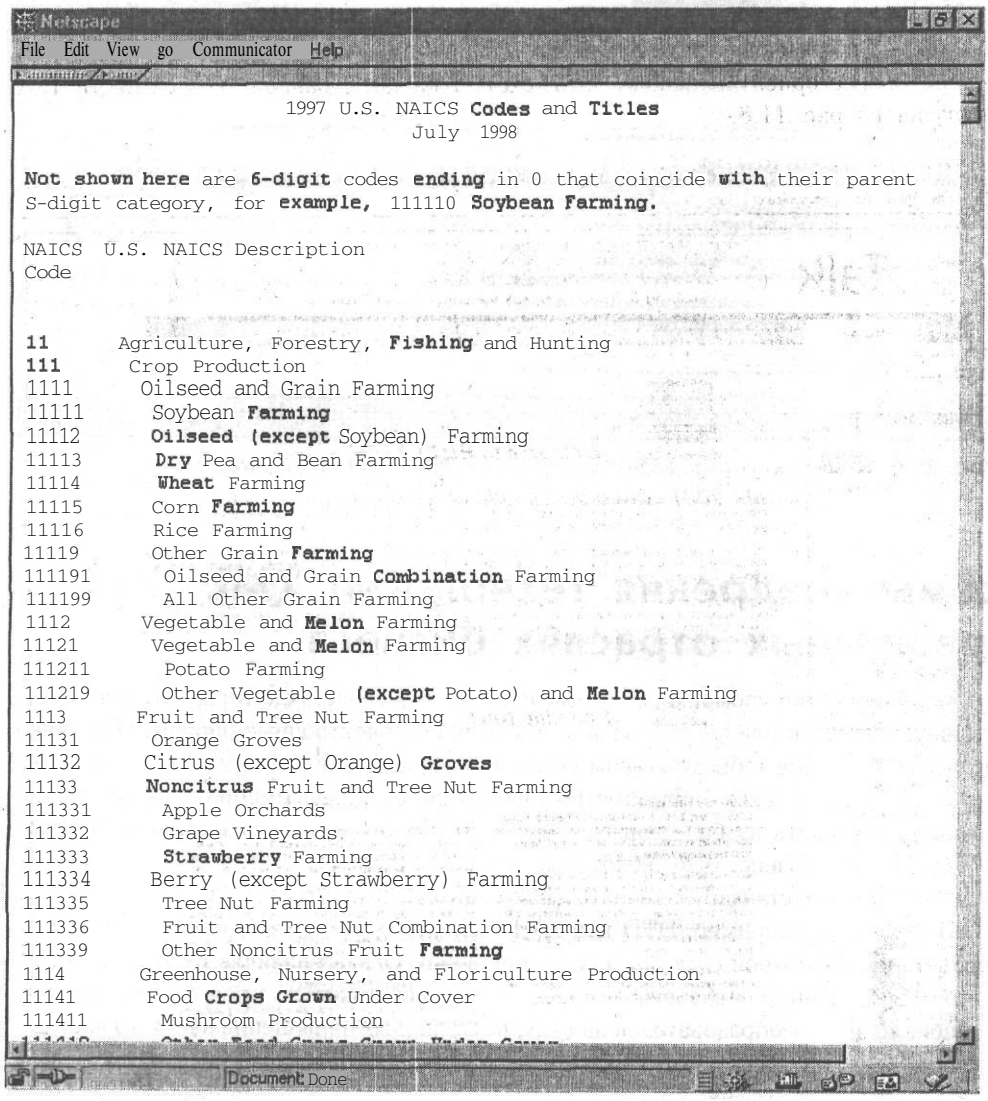


РИС. 11.7. *Web-сайт NAICS, http://www.census.gov/epcd/naics/naicscod.txt (Copyright 2001, U.S.Census Bureau. All rights reserved.)*

Основные игроки на арене XML-схем и DTD

На ум сразу приходят консорциумы OASIS и RosettaNet. Обе организации имеют механизмы, реализованные на их Web-сайтах, позволяющие запрашивать и получать DTD и XML-схемы. Например, на сайте RosettaNet классификация DTD и XML-схем соответствует категориям, определенным классификатором NAICS (North American Industry Classification System), который представляет собой полный список различных отраслей бизнеса (его фрагмент показан на рис. 11.7).

Кроме них можно вспомнить еще две организации: Microsoft BizTalk и DTD.com. Microsoft BizTalk является результатом усилий Microsoft по продвижению BizTalk-версии бизнес-ориентированных XML-схем. Главная страница Web-сайта BizTalk показана на рис. 11.8.

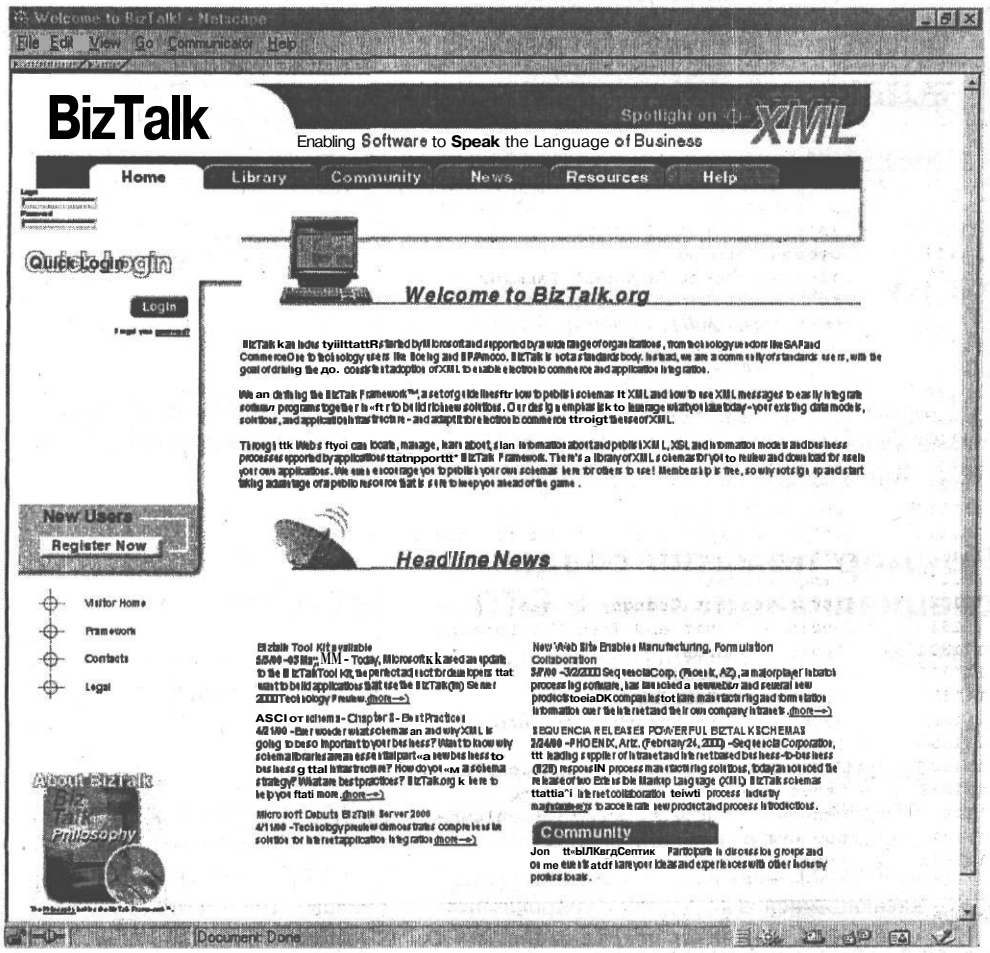


Рис. 11.8. *Web-сайт BizTalk, http://www.biztalk.org (Copyright 2001, BizTalk. All rights reserved.)*

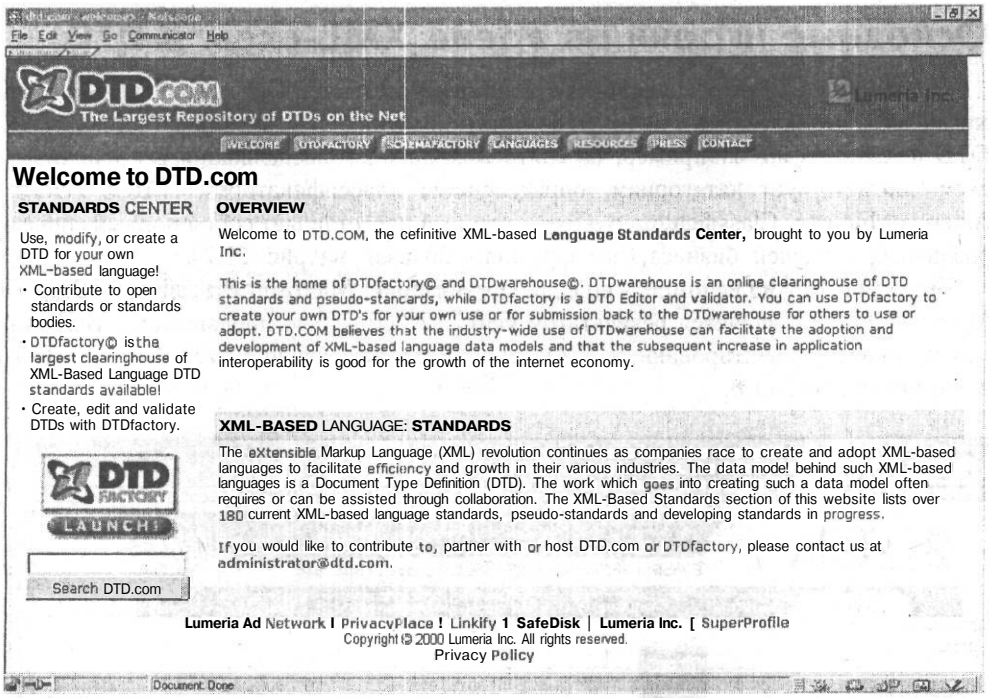


Рис. 11.9. *Web-сайт DTD.com, http://www.dtd.com (Copyright 2001, Lumeria. All rights reserved.)*

У организации DTD.com также есть репозиторий DTD и XML-схем, которые предлагаются разработчикам вместе со средствами создания, редактирования и проверки этих моделей контента. Web-сайт DTD.com показан на рис. 11.9.

Пример попытки создания специализированных DTD и XML-схем

FinXML — это организация, созданная для сбора DTD/XML-схем для компаний, работающих в области финансовых услуг. На ее Web-сайте опубликована следующая декларация:

> FinXML™ — это структура, построенная на базе технологии XML, созданная для поддержки единого универсального стандарта обмена данными между компаниями, работающими на рынках ценных бумаг, и позволяющая финансовым организациям производить обмен данными, касающимися высокоструктурированных финансовых транзакций, между собственными подразделениями или с другими организациями в электронной форме через сеть Интернет.

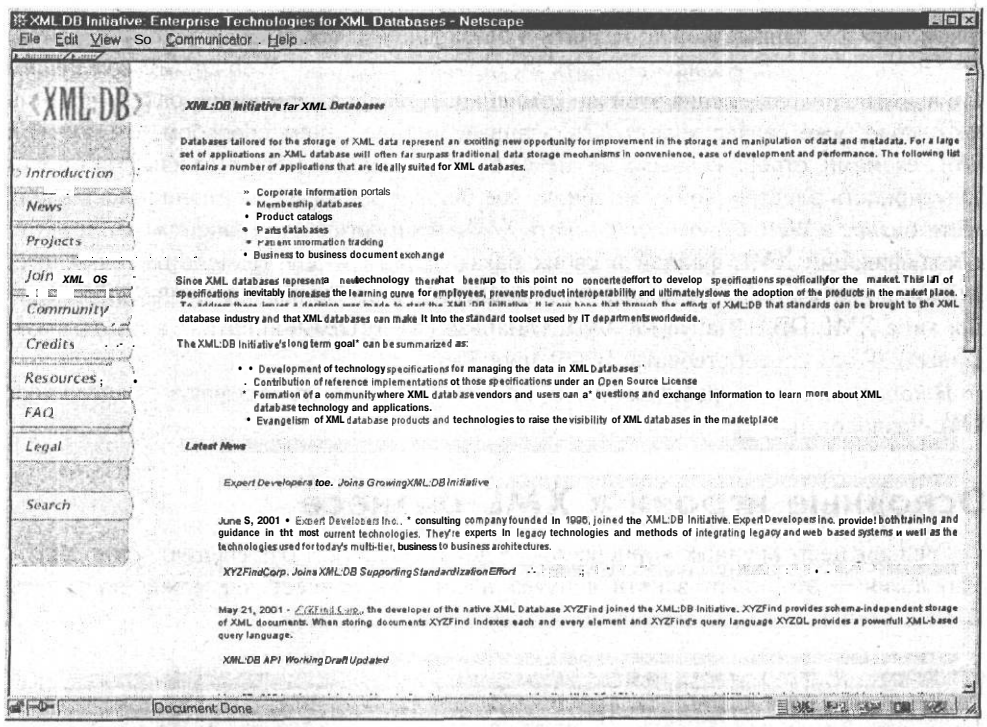


Рис. 11.10. *Web-сайт FinXML, http://www.finxml.org (Copyright 2001, FinXML.org. Ail rights reserved.)*

Консорциум, поддерживающий FinXML, состоит из участников фондового рынка, серьезно заинтересованных в стандартизации словаря, описывающего финансовые транзакции и другие связанные с ними аспекты. Для участников этого консорциума на Web-сайте FinXML публикуются новости, спецификации и готовые программные продукты. Так как FinXML построена на базе стандартов, предложенных Международной ассоциацией по свопам и производным ценным бумагам (International Swaps and Derivatives Association, ISDA), организациями BizTalk и cXML, члены консорциума могут взаимодействовать друг с другом и не заниматься проталкиванием собственных патентованных словарей. Поэтому они чрезвычайно заинтересованы в пропаганде FinXML на всех мировых фондовых рынках.

Web-сайт FinXML показан на рис. 11.10.

Влияние технологии XML на Интернет

HTML — язык разметки для представления данных, а XML — язык электронной коммерции. Основным преимуществом XML относительно HTML является то, что в нем данные можно окружать тегами, определяемыми самим пользователем, и семантика этих тегов связана с данными, к которым они относятся. Сформатированные

таким образом данные можно хранить в базах данных, обслуживающих Web-сайты, извлекать их оттуда, модифицировать и отправлять их через Web другим компаниям или в другие подразделения этой же компании. Целый ряд компаний, работающих на XML-поле, уже сейчас хранят XML-данные именно таким способом, с DTD или XML-схемами, отображенными на схемы базы данных. По мере того как будет расти популярность электронной коммерции, все больше и больше компаний, желающих вести бизнес в Web, будут использовать XML-технологию для хранения, извлечения и модификации XML-файлов в своих базах данных и для транспортировки этих файлов по сетям в другие компании и подразделения. Во всяком случае, организации типа XML:DB initiative for XML Database (XML:DB-инициатива за XML-базы данных) будут способствовать этому процессу.

В конце концов потребители тоже, прямо или косвенно, выиграют от внедрения XML-технологий.

Основные игроки в XML-бизнесе

Главная цель крупных корпораций — делать деньги. Для специалистов XML-технология — это просто занятная штука, а для тех, кто знает, как можно построить на базе этой технологии приложения электронного бизнеса, XML — это большой

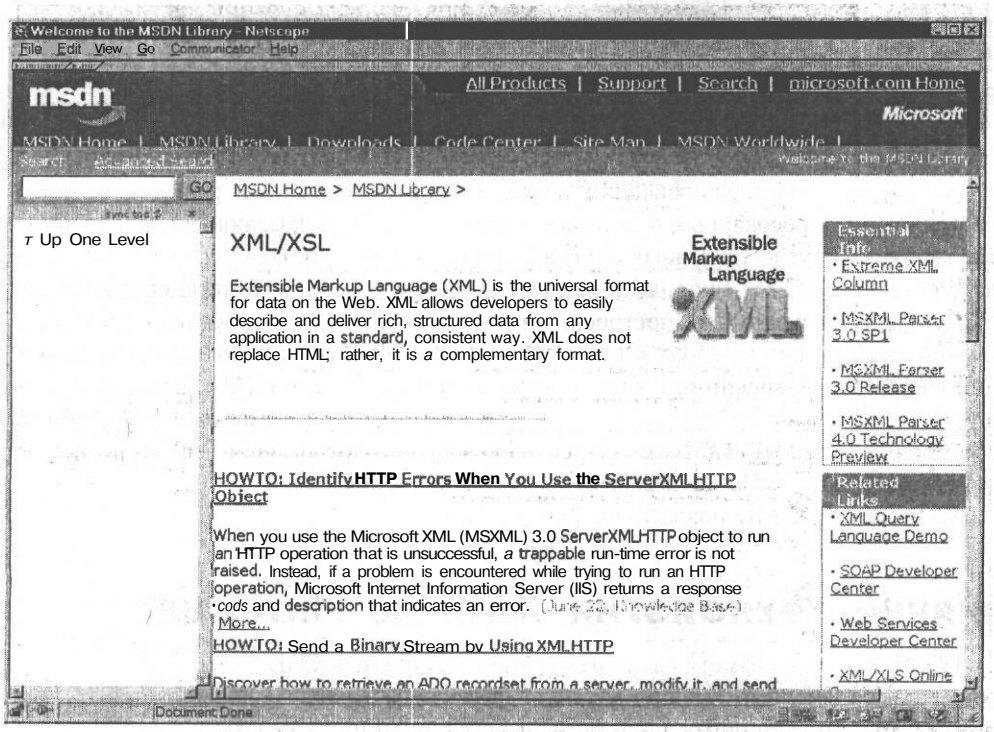


Рис. 11.11. *Web-сайт Microsoft, http://www.microsoft.com/xml (Copyright 2001, Microsoft Corporation. All rights reserved.)*

бизнес. Так как для B2B-бирж и рынков нужно сначала построить соответствующую инфраструктуру, то консалтинговые услуги, техническая поддержка и продажа программного обеспечения превращаются для компаний, работающих в XML-бизнесе, в миллиарды долларов дохода. Хотя в XML-бизнесе могут преуспеть и действительно преуспевают небольшие начинающие фирмы, все же львиная доля доходов приходится на крупные компании. Поэтому неудивительно, что такие компании, как Microsoft, IBM, Sun Microsystems и Oracle, тратят столько денег, времени и сил на технологию XML. Теперь аббревиатуру XML вы увидите на Web-сайтах всех перечисленных компаний, доказательством чему служат рис. с 11.11 по 11.14, на которых показаны соответствующие Web-страницы.

Если внимательно ознакомиться с содержимым этих сайтов, несложно заметить, что предлагаемые на них ресурсы и новости на удивление похожи. Пока не известно, кто из упомянутых компаний станет победителем на XML-арене, но триллиона долларов, которые сулит B2B-бизнес, должно хватить на всех. Ясно только, что победителями станут сама технология XML, те компании, которые будут использовать ее в своем бизнесе, и конечные потребители, так как построенные на базе XML технологии должны упростить ведение электронной коммерции через Интернет, снизить издержки производства и в конечном итоге цены.

Рис. 11.12. *Web-сайт IBM, http://www.ibm.com/xml*
(Copyright 2001, IBM Corporation. All rights reserved.)

