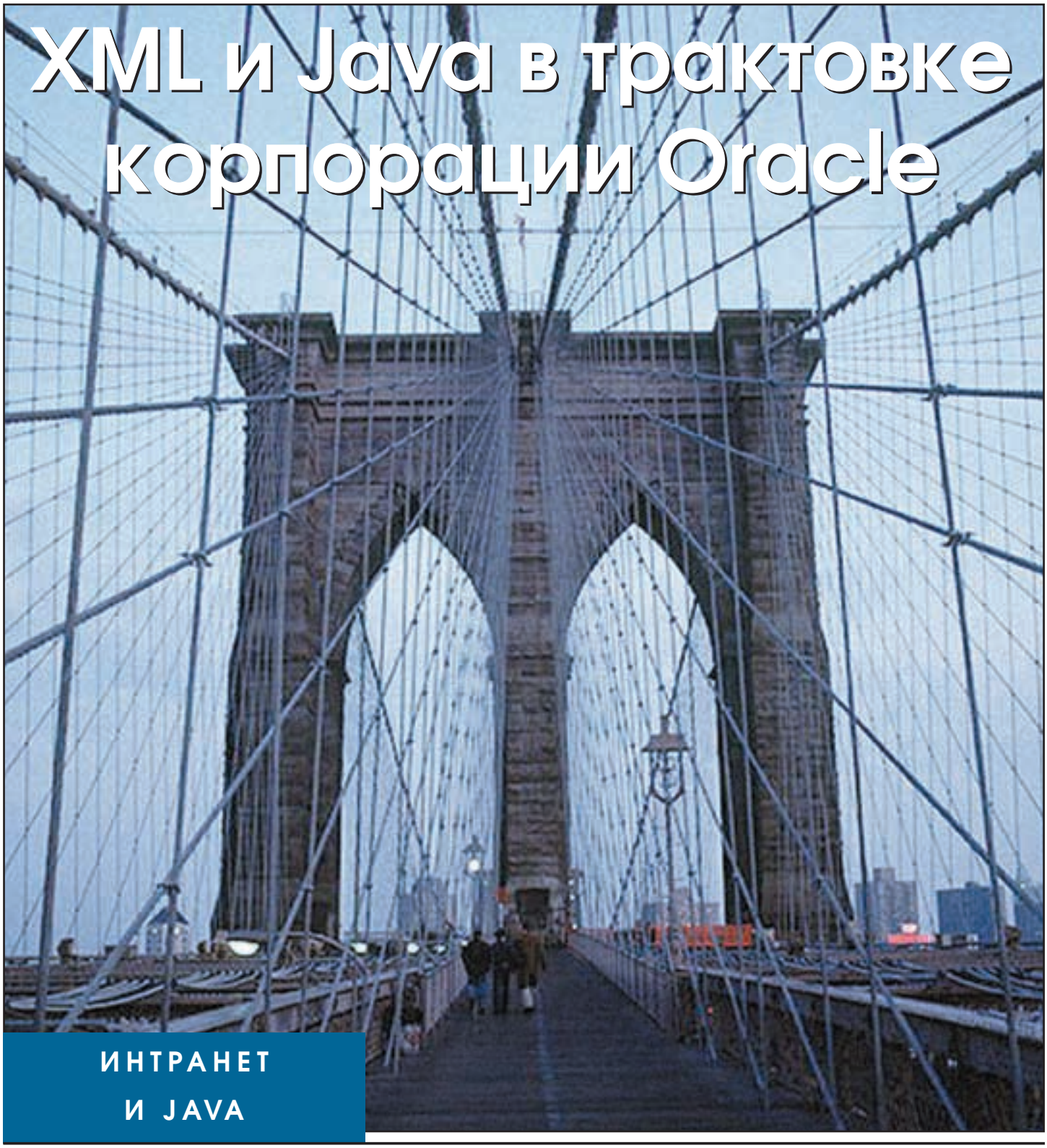
Jet Info

ИНФОРМАЦИОННЫЙ БЮЛЛЕТЕНЬ № 4 (83)/2000

XML и Java в трактовке корпорации Oracle



ИНТРАНЕТ И JAVA

XML и Java в трактовке корпорации Oracle

Глеб Ладыженский

СОДЕРЖАНИЕ

Наш комментарий

В последнее время любое околокомпьютерное издание в обязательном порядке публикует статьи, в названиях которых присутствует новое слово из трех букв — XML. Неизбежное следствие такой популярности — появление публикаций, которые не только ничего не объясняют, но, что гораздо хуже, плодят мифы и легенды.

Положение усугубляется тем, что хотя в основе XML лежат простые и понятные идеи, воплощение этих идей в стандарты, спецификации и программное обеспечение уже далеко не тривиально.

Если отвлечься от технических деталей, то в XML есть две основные идеи.

Во-первых, XML — это способ разделить содержание, логическую структуру и визуальное представление документа. О том, что дает такое разделение в практическом плане, написано достаточно много.

Во-вторых, XML есть способ работать с документом не как с текстом, а как с полноценной структурой данных, обладающей определенным типом и снабженной совокупностью стандартных методов для работы.

Статья, опубликованная в данном номере Jet Info, называется «XML и Java в трактовке корпорации Oracle». Она посвящена сугубо практическим аспектам применения технологий XML в информационных системах, более конкретно, использованию XML для разработки приложений, работающих с СУБД Oracle.

Так сложилось, что большая часть инструментария, предназначенного для разработки XML-приложений, была выполнена с помощью языка Java. Сюда относятся и грамматические анализаторы, и XSL-процессоры (подробнее о них в статье) и реализации спецификации DOM. Это привело к тому, что связка XML-Java стала довольно устойчивой, хотя надо оговориться, что изначально ни XML не создавался «под Java», ни наоборот.

Хорошо известно, что компания Oracle уделяет большое внимание применению java-технологий в своих продуктах. Не стали исключением и XML-ориентированные компоненты Oracle. Поэтому читатель, поверивший названию статьи, не будет обманут в своих ожиданиях, поскольку она достаточно подробно и с приложением конкретных примеров показывает одно из возможных и весьма перспективных применений XML при построении информационных систем.

Александр Таранов,
начальник отдела Интранет-технологий

1. Введение

В первом номере информационного бюллетеня Jet Info за 2000 год была опубликована статья [1], посвященная языкам разметки документов. Большая часть статьи касалась языка XML (eXtensible Markup Language), не сходящего ныне со страниц компьютерных изданий. На наш взгляд, такая популярность вполне заслужена, если трактовать XML не только и не столько как собственно язык разметки, но и как универсальный формат для обмена электронными документами.

Мы посчитали возможным и важным продолжить развитие темы XML, переведя обсуждение в более практическую плоскость. Возможно, читателям, получившим первоначальные и очень важные сведения из упомянутой статьи, будет интересно узнать, как реализована обработка документов в конкретных программных продуктах и, в том числе, в продуктах корпорации Oracle. Наша задача облегчена тем, что в общем-то достаточно сложная для понимания тема была изложена в статье [1] ясно и доходчиво. Были введены основные понятия, рассмотрены полезные примеры, что заложило основу для дальнейшего развития темы в техническом направлении, что мы и попытаемся сделать в предлагаемой вниманию читателя статье. При этом мы будем использовать предложенную в [1] русскоязычную терминологию.

Мы сконцентрируем внимание на одном из направлений, отмеченных в [1] как «XML и Java». Действительно, новый подход к программной обработке документов может заключаться в генерации Java-классов, исходя из структурированных посредством XML документов. Порожденный таким образом интерфейс прикладного программирования предоставляет прямой, а не опосредованный, доступ к содержанию документа и закладывает основу использования XML-данных для взаимодействия приложений.

Статья организована как набор примеров, разъясняющих механизмы обработки XML-документов. Эти примеры (с небольшими изменениями) заимствованы из [2].

Логика статьи выстроена следующим образом. Вначале рассказано обо всем, что необходимо для организации взаимодействия приложений с использованием XML, причем изложение иллюстрируется средствами из арсенала Oracle. Затем рассматривается модель взаимодействия приложений посредством XML-данных.