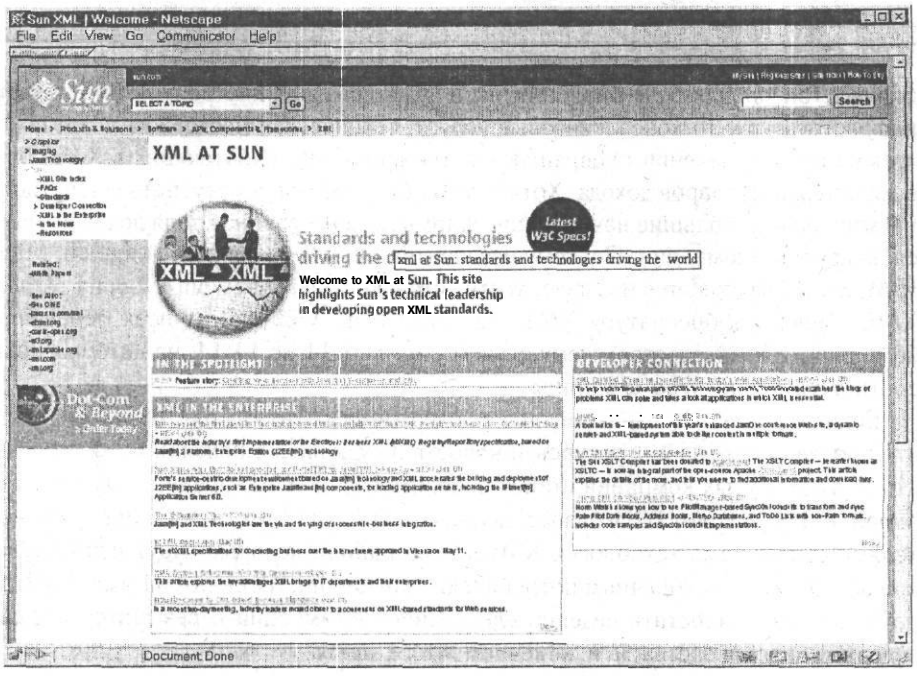


Рис. 11.13. *Web-сайт Sun Microsystems, http://www.sun.com/xml (Copyright 2001, Sun Microsystems Corporation. All rights reserved.)*

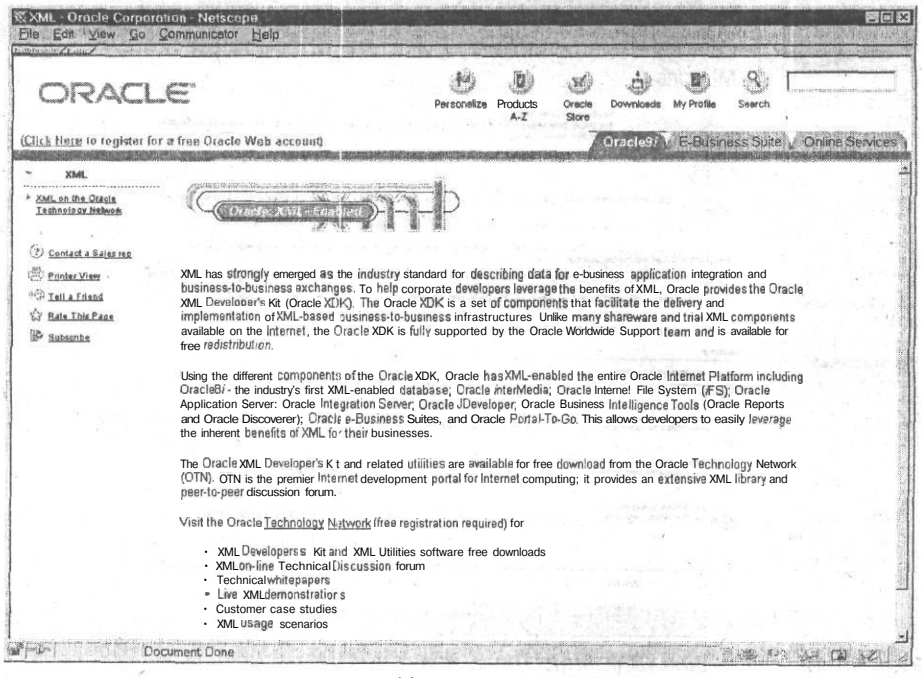


Рис. 11.14. *Web-сайт Oracle, http://www.oracle.com/xml (Copyright 2001, Oracle Corporation. All rights reserved.)*

Приложение А

Спецификации W3C XML, DOM, SAX и XSLT

Спецификация XML

Что такое XML!

Программный модуль, называемый анализатором XML-синтаксиса, используется приложениями для чтения XML-документов и получения доступа к содержимому этих документов и их структуре. Спецификация XML описывает поведение анализатора XML-синтаксиса, определяя, каким образом он должен считывать XML-данные и какую именно информацию передавать в использующее его приложение. Однако эта спецификация не описывает, как именно приложения должны интерпретировать полученные данные.

Документы

Объект данных называется XML-документом, если он *правильный* (well-formed), т. е. соответствует основным правилам построения документа, определяемым спецификацией XML, и удовлетворяет определенным специальным ограничениям, налагаемым в разных разделах спецификации. Каждый XML-документ имеет логическую и физическую структуры. Физически документ состоит из блоков, называемых *сущностями* (entities), которые могут ссылаться на другие сущности, включенные в документ. Документ начинается с сущности документа "root". Логически документ состоит из описаний, элементов, комментариев, символьных ссылок и инструкций по обработке. Все эти составные части выделяются специальной разметкой. Весь текст, который не является разметкой, представляет собой символьные данные документа.

Элементы

Каждый XML-документ содержит один или несколько *элементов.* Границы каждого элемента разделяются открывающими (например, <book>) и закрывающими тегами (например, </book>) или тегом пустого элемента (например, **</book>**) для пустых элементов. Каждый элемент может иметь набор атрибутов. Каждый атрибут имеет имя и значение. Например, **<book isbn="1234-5678-1432-1224"/>** задает элемент book с атрибутом isbn.

Комментарии

Комментарии могут находиться в любом месте документа за пределами знаков разметки. Кроме того, комментарии могут находиться в определении типа документа в тех местах, где это разрешено грамматическими правилами. Они не являются частью символьных данных документа. Комментарии имеют вид: <!-- This is a comment -->. Отметим, что знаки "--" не разрешается употреблять внутри комментариев, так как они являются частью синтаксиса комментариев.

Инструкции по обработке

Инструкции по обработке используются для передачи приложениям специальных инструкций. Эти инструкции не являются частью символьных данных документа, они просто передаются в приложение. Инструкции начинаются с указания целевого объекта, используемого для идентификации приложения, в которое эти инструкции направляются. Для описания целевых объектов инструкций по обработке может использоваться механизм XML Notation (XML-нотации). Инструкции по обработке должны иметь следующий вид:

```
<? App This is an instruction ?>
```

Названия целевого объекта XML и различные его варианты зарезервированы для стандартизации. В самом начале большинства XML-документов ставится специальная инструкция по обработке, называемая *xml-описанием*. Она имеет вид: <?xml **version=1.0 encoding="ISO-8859-1" standalone="yes"?>**. Версия номер 1.0 используется для указания на соответствие версии 1.0 спецификации XML. Атрибут кодировки определяет набор символов, использованный для данного XML-документа. По умолчанию значение этого атрибута принимается равным **"UTF-8"**. Если для атрибута "standalone" установлено значение "yes", это означает, что анализатор XML-синтаксиса, который не читает внешнее подмножество DTD (в том числе сущности внешних параметров, на которые есть ссылки из внешних сущностей), будет синтезировать такое же содержимое документа, как если бы он считывал внешнее подмножество DTD.

Секции CDATA

```
<![CDATA[<book> No need to escape <and > <?book>]]>
```

Определение типа документа

Если XML-документ содержит определение типа документа (Document Type Definition, DTD), то анализатору XML-синтаксиса, имеющему функцию проверки такого документа на допустимость (так делают все анализаторы XML-синтаксиса от компании Oracle), можно дать команду проверить документ на допустимость.

Анализатор проверяет, не нарушаются ли ограничения, выраженные в DTD. Для выполнения такой проверки анализатор XML-синтаксиса считывает и обрабатывает все определение DTD и все внешние сущности, на которые есть ссылки в данном документе. Он также проверяет и другие ограничения на допустимость, например, чтобы имя в определении типа документа соответствовало типу корневого элемента. XML-документ, который процессор проверки на допустимость счел соответствующим его DTD, называется *допустимым*. DTD содержит описания разметки, которые определяют разрешенные грамматические правила использования элементов и атрибутов в XML-документе. Существуют четыре вида описаний разметки: описания типа элемента, описания списка атрибутов, описания сущности и описания нотаций.

Описания типа элемента

Вот пример описания типа элемента:

```
<!ELEMENT book (title, author, publisher, price)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT author (#PCDATA)>
<!ELEMENT publisher (#PCDATA)>
<!ELEMENT price (#PCDATA)*>
```

В этом описании указывается, что элемент book может иметь только четыре дочерних элемента: **title, author, publisher и price.** Каждый из этих элементов может содержать только символьные данные. Кроме того, это описание указывает на то, что элемент price может определяться не всегда (значок * означает 0 или больше).

Любое описание типа элемента также может определять тип содержимого этого элемента. Этот тип может быть **EMPTY** (пустой), ANY (любой) и **Mixed** (смешанный). Тип **EMPTY** означает, что у элемента не должно быть дочерних элементов. Отметим, что такой тип не следует использовать, если наличие дочерних элементов просто не является обязательным (в этом случае нужно использовать оператор *, как это показано выше). Типы ANY и **Mixed** указывают на то, что элемент может содержать символьные данные, в нем могут быть дочерние элементы (это необязательно). То есть в элементы, которым разрешено иметь дочерние элементы, можно помещать символьные данные.

Описание списка атрибутов

Атрибуты строкового типа могут содержать только значения в виде символьных данных. Вот пример описания списка атрибутов строкового типа:

```
<!ATTLIST book isbn CDATA #REQUIRED>
```

Здесь показано, что isbn — это атрибут строкового типа для элемента *book,* а каждый элемент *book* должен содержать атрибут **isbn.**

Атрибуты с помеченным типом могут иметь одно из следующих значений: **ID, IDREF, IDREFS, ENTITY, ENTITIES,** NMTOKEN или NMTOKENS. Атрибуты типа ID предназначены для присваивания меток элементам в XML-документах и используются для описания атрибутов, которые играют роль идентификаторов элементов. Атрибуты типов **IDREF и IDREFS** предназначаются для элементов типа ссылок, которые имеют метки, присвоенные атрибутами **ID.** Элементы могут иметь несколько атрибутов типа IDREF, которые, в свою очередь, могут иметь заданные значения или значения по умолчанию. В допустимом XML-документе каждая адресуемая единица, на которую есть ссылка, определяемая атрибутом типа **IDREF** или **IDREFS,** должна быть в каком-то месте этого документа определена еще и атрибутом типа ID. Разница между атрибутами типов **IDREF** или **IDREFS** состоит в том, что атрибут типа **IDREFS** может иметь одно или несколько значений имени, разделенных между собой пробелами, которые могут ссылаться на несколько адресуемых единиц. Пример описания атрибутов, определяющего атрибуты типов ID и IDREF:

```
<!ATTLIST book isbn ID #REQUIRED>
<!ATTLIST booktitle isbnref IDREF #IMPLIED>
```

В этом примере isbn описан как идентификатор элемента *book, a* isbnref — как ссылка для элементов *book,* имеющих определенный isbn.

Атрибуты типов **ENTITY** разрешают, чтобы значения атрибутов содержали ссылки на имеющуюся сущность, не подвергаемую синтаксическому анализу (и описанную в DTD). Теперь приведем пример описания списка атрибутов, определяющего тип сущности:

```
<!ENTITY niftysound SYSTEM 'beatles.wav' NDATA WAV>
<!ENTITY Group ( #PCDATA )>
<!ATTLIST Group sound ENTITY #IMPLIED>
```

В этом примере описывается сущность **niftysound,** ссылающаяся на внешний системный файл *beatles.wav,* который имеет связанный атрибут **NOTATION** с названием WAV. В нем также описывается элемент **Group** и необязательный атрибут **sound,** который может ссылаться на тип сущности. Теперь проверяться на допустимость будет следующий XML-документ:

```
<Group sound="niftysound">The Beatles</Group>
```

Атрибуты типов **NMTOKEN и NMTOKENS** предназначаются для атрибутов, которые в качестве разрешенных значений могут иметь только метки имен. Эти метки имен имеют определенные дополнительные ограничения на разрешенные типы символов, отличные от CDATA.

Атрибуты перечислимых типов данных могут иметь несколько значений, указанных в DTD. Самих перечислимых атрибутов существует два вида: **NOTATION и Enumeration.** Атрибут **NOTATION** определяет нотацию, описанную в DTD с помощью

связанных системных и/или открытых идентификаторов. Он используется в интерпретации элемента, к которому этот атрибут относится. Пример описания списка атрибутов, которое определяет атрибут **NOTATION:**

```
<!ATTLIST Group sound NOTATION (WAV I AU I RA) #IMPLIED>
```

Внимание

S *DTD* для каждого *имени нотации WAV, AU и RA должно* быть отдельное *описание нотации.*

Атрибут **Enumeration** имеет только те значения, которые перечислены в атрибуте типов **NMTOKEN**. Следующий пример иллюстрирует описание списка атрибутов, которое определяет атрибут **Enumeration:**

```
<!ATTLIST book genre ( Mystery I Humor I Romance I Children ) #REQUIRED>
```

Здесь для элемента *book,* имеющего фиксированный набор разрешенных значений, описывается обязательный атрибут *genre.*

Сущности

```
<!ENTITY company "Oracle Corporation">
<!ENTITY soundfile SYSTEM "c:\sounds\beatles.wav">
```

Ссылки на эти сущности могут находиться внутри XML-документа. Они выглядят следующим образом:

```
<Corporation>&company;</Corporation>
<Group sound="soundfile">The Beatles</Group>
```

Сущности параметров хранятся как сущности в DTD. Они действительны только внутри DTD и могут описываться в DTD путем добавления к имени сущности префикса %:

```
<!ENTITY % bookchildren "(title, author, publisher, price )">
<! ELEMENT book %bookchildren;>
```

Здесь утверждается, что элемент *book* может иметь дочерние элементы только типов *title, author, publisher* и *price.*

Анализатором XML-синтаксиса распознаются и некоторые другие специальные заранее определенные общие сущности: **amp, It, gt, apos и quot.** Так, например, **&amp** может использоваться для генерирования символа & и для его пропуска, если он встречается в символьных данных.

Нотации

```
<!NOTATION WAV SYSTEM "c:/musicplayer/playwav.exe">
```

Здесь утверждается, что нотация WAV существует и связана с приложением **playwav.exe.**

Спецификация DOM

Что такое DOM!

DOM — это модель представления данных, построенная на базе структуры объекта, которая очень похожа на структуру документа. Для примера рассмотрим следующий XML-документ:

```
<booklist>
<book  isbn="1234-123456-1234">
      <title>C Programming Language</title>
      <author>Kernighan and Ritchie</author>
      <publisher>IEEE</publisher>
      <price>7.99</price>
   </book>
</booklist>
```

DOM-представление этого кода показано на рис. АЛ.

В DOM документы имеют древовидную логическую структуру, известную также под названием *структурная модель.* На рис. А. 1 можно видеть, что корневой элемент *booklist* является, как и следовало ожидать, корнем дерева DOM. Этот

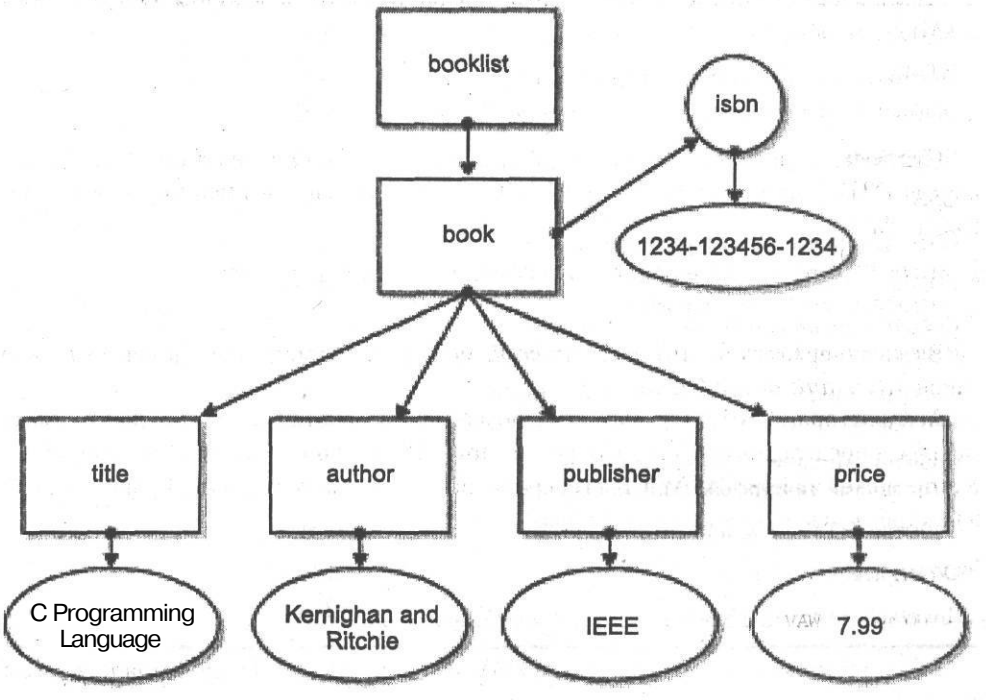


Рис. А.1. *DOM-представление*

корневой элемент содержит один дочерний элемент *book,* который, в свою очередь, имеет четыре дочерних элемента *title, author, publisher* и *price* и один атрибут *isbn*. Листьями этого дерева являются простые текстовые строковые значения. Узлы дерева DOM можно обойти методами обхода дерева (в котором не учитываются атрибуты). Структурные модели DOM обладают одним очень важным свойством — *структурным изоморфизмом*. Это означает, что если для представления одного и того же документа используются два DOM-представления, то они создают одну и ту же структурную модель. То есть для создания представления DOM можно выбирать любую структуру данных (и не обязательно древовидную).

DOM Level 2 и Level 3

Проект спецификации DOM Level 3 W3C Working Draft состоит из следующих разделов: DOM Level 3 CORE (библиотека функций), Abstract Schemas and Load/Save (абстрактные схемы и операции загрузки/сохранения), Events (события) и язык путей XPath. Остальные функции определяются пользователями DOM в соответствии с требованиями их приложений.

Таблица А.1

Функциональные строки DOM Level 2

Модуль	Функциональная строка
XML	XML
HTML	HTML
Views	Views
SlyleSheets	SlyleSheets
CSS	CSS
CSS (расширенные интерфейсы)	CSS2
Events (события)	Events
События пользовательского интерфейса User Interface Events (интерфейс UIEvent)	UIEvents
События Mouse Events, связанные с мышью (интерфейс MouseEvents)	MouseEvents
События Mutation Events, связанные с мутациями (интерфейс MutationEvent)	MutationEvents
События HTML Events	HTMLEvents
Обход дерева (Traversal)	Traversal
Диапазон значений (Range)	Range

DOM-приложение может применять метод hasFeature объекта DOMImplementation для определения того, поддерживается ли данный модуль. Объект DOMImplementation можно возвращать из документа с помощью метода getImplementation. Функциональные строки всех модулей в DOM Level 2 приведены *в* таблице А. 1.

Внимание

Строки не чувствительны к регистру вводимых символов.

В таблице А.2 приведены все зависимости между модулями, перечисленными в таблице А. 1.