Предполагается, что читатель знаком с языком программирования Java и основными терминами технологии программирования Java (сервлеты, Java server pages и т.д.). При необходимости соответствующие сведения можно почерпнуть в статье [3].

2. Основные понятия

Для тех, кто не читал [1], мы кратко изложим основные понятия, настоятельно рекомендуя, впрочем, ознакомиться с упомянутой статьей.

2.1. HTML и XML

XML — расширяемый язык разметки — быстро становится стандартом для идентификации и описания данных в рамках Web-технологии. XML является подмножеством давно существующего, но не получившего достаточно широкого распространения языка SGML (Structured Generalized Markup Language). XML — это метаязык, с помощью которого могут быть определены конкретные языки разметки.

Важное отличие разметки, сделанной с помощью XML, от HTML состоит в том, что HTML в значительной степени предназначен для описания внешнего представления документа в Web-навигаторе, в то время как задача XML — описание структуры и семантики документа. Если тэги HTML — это, зачастую, инструкции для визуализации содержания документа Web-навигатором, то тэги XML определяют структуру и смысл того, что они обрамляют. Так, в HTML запись

```
<bold>Oracle</bold>
```

означает, что при визуализации слова Oracle оно будет выделено жирным шрифтом. Напротив, в XML запись

```
<company_name>Oracle</company_name>
```

означает, что слово Oracle будет интерпретироваться как имя компании (разумеется, при условии, что тэг **<company_name>** изначально был предназначен для задания имени компании и ни для чего иного).

В листинге 1 приведено описание на языках HTML и XML простого документа, содержащего таблицу с данными о сотрудниках организации — идентификационный номер, имя, наименование должности и заработная плата.

Ключевым преимуществом XML по сравнению с HTML является то, что в XML описание внешнего представления документа отделено от его структуры и содержания. Для задания внешнего представления документов используются стили (stylesheet). XML-документ может быть представлен в различных вариантах, которые определяются примененными к нему стилями.

Очевидно, что для описания стилей также необходимо иметь некоторый язык. В качестве такого языка консорциумом W3 предложен XSL (eXtensible Stylesheet Language). Он позволяет создавать стили, служащие целям трансформации

```
<table>
  <tr><td>EMPNO</td><td>ENAME</td><td>JOB</td><td>SAL</td></tr>
  <tr><td>7654</td><td>MARTIN</td><td>SALESMAN</td><td>1250</td></tr>
  <tr><td>7788</td><td>SCOTT</td><td>ANALYST</td><td>3000</td></tr>
</table>

<?xml version="1.0"?>
  <EMPLIST>
    <EMP>
      <EMPNO> 7654 </EMPNO>
      <ENAME> MARTIN </ENAME>
      <JOB> SALESMAN </JOB>
      <SAL> 1250 </SAL>
    </EMP>
    <EMP>
      <EMPNO> 7788 </EMPNO>
      <ENAME> SCOTT </ENAME>
      <JOB> ANALYST </JOB>
      <SAL> 3000 </SAL>
    </EMP>
  </EMPLIST>
```

Листинг 1. Сопоставление HTML и XML.

XML-документов в другие форматы (например, в HTML или текстовый формат).

2.2. Расширяемость и определение типов документов

Другим важным преимуществом XML по сравнению с HTML является то, что XML разрешает пользователю задавать собственные спецификации тэгов. То есть, пользователь может создавать свои собственные тэги для того, чтобы адекватно представлять структуру данных, с которыми он работает. Создаваемые пользователем тэги могут быть определены двумя путями. Во-первых, непосредственным заданием тэга в теле самого документа. Во-вторых, они могут быть формально определены в специальной структуре, которая называется DTD (Document Type Definition).

Тэги, определяемые пользователем, придают документу необходимую гибкость. Так, если в приложении, которое работает с документом, появляются данные нового типа либо изменяется уже существующий тип данных, то пользователь может внести соответствующие изменения в описание документа, добавив новый тэг или изменив существующий. Пример DTD для упомянутого выше XML-документа приведен в листинге 2.

XML-документы, соответствующие формальным правилам, основные из которых приведены ниже, называются структурно корректными (well-formed). Такие XML-документы не имеют описания в виде DTD, но, тем не менее, исчерпывающе определяют собственные элементы данных и их отношения между собой. Правильный XML-документ обязан удовлетворять следующим требованиям (полностью требования перечислены в [1]):

- документ начинается с декларации XML <?xml version=»1.0»?>;
- все элементы организованы в древовидную структуру;
- все непустые элементы имеют тэги начала и завершения.

Все структурно корректные документы, имеющие описание тэгов в виде DTD, называются синтаксически корректными (valid). В процессе анализа (разбора) XML-документа, ссылающегося на соответствующую DTD-структуру или содержащего ее внутри себя, программа, выполняющая анализ документа (программа-анализатор, parser), проверяет, в первую очередь, соответствует ли разбираемый XML-документ его описанию в DTD.

Если XML-документ записывается в базу данных, то DTD может быть использован для установления соответствия между элементами XML и колонками реляционных таблиц. Если XML-документ ге-

```
<!DOCTYPE EMPLIST[
  <!ELEMENT EMPLIST (EMP)*>
  <!ELEMENT EMP (EMPNO, ENAME, JOB,
  SAL)>
  <!ELEMENT EMPNO (#PCDATA)>
  <!ELEMENT ENAME (#PCDATA)>
  <!ELEMENT JOB (#PCDATA)>
  <!ELEMENT SAL (#PCDATA)>
 ]>
```

Листинг 2. Пример DTD.

нерируется посредством извлечения из базы данных некоторых значений (то есть фактически XML-документ конструируется по схеме базы данных), то он, разумеется, будет синтаксически корректным.

2.3. Адаптированное представление данных

В настоящее время XML становится все более популярным как средство настройки представления данных для различных навигаторов, и специфических устройств, а также в широком смысле — для прикладных программ и пользователей. Используя XML-документы совместно со стилями в рамках архитектуры клиент/сервер (то есть как на клиенте, так и на серверах, будь то серверы баз данных или серверы приложений), мы можем организовывать, трансформировать и представлять данные, сформированные для потребностей конкретного пользователя, на широком классе различных устройств.

3. Поддержка XML в Oracle

3.1. Арсенал Oracle

Корпорация Oracle поставляет набор компонентов, утилит и интерфейсов для организации работы с XML-документами. Этот набор включает:

- сервер реляционных баз данных Oracle8i;
- расширение сервера interMedia (см. далее);
- программы-анализаторы XML-документов для Java, C, C++, PL/SQL (XML Parsers);
- XSLT-процессоры (см. далее);
- генераторы XML-классов (Java и C++),

а также специальные средства:

- XML SQL Utility for Java;
- XSQL Servlet;
- XML Transviewer Beans (в данной статье не рассматривается).

3.2. Стратегии хранения XML-документов в объектно-реляционной базе данных

Существуют три базовых стратегии хранения XML-документов в объектно-реляционной базе данных:

- Хранение XML-документов (вместе с тэгами, то есть полностью) как отдельных неделимых объектов. Документы хранятся как данные типа CLOB или BLOB (см. [4]).
- Хранение элементов XML-документов как данных (без тэгов) в объектно-реляционном представлении (фактически — в таблицах реляционной базы данных).
- Смешанное хранение документов и данных с использованием представлений (views).

Рассмотрим каждую стратегию более подробно.

Хранение XML-документов в базе как неделимых объектов подходит в том случае, когда их содержание статично и, что существенно, любое обновление документа сводится к его полной перезаписи. Типичные примеры таких документов — статьи, книги, технические руководства, контракты и т.д. Это документы в обычном смысле, они хранятся в базе данных целиком и поставляются из нее вовне также целиком.

Oracle умеет хранить документы такого типа в различных форматах (MS Word, WordPerfect, Acrobat и т.д.). Более того, возможно проводить по ним эффективный изощренный поиск, в том числе с использованием морфологии русского языка. Для этого нужно только установить опцию сервера под названием interMedia (сейчас она уже включена в состав сервера и лицензируется вместе с ним), а также добавить к нему программный продукт Russian ConText Optimizer (его можно приобрести у компании Гарант-Парк). В смысле особенностей хранения и обработки XML-документы ничем не отличаются от документов других форматов и сервер Oracle хранит их как большие объекты, не делая различия между ними и, например, документами в формате MS Word.

Если документ структурно корректен и содержит элементы, которые могут обновляться и использоваться по отдельности, а не как единое целое, то такой документ можно назвать датацентрическим (приносим извинения за слово-кальку с data-centric). Обычно подобные документы включают один или несколько элементов со сложной структурой (называемых микродокументами). Примерами могут служить бланки заказов, финансовые счета и т.д., то есть документы на базе сложных форм. Сервер Oracle8i предоставляет адекватные структуры для хранения и обработки элементов сложных документов. Речь идет об объектах в базе данных Oracle, конкретно — об определяемых типах, ссылках и коллекциях (collections). Возможны два варианта отображения структурированных XML-документов в объектно-реляционные структуры базы данных Oracle:

- хранение атрибутов и содержимого элементов XML-документов только в таблицах базы данных и использование объектных представлений для воспроизведения структуры XML-документов;
- хранение структурированных элементов XML в объектных таблицах.

Будучи сохраненными в объектно-реляционной базе данных, элементы документа становятся потенциальными операндами различных операций, таких как выборка, обновление и т.д., осуществляемых с помощью операторов языка SQL. Собственно

процедура отображения документа в объектно-реляционную базу данных, равно как и различные поисковые операции над данными-элементами документа, выполняются программой XML SQL Utility (о ней подробнее будет сказано ниже).

Если документ структурирован, но его структура в целом не соответствует схеме базы данных, необходимо преобразовать документ к нужному формату до его записи в базу. Этого можно достичь посредством механизма стилей.

Наконец, если необходимо обрабатывать документы смешанных типов, когда имеются как структурированные, так и неструктурированные данные в формате XML, рассматриваемые, тем не менее, как единый документ, целесообразно использовать представления (view). Они позволяют конструировать объекты «на лету», комбинируя данные, которые хранятся в различных видах. Таким образом, можно хранить структурированные данные (такие, например, как данные о сотрудниках, заказчиках и т.д.) с использованием объектно-реляционных таблиц, а неструктурированные данные (такие как описания и комментарии) — как данные типа CLOB. Когда необходимо обновить данные в целом, можно попросту создать структуру из различных «кусочков» с использованием конструктора типов в операторе SELECT, примененном к view. Утилита XML SQL даст возможность поиска сконструированных данных в view.

3.3. Программы-анализаторы XML-документов

Oracle поставляет набор программ-анализаторов XML-документов для среды Java, C, C++ и PL/SQL (см. рис. 1). Каждый из них представляет собой отдельно устанавливаемый компонент, который анализирует (разбирает) XML-документ (или DTD) таким образом, что далее с ним (документом) может продолжить работу некоторое приложение. Все программы-анализаторы поддерживают интерфейсы DOM (Document Object Model) и SAX (Simple API for XML), механизм XML Namespace, обеспечивают проверку структурной и синтаксической корректности. Программы-анализаторы доступны на всех платформах, где работает Oracle Server.

3.4. Поддержка XSL-трансформаций

Версия 2 программ-анализаторов XML-документов включает специальную утилиту для трансформации XML-данных с использованием механизма стилей. Это — так называемый XSL Transformation (XSLT) Processor или, для краткости, XSLT-процессор (см. рис. 2). Используя его, мы получаем возможность трансформации документов различных форматов, как-то: XML в XML, в HTML или любой иной текстовый формат.

3.5. Области действия имен

Для адекватного именования элементов XML-документов вводятся так называемые области действия имен — XML Namespaces. Идея состоит в том, что каждый XML-документ получает собственное пространство имен элементов и атрибутов, в котором имена должны быть уникальными. Однако в ряде случаев может потребоваться распространить действие некоторых имен за пределы конкретного документа и применять его для других документов.

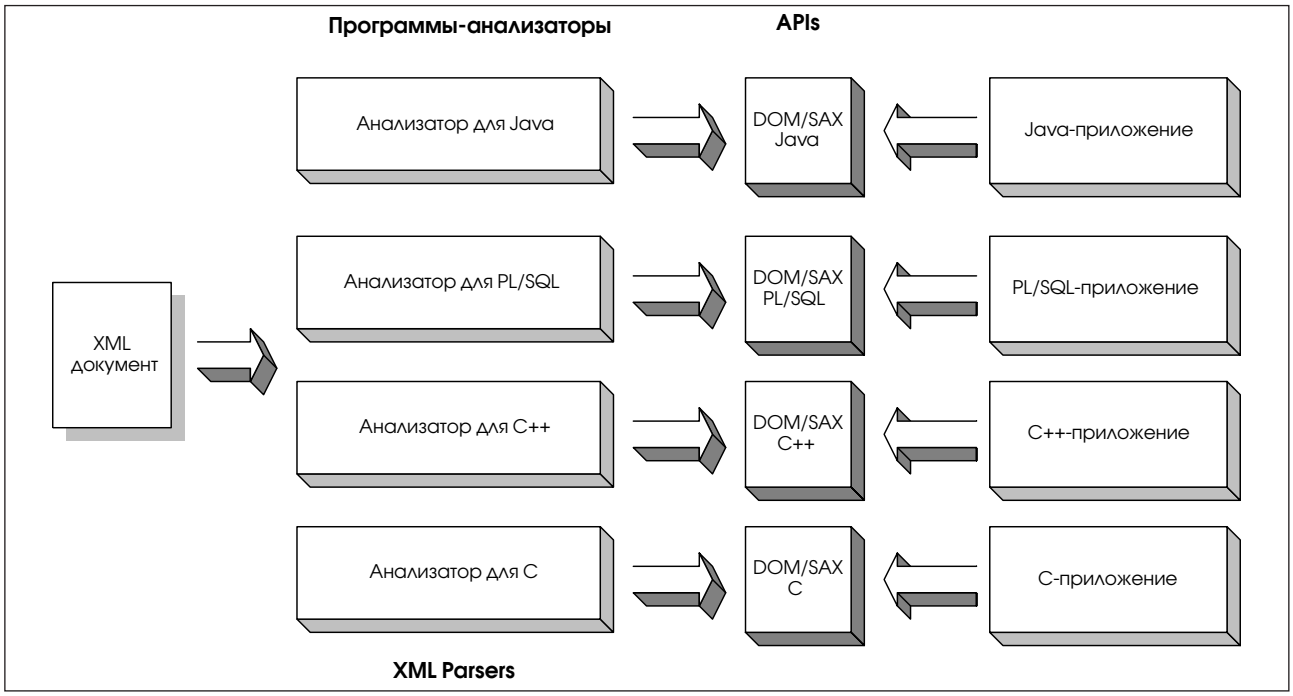


Рис. 1. Анализаторы XML-документов.