Таблица А.2

Зависимости между модулями DOM Level 2

Модуль	Соответствия
Views	XML или HTML
SlyleSheets	SlyleSheets и XML или HTML
CSS	SlyleSheets, Views и XML или HTML
CSS2	CSS, SlyleSheets, Views и XML или HTML
Events	XML или HTML
UIEvents	Views, Events и XML или HTML
MouseEvents	UIEvents, Views, Events и XML или HTML
MutationEvents	Events и XML или HTML
HTML Events	Events и XML или HTML

Ядро DOM

DOM-дерево состоит из узловых (Node) объектов. В представлении XML-документов используются несколько видов узлов: Document, **DocumentFragment**, Attr, Element, Text, DocumentType, **ProcessingInstruction**, Comment, CDATASection, **EntityReference** и Notation. DOM также определяет некоторые другие типы, представляющие список узлов — NodeList, **NamedNodeMap**,— и вводит тип **DOMString**, являющийся строкой символов в кодировке UTF-16. Наконец, DOM представляет тип исключений DOMException, который используется разными DOM-интерфейсами при выполнении ошибочной операции или если во время исполнения программы происходят какие-то другие ошибки.

В таблицах А.3 и А.4 описаны различные DOM-типы, представленные в разных анализаторах синтаксиса Oracle XML Parser.

Таблица А.3

DOM-типы и соответствующие им Oracle-типы

DOM-тип	Java	PL/SQL
Node	XMLNode	DOMNode
Document	XMLDocument	DOMDocument
Element	XMLElement	DOMElement
Attr	XMLAttr	DOMAttr
Text	XMLText	DOMText
DocumentFragment	XMLDocumentFragment	DOMDocumentFragment
ProcessingInstruction	XMLPI	DOMPI
DocumentType	DTD	XMLDTD
EntityReference	XMLEntityReference	DOMEntityReference
Comment	XMLComment	DOMComment
CDATASection	XMLCDATA	DOMCDataSection
NodeList	Не определен (частный класс)	DOMNodeList
NamedNodeMap	Не определен (частный класс)	DOMNamedNodeMap
Notation	XMLNotation	DOMNotation
DOMString	java.lang.String	VARCHAR2
DOMException	Не определен (частный класс)	EXCEPTION

Таблица А.4

DOM-типы и соответствующие им Oracle-типы

DOM-тип	C	C++
Node	xmlnode	Node
Document	xmlnode	Document
Element	xmlnode	Element
Attr	xmlnode	Attr
Text	xmlnode	Text
DocumentFragment	xmlnode	DocumentFragment
ProcessingInstruction	xmlnode	ProcessingInstruction
DocumentType	xmlnode	DocumentType
EntityReference	xmlnode	EntityReference
Comment	xmlnode	Comment

Таблица А.4 (*продолжение*)

DOM-типы и соответствующие им Oracle-типы

DOM-тип	C	C++
CDATASection	xmlnode	CDATA section
NodeList	xmlnodes	NodeList
NamedNodeMap	xmlnodes	NamedNodeMap
Notation	xmlnode	Notation
DOMString	oratext*	DOMString
DOMException	Не определен	Не определен

Node

Чтобы получить в узле информацию без спуска к специальному производному интерфейсу, используются атрибуты nodeName и **nodeValue**. Эти атрибуты можно получить с помощью методов getNodeName и getNodeValue, доступных в объекте Node. Если не существует очевидного отображения этих атрибутов для какого-то конкретного типа Node (например, узел Element не имеет собственного значения), возвращается пустое значение. Все варианты подобных случаев приведены в таблице А.5.

ТаблицаА.5

Типы узлов и атрибуты

Тип узла	nodeName	**nodeValue**
Document	#document	Null
Element	имя тега	Null
Attr	Имя атрибута	Значение атрибута
Text	#text	Содержимое текстового узла
DocumentFragment	#document-fragment	Null
ProcessingInstruction	Целевое приложение	Все содержимое за исключением целевого приложения
DocumentType	Имя типа документа	Null
EntityReference	Имя сущности, на которую сделана ссылка	Null
Comment	#comment	Содержимое комментария
CDATA section	#cdata-section	Содержимое секции CDATA

Отметим, что специализированные интерфейсы могут содержать дополнительные и более удобные механизмы получения и установки релевантной информации.

Наконец, заметим, что благодаря DOM Level 2 в узлы были добавлены пространства имен. Это было сделано через атрибуты **namespaceURI, prefix и localName.**

Document

Любой документ **(Document)** может иметь дочерние элементы следующих типов: **Element, ProcessingInstruction, Comment и DocumentType,** при этом в **Document** может существовать только один дочерний элемент **Element** (корневой элемент) и один дочерний элемент **DocumentType** (DTD).

Element

Элементы, которые представляют собой теги XML-разметки, это самые распространенные объекты в XML-документах. Они составляют основную часть структуры XML-документа. Рассмотрим, например, следующий документ:

```
<booklist>
<book isbn="0-04-823188-6">
      <title>The Hobbit</title>
      <author>J.R.R. Tolkien</author>
      <publisher>Unwin Paperbacks</publisher>
      <price>5.79</price>
   </book>
</booklist>
```

Если представить этот документ с помощью модели DOM, он будет выглядеть следующим образом: узел **Document** содержит узел **Element,** который называется **book;** в свою очередь узел **Element** содержит четыре дочерних узла элементов — по одному для **title, author, publisher и price.** Каждый из этих элементов содержит один дочерний элемент **Text.** Все элементы могут иметь связанные с ними атрибуты. Например, здесь элемент **booklist** имеет один атрибут **isbn,** который представлен с помощью объекта **Attr.** Для возвращения значения атрибута (или представления объекта **Attr,** или его строкового значения) по его имени в интерфейсе **Element** существуют специальные методы.

Element может иметь дочерние элементы следующих типов: **Element, Text, Comment, ProcessingInstruction, CDATAsection и EntityReference.**

Кроме того, в DOM Level 2 были добавлены методы установки/получения/удаления пространств имен по их атрибутам.

Attr

Attr представляет собой атрибут в объекте **Element.** Его значение можно извлечь с помощью метода **getValue.** Это значение может быть либо явно заданным, либо

значением по умолчанию, определенным в DTD. Attr может иметь дочерние элементы двух типов: **Text** или **EntityReference.** Так как объекты Attr не являются в действительности дочерними узлами элемента, который они описывают, они и не считаются частью дерева DOM. Поэтому методы типа **getParentNode, getPreviousSibling** и **getNextSibling**, которые вполне ожидаемым образом работают с большинством других узлов, при вызове из Attr вернут пустое значение. Для навигации по атрибутам объекта **Element** нужно прежде всего вернуть все атрибуты элемента (как **NamedNodeMap**), а потом уже совершать их обход. >.

CharacterData

CharacterData расширяет Node целым набором атрибутов и методов для доступа к символьным данным в дереве DOM. Ни один из объектов DOM напрямую не соответствует **CharacterData.** Однако **Text, Comment и CDATASection** являются наследниками **CharacterData.**

Text

Text является наследником **CharacterData** и представляет собой текстовое содержимое **Element** или **Attr.** Если внутри содержимого элемента нет никакой разметки, т. е. в нем нет никаких других элементов, комментариев и т. п., то текст содержится в одном объекте **Text,** являясь его единственным дочерним элементом.

После синтаксического разбора XML-документа Oracle XML-анализаторы нормализуют весь текст в документе. То есть для каждого блока текста будет существовать только один узел **Text.** С помощью DOM можно создать смежный узел **Text,** в котором будет располагаться содержимое заданного элемента без разметки. Если затем распечатать этот документ и провести повторный синтаксический разбор, смежные узлы объединятся. При желании можно это сделать и вручную с помощью метода **normalize(),** доступного в родительском узле **Element.**

Узлы **Text** всегда являются листьями на дереве DOM, т. е. у них нет никаких дочерних элементов.

DocumentFragment

DocumentFragment — это облегченный или минимальный объект **Document. DocumentFragment** может представлять собой часть имеющегося дерева документа или быть новым фрагментом, который можно вставить в документ. Когда **DocumentFragment** вставляется в **Document** (или в любой другой узел **Node,** которому разрешено иметь дочерние элементы), то в **Node** вставляется дочерний по отношению к **DocumentFragment** элемент, но не сам **DocumentFragment.** Поэтому **DocumentFragment** может оказаться полезным при необходимости создать узлы, которые являются братьями по отношению друг к другу. **DocumentFragment** будет для этих

братьев родительским узлом, поэтому из интерфейса Node можно будет использовать стандартные методы, например **insertBefore и appendChild.**

DocumentFragment может иметь дочерние элементы следующих типов: **Element, ProcessingInstruction, Comment, Text, CDATASection и EntityReference.**

ProcessingInstruction

Объект **ProcessingInstruction** представляет собой инструкцию по обработке, которая используется в XML для сохранения в тексте документа информации для процессора, обрабатывающего этот документ.

DocumentType

Объект **DocumentType** предоставляет доступ к списку сущностей и нотаций, определенных для данного документа. **DocumentType** можно вернуть из **Document** с помощью метода **getDocType**. Если для данного **Document** нет никакого DTD, вернется пустое значение. В настоящее время спецификация DOM не позволяет редактировать узлы **DocumentType,** но если нужно изменить что-нибудь в DTD, это можно сделать только путем модификации его текста, после чего следует провести повторный синтаксический разбор XML-документа.

Notation

Notation — это описанная в DTD нотация. Нотация либо описывает по имени формат сущности, еще не подвергнутой синтаксическому разбору, либо используется для формального описания целевых приложений для инструкции по обработке **Processing Instruction.** Редактировать нотации с помощью DOM нельзя. У **Notation** не может быть никакого родительского элемента, и ее значение извлекается из **DocumentType.**

Entity

Entity представляет собой сущность, подвергнутую или не подвергнутую синтаксическому разбору, которая входит в состав XML-документа. Отметим, что Entity представляет собой модель самой сущности, а не ее описание. Редактировать узлы Entity с помощью DOM нельзя. Для того чтобы внести изменения в содержимое **Entity,** нужно в DOM-дереве каждый связанный узел **EntityReference** заменить клоном содержимого Entity, после чего произвести эти изменения в каждом таком клоне. Все потомки узла **Entity** доступны только для чтения. Важно также отметить, что узлы **Entity** не имеют никакого родительского узла и не являются частью DOM-дерева. Узлы **Entity** можно вернуть только из **DocumentType.** Сущности могут иметь дочерние элементы следующих типов: **Element, ProcessingInstruction, Comment, Text, CDATASection и EntityReference,**

EntityReference

EntityReference — это ссылка на сущность XML-документа. Анализаторы XML-синтаксиса от Oracle при построении дерева DOM распространяют ссылки на сущности и, следовательно, никогда не создают объект **EntityReference.** Однако создать и вставить **EntityReference** в дерево DOM можно. Как и в случае с узлом Entity, все потомки **EntityReference** доступны только для чтения и могут иметь такие же типы, что и дочерние элементы **Entity.**

Comment

Comment является наследником **CharacterData** и представляет собой содержимое комментариев в XML-документе, т. е. все символы, которые располагаются между знаками "<!" и "-->".

CDATASection

CDATASection используются для пропуска блоков текста, содержащих символы, которые, если этого не сделать, могут рассматриваться как разметка. Единственный разделитель, который распознается в **CDATASection,** — это строка **]]>,** которая заканчивает **CDATASection.** Элементы **CDATASection** не могут иметь вложений. Основная цель этого элемента — включить в документ XML-фрагменты, не удаляя при этом всех разделителей. Интерфейс **CDATASection** был получен из интерфейса **CharacterData** с использованием интерфейса **Text.** Отметим также, что смежные узлы **CDATASection** нельзя объединять с помощью метода нормализации интерфейса **Element.**

NodeList

NodeList представляет собой упорядоченную коллекцию узлов. Все дочерние узлы любого узла можно получить в виде списка **NodeList** с помощью метода **getChildNodes.** Причем в этом списке все узлы будут располагаться в том же порядке, в каком они представлены в исходном XML-документе.

NamedNodeMap

NamedNodeMap представляет собой коллекцию узлов, доступ к которым можно получить по имени. **NamedNodeMap** не является последователем NodeList, и в списках **NamedNodeMap** нет никакого определенного порядка. К объектам, расположенным в объекте, имеющем **NamedNodeMap,** можно также получить доступ по порядковому индексу, но это просто позволяет произвести удобную нумерацию содержимого **NamedNodeMap** и не предполагает упорядочивания узлов в дереве DOM. Здесь можно привести следующий пример: атрибуты XML-элемента не имеют никакого порядка, следовательно, метод **getAttributes** вернет **NamedNodeMap,** а не **NodeList.**

Спецификация SAX

Что такое SAX?

Simple API for XML (SAX) — это стандартный интерфейс для основанного на событиях разбора XML-синтаксиса. Анализатор синтаксиса, который поддерживает SAX, например Oracle XML Parser for Java, сообщает о событиях синтаксического разбора (таких, как начало и конец элемента) непосредственно в приложение через функции обратного вызова и обычно не строит внутреннее дерево. В приложении для обработки различных событий есть специальные программы обработки, которые действуют аналогично тому, как производится обработка событий в графическом пользовательском интерфейсе.

Благодаря простому низкоуровневому доступу к XML-документу, предоставляемому SAX API-интерфейсом, можно производить синтаксический разбор документов, размер которых намного превышает возможности доступной системной памяти. Можно также строить собственные структуры данных с помощью обработчиков событий обратного вызова. Это может оказаться полезным, если для приложения нужно представить XML-документы каким-то особым способом.

Рассмотрим старый знакомый пример с продажей книг:

```
<booklist>
...
   <book  isbn="1234-123456-1234">
      <title>C Programming Language</title>
      <author>Kernighan and Ritchie</author>
      <publisher>IEEE</publisher>
      <price>7.99</price>
   </book>
...
   <book isbn="1230-23498-2349879">
      <title>Emperor's New Mind</title>
      <author>Roger Penrose</author>
      <publisher>Oxford Publishing Company</publisher>
      <price>15.99</price>
   </book>
...
</booklist>
```

Предположим, что нужно определить местоположение книги *Emperor's New Mind,* чтобы найти ее цену. Так как количество продаваемых книг может быть довольно большим (и размер документа может составлять несколько мегабайт), построение в памяти и обход дерева синтаксического разбора для поиска одного фрагмента

контекстной информации будет неэффективным. В такой ситуации найти книгу за один проход, используя для этого лишь малую долю системной памяти, можно с помощью API-интерфейса SAX. Для этого нужно разбить структуру документа на серии линейных событий:

```
start document
start element: booklist
...
start element: book
start element: title
character: Emperor's New Mind
end element: title
start element: author
characters: Roger Penrose
end element: author
start element: publisher
characters: Oxford Publishing Company
end element: publisher
start element: price
characters: 15.99
end element: price
end element: book
...
end element: booklist
end document
```

Затем можно написать простое SAX-приложение для обработки этих событий. Таким образом, не отправляя весь документ для кэширования в памяти или на каком-то вторичном устройстве хранения данных, можно быстро просматривать документ, пока не будет найдено то, что нужно. При таком алгоритме игнорируются все книги, названием которых не является *Emperor's New Mind*. А как только будет получен список книг с такими названиями, можно приступить к обработке событий для определения цены книги.

Интерфейсы и классы SAX

SAX API-интерфейс состоит из совокупности интерфейсов и классов. Некоторые из этих интерфейсов реализованы SAX-анализатором синтаксиса (например, Oracle XML Parser for Java). Другие интерфейсы придется реализовывать или расширять с помощью приложения. Кроме того, в интерфейсах и методах SAX Level 2 есть поддержка пространства имен, фильтров и некоторых других функций. Следовательно,

благодаря поддержке пространства имен некоторые из интерфейсов можно заменить новыми. Интерфейсы и классы SAX можно классифицировать следующим образом.

Интерфейсы, реализованные SAX-анализатором синтаксиса

Parser Это основной интерфейс для SAX-анализатора синтаксиса. Он позволяет регистрировать программы-обработчики для обратных вызовов, устанавливать место для сообщений об ошибках и запускать синтаксический анализ XML-документа. В SAX Level 2 его заменили интерфейсом **XMLReader**.

AttributeList Этот интерфейс позволяет повторно просматривать список атрибутов. В SAX Level 2 его заменили интерфейсом Attributes.

Locator С помощью этого интерфейса можно находить текущее местоположение в исходном XML-документе.

Интерфейсы, реализованные приложениями

DocumentHandler Это самый распространенный интерфейс, причем очень часто **DocumentHandler** является единственным интерфейсом, который нужно реализовать. В SAX Level 2 его заменили интерфейсом **ContentHandler.** Если в приложении есть реализация этого интерфейса, приложение будет получать уведомление об основных событиях, связанных с документом, в том числе о начале и конце какого-нибудь элемента.

ErrorHandler Если в приложении нужно использовать специальную программу обработки ошибок, требуется именно этот интерфейс.

DTDHandler Если приложению приходится работать с нотациями и синтаксически не разобранными (бинарными) сущностями, оно должно реализовать этот интерфейс, чтобы получать уведомления об описаниях **NOTATION** и **ENTITY в** XML-документе.

EntityResolver Если приложению нужно в документах выполнять перенаправление URI-идентификаторов (или производить какую-либо другую специфическую обработку данных), в нем должен быть интерфейс **EntityResolver.**

Стандартные SAX-классы

InputSource Этот класс содержит всю необходимую информацию для одного входного источника, в том числе его открытый идентификатор, системный идентификатор, байтовый поток и символьный поток. Приложение должно назначать как минимум один **InputSource** для анализатора синтаксиса, а **EntityResolver** может назначать остальные.

SAXException Этот класс представляет собой общую исключительную ситуацию для SAX.

SAXParseException Этот класс представляет собой исключительную ситуацию для SAX, привязанную к какой-то конкретной точке исходного XML-документа.

HandlerBase Данный класс выдает реализации по умолчанию для интерфейсов DocumentHandler, ErrorHandler, DTDHandler и EntityResolver. В SAX Level 2 этот интерфейс заменили интерфейсом DefaultHandler. Приложение может классифицировать его как подкласс, чтобы упростить написание программы-обработчика.

Дополнительные вспомогательные классы, специфические для языка Java

ParserFactory Приложение может использовать в этом классе статические методы для динамической загрузки в реальном времени SAX-анализаторов синтаксиса.

AttributeListImpl Приложение может использовать этот класс для создания постоянной копии интерфейса AttributeList. В SAX Level 2 этот интерфейс заменили интерфейсом AttributesImpl.

LocatorImpl Этот класс приложение может использовать для создания постоянной копии значений интерфейса Locator в какой-то заданной точке при синтаксическом разборе.

Спецификация пространства имен XML

Что такое пространство имен XML?

Пространство имен XML — это совокупность имен, идентифицированных URI-ссылкой, которые используются в XML-документах как типы элементов и имена атрибутов. Пространство имен удобно использовать, если один XML-документ содержит элементы и атрибуты, которые определены и используются в нескольких программных модулях.

Пространство имен вводится в XML-документе путем определения специального префикса пространства имен. Этот префикс ставится перед именами элементов и атрибутов и отделяется от них двоеточием. Префиксы пространства имен — это атрибуты, которые сами определяются с помощью специального префикса xmlns. Например, рассмотрим следующий XML-документ:

```
<doc>
<title>Book List</title>
<author>Oracle XML Team</author>
```

```
<booklist>
   <book isbn="1234-5678-1234">
      <title>Oracle XML Handbook</title>
      <author>Oracle XML Team</author>
   </book>
   <book isbn="24345-564478-1344234">
      <title> The C programming language</title>
      <author>Kernighan and Ritchie</author>
   </book>
</booklist>
</doc>
```

В этом документе имена **title и author** имеют два набора тегов. Очевидно, что один набор относится к самому документу, а другой — к элементу **booklist**. Таким образом, приложение при работе с этим документом должно, в зависимости от тега, выполнять разные действия. Чтобы упростить эту дифференциацию, можно использовать пространство имен. Если ввести в документ пространства имен, он будет выглядеть так:

```
<doc xmlns:document="http://www.osborne.com/document"
     xmlns:books="http://www.osborne.com/books">
<document:title>Book List</document:title>
<document:author>Oracle XML Team</document:author>
<books:booklist>
   <books:book books:isbn="1234-5678-1234">
      <books:title>Oracle XML Handbook</books:title>
      <books:author>Oracle XML Team</books:author>
   </books:book>
   <books:book books:isbn="24345-564478-1344234">
      <books:title>C programming language</books:title>
      <books:author>Kernighan and Ritchie</books:author>
   </books:book>
</books:booklist>
</doc>
```

В этом документе введены два пространства имен, имеющие префиксы **document и books.** Они четко разделены тегами и приложение, знакомое с обозначениями www.osborne.com, легко обработает их должным образом.

У пространств имен есть и другие достоинства, а именно: модульность и возможность повторного использования. Предположим, есть приложение для обработки списков книг и нужно произвести обработку так, как это делается на сайте www.osborne.com. Можно просто использовать в приложении имеющийся на сайте программный модуль, который знает, как обрабатывать разметку (элементы и атрибуты) в этом пространстве имен.

Терминология пространства имен

В спецификации пространств имен XML вводятся следующие термины: *Local Name* (локальное имя), *Qualified Name* (составное имя), *Namespace Prefix* (префикс пространства имен) и *Expanded Name* (расширенное имя).

Локальное имя

Представляет собой имя элемента или атрибута без префикса. В предыдущем примере локальными именами были: **book, title, author, isbn и т. д.**

Составное имя

Представляет собой полное имя вместе с префиксом. В предыдущем примере встречаются следующие составные имена: document:title, **books:book и др.**

Префикс пространства имен

Это префикс, описанный с помощью специального префикса **xmlns**. В рассмотренном выше примере есть два префикса пространства имен: **document и books.**

Расширенное имя

Это результат применения пространства имен, определенного префиксом пространства имен к составному имени. Например, **books:booklist** можно расширить до http://www.osborne.com/books:booklist. Расширенного имени в самих XML-документах не бывает, но это очень важное концептуальное понятие.

Атрибуты пространства имен

Существует два вида атрибутов пространства имен: префиксные и по умолчанию. Префиксный атрибут пространства имен имеет вид **nsprefix:attr,** где **nsprefix** — это префикс пространства имен, определенный выше. После описания префикса его можно использовать для определения пространства имен для любых элементов и атрибутов в рамках того элемента, в котором он был описан. Поэтому глобальные префиксы, т. е. используемые на протяжении всего документа, нужно описывать как атрибуты корневого элемента.

Атрибут пространства имен по умолчанию имеет вид **xmlns.** Он определяет пространство имен по умолчанию на протяжении всего элемента (включая сам элемент). Однако этот атрибут по умолчанию не применяется к атрибутам в поддереве. Снова обратимся к примеру с каталогом книг:

```
<booklist xmlns="http://www.osborne.com/books">
   <book isbn="1234-5678-1234">
      <title>Oracle XML Handbook</title>
      <author>Oracle XML Team</author>
   </book>
```

```
    <book isbn="24345-564478-1344234">
      <title> The C programming language</title>
      <author>Kernighan and Ritchie</author>
    </book>
  </booklist>
```

В этом примере показано, что все элементы, расположенные ниже booklist (book, title, author), находятся в пространстве имен http://www.osborne.com/books, а атрибут isbn — нет. Пространства имен по умолчанию можно определить на любом уровне документа, и они будут действовать поверх предыдущих описаний. Таким образом, установив xmlns="", мы удалим описание пространства имен по умолчанию для конкретного поддерева документа.

Пространства имен усложняют определение уникальности атрибутов. Это можно увидеть на примере:

```
<booklist xmlns:dollars="USA" xmlns:pounds="Britain">
  <book dollars:price="7.99" pounds:price="3.99">
    <title>The Code of Woosters</title>
    <author>P.G. Wodehouse</author>
  </book>
</booklist>
```

Здесь два атрибута **price** нужно рассматривать отдельно, несмотря на то, что они имеют одно локальное имя, ведь расширенные имена у них разные. А вот следующий документ составлен неправильно:

```
<booklist xmlns:dollars="USA" xmlns:currency="USA">
  <book dollars:price="7.99" currency:price="3.99">
    <title>The Code of Woosters</title>
    <author>P.G. Wodehouse </author>
  </book>
</booklist>
```

Несмотря на то, что здесь **dollars:price и currency:price** имеют разные составные имена, расширенное имя у них одно и то же, следовательно, они являются одним и тем же атрибутом, который дважды описан в элементе book.

Спецификация XPath

Что такое XPath?

Язык *XML Path (XPath)* определяет способ адресации частей XML-документа и некоторые базовые функции для манипуляции строками, числами и логическими переменными. Этот язык использует и процессор XSLT, и язык указателей XPointer. XPath моделирует XML-документ в виде дерева узлов. Эти узлы бывают следующих

типов: корневые узлы, узлы элементов, текстовые узлы, узлы атрибутов, узлы пространства имен, узлы инструкций по обработке и узлы комментариев. XPath также определяет способ вычисления строкового значения для каждого типа узлов. Для некоторых типов узлов это строковое значение является частью узла, а для других типов оно вычисляется из строкового значения узлов-потомков. Для узлов, имеющих имя, например узлы элементов, XPath моделирует расширенное имя, т. е. пару имя-значение, состоящую из локального имени и значения пространства имен.

Выражения языка XPath

Результатом вычисления выражения XPath является объект одного из следующих типов: совокупность узлов *node-set* (неупорядоченная совокупность неповторяющихся узлов), *логический* (принимает значения истина (true) или ложь (false)), *числовой* (число с плавающей точкой) или *строковый* (строка, т. е. последовательность UCS-символов). Вычисление выражение производится с учетом контекста. Как определяется контекст для выражений XPath, которые потом используются в XSLT и XPointer, устанавливается соответственно в XSLT и XPointer.

Совокупность узлов node-set

Совокупность узлов node-set обычно порождается выражением для пути, определяющего местоположение. Это выражение может фильтроваться с помощью условия логического выражения (предиката). Выражения для пути, определяющего местоположение, очень важны, так как представляют собой мощное средство выбора узлов из дерева XML-документа. Путь, определяющий местоположение, может определяться как относительный путь (из текущего контекстного узла) или как абсолютный путь (из корня дерева). Для определения соотношений между узлами дерева, отобранными на разных шагах пути, определяющего местоположение, используются специальные осевые операторы (axis operators).

Логический тип

Логические объекты могут иметь только два значения: истина (true) или ложь (false). Эти значения можно получить путем применения логической функции к другому объекту или путем вычисления выражений И, ИЛИ либо РАВНО.

Числовой тип

Числовой результат вычисления выражения XPath имеет вид числа с плавающей точкой. Причем это число может иметь 64-разрядный формат с удвоенной точностью стандарта IEEE 754. К этому же типу относятся и специальные нечисловые значения — плюс или минус бесконечность и плюс или минус ноль. Числовой объект можно получить, вызвав числовую функцию или вычислив числовое выражение, т. е. выражение, содержащее один из числовых операторов, например +, -, *, / и т. д.

Строковый тип

Строки состоят из последовательности символов, количество которых может равняться нулю или быть больше нуля. Один символ в языке XPath соответствует одному абстрактному символу в кодировке Unicode с одним скалярным значением, соответствующим этому символу Unicode. Строковый объект обычно получается при вызове строковой функции, которая действует на объект.

Функции

В языке XPath есть обширная библиотека функций, которую должны поддерживать XPath-процессоры. Эти функции можно разбить на четыре группы: функции совокупности узлов, строковые, логические и числовые функции.

Функции совокупности узлов

Операции над совокупностями узлов выполняют следующие функции:

last Возвращает из выражения вычисления контекста число, равное размеру контекста.

position Возвращает из выражения вычисления контекста число, равное местоположению контекста.

count Берет аргумент совокупности узлов и возвращает число, равное количеству узлов в этой совокупности.

id Выбирает элементы по их уникальным идентификаторам ID, т. е. значение атрибута, описанного в DTD как имеющий тип ID.

local-name Принимает аргумент совокупности узлов и возвращает локальную часть расширенного имени того узла из этой совокупности, который в документе встречается первым.

namespace-uri Принимает аргумент совокупности узлов и возвращает URI-идентификатор пространства имен расширенного имени того узла из совокупности, который в документе встречается первым.

name Принимает аргумент совокупности узлов и возвращает строку, содержащую расширенное имя того узла из этой совокупности, который в документе встречается первым.

Строковые функции

string Принимает аргумент объекта и преобразует его в строку в соответствии с разными правилами.

concat Возвращает конкатенацию ее аргументов.

starts-with Принимает строки двух аргументов и возвращает значение истина, если строка первого аргумента начинается со строки второго аргумента, во всех других случаях возвращается значение ложь.

contains Принимает строки двух аргументов и возвращает значение истина, если строка первого аргумента содержит строку второго аргумента, во всех других случаях возвращается значение ложь.

substring-before Принимает строки двух аргументов и возвращает подстроку строки первого аргумента, которая предшествует первому появлению строки второго аргумента в строке первого аргумента. Если же строка первого аргумента не содержит строки второго аргумента, возвращается пустая строка.

substring-after Принимает строки двух аргументов и возвращает подстроку строки первого аргумента, которая находится после первого появления строки второго аргумента в строке первого аргумента. Если же строка первого аргумента не содержит строки второго аргумента, возвращается пустая строка.

substring Принимает строковый аргумент, числовой индекс местоположения и длину и возвращает подстроку первого аргумента, которая начинается с позиции, определяемой во втором аргументе, и имеет длину, определяемую в третьем аргументе. Если третий аргумент не определен, возвращается подстрока, которая начинается с позиции, определенной во втором аргументе, и продолжается до конца строки.

Внимание

В отличив от многих других языков программирования, например C и Java, значение положения первого символа здесь равно 1, а не 0.

string-length Принимает строку аргумента и возвращает количество символов в этой строке. Если аргумент пропущен, то по умолчанию контекстный узел преобразуется в строку, другими словами, в строковое значение контекстного узла.

normalize-space Принимает строку аргумента и возвращает ее, удалив первый и последний пробелы, и заменив последовательности пробелов на одиночные пробелы. Если аргумент пропущен, по умолчанию контекстный узел преобразуется в строку, т. е. в строковое значение контекстного узла.

translate Принимает строки трех аргументов и возвращает строку первого аргумента, в которой символы, встречающиеся в строке второго аргумента, заменены символами, стоящими на тех же местах в строке третьего аргумента.

Логические функции

boolean Принимает аргумент объекта и преобразует его в логическое значение в соответствии с разными правилами.

not Возвращает значение истина, если аргумент имеет значение ложь; если аргумент имеет значение истина, возвращается значение ложь.

true Всегда возвращает значение истина.

false Всегда возвращает значение ложь.

lang Принимает строку аргумента и возвращает значение истина или ложь в зависимости от того, совпадает ли язык контекстного узла, определенный в атрибутах xml:lang, с языком, определенным в строке аргумента, или он является подъязыком языка, определенного в строке аргумента.

Числовые функции

number Преобразует аргумент своего объекта в число в соответствии с разными правилами.

sum Принимает аргумент совокупности узлов и возвращает для каждого узла из совокупности узлов аргумента сумму результатов преобразования строковых значений узла в число.

floor Принимает числовой аргумент и возвращает максимально возможное целое число, значение которого не превышает значение аргумента.

ceiling Принимает числовой аргумент и возвращает наименьшее целое число, значение которого не меньше, чем значение аргумента.

round Принимает числовой аргумент и возвращает целое число, максимально близкое по значению к этому аргументу.

Узлы XPath

XPath создает модель XML-документа в виде дерева узлов разных типов. Эти узлы XPath располагает в том же порядке, в каком они расположены в документе. Согласно этому порядку первым всегда будет корневой узел. Узлы элементов располагаются в порядке появления в XML-документе их открывающих тегов (после расширения сущностей). Узел элемента, его атрибут и узлы пространства имен по определению располагаются перед его дочерним элементом. Пространство имен и узлы атрибута по определению располагаются перед узлами атрибута.

Корневые узлы и узлы элементов имеют упорядоченный список дочерних узлов. Каждый узел, отличный от корневого, имеет только один родительский элемент, который одновременно является либо узлом элемента, либо корневым узлом. Корневой узел или узел элемента является родительским для каждого из своих дочерних узлов. Узлы-потомки любого узла являются дочерними для этого узла или потомками для дочерних узлов данного узла.

Корневой узел

Корневой узел — это корень дерева. Корневой узел существует только в корне дерева. *Узел элемента* для элемента документа является дочерним для корневого узла. Корневой узел также имеет дочерние узлы инструкций по обработке и узлы комментариев для инструкций по обработке, а также комментарии, которые располагаются и в прологе, и после окончания дочернего элемента. Строковое значение корневого узла представляет собой конкатенацию (объединение) строковых значений всех текстовых узлов-потомков корневого узла в том порядке, в каком они встречаются в документе.

Узлы элементов

Узел элемента есть у каждого элемента в документе. Дочерними для узла элемента являются узлы элементов, узлы комментариев, узлы инструкций по обработке и текстовые узлы для его содержимого. Строковое значение узла элемента представляет собой конкатенацию (объединение) строковых значений всех текстовых узлов-потомков этого узла в том порядке, в каком они встречаются в документе.

Узлы атрибутов

Каждый узел элемента имеет связанный с ним набор *узлов атрибутов.* Элемент является родительским для каждого из этих узлов атрибута, однако узел атрибута не является дочерним для своего родительского элемента. Отметим, что в этом XPath отличается от DOM, в котором элемент не является родителем по отношению к своему атрибуту. Узел атрибута имеет строковое значение, которое представляет собой нормированное значение атрибута.

Узлы пространства имен

Каждый элемент имеет связанную с ним совокупность *узлов пространства имен,* по одному узлу на каждый отдельный префикс пространства имен в рамках данного элемента, включая заранее заданный префикс xml и один префикс для пространства имен по умолчанию, если таковое есть в данном элементе. Элемент является родительским для каждого из этих узлов пространства имен, но узел пространства имен не является дочерним по отношению к своему родительскому элементу. Узел пространства имен имеет строковое значение, которое представляет собой URI-идентификатор пространства имен, связанный с префиксом этого пространства имен.

Узлы инструкций по обработке

Узел инструкции по обработке существует для каждой инструкции по обработке, за исключением тех инструкций, которые находятся в определении типа документа (DTD). Строковое значение узла инструкции по обработке является частью инструкции по обработке, которая следует за целью и любым пробелом. Это строковое значение не содержит завершающих значков "?>".

Узлы комментариев

Узел комментариев существует для каждого комментария, за исключением тех комментариев, которые находятся в определении типа документа (DTD). Строковое значение узла комментария — это содержимое данного комментария без открывающих и закрывающих значков "<!--" и "-->".

Текстовые узлы

Символьные данные группируются в текстовые узлы неким нормализованным способом. Строковое значение текстового узла — это символьные данные. Текстовый узел имеет хотя бы один символ в данных. Если текстовый узел, который содержит символ "<", переписать в XML-формате, то символ "<" нужно удалить, например, с помощью **&lt** или включить его в секцию CDATA. Комментарии с символами внутри, инструкции по обработке и значения атрибутов текстовых узлов не образуют. '

Спецификация XSLT

Что такое XSLT?

Язык *Extensible Stylesheet Language Transformation (XSLT)* используется для описания правил преобразования исходного древовидного представления документа в результирующее древовидное представление. Это преобразование выполняется с помощью соответствующих схем с использованием шаблонов, содержащихся в таблице стилей. XSLT-процессор ставит в соответствие элементам исходного дерева определенные схемы и применяет к ним соответствующие шаблоны для создания различных частей результирующего дерева. При конструировании результирующего древовидного представления документа процессор может отфильтровывать элементы исходного дерева, изменять их порядок и добавлять произвольную структуру — все это задается в XSLT-преобразовании. Результирующее древовидное представление документа располагается отдельно от исходного, т. е. ни один узел не принадлежит одновременно обоим деревьям. Структуры исходного и результирующего деревьев могут быть совершенно не похожими друг на друга.

XSLT-таблицы стилей — это правильные XML-документы, содержащие элементы и атрибуты в пространстве имен XSLT (**http://www.w3.org/1999/XSLT/Transform**). Эти элементы и атрибуты используются для выдачи инструкций XSLT-процессору

относительно того преобразования, которое необходимо произвести. Таблица стилей может также содержать элементы и атрибуты, находящиеся за пределами пространства имен XSLT. XSLT-процессоры обычно интерпретируют их как разметку, которая добавляется непосредственно к результирующему древовидному представлению. Исключением являются только элементы расширения, которые выполняют функции инструкций для XSLT-процессора и находятся в отдельном пространстве имен, определяемом с помощью специального механизма.

XSLT-инструкции находятся в шаблонах. Результирующее древовидное представление документа строится путем поиска шаблонного правила для корневого узла и назначения его шаблона. После назначения шаблона каждая инструкция в нем исполняется и заменяется созданным фрагментом результирующего дерева. Эти инструкции способны выбирать и обрабатывать элементы-потомки исходного дерева, которые могут вызывать назначение других шаблонов. Важно отметить, что элементы обрабатываются только после того, как они были отобраны в процессе выполнения инструкции. Для отбора элементов из исходного дерева и выполнения их условной обработки XSLT использует язык XPath.

XSLT-процессор назначает шаблоны в соответствии с правилами, соответствующими определенной структуре. В процессе поиска подходящего шаблонного правила может оказаться, что данному элементу соответствует структура, которую имеют несколько шаблонных правил. Тогда процессор использует специальную логику разрешения конфликтов (в которой используются приоритеты шаблонов), чтобы гарантировать применение самого лучшего шаблонного правила.

Шаблоны

Шаблоны можно сравнить с процедурами в структурном языке программирования. Они содержат программную логику, выраженную в терминах XSLT-инструкций, которые обрабатывают исходное древовидное представление и изменяют результирующее дерево. В некоторых случаях для преобразования исходного дерева в результирующее может использоваться один-единственный шаблон. В языке XSLT есть особый механизм, который называется *литеральный результирующий элемент как таблица стилей* (Literal Result Element as Stylesheet), позволяющий встраивать всю логику XSLT-обработки в тело результирующего документа.

Шаблоны могут иметь явное имя или содержать выражение для установки соответствия. Если XSLT-процессору встречается инструкция **xsl:apply-templates**, он пытается подобрать узлы, которые были выбраны с помощью наиболее подходящего шаблона (с помощью выражения для установки соответствия). Иногда преобразованию соответствует несколько шаблонов. Для разрешения этого конфликта в XSLT-процессоре есть сложный набор правил, по которым различным выражениям для установки соответствия назначаются разные приоритеты. Кроме того, шаблон сам может задавать явный приоритет, и XSLT-процессор рассмотрит его при определении нужного для преобразования шаблона.

Инструкции языка XSLT

Язык XSLT имеет полностью проработанный набор инструкций, который позволяет использовать его почти как самостоятельный язык программирования. В нем есть условные операторы, переменные, параметры и циклы, все, что свойственно любому языку программирования. Конечно, XSLT больше подходит для преобразований, чем для написания программ общего назначения, но богатый набор инструкций делает его действительно мощным средством.

Далее приведен список инструкций, которые можно использовать в языке XSLT, с синтаксисом и кратким описанием того, что они делают.