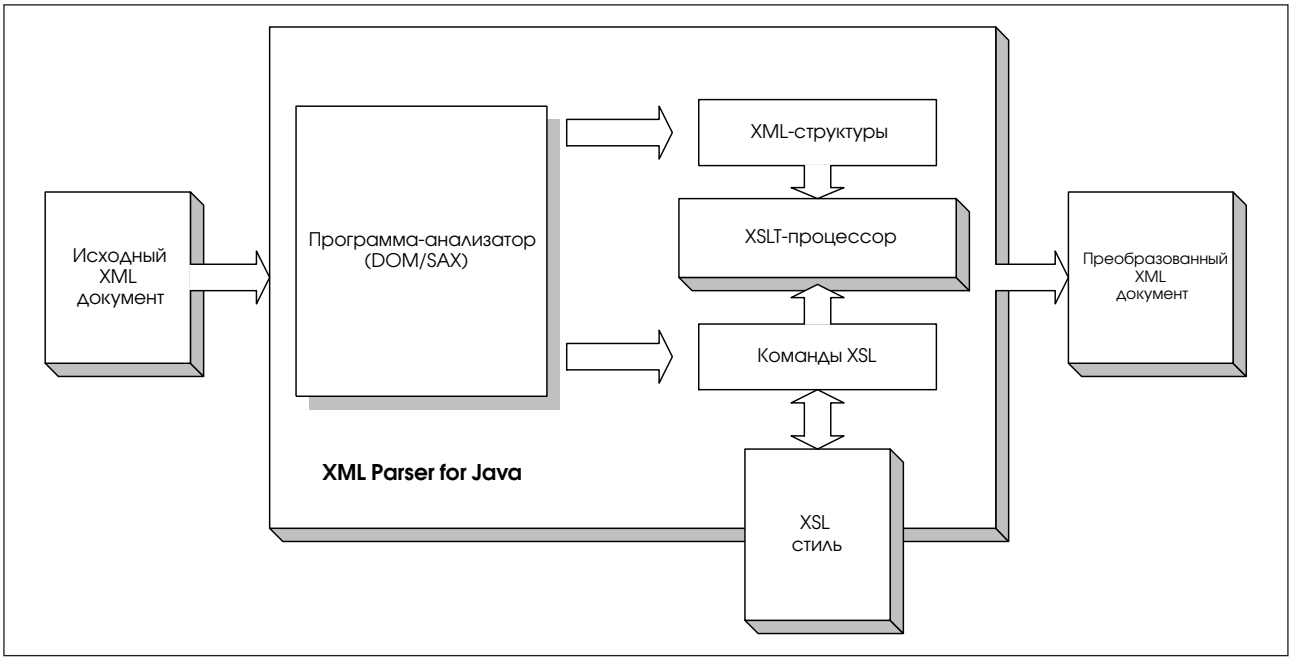


Рис. 2. Схема XSL-трансформаций.

XML Namespace — это механизм для разрешения конфликтов между именами элементов (тэгами) или атрибутов в документах XML. Механизм обеспечивает универсальное именование объектов (элементов и атрибутов), чья область действия выходит за пределы конкретного документа. Такие пространства определяются при помощи универсальных идентификаторов ресурсов (Uniform Resource Identifiers — URI).

Анализаторы XML-документов поддерживают механизм XML Namespaces за счет умения распознавать и анализировать имена элементов и атрибутов, равно как и квалифицировать их соответствующим образом, то есть как универсальные либо локальные.

3.6. Проверка корректности документов

Программы-анализаторы XML-документов используются, помимо своего основного назначения — разбора XML-документов, также для проверки структурной и синтаксической корректности документов. Для этого предусмотрены два режима (non-validating и validating).

В первом случае программа-анализатор проверяет, что XML-документ является структурно корректным и собирает данные в дерево объектов, которыми можно манипулировать посредством вызовов DOM API. Во втором случае программа-анализатор проверяет, является ли документ синтаксически корректным и сравнивает данные XML с соответствующим DTD. Проверяется, например, являются ли типы элементов и атрибуты разрешенными (легальными), находятся ли вложенные элементы под элементами, которым они принадлежат, и т.д.

4. Интерфейсы прикладного программирования

В программах анализа XML-документов интерфейс прикладного программирования распадается на две категории:

- интерфейс, ориентированный на события;
- интерфейс, служащий для манипулирования древовидными структурами.

Прежде чем переходить к описанию этих интерфейсов, посмотрим, в чем же заключается задача прикладной программы, использующей один из двух перечисленных вариантов API. Собственно говоря, программа-анализатор не делает ничего иного, кроме как строит в собственном пространстве памяти некоторую (как мы увидим, не обязательно древовидную) структуру, позволяющую оперировать с ней некоторому приложению с конкретными целями — например, с целью преобразования структуры документа.

Иными словами, приложение, желающее работать с XML-документом, обязано вызвать предварительно программу-анализатор и получить с ее помощью требуемую структуру, а затем оперировать с ее элементами способом, зависящим от типа используемой программы-анализатора.

Суть API, основанного на событиях, состоит в интерпретации структуры документа как цепочки

```
<?xml version="1.0"?>
  <EMPLIST>
    <EMP>
      <ENAME> Martin </ENAME>
    </EMP>
    <EMP>
      <ENAME> Scott </ENAME>
    </EMP>
  </EMPLIST>
```

Листинг 3. Пример простого XML-документа.

```
start document
start element: EMPLIST
start element: EMP
start element: ENAME
characters: Martin
end element: EMP
start element: EMP
start element: ENAME
characters: Scott
end element: EMP
start element: EMPLIST
end document
```

Листинг 4. Структура XML-документа в виде цепочки событий.

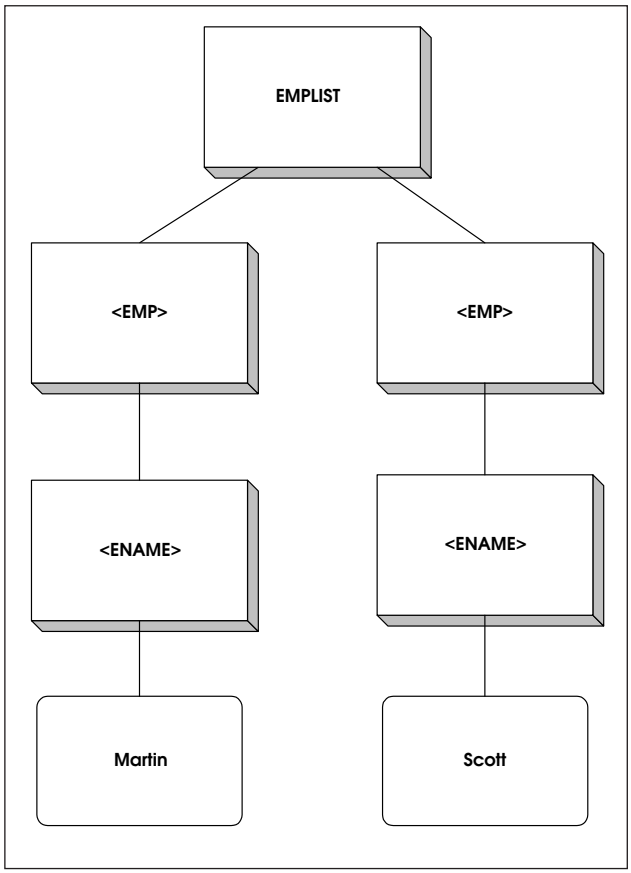


Рис. 3. Древовидная структура XML-документа.

событий начала и завершения элементов в определенном контексте, который достаточно прост. Событие начала элемента означает, что далее должно следовать содержимое этого элемента, событие завершения — что обработку содержимого следует прекратить и т.д.

Например, XML-документ, представленный в листинге 3, преобразуется в линейную последовательность событий, показанную в листинге 4.

В отличие от API, опирающихся на деревья, событийно-ориентированные интерфейсы не воспроизводят в памяти древовидное представление XML-документа. В основном, SAX API полезны для приложений, которым не нужно манипулировать подобным представлением при выполнении таких операций, как, скажем, поиск.

Интерфейсы прикладного программирования, оперирующие с деревьями (такие как DOM), направлены на построение в памяти древовидного представления XML-документов. Они предоставляют классы и методы, позволяющие приложению манипулировать элементами дерева и перемещаться по нему. Ясно, что DOM API полезен для таких манипуляций, как перестановка элементов, добавление или удаление элементов и атрибутов, переименование элементов и т.д. Например, взяв XML-документ, представленный в листинге 3, DOM API сгенерирует структуру, показанную на рис. 3.

Теперь мы готовы приступить к рассмотрению (в этом и следующих разделах) относительно длинной серии примеров программ на языке Java.