<xsl:apply-imports/>

Эта инструкция, которая может встретиться при обработке узла с помощью шаблона, предписывает процессору применить в данном узле шаблонные правила, импортированные в таблицу стилей. Инструкция не имеет ни атрибутов, ни содержимого.

<xsl:apply-templates/>

```
<xsl:apply-templates select = node-set-expression mode, = qname>
        <!-- Content: (xsl:sort I xsl:with-param) * -->
</xsl:apply-templates>
```

Эта инструкция предписывает процессору назначить шаблон, соответствующий выражению XPath, которое задано значением атрибута **select.** Если атрибут **select** отсутствует, то инструкция обрабатывает все дочерние узлы текущего узла, включая и текстовые узлы. Для выбора способа, каким будет определен шаблон при возникновении конфликта, можно использовать необязательный атрибут **mode.** Эту инструкцию также можно использовать для передачи параметров в назначаемый шаблон (с помощью **xsl:with-param)** и для изменения порядка обработки выбранных узлов (это определяется с помощью **xsl:sort**).

<xsl:attribute>

```
<xsl:attribute name = { qname } namespace = { uri-reference }>
        <!-- Content: template -->
</xsl:attribute>
```

Эта инструкция предписывает процессору создать новый атрибут для элементов, созданных в результирующем древовидном представлении документа. В данной инструкции можно также установить имя и пространство имен. В результате выполнения этой инструкции выдается значение созданного атрибута. Следует отметить, что при обработке содержимого данной инструкции будет создан только текстовый узел (так как значение атрибута может состоять только из символов).

<xsl:attribute-set>

```
<xsl:attribute-set name = qname use-attribute-sets = qnames>
        <!-- Content: attribute* -->
</xsl:attribute-set>
```

Эту инструкцию можно использовать для определения именованной совокупности атрибутов. Ее содержимое состоит из набора инструкций **xsl:attribute**, которые устанавливают атрибуты. Атрибут **use-attribute-sets** можно использовать для определения разделенного пробелами списка других совокупностей атрибутов **attribute-set.** Этот список нужен для того, чтобы добавить инструкции **xshattribute,** содержащие упомянутые совокупности атрибутов, в начало содержимого текущего набора атрибутов.

<xsl:call-template>

```
<xsl:call-template name = qname>
        <!-- Content: xsl:with-param* -->
</xsl:call-template>
```

Эта инструкция используется для вызова именованных шаблонов и для передачи им параметров с помощью оператора **xsl:with-param**.

<xsl:choose>

```
<xsl:choose>
        <!-- Content: (xsl:when+, xsl:otherwise?) -->
</xsl:choose>
```

Эта инструкция используется для выбора одного из нескольких вариантов. Она состоит из последовательности элементов **xshwhen, за** которыми может следовать необязательный элемент **xsl:otherwise**. Каждый элемент **xshwhen** имеет один атрибут (критерий), который задает XPath-выражение. Содержимое элементов **xshwhen и xshotherwise** — это шаблон. При обработке элемента **xsl:choose** происходит поочередная проверка всех элементов **xshwhen.** Это делается путем вычисления выражения и преобразования результирующего объекта к логическому виду, как если бы это происходило с помощью вызова функции boolean из языка XPath. Результатом работы этой инструкции будет содержимое первого по порядку элемента **xshwhen,** значение которого оказалось равным истине. Если таких элементов xshwhen не нашлось, результатом будет содержимое элемента **xshotherwise.** Если же ни один из элементов **xshwhen** не является истиной, а элемент **xshotherwise** отсутствует, то в результате работы этой инструкции ничего не создается.

<xsl:comment>

```
<xsl:comment>
        <!-- Content: template -->
</xsl:comment>
```

Эта инструкция создает новый комментарий в результирующем древовидном представлении.

<xsl:copy>

```
<xsl:copy use-attribute-sets = qnames>
        <!-- Content: template -->
</xsl:copy>
```

Эта инструкция позволяет легко скопировать текущий узел. Она создает копию текущего узла, но без его атрибутов и дочерних узлов. Если текущий узел является элементом, то для создания атрибутов узла-копии (в результирующем дереве) может использоваться значение, содержащееся в атрибуте **use-attribute-sets.** Содержимое этой инструкции является шаблоном для атрибутов и дочерних узлов созданного узла.

<xsl:copy-of>

```
<xsl:copy-of select = expression />
```

Эта инструкция позволяет копировать совокупность узлов, получающихся при вычислении XPath-выражения, заданного выбранным атрибутом для результирующего дерева. Если результат этого выражения не является совокупностью узлов, он преобразуется в строку и копируется как текстовый узел.

<xsl:decimal-format>

```
<xsl:decimal-format
    name = qname decimal-separator = char grouping-separator = char
    infinity = string minus-sign = char NaN = string percent = char
    per-mille = char zero-digit = char digit = char
    pattern-separator = char />
```

Эта инструкция описывает десятичный формат, который управляет интерпретацией шаблона форматирования, используемого функцией format-number. Если имя атрибута существует, то описывается именованный десятичный формат, в противном случае описывается десятичный формат по умолчанию. Его атрибуты соответствуют методам класса DecimalFormatSymbols из пакета разработчика JDK 1.1. Атрибут определяется для каждой пары методов get/set.

<xsl:element>

```
<xsl:element name = {qname}
             namespace = {uri-reference}
             use-attribute-sets = qnames>
          <!-- Content: template -->
</xsl:element>
```

Эта инструкция создает новый элемент с помощью заданных имени тега и пространства имен в результирующем дереве. Для определения атрибутов в новом элементе можно использовать атрибут use-attribute-sets. Атрибуты и дочерние элементы вновь созданного элемента определяются содержимым этой инструкции.

<xsl:fallback>

```
<xsl:fallback>
          <!-- Content: template -->
</xsl:fallback>
```

Эта инструкция исполняется, если XSLT-процессор выполняет нейтрализацию неисправности (fallback) для какого-то элемента инструкции. Если элемент инструкции имеет один или несколько дочерних элементов **xsl:fallback**, то содержимое каждого из дочерних элементов xshfallback назначается последовательно.

<xsl:for-each>

```
<xsl:for-each select = node-set-expression>
          <!-- Content: (xsl:sort*, template) -->
</xsl:for-each>
```

Содержимое этой инструкции является шаблоном, который берет приписанное каждому узлу значение, выбранное XPath-выражением, заданным атрибутом выбора. Порядок обработки этих узлов можно изменить с помощью необязательной инструкции **xsl:sort**.

<xsl:if>

```
<xsl:if test = boolean-expression>
          <!-- Content: template -->
</xsl:if>
```

Эта инструкция имеет атрибут test, который задает выражение. Заданное выражение вычисляется, а результирующий объект преобразуется в логическую форму, как если бы это было выполнено путем вызова функции boolean. Если результат получается истинным, то назначается шаблон содержимого, в противном случае ничего не создается.

<xsl:import>

```
<xsl:import href = uri-reference />
```

Эта инструкция используется для импорта другой таблицы стилей (заданной значением атрибута **href).** Результатом выполнения этой инструкции является то, что шаблоны и определения, находящиеся в импортированной таблице стилей, становятся видимыми в импортирующей таблице стилей, но здесь они будут иметь более низкий приоритет.

<xsl:include>

```
<xsl:include href = uri-reference />
```

Эта инструкция используется для включения другой таблицы стилей (заданной значением атрибута **href**). Результатом выполнения этой инструкции является то, что шаблоны и определения, находящиеся во включенной таблице стилей, становятся видимыми в той таблице стилей, в которую их включили, и здесь они будут рассматриваться, как если бы были определены именно в этой таблице.

<xsl:key>

```
<xsl:key name = qname match = pattern use = expression />
```

Эта инструкция используется для описания ключей. Атрибут **name** дает имя ключу, а атрибут use — значение. Эта инструкция выдает информацию о ключах любого узла, который соответствует структуре pattern, заданной в атрибуте **match.**

<xsl:message>

```
<xsl:message terminate = "yes* I "no">
            <!-- Content: template -->
</xsl:message>
```

Эта инструкция посылает сообщение способом, который определяется процессором. Например, анализатор синтаксиса Oracle XML Parser for Java просто печатает сообщение на стандартном устройстве вывода. Для возвращения значения этого сообщения назначается содержимое данной инструкции. Если атрибут **terminate** имеет значение "yes", то XSLT-процессор заканчивает обработку после отправки сообщения. Значением по умолчанию является "no".

<xsl:namespace>

```
<xsl:namespace-alias stylesheet-prefix = prefix I "#default"
                     result-prefix = prefix I "#default" />
```

Данная инструкция описывает тот факт, что пространство имен URI, связанное с префиксом, заданным атрибутом **stylesheet-prefix**, представляет собой псевдоним пространства имен URI, связанного с префиксов, заданным атрибутом **result-prefix.** Атрибут **stylesheet-prefix** задает пространство имен URI, которое фигурирует в этой таблице стилей, а атрибут **result-prefix** задает соответствующее пространство имен URI, которое имеет место в результирующем древовидном представлении документа.

<xsl:number>

```
<xsl:number level = "single" I "multiple" I "any"
            count = pattern from = pattern
            value = number-expression
            format = {string } lang = { nmtoken }
            letter = value = {"alphabetic" I "traditional" }
            grouping-separator = { char }
            grouping-size = { number } />
```

Эта инструкция используется для вставки сформатированного числа в результирующее древовидное представление. Значение этого числа задается выражением, заключенным в атрибуте **value.** Это выражение вычисляется, и результирующий объект преобразуется в число, как если бы все это было выполнено путем обращения к функции **number.** Это число округляется до целого и преобразуется в строку с помощью набора заранее заданных атрибутов. После преобразования результирующая строка вставляется в результирующее древовидное представление.

<xsl:output>

```
<xsl:output method = "xml" I "html" I "text" I qname-but-not-ncname
            version = nmtoken
            encoding = string
            omit-xml-declaration = "yes" I "no"
            standalone = "yes" I "no"
            doctype-public = string
            doctype-system = string
            cdata-section-elements = qnames
            indent = "yes" I "no"
            media-type = string />
```

Эта инструкция управляет выводом XSLT-процессора, когда на него приходит запрос о выводе результата в виде последовательности байтов, а не в виде дерева. Специфические аспекты этого вывода управляются следующими атрибутами:

- **Method** Задает общий метод, который следует использовать для вывода результирующего дерева.
- **Version** Задает версию метода вывода.

- **Indent** Определяет, может ли XSLT-процессор добавить дополнительный пробел при выводе результирующего дерева.
- **Encoding** Определяет, какую предпочтительную кодировку символов следует использовать XSLT-процессору для кодирования последовательности символов как последовательности байтов.
- **Media-type** Задает тип содержимого (в стандарте MIME) данных, которые получаются при выводе результирующего дерева,
- **Doctype-system** Задает идентификатор системы, который будет использоваться в определении типа документа.
- **Doctype-public** Задает открытый идентификатор, который будет использоваться в определении типа документа.
- **Omit-xml-declaration** Определяет, следует ли XSLT-процессору выводить XML-описание.
- **Standalone** Определяет, следует ли XSLT-процессору выводить описание автономного документа.
- **Cdata-section-elements** Задает список имен элементов, чьи дочерние текстовые узлы следует выводить с использованием секций CDATA.

<xsl:param>

```
<xsl:param name = qname select = expression>
        <!-- Content: template -->
</xsl:param>
```

Эта инструкция описывает параметр для шаблона или таблицы стилей. Его имя задается атрибутом **name.** Значение по умолчанию для этого параметра можно задать, используя либо атрибут select, либо его содержимое. Это значение можно заменить при вызове шаблона или таблицы стилей.

<xsl:preserve-space>

```
<xsl:preserve-space elements = tokens />
```

Эта инструкция задает в исходом древовидном представлении элементы, для которых нужно сохранить пробел.

<xsl:processing-instruction>

```
<xsl:processing-instruction name = {ncname}>
        <!-- Content: template -->
</xsl:processing-instruction>
```

Эта инструкция создает новую инструкцию по обработке Processing Instruction, задаваемую атрибутом **name.** Цель этой инструкции определяется ее содержимым.

<xsl:sort>

```
<xsl:sort select = string-expression lang = { nmtoken }
          data-type = {"text" I "number" I qname-but-not-ncname}
          order = {"ascending" I "descending"}
          case-order = {"upper-first" I "lower-first"}/>
```

Эта инструкция используется в инструкциях **xsl:apply-templates и xsl:for-each** для задания порядка обработки узлов. Первый результат выполнения инструкции **xsl:sort** задает первичный ключ сортировки, а каждое следующее ее выполнение — вторичный ключ сортировки. Сортировочный ключ для каждой инструкции получается путем вычисления XPath-выражения, содержащегося в атрибуте select, с использованием обрабатываемых узлов. По умолчанию в качестве ключа сортировки будет использоваться строковое значение обрабатываемого узла. Затем ключ сортировки можно изменить с помощью атрибутов этой инструкции:

- **Order** Определяет, каким должен быть порядок сортировки строк — восходящим или нисходящим.
- **Lang** Задает язык ключей сортировки.
- **Data-type** Определяет тип данных строк (обычно он бывает текстовым или числовым).
- **Case-order** Имеет значение **upper-first** или **lower-first.** Применяется, если **data-type="text"**, и задает порядок, в котором должна проводиться сортировка: сначала прописные буквы, а потом строчные, или наоборот.

<xsl:strip-space>

```
<xsl:strip-space elements = tokens />
```

Эта инструкция задает в исходном древовидном представлении документа элементы, для которых пробелы сохранять не следует.

<xsl:stylesheet>

```
<xsl:stylesheet id = id extension-element-prefixes = tokens
          exclude-result-prefixes = tokens version = number>
          <!-- Content: (xsl:import*, top-level-elements) -->
</xsl:style-sheet>
```

Эта инструкция представляет собой XSLT-таблицу стилей. В нее входят шаблоны и другие инструкции, разрешенные на верхнем уровне. Атрибут версии является

обязательным, он задает версию применяемого языка XSLT. В момент написания этой книги поддерживалась только версия 1.0. Атрибут **extension-element-prefixes** задает префиксы пространства имен для специальных инструкций, распознаваемых процессором. Атрибут **exclude-result-prefixes** используется для задания префиксов пространства имен, которые не следует выводить в результирующем древовидном представлении. Это бывает полезно, когда таблица стилей использует описание пространства имен только для адресации исходного дерева. Задавая префикс в атрибуте **exclude-result-prefixes**, можно избежать перегрузки результирующего дерева многочисленными описаниями пространств имен.

<xsl:template>

```
<xsl:template match = pattern name = qname
              priority = number mode = qname>
          <!-- Content: (xsl:param*, template) -->
</xsl:template>
```

Эта инструкция задает шаблон. Он может быть именованным с использованием атрибута **name** или назначенным, если схема соответствия связана с каким-то конкретным узлом с помощью инструкции **<xsl:apply-templates>**. Для этого шаблона можно также задать атрибуты mode и priority. После назначения шаблона начинается обработка его содержимого. При обработке всех параметров шаблона их значения по умолчанию заменяются любыми значениями, переданными в этот шаблон.

<xsl:text>

```
<xsl:text disable-output-escaping = "yes" I "no">
          <!-- Content: #PCDATA -->
</xsl:text>
```

Эта инструкция используется для создания в результирующем древовидном представлении нового текстового узла. Его содержимое может быть только символьным. Если атрибут **disable-output-escaping** имеет значение "yes", то процессор, на который отправляется запрос о выводе потока байтов, выдает в качестве результата символ как он есть и не пытается представить его с помощью символьной ссылки. Например, если атрибут **disable-output-escaping** имеет значение "yes", символ < будет представлен процессором в виде **<**.

<xsl:transform>

```
<xsl:transform id = id extension-element-prefixes = tokens
               exclude-result-prefixes = tokens version number>
          <!-- Content: (xsl:import*, top-level-elements) -->
</xsl:transform>
```

Эта инструкция является синонимом инструкции **xsl:stylesheet** и выполняет ту же самую функцию.

<xsl:value-of>

```
<xsl:value-of select = string-expression
              disable-output-escaping = "yes" I "no" />
```

Эта инструкция вычисляет XPath-выражение, заданное атрибутом **select,** с использованием текущих обрабатываемых узлов и преобразует его в строку.

<xsl:variable>

```
<xsl:variable name = qname select = expression>
          <!-- Content: template -->
</xsl:variable>
```

Эта инструкция определяет переменную, имя которой задается атрибутом пате. Ее значение может определяться или атрибутом **select,** или содержимым этой инструкции. Отметим, что это значение нельзя менять в пределах одного элемента.

<xsl:with-param>

```
<xsl:with-param name = qname select = expression>
          <!-- Content: template -->
</xsl:with-param>
```

Эта инструкция используется внутри инструкций **xsl:apply-templates** или **xsl:call-template,** чтобы перезаписать значение параметра шаблона, заданного атрибутом **пате.** Значение этой инструкции может определяться либо атрибутом **select,** либо содержимым этой инструкции.

Функции XSLT

В языке XSLT есть несколько дополнительных функций, отсутствующих в библиотеке XPath.

document

Эта функция задает кроме основного способа обращения к данным в исходном XML-документе еще один способ. Она производит синтаксический разбор XML-документа, к которому производится обращение, и возвращает узел, представляющий этот документ.

key

Эта функция позволяет извлекать значения ключей, описанных в XSLT-таблице стилей. Ключи имеют такие же функции, как идентификаторы ID, но кроме этого позволяют употреблять значения из нескольких слов и несколько ключей для одного и того же имени.

format-number

Эта функция преобразует число в строку с помощью заданных форматирующих строк.

current

Эта функция возвращает совокупность узлов, в которую входит текущий узел.

unparsed-entity-uri

Эта функция возвращает идентификатор URI не подвергнутой синтаксическому разбору сущности, которая имеет заданное имя в DTD того же документа, что и контекстный узел.

generate-id

Эта функция принимает аргумент совокупности узлов и возвращает строку, которая уникальным образом идентифицирует тот узел в совокупности, который первым встречается в документе,

system-property

Эта функция принимает строку имени и возвращает объект, представляющий значение свойства системы, идентифицируемого его именем. Если такого свойства системы не существует, возвращается пустая строка.

element-available

Эта функция возвращает значение истина, если ее аргумент **string** является именем какой-то инструкции. Если аргумент **name** имеет тот же URI-идентификатор пространства имен, что и URI-идентификатор пространства имен XSLT, то функция обращается к элементу, определенному XSLT-процессором. В противном случае функция обращается к элементу расширения. Если URI-идентификатор пространства имен является пустым, функция возвращает значение ложь.

function-available

Эта функция возвращает значение истина, если ее аргумент **string** является именем функции из библиотеки функций. Если аргумент **name** имеет непустой URI-идентификатор пространства имен; происходит обращение к функции расширения, в противном случае происходит обращение к функции, определенной XPath или XSLT.

Приложение

B

Спецификация W3C XML Schema

Что такое XML Schema?

В феврале 1999 г. был выпущен документ W3C Note, где были подробно изложены требования, разработанные группой XML Schema Working Group. В документе был сделан обзор языка XML Schema, изложены основные принципы его построения, требования к структуре и типам данных и к соответствию этим требованиям. В обзоре рабочая группа XML Schema Working Group подробно рассмотрела ограничения, накладываемые на XML-документ (DTD в этом документе уделяется гораздо меньше внимания). Приведем лишь некоторые из этих ограничений. XML-документ должен:

- Поддерживать как примитивные, так и сложные типы данных
- Поддерживать как ограничения, так и расширения типов данных
- Быть написанным на языке XML

Например, следующий фрагмент DTD:

```
<!ELEMENT book (title, author, publisher, price)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT author (#PCDATA)>
<!ELEMENT publisher (#PCDATA)>
<!ELEMENT price (#PCDATA)*>
```

в формате XML Schema превратится в XSD-файл:

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:bk="http://www.mypublishsite.com/book>
   <xsd:annotation>
      <xsd:documentation xml:lang="en">
      Possible XML Shema equivalent of DTD shown in Listing 1.
     </xsd:documentation>
   </xsd:annotation>
   <xsd:element name="title" type="xsd:string" minOccurs="1" maxOccurs="1"/>
   <xsd:element name="author" type="xsd:string" minOccurs="1"
    maxOccurs="unbounded"/>
```

```
    <xsd:element name="publisher" type="xsd:string" minOccurs="1"
     maxOccurs="unbounded"/>
    <xsd:element name="price" type="xsd:string" minOccurs="0" maxOccurs="*"/>
    <xsd:element name="Book"/>
       <xsd:complexType>
          <xsd:sequence>
             <xsd:element ref="bk:title"/>
             <xsd:element ref="bk:author"/>
             <xsd:element ref="bk:publisher"/>
             <xsd:element ref="bk:price"/>
          </xsd:sequence>
       </xsd:complexType>
    </xsd:element>
</xsd:schema>
```

В этих требованиях говорится:

> "Цель создания схемы состоит в том, чтобы определить и описать класс XML-документов, используя эти конструкции, для того чтобы ограничить и документировать значения, правила использования и взаимоотношения их составных частей: типов данных, элементов и их содержимого, атрибутов и их значений, сущностей и их содержимого, *а* также нотаций. Конструкции схемы могут быть использованы для создания спецификации такой неявной информации как значения по умолчанию. Схемы документируют свои собственные значения, правила использования и функции. Таким образом, язык XML Schema можно использовать для определения, описания и каталогизации XML-словарей для классов XML-документов".

/ Кроме того, в этом документе с требованиями перечислены следующие сценарии использования XML-приложений, которые получат явные преимущества от применения XML Schema:

- Публикация и синдицирование
- Обработка транзакций в электронной коммерции
- Управление в режиме супервизора и сбор данных
- Традиционные разработка и редактирование документов, управляемые ограничениями схемы
- Использование схемы для формулировки и оптимизации запросов
- Открытие и равномерная передача данных между приложениями, в том числе между базами данных
- Обмен метаданными

Наконец, кроме перечисленных принципов построения схемы в этом документе указаны структурные требования к тому, что должен определять язык XML Schema: требования к типам данных и к соответствию.

Затем на основе этих требований рабочая группа XML Schema Working Group последовательно выпустила несколько проектов документов, и завершилась эта работа в мае 2001 г. выпуском рекомендаций для XML Schema — W3C Recomendation for XML Schema. Рекомендации XML Schema Recomendations состоят из трех частей: "Основные принципы", глава 0; "Раздел структур", глава 1; и "Раздел типов данных", глава 2. К "Основным принципам" затем будет выпущено приложение, иллюстрирующее, как можно работать с механизмами, определенными в разделах структур и типов данных спецификаций XML Schema.

Введение

Эта часть спецификаций представляет собой очень длинное введение в XML Schema, ее конструкцию и правила использования. В разделе 2 "Основных принципов" рассматриваются главные концепции, причем несколько подробнее обсуждается схема заказа на покупку. Рассмотрим пример определения сложного типа для адреса:

```
<xsd:complexType name="USAddress">
   <xsd:sequence>
      <xsd:element name="name" type="xsd:string"/>
      <xsd:element name="street" type="xsd:string"/>
      <xsd:element name="city" type="xsd:string"/>
      <xsd:element name="state" type="xsd:string"/>
      <xsd:element name="zip" type="xsd:decimal"/>
   </xsd:sequence>
   <xsd:attribute name="country" type="xsd:NMTOKEN" fixed="US"/>
</xsd:complexType>
```

Важно отметить, что здесь пространства имен XML Schema вводятся даже для встроенных типов данных, таких как строковый (string). Кроме того, сложный тип данных окружен внутренними тегами. А такие ограничения, как **minOccurs и maxOccurs,** у которых значения по умолчанию равны 1, можно размещать в элементе имени следующим образом:

```
<xsd:element name="name type="xsd:string" minOccurs="1" maxOccurs="2"/>
```

В разделе 2 "Основных принципов" есть также полный список всех возможных простых встроенных типов данных для XML Schema, представленный в таблице В. 1.

Кроме того, для этих простых встроенных типов данных существуют определенные ограничения или аспекты. Например, для типов **string, normalizedString, token, base64Binasry, hexBinary, Name, Qname,** NCName, **anyURI, language, ID, IDFERS,**

ENTITY, ENTITIES, NOTATION, NMTOKEN и NMTOKENS существуют следующие ограничения: **length, minLength, maxLength, pattern** (это может быть какое-либо устойчивое выражение, например, дата в формате ММ/ДД/ГГГГ), **enumeration, whiteSpace.** Численно-ориентированные типы данных **byte, unsignedByte, integer, positiveInteger, negativeInteger, nonNegativeInteger, nonPositiveInteger, int, unsignedInt, long, unsignedLong, short, unsignedSort, decimal** имеют следующие ограничения: **maxInclusive, maxExclusive, minInclusive, minExclusive, totalDigits, fractionDigits.**

Таблица В.1

Простые встроенные типы данных для XML Schema

Простой встроенный тип данных	Пример (комментарии)
string	это строка
normalizedString	это может быть другая строка (разделители строк, знаки табуляции, возврат каретки и т. д. транслируются в пробелы)
token	это может быть другая строка (разделители строк, знаки табуляции, возврат каретки и т. д. транслируются в пробелы; расположенные рядом пробелы сжимаются до одного пробела; конечные и начальные пробелы удаляются)
byte	-1, 126
unsignedByte	0, 126
base64Binary	GpM7
hexBinary	Offf
integer	–126 789, 0, 126789 (только целые числа)
positiveInteger	1, 2, 126 789 (только целые положительные числа)
negativeInteger	-126 789, -2, -1 (только целые отрицательные числа)
nonNegativeInteger	O, 1, 126789
nonPositiveInteger	–126 789, -1, O
int	-1, 0, 2, 126 789 675
unsigned int	O, 1, 1 267 896 754
long	-1, -2, 0, 12 678 967 543 233
short	-1, -2, -5, O, 1, 12 678
unsigned Short	O, 1, 5, 12 678
decimal	-1.2, 0, 1.2, 100 000.00

Таблица В.1 *(продолжение)*

Простые встроенные типы данных для XML Schema

Простой встроенный тип данных	Пример (комментарии)
float	-0, 0, 12, INF, NaN 1.0E-2 (32-разрядное число с плавающей точкой)
double	-O, 0, 13, INF, NaN 1.0E-20 (64-разрядное число с плавающей точкой)
boolean	истина, ложь, 1, О
time	21:21:21.0000-01:00 (UTC — время по Гринвичу)
dateTime	2001-01-01T121:21:21.0000-01:00 (дата + временная зона + UTC)
duration	P1Y2M3DT10H30M12.0S (год, месяц, день, час, минута, секунда)
Date	2001-01-01
gMonth	-01-
gYear	2001
gYearMonth	2001-01
gDay	---31
gMonthDay	-01-01
Name	любое имя (XML 1.0 Name)
QName	xsd:anyname (XML Namespace Qualified Name)
NCName	любое имя (XML Namespace Qualified Name без префикса и двоеточия)
anyURI	http://www.oracle.com
language	en-US(американскийанглийский)
ID	(уникальный символ, атрибут XML 1.0 ID)
IDREF	(символ, соответствующий ID, атрибут XML 1.0 IDREF)
IDREFS	(список IDREF, атрибут XML 1.0 IDREFS)
ENTITY	(атрибут XML 1.0 ENTITY)
ENTITIES	(атрибут XML 1.0 ENTITIES)
NOTATION	(атрибут XML 1.0 NOTATION)
NMTOKEN	US, Canada (атрибут XML 1.0 NMTOKEN)
NMTOKENS	US UK Canada (атрибут XML 1.0 NMTOKENS)

Определяемые пользователем типы, такие как списки типов,— например *<mystates>CA NY MA</mystates>*— можно задать через:

```
<xsd:simpleType name="mystates">
   <xsd:list itemType="Juanstates"/>
</xsd:simpleType>
```

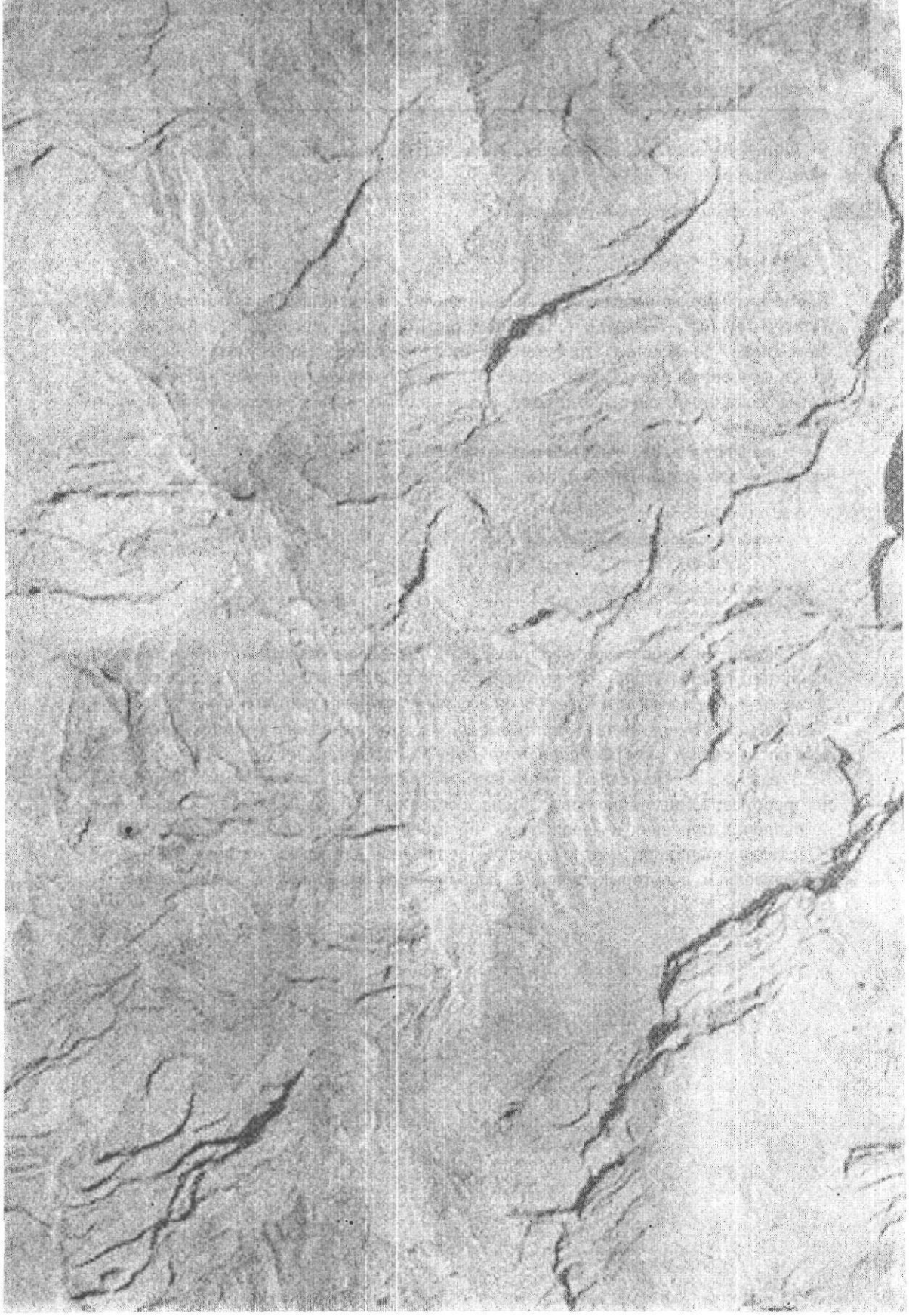
Кроме того, можно определить: объединенные типы, например *<xsd:union member Types="mystatesallstate"/>*; сложные типы complexType, образованные из простых типов; смешанные атрибуты для сложных типов, чтобы указать на данные между дочерними элементами; любой тип anyType, например *type="xsd:anyType"*, указывающий, что данный элемент может принадлежать к любому разрешенному типу данных.

Разрешаются также аннотации вида *<xsd:annotation>*. Они представляют собой механизм для встраивания документации в схему:

```
<xsd:annotation>
   <xsd:documentation xml:lang="en">
      hi there
   </xsd:documentation>
</xsd:annotation>
```

Наконец, пользователь может создавать им самим определенные механизмы, например, группы атрибутов, которые создаются для того, чтобы связать с одним элементом несколько атрибутов. В определение группы атрибутов нужно включить *<xsd:attributeGroup name="BookDelivery">*, а в определение complexType нужно включить ссылку *<xsd:attributeGroup ref="BookDelivery">*.

В разделе 3 "Основных принципов" обсуждаются пространства имен цели, не преобразованные локальные имена, составные локальные имена, глобальные и локальные описания и неописанные пространства имен цели. В разделах 4 и 5 "Основных принципов" еще раз рассмотрена схема для заказа на покупку, но уже с добавлением некоторых функций, дополняющих возможности такой схемы.



Приложение C

Другие спецификации консорциума W3C

Что такое XML Query?

В феврале 2001 г. был выпущен документ W3C Note, где были подробно изложены требования, разработанные группой XML Schema Working Group. В этом документе изложены цели, требования и сценарии использования модели данных, алгебры и языка запросов W3C XML Query. Согласно заявлению группы XML Schema Working Group, выпуская этот документ, они собирались "предоставить универсальные средства для составления запросов на извлечение данных из реальных и виртуальных документов в Web, обеспечивая таким образом необходимое взаимодействие между миром Web и миром баз данных", делая доступ к коллекциям XML-файлов в Web похожим на доступ в базы данных.

В числе указанных сценариев применения XML Query были следующие:

- Документы, пригодные для чтения человеком
- Документы, централизованные по данным
- Документы, построенные по смешанным моделям
- Административные данные
- Фильтрация потоков
- Document Object Model (DOM)
- Корпоративные XML-репозитории и Web-серверы
- Поиск по каталогам
- Среды с разнородным синтаксисом

Основные требования к языку XML Query касаются следующих проблем:

- Синтаксис XML Query должен был пригоден для чтения человеком
- XML Query должен быть декларативным языком
- XML Query должен быть языком, не зависящим от протоколов
- Язык XML Query должен поддерживать обработку ошибок
- Язык XML Query должен поддерживать обновления
- Язык XML Query определяется для конечных экземпляров документов (finite instances).

Основные требования к модели данных XML Query Data Model касаются следующих аспектов:

- Зависимость от информационного множества XML
- Типы данных
- Коллекции документов и простые/сложные значения
- Поддержка ссылок в XML-документе и ссылок из одного XML-документа в другой
- Доступность схемы
- Поддержка пространства имен

Основные требования к функциональным возможностям языка XML Query касаются следующих вопросов:

- Поддержка операций со всеми типами данных XML Query Data Model
- Границы между текстами и элементами
- Кванторы всеобщности и кванторы существования
- Иерархия и последовательность
- Комбинация из разных частей одного или нескольких XML-документов
- Агрегирование информации
- Сортировка
- Состав операций
- Значения NULL
- Сохранение структуры
- Преобразование структуры
- Ссылки
- Сохранение целостности данных
- Операции над литералами, т. е. над экземплярами XML Query Data Model, заданными с помощью запроса
- Операции с именами
- Операции со схемами
- Операции со схемой PSV infoset
- Расширяемость
- Информация об оборудовании
- Закрытие системы

В июне 2001 г. был выпущен проект документа XML Query Use Cases, где рассматривались подробности следующих сценариев работы с информацией, касающиеся описаний, DTD, выборки данных, запросов и их результатов:

- **XMP** Примеры использования
- **Tree** Запросы, сохраняющие иерархию
- **SEQ** Запросы, построенные на базе последовательностей
- R Доступ к реляционным базам данных
- **SGML** Язык Standard Generalized Markup Language
- **Text** Полнотекстовый поиск
- NS Запросы, использующие пространства имен
- **Parts** Рекурсивное размножение частей (Recursive Parts Explosion)
- REF Запросы, построенные на базе ссылок
- **FNPARM** Функции и параметры

В разделе XMP рассмотрен сценарий со следующими DTD и примером данных:

```
<!ELEMENT bib (book*)>
<!ELEMENT book (title, (author+ I editor+), publisher, price)>
<!ATTLIST book year CDATA #REQUIRED>
<!ELEMENT title (#PCDATA)>
<!ELEMENT editor (last, first, affiliation)>
<!ELEMENT author (last, first)>
<!ELEMENT publisher (#PCDATA)>
<!ELEMENT last (#PCDATA)>
<!ELEMENT first (#PCDATA)>
<!ELEMENT affiliation (#PCDATA)>
<!ELEMENT publisher (#PCDATA)>
<!ELEMENT price (#PCDATA)*>
```

```
<bib>
   <book year="1994">
      <title>TCP/IP Illustrated</title>
      <author><last>Stevens</last><first>W.</first></author>
      <publisher>Addison-Wesley</publisher>
      <price>65.95</price>
   </book>
   <book year="1992">
      <title>Advanced Programming in the Unix environment</title>
      <author><last>Stevens</last><first>W.</first></author>
      <publisher>Addison-Wesley</publisher>
      <price>65.95</price>
   </book>
```

```
<book year="2000">
   <title>Data on the Web</title>
   <author><last>Abiteboul</last><first>Serge</first></author>
   <author><last>Buneman</last><first>Peter</first></author>
   <author><last>Suciu</last><first>Dan</first></author>
   <publisher>Morgan Kaufmann Publishers</publisher>
   <price>39.95</price>
</book>
<book year="1999">
   <title>The Economics of Technology and Content for Digital TV</title>
   <editor>
      <last>Gerberg</last><first>Darcy</first>
      <affiliation>CITI</affiliation>
   <editor>
   <publisher>Kluwer Academic Publishers</publisher>
   <price>129.95</price>
</book>
</bib>
```

© 2001, W3C

Чтобы найти пары книг, имеющие разные названия, но одних и тех же авторов, составляется запрос XQUERY:

```
<bib>
{  FOR $book1 IN document ("www.bn.com/bib.xml">//book,
      $book2IN document ("www.bn.com/bib.xml">//book
   WHERE $book1/title/text() > $book2/title/text()
   AND bags-are-equal ($book1/author, $book2/author)
   RETURN
      <book-pair>
         {$book1/title}
         {$book2/title}
      </book-pair>
}
</bib>
```

- на который будет получен следующий ответ:

```
<bib>
   <book-pair>
      <title>TCP/IP Illustrated</title>
      <title>Advanced Programming in the Unix environment</title>
   </book-pair>
</bib>
```

© 2001, W3C

В примере из раздела TREE текст смешан с элементами, элементы определены в нескольких DTD и в одном XML-файле, и не все они являются обязательными. Для подготовки вложенной таблицы, в которой перечисляются все секции и заголовки с сохранением атрибутов каждого элемента <section>, составляется запрос XQUERY:

```
<toc>
{  LET $b :=document ("book1,xml")
   RETURN
            Filter($b//section I $b//section/title
             I $b//section/title/text())
}
</toc>
```

Результат выполнения этого запроса:

```
<toc>
   <section id="intro" difficulty="easy">
      <title>Introduction</title>
      <section><title>Audience</title></section>
      <section><title>Web Data and the Two Cultures</title>
   </section>
   <section id="syntax" difficulty="medium">
      <title>A Syntax for Data</title>
      <sectionxtitle>Base Types</tit le></sect ion>
      <sectionxtitle>Representing Relational Databases</title>
      <section><title>Representing Object Databases</title></section>
</toc>
```


В разделе SEQ приведен сценарий приложения с несколькими запросами, построенными на последовательности, с которой элементы появляются в документе. В примере фигурирует другой DTD и XML-файл примера, который построен на основе медицинского отчета. Для того чтобы определить, какой из двух инструментов нужно применить первым в разделе "Процедура" Отчета1, составляется следующий запрос XQUERY:

```
FOR $s IN document ("report1.xml")//section[section.title = "Procedure"]
RETURN ($s//instrument) [1 TO 2]
```

На него приходит ответ:

```
IP <instrument>using electrocautery.</instrument>
<instrument>electrocautery</instrument>
```

В разделе R рассмотрен сценарий приложения для доступа к информации, хранящейся в реляционной базе данных. В примере приведен возможный DTD и XML-файл примера. Для того чтобы определить, сколько лотов было выставлено на аукцион в марте 1999 г., составляется следующий запрос XQUERY:

```
LET $item := document ("items.xml")//item_tuple
    [end_date >= date("1999-03-01") AND end_date <= date("1999-03-31")]
RETURN
   <item_count>
   { count ($item) }
   </item_count>
```

Ответ на этот запрос:

```
<item_count>3</item_count>
```


В разделе SGML в качестве примера приведен сценарий приложения, когда DTD и пример документа транслируются с языка SGML на язык XML, и все это иллюстрируется несколькими запросами. Для определения местоположения всех секций с заголовками, в которых есть слова "is SGML", составляется следующий запрос XQUERY:

```
<result>
{ //section[contains(string(.//title), "is SGML")] }
</result>
```

На этот запрос будет получен ответ:

```
Elements whose start-tags are on lines 51, 60
```


В разделе TEXT разобран сценарий приложения, в котором DTD и пример XML-файла построены на основе профилей компании, а запросы составляются для поиска в новостях документов, где упоминается название компании. Для определения местоположения всех статей, где в заголовке появляется имя "Foobar Corporation", составляется запрос XQUERY:

```
//news_item/title[contains(./text(), "Foobar Corporation")]
```

Ответ на этот запрос:

```
<title>Foobar Corporation releases its new line of Foo products today</title>
<title>Foobar Corporation is suing Gorilla Corporation for patent
infringment</title>
```

В разделе NS использован сценарий приложения, где DTD и пример документа выдают иллюстрации в ответ на несколько запросов на имена с составными пространствами имен. Для выбора заголовка каждой выставленной на продажу музыкальной записи составляется следующий запрос XQUERY:

```
NAMESPACE music = "http://www.example.org/music/records"
<Q2>
{  //music:title  }
</Q2>
```

На этот запрос будет получен ответ:

```
<Q2 xmlns:music="http://www.example.org/music/records">
    <music:title>In a Silent Way</music:title>
    <music:title>Think of One ... </music:title>
</Q2>
```

В разделе PARTS представлен сценарий приложения, иллюстрирующий использование рекурсивного запроса для построения структурированного документа некоторого раздела из реляционных таблиц, хранящихся в базе данных. В разделе REF представлен сценарий приложения, который иллюстрирует использование запросов на базе ссылок. В качестве примеров там приведены DTD и XML-файл, представляющий собой базу данных, в которой очень важную роль играют ссылки. Наконец, в разделе FNPARM представлен сценарий, где исследуются некоторые способы определения и вызова функций с использованием нескольких типов данных из XML Schema.

Что такое XML Protocol!

По заявлению рабочей группы XML Protocol, ее целью является "разработка технологий, которые позволяют взаимодействовать друг с другом двум" и более узлам в распределенной среде, используя XML в качестве языка инкапсуляции. Решения, разработанные в результате этих усилий, позволяют строить поверх простого формата обмена сообщениями многоуровневую архитектуру, и такая конструкция обладает надежностью, простотой, совместимостью с разными платформами, а также допускает повторное использование разработанных программных модулей". В июле 2001 г. упомянутой группой были опубликованы проекты рабочих документов XML Protocol Abstract Model и SOAP V1.2.

Проект документа XML Protocol Abstract Model описывает "структуру для систем обмена сообщениями в XML-формате, которая определяет формат конверта сообщения и метод сериализации данных. Эта структура может быть использована в приложениях с удаленным вызовом процедур (но не только в них), что соответствует принципу повторного использования кода". Частью этой архитектуры является XMLP Application — клиент или пользователь служб, предоставляемых слоем

XML Protocol Layer. Приложение XML Protocol (XMLP) Application может инициировать или отвечать на однонаправленные или двунаправленные операции отклика. В приложения XMLP Application инкапсулированы программы-обработчики XML Protocol Handlers. Кроме того, XMLP Layer представляет собой "абстракцию, которая предоставляет службы или операции, передающие пакеты XML Protocol Blocks между равноправными приложениями XML Protocol Applications непосредственно или через программы-посредники XML Protocol Intermediaries". Некоторые примитивы XMLP-слоя включают операции XMLP_UnitData по отправке, приему, определению статуса и перенаправлению сообщений. Такие операции представляют собой службы, предлагаемые слоем XMLP. Они моделируются как последовательность событий, пересекающих границы между XMLP-процессорами и приложениями.

Что касается протокола SOAP, то его "версия 1.2 представляет собой облегченный протокол для обмена информацией в децентрализованной распространенной среде. Этот протокол построен на базе технологии XML. Он состоит из четырех частей: конверта, который определяет структуру описания того, что находится в сообщении, и как это обрабатывать; набора правил кодирования для выражения экземпляров типов данных, определяемых приложением; соглашения для представления вызовов и ответов удаленных процедур; и соглашения о связях для обмена сообщениями с использованием базового протокола. Протокол SOAP потенциально можно использовать в комбинации с другими протоколами. Однако пока существуют только соглашение о том, как использовать протокол SOAP в комбинации с протоколом HTTP, и экспериментальная среда HTTP Extention Framework".

Далее приведены примеры конвертов SOAP-сообщений для вызова запроса о курсе акций компании Oracle и ответа на него.

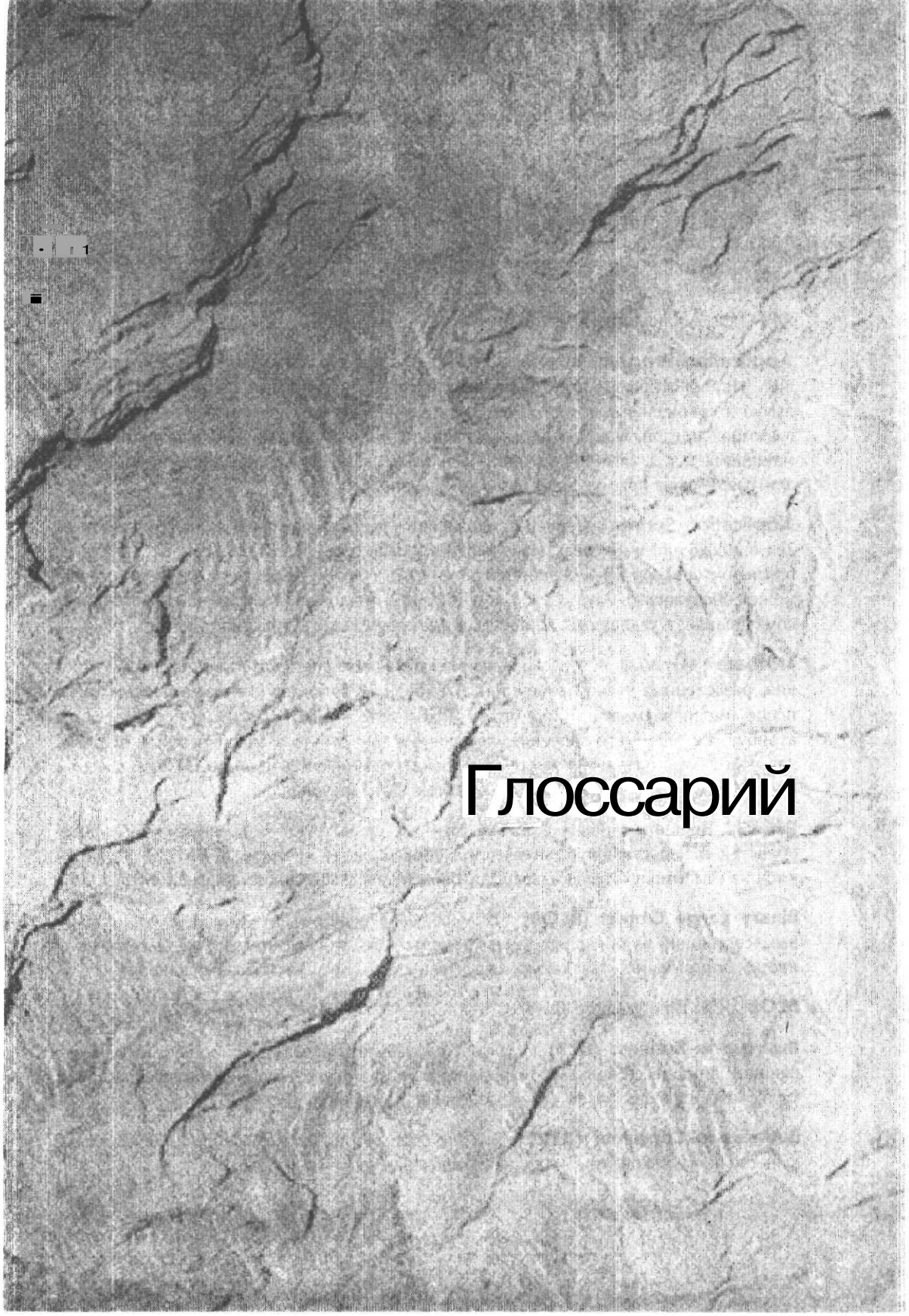
Запрос:

```
POST /StockQuote HTTP/1.1
Host: www.stockquoterserver.com
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn
SOAPAction: "http://example.org/2001/06/quotes"

<env:Envelope
   xmlns:env="http://example.org/2001/06/soap-envelope">
   <env:Header>
         <t:Transaction
            xmlns:t="http://example.org/2001/06/tx"
            env:encodingStyle="http://www.w3.org/2001/06/soap-encoding"
            env:mustUnderStand="1">
            5
         </t:Transaction>
   </env:Header>
```

```
    <env:Body>
          <m:GetLastTradePrice
             env:encodingStyle="http://ww.w3.org/2001/06/soap-encoding"
             xmlns:m="http://example.org/2001/06/quotes">
             <m:symbol>ORCL</m:symbol>
          </m:GetLastTradePrice>
    </env:Body>
</env:Envelope>
```

Ответ:

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn

<env:Envelope
   xmlns:env="http://example.org/2001/06/soap-envelope">
   <env:Header>
         <t:Transaction
            xmlns:t="http://example.org/2001/06/tx"
            xmlns:xsi="http://w3.org/2001/XMLSchema-instance"
            xmlns:xs=http://w3.org/2001/XMLSchema
            xsi:type="xs:int"
            env:encodingStyle="http://www.w3.org/2001/06/soap-encoding"
            env:mustUndersStand="1" >
            5
         </t:Transaction>
   </env:Header>
   <env:Body>
         <m:GetLastTradePriceResponse
            env:encodingStyle="http://www.w3.org/2001/06/soap-encoding"
            xmlns:m="http://example.org/2001/06/quotes">
            <Price>93.5</Price>
         </m:GetLastTradePriceResponse>
   </env:Body>
</env:Envelope>
```



Глоссарий

API *См.* Application Program Interface.

Application Program Interface (API) Интерфейс прикладного программирования. Набор открытых программных интерфейсов, который состоит из формата языка и сообщения для взаимодействия с операционной системой или другой программной средой, например базой данных, Web-серверами, виртуальными Java-машинами и т. д. Эти сообщения обычно вызывают функции и методы, доступные для разработки приложения.

Application Server Сервер приложений — сервер, который является основным для приложений и их сред, позволяющих запуск серверных приложений. Типичным примером сервера приложения является OAS (Oracle Application Server), который может обслуживать Java, C, C++ и PL/SQL-приложения в средах, где интерфейсом управляет удаленный клиент. *См. также* Oracle Application Server

attribute Атрибут — это свойство элемента, которое состоит из имени и значения, разделенных знаком равенства. Атрибут находится внутри открывающих тегов после имени элемента. В примере <Price units='USD'>5</Price> *Price* — это атрибут, а *USD* — его значение, которое должно стоять в одинарных или двойных кавычках. Атрибуты могут постоянно храниться в документе или в DTD. Элементы могут иметь много атрибутов, но порядок их извлечения не определен.

BFILES Внешние бинарные файлы, которые существуют за пределами табличных областей БД, постоянно хранящихся в операционной системе. К BFILES обращаются из семантики базы данных. Они также известны под названием *External LOB.*

Binary Large Object (BLOB) Большой двоичный объект — один из типов данных, состоящий из бинарных данных. Кроме того, данные этого объекта считаются *необработанными,* так как их структура не распознается базой данных.

BLOB *См.* Binary Large Object

Business-to-Business (B2B) Термин, описывающий взаимодействие между компаниями в процессе продажи товаров и услуг друг другу. Программная инфраструктура, разрешающая такое взаимодействие, называется *биржей.*

Business-to-Consumer (B2C) Термин, описывающий взаимодействие между компаниями и их клиентами в процессе продажи товаров и услуг.

callback Обратный вызов — метод программирования, в котором один процесс дает старт другому и продолжает свою работу. Второй процесс затем вызывает первый в зависимости от действия, значения или другого события. Этот метод используется в большинстве программ, которые имеют пользовательский интерфейс, разрешающий продолжительное взаимодействие.

cartridge Картридж — хранимая программа в языках Java и PL/SQL, которая добавляет в базы данных необходимую функциональность, чтобы можно было опознавать новый тип данных и манипулировать им. Картриджи взаимодействуют через среду расширения Extensibility Framework в Oracle 8 или 8i. Например, картриджем является программа interMedia Text, поддерживающая чтение, запись и поиск текстовых документов, хранящихся в базе данных.

CDATA *См.* Character data

CGI *См.* Common Gateway Interface

Character data (CDATA) Текст в документе, который нельзя синтаксически проанализировать, помещается в секцию CDATA. В эту секцию можно включить символы, которые в другой ситуации будут восприниматься как специальные функции, например, &, <, > и т. д. Секции CDATA могут использоваться как в содержимом элемента, так и в атрибутах. Синтаксическая структура таких данных: <![CDATA[*здесь находится текст*]]>.

Character Large Object (CLOB) Тип данных большого объекта (LOB), значение которого состоит из символьных данных, соответствующих набору символов базы данных. Объект CLOB может быть индексирован, а данные в нем можно искать с помощью поискового средства interMedia Text.

child element Дочерний элемент — элемент, полностью находящийся внутри другого элемента, который по отношению к первому элементу называется *родительским.* Например, выражение **<Parent><Child></Child></Parent>** означает, что элемент Child находится внутри элемента Parent.

Class Generator Генератор классов — утилита, принимающая входной файл и создающая набор выходных классов, которые имеют соответствующие функциональные возможности. В случае генератора классов XML Class Generator входным файлом является DTD (определение типа документа), а на выходе получается несколько классов, которые могут быть использованы для создания XML-документов, соответствующих DTD.

CLASSPATH Внешняя переменная операционной системы, которую виртуальная Java-машина использует для поиска классов, необходимых для запуска приложений.

client/server Клиент/сервер — термин, используемый для описания архитектуры приложения, в которой приложение работает фактически на клиентской машине, но получает доступ к данным и другим внешним процессам через сеть на сервере.

CLOB *См.* Character Large Object.

command line Командная строка — интерфейс, в котором пользователь вводит команды в строку в ответ на приглашение системы.

Common Gateway Interface (CGI) Аббревиатура для интерфейсов программирования, допускающих, чтобы Web-серверы исполняли другие программы и передавали их результаты в HTML-страницы, в графические, звуковые и видеофайлы, направляемые на браузеры.

Common Object Request Broker API (CORBA) Стандарт консорциума Object Management Group, определяющий взаимодействие между распределенными в сети объектами. Объекты CORBA представляют собой независимые программные модули, которые могут использоваться приложениями, работающими на различных платформах или в разных операционных системах. Форматы и функции объектов CORBA определяются в языке *IDL (Interface Definition Language),* который может быть скомпилирован на самых разных языках, в том числе на Java, C, C++, Smalltalk и COBOL.

Common Oracle Runtime Environment (CORE) Библиотека функций, написанных на языке C, предоставляющая разработчикам возможность создавать код, легко переносимый практически на любую платформу и операционную систему.

CORBA *См.* Common Object Request Broker API.

Database Access Descriptor (DAD) Именованный набор значений конфигурации, используемый для доступа в базу данных. В DAD могут быть определены имя базы данных или название службы SQL*Net V2; каталог ORACLE_HOME; или информация о конфигурации NLS, например используемый язык, тип сортировки и формат записи даты.

datagram Дейтаграмма — это фрагмент текста, который может быть написан в формате XML. Такой фрагмент возвращается в запрашивающую программу, встроенную в HTML-страницу, после обработки SQL-запроса сервлетом XQSL Servlet.

DBURITYPE Тип данных в Oracle9i, используемый для хранения экземпляров типа данных и позволяющий осуществлять навигацию по схемам баз данных с помощью команд языка XPath.

DOCTYPE Термин, используемый как имя тега, характеризующий DTD или его ссылку в XML-документе. Например, **<!DOCTYPE person SYSTEM "person.dtd">** означает, что корневой документ имеет название *person* и внешнее описание типа документа DTD, которое находится в файле *person.dtd.* Внутренние DTD по определению описываются внутри описания DOCTYPE.

Document Object Model (DOM) Находящееся в памяти древовидное объектное представление XML-документа, которое предоставляет программный доступ к своим элементам и атрибутам. Объект DOM и его интерфейс являются стандартами, рекомендованными консорциумом W3C.

Document Type Definition (DTD) Описание типа документа — набор правил, которые определяют разрешенную структуру XML-документа. DTD — это текстовые файлы, формат которых определяется стандартом SGML. DTD может быть включен в XML-документ с помощью элемента DOCTYPE или с использованием внешнего файла через ссылку в DOCTYPE.

DOM *См.* Document Object Model.

DTD *См.* Document Type Definition.

element Элемент — основная логическая единица XML-документа, которая может также выполнять функции контейнера для других элементов, например элементов второго уровня, дат, атрибутов и их значений. Элементы задаются открывающими (<name>) и закрывающими тегами (</пате>). Если же элемент пустой, он может иметь вид <name/>.

empty element Пустой элемент — элемент, который не содержит текста или элементов второго уровня. Он может содержать только атрибуты и их значения. Пустые элементы имеют вид <name/> или <name></name>, где между тегами нет пробелов.

Enterprise Java Bean (EJB) Независимый программный модуль, который работает в виртуальной Java-машине на сервере. Инфраструктура для модулей EJB создается объектами CORBA. Безопасность, поддержка транзакций и другие функции на любом поддерживаемом сервере обеспечиваются контейнерным слоем.

entity Сущность — это строка символов, которая может представлять другую строку символов или специальные символы, не являющиеся частью набора символов документа. Сущности и текст, который анализатор синтаксиса заменяет сущностями, описываются в DTD.