4.1. Пример 1: программа-анализатор XML-документов для Java

В листинге 5 приведен пример программы на языке Java, в которой использован анализатор XML-документов (пакет oracle.xml.parser.v2). Суть программы проста. Ее аргументами являются имя файла с описанием XML-документа и имя файла, хранящего стиль, который предстоит применить к документу. Программа принимает имена файлов, формирует URL для документа и стиля и последовательно передает их (URLs) анализатору (класс DOMparser). Далее порождается объект класса XSLProcessor, который, собственно, и применяет стиль xsl к XML-документу xml (метод processXSL). Результат операции прописывается в result. Далее result подстраивается в структуру (начиная от корневого элемента) результирующего документа out.

Обратим внимание на то, что out — это объект класса XMLDocument. С ним можно работать, применяя методы классов из пакета, реализующего DOM API (пакет org.w3c.dom). Таковыми являются типичные операции с элементами XML-документа (`createElement()`) или над древовидной структурой (`appendChild()`).

```
import org.w3c.dom.*;
import java.util.*;
import java.io.*;
import java.net.*;
import oracle.xml.parser.v2.*;
public class XSLTransform
{
  public static void main (String args[]) throws Exception
  {
    // Анализатор на базе DOM API
    DOMParser parser;
    // Используемые XML, XSL
    XMLDocument xml, xsldoc, out;
    URL xslURL;
    URL xmlURL;
    try
    {
      if (args.length != 2)
      {
        System.err.println("Usage: java XSLTransform xslfile xmlfile");
        System.exit(1);
      }
      // Проанализировать XSL и XML-документы
      parser = new DOMParser();
      parser.setPreserveWhitespace(true);
      xslURL = createURL(args[0]);
      parser.parse(xslURL);
      xsldoc = parser.getDocument();
      xmlURL = createURL(args[1]);
      parser.parse(xmlURL);
      xml = parser.getDocument();
      // Создать экземпляр стиля
      XSLStylesheet xsl = new XSLStylesheet(xsldoc, xslURL);
      // Создать XSL-процессор
      XSLProcessor processor = new XSLProcessor();
      // Вывести возможные предупреждения
      processor.showWarnings(true);
      processor.setErrorStream(System.err);
      // Имеем готовые xsl и xml-документ
      // Применить xsl к xml-документу записать в result
      DocumentFragment result = processor.processXSL(xsl, xml);
      // Создать выходной документ
      out = new XMLDocument();
      // Создать корневой элемент для выводимого документа
      Element root = out.createElement("root");
      out.appendChild(root);
      // Добавить преобразованный документ в дерево
      root.appendChild(result);
      // Печатать преобразованный документ
      out.print(System.out);
    }
    catch (Exception e)
    {
      e.printStackTrace();
    }
  }
}
```

Листинг 5. Анализ XML-документа на Java.

5. Генератор классов XML для Java

До сих пор мы говорили о самоопределенных, структурно корректных документах. Однако, было бы очень полезно уметь порождать программные единицы на основе данных, предоставляемых нам DTD. Пусть, например, существуют два приложения и предполагается, что они будут взаимодействовать, используя для этого XML-данные.

Было бы правильно, если оба эти приложения работали на основе DTD, единообразно и исчерпывающе задающего рабочий XML-документ. Однако, сам по себе DTD в том виде, в каком он представлен (см. листинг 6), неудобен для этой цели. Возникает идея сгенерировать на основе DTD, имеющего очевидно классовую структуру, интерфейс прикладного программирования для Java. Тогда наши приложения могли бы пользоваться этим сгенерированным API для непосредственной и, что важнее, стандартной работы с документом. Эта идея и положена в основу генератора классов для DTD.

Генератор создает исходные файлы из XML DTD. Это полезно, например, когда приложение хочет послать XML-сообщение другому приложению, опираясь на согласованный DTD. Используя эти классы, Java-приложение может конструировать, проверять и печатать XML-документы, которые совместимы с входным DTD. Генератор классов работает в соединении с анализатором XML для Java, который разбирает DTD и передает разобранный документ генератору классов (как это показано на рис. 4).

5.1. Пример 2: генератор классов XML для Java

Пример показывает, как генератор классов для Java может быть использован для обработки DTD и генерации классов для элементов DTD. Далее мы увидим, как можно использовать методы сгенерированных классов программным путем с тем, чтобы сконструировать синтаксически корректный XML-документ.

5.1.1. Исходный DTD

В листинге 6 приводится DTD, представляющий XML-документ с данными по служащим некоторой организации. Будем считать, что DTD хранится в файле employee.dat. Этот файл используется в качестве входного для генератора классов.

Посмотрим более детально, что же определяет DTD. Первая строка — строка комментария. Далее определяется, что имеется два элемента — EMP и EMP_ROW, причем первый является корневым и содержит один или более элементов EMP_ROW. Каждый элемент EMP_ROW содержит обязательный атрибут EMPNO (номер служащего), так же как и некоторые опциональные атрибуты, такие как ENAME (имя служащего), JOB (занимаемая должность), MGR (имя руководителя) и т.д. Опциональные (необязательные) атрибуты обозначаются знаком вопроса.

5.1.2. Обработка DTD для генерации Java-классов

В листинге 7 представлен код, обрабатывающий DTD (см. листинг 6) и генерирующий соответ-

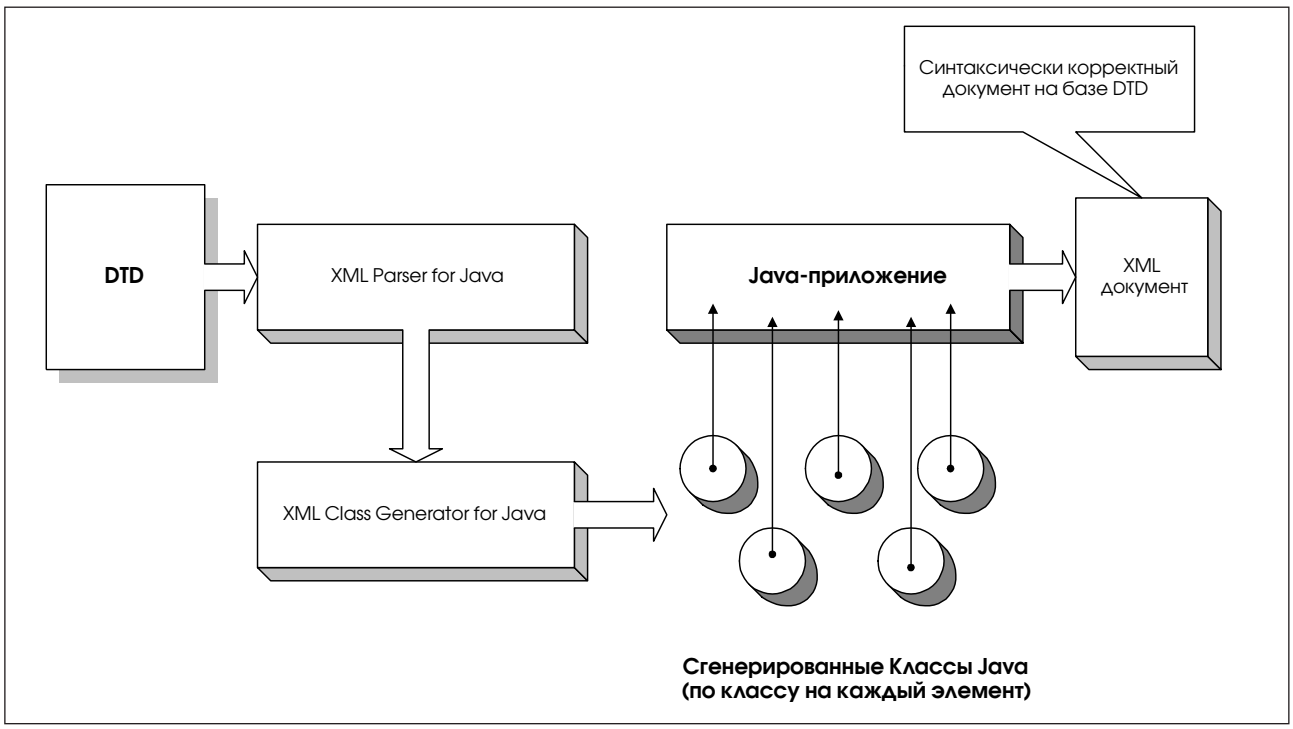


Рис. 4. Генератор классов для Java.

```
<!-- DTD for Employee Data -->
<!ELEMENT EMP (EMP_ROW)*>
<!ELEMENT EMP_ROW (EMPNO, ENAME?, JOB?, MGR?, HIREDATE?, SAL?, COMM?, DEPTNO?)>
<!ATTLIST EMP_ROW ROWNO CDATA #REQUIRED>
<!ELEMENT EMPNO (#PCDATA)>
<!ELEMENT ENAME (#PCDATA)>
<!ELEMENT JOB (#PCDATA)>
<!ELEMENT MGR (#PCDATA)>
<!ELEMENT HIREDATE (#PCDATA)>
<!ELEMENT SAL (#PCDATA)>
<!ELEMENT COMM (#PCDATA)>
<!ELEMENT DEPTNO (#PCDATA)>
```

Листинг 6. Пример DTD XML-документа.

ствующие классы для элементов DTD. Классы создаются для каждого элемента (EMP, EMP_ROW, EMPNO, ENAME и т.д.). Java-приложение далее может использовать методы классов для создания структурно корректных документов, содержащих данные о сотрудниках организации.

5.2. Пример 3: создание синтаксически корректных XML-документов из Java-классов

Приведенный в листинге 8 пример показывает, как можно использовать сгенерированные в

```
import java.io.*;
import java.net.*;
import oracle.xml.parser.*;
import oracle.xml.classgen.*;
import org.w3c.dom.Element;

public class SampleMain
{
  public SampleMain()
  {
  }
  public static void main (String args[])
  {
    // Проверить аргументы
    if (args.length < 1)
    {
      System.out.println("Usage: java SampleMain " +
                         "[-root <rootName>] <fileName>");
      System.out.println("fileName\t Input file, XML document or " +
                         "external DTD file");
      System.out.println("-root <rootName>  Name of the root Element " +
                         "(required if the input file is an external DTD)");
      return;
    }
    try // Открыть DTD-файл
    {
      // Создать экземпляр анализатора
      XMLParser parser = new XMLParser();

      if (args.length == 3)
        parser.parseDTD(fileToURL(args[2]), args[1]);
      else
        parser.parse(fileToURL(args[0]));

      XMLDocument doc = parser.getDocument();
```

Листинг 7. Генерация классов для DTD (продолжение на стр. 12).

предыдущем примере классы. Для простоты создадим две новые записи о сотрудниках организации. Фактически создаются экземпляры классов EMP (список сотрудников), EMP_ROW (запись о сотруднике) и атрибуты сотрудника (EMPNO, ENAME и т.д.). Для построения дерева, отражающего структуру XML-документа, элементы данных группируются посредством отнесения их к строковым элементам. Каждый строковый элемент затем добавляется как узел к корневому элементу документа

```
      DTD dtd = (DTD)doc.getDoctype();
      String doctype_name = null;

      if (args.length == 3)
      {
        doctype_name = args[1];
      }
      else
      {
        // Получить имя корневого элемента из XMLDocument
        doctype_name = doc.getDocumentElement().getTagName();
      }

      // Сгенерировать Java-классы
      ClassGenerator generator = new ClassGenerator();

      // Будем генерировать и комментарии
      generator.setGenerateComments(true);
      // Установить каталог, в который будет направлен вывод
      generator.setOutputDirectory(".");
      // Установить validating mode
      generator.setValidationMode(true);

      // Сгенерировать исходный код Java
      generator.generate(dtd, doctype_name);
    }
    catch (Exception e)
    {
      System.out.println ("XML Class Generator: Error " + e.toString());
      e.printStackTrace();
    }
  }

  static public URL fileToURL(String sfile)
  {
    File file = new File(sfile);
    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");
    }
  }
}
```

Листинг 7. Генерация классов для DTD (окончание).

```
import oracle.xml.classgen.*;
import oracle.xml.parser.*;

public class CreateEmployees
{
  public static void main (String args[])
    {
      try
      {
        EMP EMPLIST = new EMP();

        // get static from base document
        DTD dtd = EMPLIST.getDTDNode();

        // Запись о сотруднике emp_row1
        // Создать запись и установить ROWNO
        EMP_ROW emp_row1 = new EMP_ROW(1);
        EMPNO empno1 = new EMPNO("7654");
        ENAME ename1 = new ENAME("MARTIN");
        JOB job1 = new JOB("SALESMAN");
        MGR mgr1 = new MGR("7698");
        HIREDATE hiredate1 = new HIREDATE("1981-09-28");
        SAL sal1= new SAL("1250");
        COMM comm1= new COMM("1400");
        DEPTNO deptno1 = new DEPTNO("30");

        // Запись о сотруднике emp_row2
        // Создать запись и установить ROWNO
        EMP_ROW emp_row2 = new EMP_ROW(2);
        EMPNO empno2 = new EMPNO("7788");
        ENAME ename2 = new ENAME("SCOTT ");
        JOB job2 = new JOB("ANALYST ");
        MGR mgr1 = new MGR("7566");
        HIREDATE hiredate2 = new HIREDATE("1987-04-19");
        SAL sal2= new SAL("3000");
        COMM comm2= new COMM("");
        DEPTNO deptno2 = new DEPTNO("20");

        // Добавить данные как потомки узла emp_row1
        emp_row1.addnode(empno1);
        emp_row1.addnode(ename1);
        ...

        // Добавить данные как потомки узла emp_row2
        emp_row2.addnode(empno2);
        emp_row2.addnode(ename2);
        ...

        // Добавить узел emp_row1 как потомок EMPLIST
        EMPLIST.addNode(emp1);

        // Добавить узел emp_row2 как потомок EMPLIST
        EMPLIST.addNode(emp2);

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

Листинг 8. Пример Java-приложения, генерирующего XML-документ.

```
<?xml version="1.0"?>
<!DOCTYPE EMP SYSTEM "employee.dtd">
<EMP>
  <EMP_ROW ROWNO = "1">
    <EMPNO> 7654 </EMPNO>
    <ENAME> MARTIN </ENAME>
    <JOB> SALESMAN </JOB>
    <MGR> 7698 </MGR>
    <HIREDATE> 1981-09-28 00:00:00.0
    </HIREDATE>
    <SAL> 1250 </SAL>
    <COMM> 1400 </COMM>
    <DEPTNO> 30 </DEPTNO>
  </EMP_ROW>
  <EMP_ROW ROWNO = "2">
    <EMPNO> 7788 </EMPNO>
    <ENAME> SCOTT </ENAME>
    <JOB> ANALYST </JOB>
    <MGR> 7566 </MGR>
    <HIREDATE> 1987-04-19 00:00:00.0
    </HIREDATE>
    <SAL> 3000 </SAL>
    <COMM></COMM>
    <DEPTNO> 20 </DEPTNO>
  </EMP_ROW>
</EMP>
```

Листинг 9. XML-документ, сгенерированный Java-приложением.

EMPLIST. В листинге 8 сгенерированные классы выделены большими буквами.

Приведенное в листинге 8 Java-приложение создаст XML-документ следующей структуры (см. листинг 9).

6. XML SQL Utility for Java

Утилита XML SQL Utility for Java представляет собой набор классов Java, которые:

```
SELECT EMPNO, ENAME FROM EMP WHERE
EMPNO = 7654;
```

Листинг 10. Пример SQL-запроса.

- передают SQL-запрос серверу баз данных и генерируют XML-документ, исходя из результирующего набора данных, возвращаемого по запросу (result set);
- записывают данные XML в соответствующие таблицы базы данных.

6.1. Генерация XML-документов на основе результатов SQL-запроса

Как показано на рис. 5, XML SQL Utility обрабатывает SQL-запросы и возвращает результат в виде XML-документа.

Структура результирующего XML-документа опирается на внутреннюю структуру схемы базы данных, которая возвращает результат запроса. Колонки таблицы базы данных отображаются в элементы верхнего уровня. Скалярные значения отображаются в элементы с текстом. Объектные типы — в элементы с атрибутами, возникающими как подчиненные элементы. Коллекции отображаются в списки элементов. Ссылки на объекты и ограничения по ссылкам — в XML IDREFs.

Утилита может генерировать как текстовое представление XML-документа, так и представление структуры XML-документа в виде дерева. Последнее целесообразно использовать, если в дальнейшем планируется работать с XML-документом программно, например, трансформировать его в другие форматы с применением XSLT-процессора

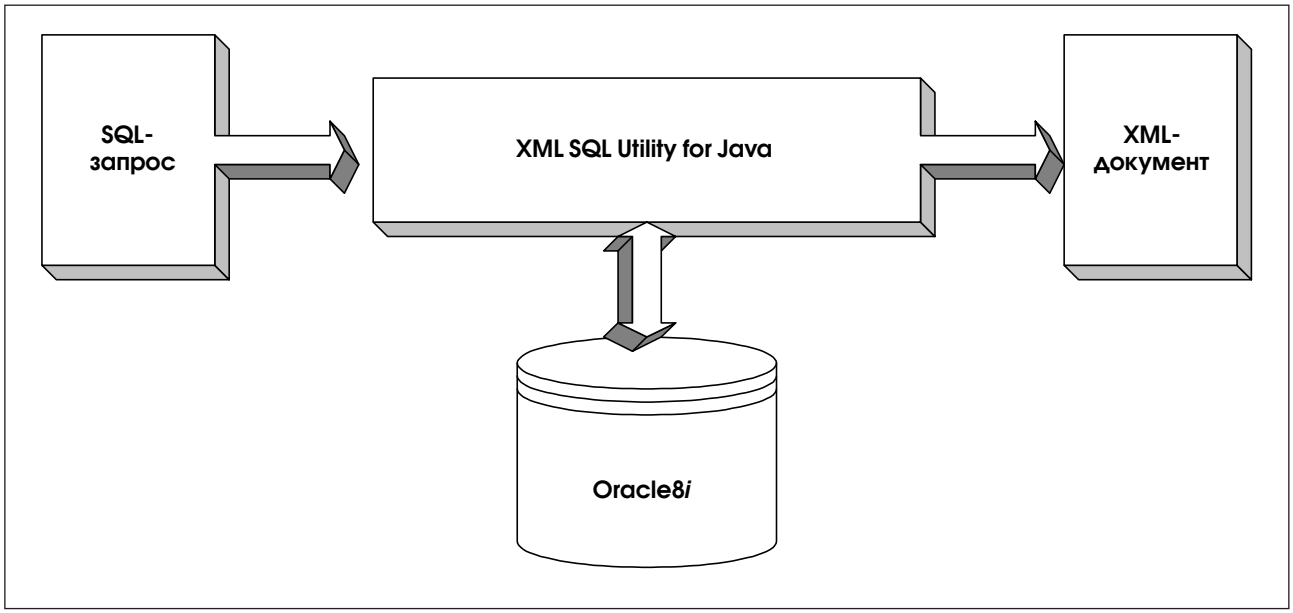


Рис. 5. Логика работы XML SQL Utility.

```
<?xml version="1.0"?>
<ROWSET>
  <ROW id="1">
    <EMPNO>7654</EMPNO>
    <ENAME>MARTIN</ENAME>
  </ROW>
</ROWSET>
```

Листинг 11. XML-документ, сгенерированный по SQL-запросу.

```
SELECT EMPNO, ENAME from EMP
```

Листинг 12. Исходный запрос для примера 5.

или использовать DOM API-методы для поиска в документе или для изменения его структуры.

Утилита также может быть использована для генерации DTD на основе схемы таблицы, к которой был обращен запрос.

```
import java.sql.*;
import java.math.*;
import oracle.xml.sql.query.*;
import oracle.jdbc.*;
import oracle.jdbc.driver.*;

public class xmlquerydb
{
  public static void main(String args[]) throws SQLException
  {
    // Имя таблицы базы данных
    String tabName = "emp";

    // Имя и пароль пользователя
    String user = "scott/tiger";

    DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());

    //Инициировать соединение с БД через JDBC
    Connection conn =
        DriverManager.getConnection("jdbc:oracle:oci8:"+user+"@");

    // Инициировать OracleXMLQuery
    OracleXMLQuery qry = new OracleXMLQuery(conn,"select EMPNO,
        ENAME from "+tabName);

    // Структура генерируемого XML-документа
    // Установить максимальное число возвращаемых строк
    qry.setMaxRows(2);
    // Установить корневой элемент документа
    qry.setRowsetTag("ROOTDOC");
    // Установить разделитель строк
    qry.setRowTag("DBROW");
    // Установить стиль
    qry.setStyleSheet("emp.xsl");

    // Получить XML-документ в строковом формате
    String xmlString = qry.getXMLString();

    // Напечатать XML-документ
    System.out.println(" OUTPUT IS:\n"+xmlString);
  }
}
```

Листинг 13. Код приложения, генерирующего XML-документ по SQL-запросу.

6.2. Пример 4: генерация XML-документа на базе результатов SQL-запроса

В листинге 10 приведен пример простого запроса на языке SQL. К нему применяется XML SQL Utility для создания документа. В результате генерируется XML-документ, представленный в листинге 11.

По умолчанию ROWSET есть имя корневого элемента. ROW есть имя элемента для каждой строки в результате, возвращаемом по запросу. Данные, такие как EMPNO, ENAME, также представлены как элементы, подчиненные элементу ROW.

6.3. Пример 5: генерация XML-документа на базе результатов SQL-запроса и структурирование данных

Используя прикладной программный интерфейс рассматриваемой утилиты, можно также ограничивать данные, представленные в XML-документе. Например, можно определить максимальное число возвращаемых строк.

```
<?xml version="1.0"?>
<ROOTDOC>
  <DBROW id="1">
    <EMPNO>7876</EMPNO>
    <ENAME>ADAMS</ENAME>
  </DBROW>
  <DBROW id="2">
    <EMPNO>7499</EMPNO>
    <ENAME>ALLEN</ENAME>
  </DBROW>
</ROOTDOC>
```

Листинг 14. Сгенерированный XML-документ для примера 5.

Приведенный в листинге 13 код на Java запрашивает базу данных и конструирует файл, содержащий результат. Исходным запросом является запрос, приведенный в листинге 12.

В результате работы приведенного в листинге 13 приложения будет сформирован XML-документ, представленный в листинге 14. Обратим внимание на то, что в документ помещены первые две

```
import oracle.xml.sql.dml.*;
import java.sql.*;
import oracle.jdbc.driver.*;
import oracle.jdbc.*;
import java.net.*;

public class xmlwritedb
{
  public static void main(String args[]) throws SQLException
  {
    // Имя таблицы, в которую добавляются XML-данные
    String tabName = "EMP";
    // Имя файла, хранящего XML-документ
    String fileName = "emp.xml";
    DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());

    // Инициализация подключения через JDBC
    Connection conn =
        DriverManager.getConnection("jdbc:oracle:oci8:scott/tiger@");

    // Добавить XML-данные из файла в таблицу
    OracleXMLSave save = new OracleXMLSave(conn, tabName);
    URL url = save.createURL(fileName);
    int rowCount = save.insertXML(url);