Extensible Markup Language (XML) Открытый стандарт языка для описания данных, разработанный консорциумом W3C. Этот язык использует подмножество синтаксиса языка SGML и предназначается для использования в Интернете. Текущий стандарт XML имеет номер версии 1.0, он опубликован в феврале 1998 г. как рекомендация консорциума W3C.

Extensible Stylesheet Language (XSL) Язык, использующийся внутри таблиц стилей для преобразования или формирования XML-документов. Существуют две рекомендации W3C, касающиеся таблиц стилей XSL — XSL Transformations (XSLT) и XSL Formatting Objects (XSLFO).

Extensible Stylesheet Language Formatting Objects (XSLFO) Спецификация стандарта W3C, которая определяет словарь XML для определения семантики форматирования.

Extensible Stylesheet Language Transformation (XSLT) Спецификация стандарта XSL W3C, которая определяет язык преобразования одного XML-документа в другой.

Extract SQL-оператор Oracle9i, который возвращает фрагменты XML-документов, хранящихся как данные типа XMLTYPE.

Existnode SQL-оператор Oracle9i, который возвращает значение истина (TRUE) или ложь (FALSE), исходя из существования XPATH в XMLTYPE.

Functional Index Индекс базы данных, который после создания позволяет намного быстрее получать результаты известных запросов.

HASPATH SQL-оператор Oracle9i, являющийся частью программы Oracle Text. Оператор используется для запросов к типам данных XMLTYPE о существовании конкретного оператора Xpath.

HTML *См.* Hypertext Markup Language.

HTTP *См.* Hypertext Transport Protocol.

HTTPURITYPE Тип данных в Oracle9i, используемый для хранения экземпляров типа данных. HTTPURITYPE позволяет осуществлять навигацию по схемам баз данных в удаленных БД с помощью операторов языка XPath.

hypertext Гипертекст — метод создания и публикации текстовых документов, который позволяет пользователям перемещаться по другим документам или графическим файлам, выбирая слова или фразы, обозначенные как гиперссылки.

Hypertext Markup Language (HTML) Язык разметки, используемый для создания файлов, отправляемых на Web-браузеры. Этот язык является основой системы World Wide Web. Следующая версия языка HTML будет называться *xHTML* и станет приложением языка XML.

Hypertext Transport Protocol (HTTP) Протокол, используемый для транспортировки HTML-файлов по сети Интернет между Web-серверами и браузерами.

IDE *См.* Integrated Development Environment.

IFS *См.* Internet File System.

INPATH SQL-оператор Oracle9i, который является частью программы Oracle Text. Он используется в языке Xpath для организации запросов о типах данных XMLTYPE для поиска заданного текста.

instantiate Приписывать значение — термин, используемый в объектных языках, например Java и C++, для описания создания объекта определенного класса.

Integrated Development Environment (IDE) Набор программ, призванных помочь в разработке ПО, работающего с единого пользовательского интерфейса. Например, JDeveloper является IDE для разработки на языке Java, так как в него входит редактор, компилятор, отладчик, программа проверки синтаксиса, система справочной информации и т. д., что позволяет разрабатывать Java-программы через единый пользовательский интерфейс.

interMedia Термин, используемый для описания набора сложных типов данных и доступа к ним внутри Oracle8i. Сюда относятся текстовые данные, видео, данные временного ряда и пространственные данные.

Internet File System (iFS) Файловая система Oracle и среда разработки на базе Java, которая запускается или внутри СУБД Oracle8i, или на промежуточном слое. Эта файловая система предоставляет средства для создания, хранения и управления несколькими типами документов в одном репозитории базы данных.

Internet Inter-ORB Protocol (HOP) Протокол, используемый в технологии CORBA для обмена сообщениями в сетях TCP/IP, в частности, в Интернете.

Java Язык программирования высокого уровня, разработанный и поддерживаемый компанией Sun Microsystems. Этот язык предусматривает запуск приложения на виртуальной машине, называемой JVM (виртуальная Java-машина). JVM отвечает за все интерфейсы с операционной системой. Такая архитектура позволяет разработчикам создавать Java-приложения и апплеты, которые могут работать в любой операционной системе и на любой платформе, где есть JVM.

Java Bean Независимый программный модуль, работающий внутри виртуальной Java-машины. Он обычно используется для создания пользовательских интерфейсов на стороне клиента. Серверный эквивалент Java Bean называется *Enterprise Java Bean* (*EJB*). *См. также* Enterprise Java Bean.

Java Database Connectivity (JDBC) API-интерфейс, который позволяет Java-приложениям получать доступ к базе данных с помощью языка SQL. Драйверы JDBC написаны на языке Java для независимости от платформы, но у каждой СУБД есть свои версии этих драйверов.

Java Developer's Kit (JDK) Набор программного обеспечения, состоящий из Java-классов, среды реального времени, компилятора, отладчика и, как правило, исходного кода для одной из версий Java. С помощью JDK можно создать среду разработки Java-приложений. Инструментарии JDK выпускаются для каждой версии Java. Версия Java 2 используется в версиях JDK с номером 1.2 и выше.

Java Runtime Environment (JRE) Коллекция скомпилированных классов, образующая виртуальную Java-машину на некой платформе. Уже выпущено несколько версий сред JRE. Язык Java 2 используется в версиях с номерами 1.2 и выше.

Java Server page (JSP) Расширение функциональных возможностей сервлета, поддерживающее простой программный интерфейс для Web-страниц. JSP — это HTML-страница со специальными тегами и встроенным Java-кодом, который исполняется на Web-сервере или сервере приложений, добавляя в HTML-страницы динамические функциональные возможности. JSP-страницы после первого запроса и запуска на серверной виртуальной Java-машине компилируются в сервлеты.

Java virtual machine (JVM) Интерпретатор Java, который преобразует скомпилированный байт-код Java в машинный язык конкретной платформы и исполняет его. Виртуальные Java-машины могут работать на клиентском компьютере, в браузере, в промежуточном слое, в Web, на сервере приложений типа OAS или на сервере баз данных, например Oracle8i.

JDBC *См.* Java Database Connectivity.

JDeveloper Интегрированная среда разработки Oracle на языке Java, которая позволяет разрабатывать приложения, апплеты и сервлеты. В ее состав входит редактор, компилятор, отладчик, программа проверки синтаксиса, справочная система и т. д. В версии 3.1 в JDeveloper была реализована расширенная поддержка разработки на базе технологии XML, так как в это ПО интегрирован инструментарий Oracle XDK for Java, а кроме того, в редактор добавлена поддержка XML.

JDK *См.* Java Developer's Kit.

JServer Виртуальная Java-машина, функционирующая в памяти СУБД Oracle8i. В версии Oracle8i Release 1 виртуальная Java-машина была совместима с Java 1.1, а версия Release 2 стала совместима с Java 1.2.

JVM *См.* Java virtual machine.

LAN *См.* local area network.

Large Object (LOB) Класс SQL типа данных, который впоследствии разделился на *Internal LOB* (внутренние большие объекты) и *External LOB* (внешние большие объекты). Внутренние LOB содержат BLOB, CLOB и NCLOB, а внешние LOB включают BFILES. *См. также* BFILES, Binary Large Object (BLO, бинарные большие объекты), Character Large Object (CLOB, символьные большие объекты) и Non-Character Large Object (NCLOB, несимвольные большие объекты).

listener Слушающая программа — это отдельный процесс приложения, который следит за вводом информации.

LOB *См.* Large Object.

local area network (LAN) Локальная сеть — компьютерная сеть, обслуживающая пользователей на ограниченной территории. ЛС состоит из серверов, рабочих станций, коммуникационного оборудования (маршрутизаторы, мосты, сетевые карты и т. д.) и сетевой операционной системы.

namespace Пространство имен — термин для описания набора связанных имен или атрибутов элемента в XML-документе. Синтаксис пространства имен и его использование определяются W3C Recomendation. Например, элемент <xsl:apply-templates/> идентифицирован как часть пространства имен XSL. Описание пространства имен осуществляется в XML-документе или в DTD с помощью синтаксиса атрибута xmlns:xsl="http://www.w3.org/TR/WD-xsl" до их использования.

NCLOB *См.* Non-Character Large Object.

node Узел — в XML этот термин используется для обозначения каждого адресуемого объекта на дереве DOM.

Non-Character Large Objec Тип данных большого объекта LOB, значение которого состоит из символьных данных, соответствующих набору символов БД.

NOTATION Нотация — в XML это определение типа содержимого, не являющегося частью типа, распознаваемого программой синтаксического анализа. К таким типам относятся звуковые, видео и другие мультимедийные данные.

n-tier *N*-уровневый — обозначение архитектуры компьютерной сети, которая состоит из одного и более уровней, составленных из клиентов и серверов. Обычно *двухуровневые системы* состоят из одного клиентского и одного серверного уровня. В *трехуровневой системе* используются два серверных уровня, обычно первый — это сервер базы данных, второй — Web-сервер или сервер приложения, третий — клиентский уровень.

OAS *См.* Oracle Application Server.

OASIS *См.* Organization for the Advancement of Structured Information Standards.

object-relational Объектно-реляционный — термин для описания реляционной СУБД, которая также может хранить и манипулировать данными более высоких типов: текстовыми документами, аудио- и видеофайлами и определяемыми пользователем объектами.

Object Request Broker (ORB) Брокер объектных запросов — программа, которая управляет обменом сообщениями между запрашивающими программами на клиентских машинах и между объектами на серверах. ORB передает запрос на совершение действия и его параметры к объекту и возвращает результаты. Обычно брокер объектных запросов реализован в виде программных модулей CORBA и EJB. *См. также* Common Object Request Broker API и Enterprise Java Bean.

Object View Специальное представление данных, содержащихся в одной или более таблицах объектов или в других типах представлений. Ответом на запрос к Object View является таблица. Object View может использоваться в большинстве ситуаций, в которых требуются таблицы.

OIS *См.* Oracle Integration Server.

Oracle Application Server (OAS) Сервер Oracle, интегрирующий все основные службы и функции, требуемые для построения, разворачивания и управления высокопроизводительными, *n*-уровневыми, ориентированными на выполнение транзакций Web-приложениями в рамках открытых стандартов.

ORACLE_HOME Переменная среды операционной системы, идентифицирующая местоположение инсталляции СУБД Oracle для ее использования приложениями.

Oracle Integration Server (OIS) Серверный продукт Oracle, который функционирует, как концентратор при передаче сообщений для интеграции приложения. OIS содержит СУБД Oracle8i с AQ и Oracle Workflow и интерфейсы для приложений, использующих программу Oracle Message Broker для передачи между ними сообщений в формате XML.

ORB *См.* Object Request Broker.

Organization for the Advancement of Structured Information (OASIS) Организация, члены которой занимаются продвижением открытых информационных стандартов, организуя конференции, семинары, выставки и другие образовательные мероприятия. Например, эта организация активно продвигает стандарт XML, а также SGML (Standard Generalized Markup Language, стандартный обобщенный язык разметки).

parent element Родительский элемент — это элемент, окружающий другой элемент, считающийся по отношению к первому дочерним элементом (child element). Например, `<Parent><Child></Child></Parent>` показывает, что элемент Parent охватывает свой дочерний элемент Child.

Parsed Character Data (PCDATA) Содержимое элемента, состоящее из текста, который нужно проанализировать. PCDATA не является частью тега или не анализированных данных.

parser Программа синтаксического анализа — в XML эта программа принимает на входе XML-документ и определяет, правильно ли он сформатирован. Этот анализатор XML-синтаксиса может также проверять документ на допустимость (valid). Программа синтаксического анализа из Oracle XML поддерживает интерфейсы SAX и DOM.

PCDATA *См.* Parsed Character Data.

PL/SQL Процедурный язык базы СУБД Oracle, представляющий собой расширение языка SQL для создания программ, которые можно запускать на исполнение внутри базы данных.

prolog Пролог — начальная часть XML-документа, содержащая XML-описание, DTD или другие описания, необходимые для обработки документа.

PUBLIC Термин, используемый для определения местоположения следующей ссылки в Интернете.

renderer Программа визуализации — программный процессор, который представляет документ в заранее заданном формате.

result set Набор результатов — это набор возвращаемых по SQL-запросу данных, который состоит из одной или более строк данных.

root element Корневой элемент — элемент, содержащий все другие элементы в XML-документе. Он находится между прологом и эпилогом, т. е. между начальным и заключительным элементами XML-документа. Любой XML-документ может содержать только один корневой элемент.

SAX *См.* Simple API for XML.

schema Схема — определение структуры и типов данных в базе данных. Схема также может использоваться для ссылки на XML-документ, который поддерживает спецификацию XML Schema W3C Recomendation.

Secure Sockets Layer (SSL) Основной протокол, гарантирующий безопасную передачу данных в Интернете, который использует систему шифрования с открытым и частным ключом между браузерами и серверами.

Server Side Includes (SSI) HTML-команда, используемая для размещения данных или другого, содержимого на Web-странице до их отправки на запрашивающий браузер.

servlet Сервлет — Java-приложение, которое работает на сервере (обычно на Web-сервере или сервере приложений) и выполняет здесь обработку данных. Сервлеты являются Java-эквивалентом CGI-скриптов.

session Сеанс — активное соединение между двумя уровнями.

SGML *См.* Standard Generalized Markup Language.

Simple API for XML (SAX) Стандартный интерфейс XML, предоставляемый программами анализа XML-синтаксиса и используемый приложениями, основанными на событиях.

SQL *См.* Structured Query Language.

SSI *См.* Server Side Includes.

SSL *CM.* Secure Sockets Layer.

SOAP *См.* Simple Object Access Protocol.

Simple Object Access Protocol Облегченный протокол на базе XML для обмена информацией в децентрализованной распределенной среде.

Standard Generalized Markup Language (SGML) Стандарт ISO для определения формата текстового документа, созданного с использованием разметки и DTD.

Structured Query Language (SQL) Стандартный язык, используемый для доступа к данным и для их обработки в реляционной СУБД.

Stylesheet Таблица стилей — в XML данный термин используется для описания XML-документа, который состоит из инструкций XSL-обработки, используемых XSL-процессором для преобразования или форматирования входного XML-документа и превращения его в выходной документ.

SYSTEM Термин, используемый для определения местоположения ссылки в основной операционной системе.

SYS_XMLAGG Собственная SQL-функция Oracle9i, которая возвращает в виде единого XML-документа результаты переданного SYS_XMLGEN SQL-запроса. Эта функция может также использоваться для назначения типа данных XMLTYPE.

SYS_XMLGEN Собственная SQL-функция Oracle9i, которая возвращает в виде XML-документа результаты переданного SQL-запроса. Эту функцию также можно использовать для назначения типа данных XMLTYPE.

tag Тег — фрагмент XML-разметки, определяющий границы начала или конца любого элемента. Теги начинаются с открывающей угловой скобки < и заканчиваются закрывающей скобкой >. В XML существуют открывающие (<name>), закрывающие (</name>), и пустые теги (<name/>).

TCP/IP *См.* Transmission Control Protocol/Internet Protocol.

thread Поток — в программировании это путь выполнения одного сообщения или процесса в операционной системе. Причем этот поток поддерживает несколько операционных систем, в том числе Windows, UNIX и Java.

Transmission Control Protocol/Internet Protocol (TCP/IP) Коммуникационный сетевой протокол, который состоит из протокола TCP, управляющего транспортными функциями, и протокола IP, который обеспечивает работу механизма маршрутизации. Протокол TCP/IP является стандартом для сети Интернет.

TransViewer Термин компании Oracle, который применяется для описания модулей Oracle XML Javabeans, включенных в инструментарий XDK for Java. В числе этих программных модулей есть XML Source Viewer bean, Tree Viewer bean, DOM Builder bean, Transformer bean и TransViewer bean.

Uniform Resource Identifier (URI) Унифицированный идентификатор ресурса — синтаксис адреса, который используется для создания URL и XPath.

Uniform Resource Locator (URL) Адрес, который определяет местоположение и путь к какому-то файлу в Интернете. Адреса URL используются браузерами для перемещений по World Wide Web. Они состоят из префикса протокола, необязательного номера порта, доменного имени, названий каталога и подкаталога, а также имени файла. Например, URL-адрес http://technet.oracle.com:80/tech/xml/index.htm определяет местоположение и путь, которым пройдет браузер, чтобы найти в World Wide Web XML-сайт Oracle Technology Network.

URI *СМ,* Uniform Resource Identifier.

URL *СМ.* Uniform Resource Locator.

user interface (UI) Пользовательский интерфейс — комбинация меню, экранов, клавиатурных команд, щелчков мышью и командного языка, который определяет, как пользователь взаимодействует с программным приложением.

valid Допустимый — термин, используемый по отношению к XML-документу, если его структура и содержащиеся элементы совпадают с тем, что утверждается во включенном в его состав DTD или в DTD, ссылка на который есть в XML-документе.

W3C *См.* World Wide Web Consortium (W3C).

WAN *См.* wide area network.

Web Request Broker (WRB) Картридж внутри сервера OAS, который *обрабатывает* URL-адреса и отправляет их в соответствующий картридж.

well formed Правильный — термин, используемый по отношению к XML-документу, соответствующему синтаксису той XML-версии, которая задается в описании XML. Этот синтаксис предполагает наличие одного корневого элемента, соответствующим образом сформированных тегов и т. д.

wide area network (WAN) Компьютерная коммуникационная сеть, которая обслуживает пользователей на довольно обширной территории, например штата или страны. Сеть WAN состоит из серверов, рабочих станций, коммуникационного оборудования (марштутизаторы, мосты, сетевые карты и т. д.) и сетевой операционной системы.

Working Group (WG) Комитет в составе консорциума W3C, который состоит из представителей целого ряда компаний, работающих в компьютерной индустрии. Этот комитет разрабатывает рекомендации для конкретных технологических областей сети Интернет.

World Wide Web Consortium (W3C) Международный промышленный консорциум, созданный в 1994 г. для разработки стандартов для системы World Wide Web. В Интернете его сайт располагается по адресу http://www.w3c.org.

wapper Упаковщик — термин, описывающий структуру данных или программу, которая *оборачивается вокруг* других данных или ПО. Обычно упаковщик используется для реализации объектного интерфейса.

XLink Язык XML Linking, состоящий из правил использования гиперссылок в XML-документах. Эти правила разрабатываются группой XML Linking Group в соответствии с рекомендациями W3C.

XML *См.* Extensible Markup Language.

XML Developer's Kit (XDK) Набор библиотек, компонентов и утилит, которые предоставляют разработчикам ПО основанные на стандартах функции с тем, чтобы они могли реализовать в своих приложениях поддержку XML-технологии. Например, инструментарий Oracle XDK. for Java содержит программу анализа XML-синтаксиса, генератор классов XML Class Generator, программные модули TransViewer Java bean и XSQL Servlet.

XML query Проект стандарта консорциума W3C для языка и синтаксиса запросов к XML-документам

XML Schema Проект стандарта консорциума W3C для описания простых типов данных и сложных структур в XML-документе.

XPath Язык адресации элементов в документе, используемый в языках XSL и XPointer. XPath сейчас является рекомендацией консорциума W3C.

XPointer Термин и рекомендация консорциума W3C для адресации элементов внутри XML-документа. Указатель XPointer может использоваться на конце URI-идентификатора, сформатированного в соответствии с правилами языка XPath.

XSL *См.* Extensible Markup Language.

XSLFO *См.* Extensible Stylesheet Language Formatting Objects.

XSLT *СМ.* Extensible Stylesheet Language Transformation.

XSQL Обозначение, используемое сервлетом Oracle Servlet для динамичных XML-документов, создаваемых в результате выполнения одного или нескольких SQL-запросов. Сервлет может преобразовывать этот документ с использованием XSL-таблицы стилей.