    System.out.println("Successfully inserted " + rowCount +
                       " rows into " + tabName);
    conn.close();
  }
}
```

Листинг 15. Запись XML-данных в таблицу базы данных.

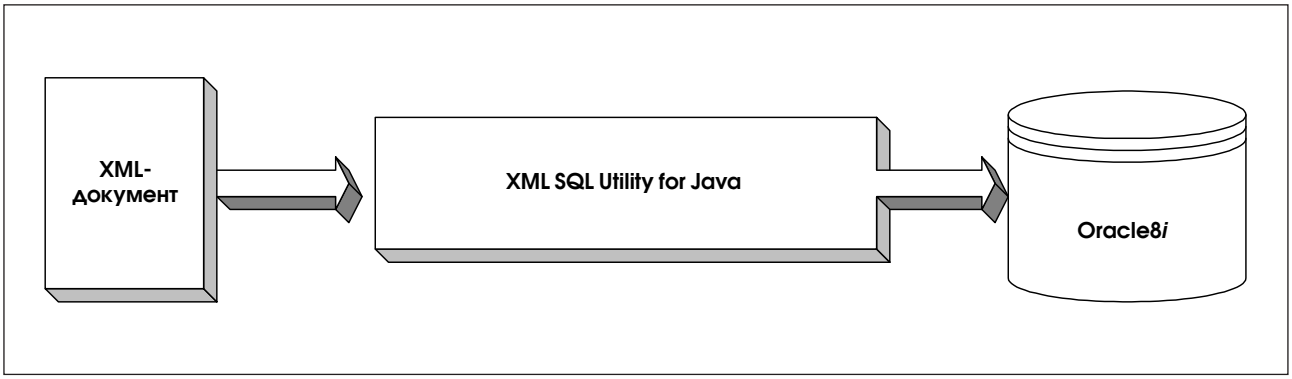


Рис. 6. Запись XML-данных в базу данных.

строки, выбранные из таблицы EMP по запросу, приведенному в листинге 12.

6.4. Запись XML-данных в таблицы базы данных

XML SQL Utility используется и для записи XML-данных в таблицы базы данных, причем в качестве сервера БД используется Oracle8i. Схема такой записи представлена на рис. 6.

Запись XML-документа в базу данных под управлением Oracle8i сохраняет структуру документа. Имена элементов преобразуются в имена столбцов таблицы. Элементы документа, содержащие только текст, преобразуются в скалярные столбцы, элементы, содержащие вложенные элементы, преобразуются в объектные типы. Списки элементов преобразуется в коллекции. Неструктурированные данные не могут быть преобразованы к хранимым в базах типам и должны быть сохранены как CLOB.

Целесообразно сопоставить запись XML-документа в базу данных и генерацию XML-документов на основе результатов SQL-запроса (см. выше

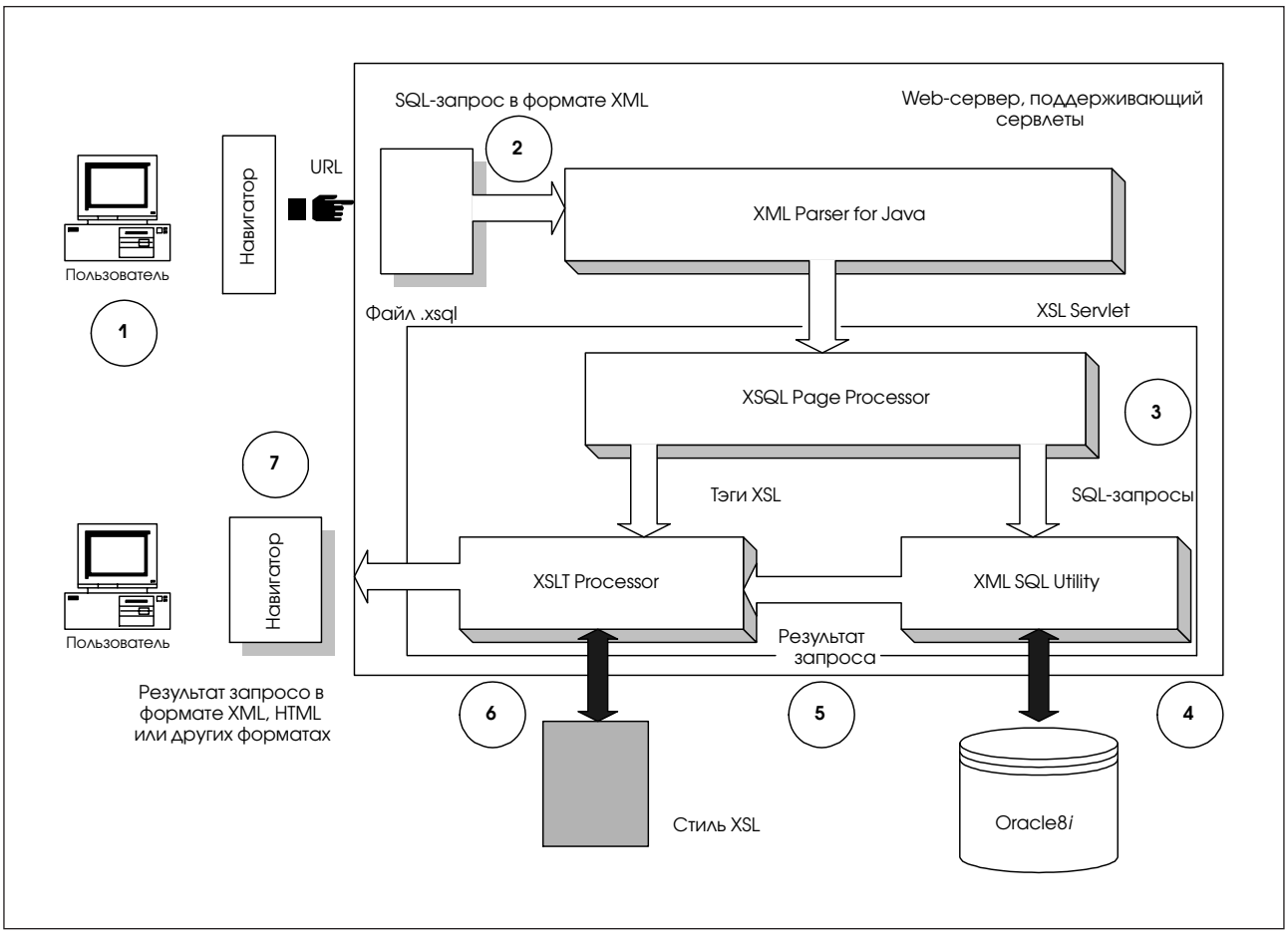


Рис. 7. Запись XML-данных в базу данных.

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="rowcol.xsl"?>
  <query connection="demo"
    find="%"
    sort="ENAME"
    null-indicator="yes" >
    SELECT *  FROM EMP
      WHERE ENAME LIKE '%{@find}%'
        ORDER BY {@sort}
  </query>
```

Листинг 16. Исходный XSQL-файл для примера 7.

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <html>
      <body class="page">
        <xsl:apply-templates/>
      </body>
    </html>
  </xsl:template>

  <xsl:template match="ROWSET">
    <center>
    <table border="0" cellpadding="4">
    <xsl:choose>
      <xsl:when test="ROW">
        <!-- present headings: row[1] columns +-->
          <xsl:for-each select="ROW[1]">
          <tr>
            <xsl:for-each select="*">
              <th align="left">
                <xsl:attribute name="class">
                </xsl:attribute>
                <xsl:value-of select="name(.)"/>
              </th>
            </xsl:for-each>
          </tr>
        </xsl:for-each>
        <xsl:apply-templates/>
      </xsl:when>
      <xsl:otherwise>
        <tr><td>No Matches</td></tr>
      </xsl:otherwise>
    </xsl:choose>
    </table>
    </center>
  </xsl:template>

  <!-- present rows and columns +-->
  <xsl:template match="ROW">
    <tr>
      <xsl:attribute name="class">
      </xsl:attribute>
      <xsl:for-each select="*">
        <td>
          <xsl:attribute name="class">
```

Листинг 17. Стиль для примера 7 (продолжение на стр. 19).

```
          </xsl:attribute>
          <xsl:apply-templates select='.'/>
        </td>
      </xsl:for-each>
    </tr>
  </xsl:template>
</xsl:stylesheet>
```

Листинг 17. Стиль для примера 7 (окончание).

тов на основе результатов SQL-запроса (см. выше одноименный раздел).

6.5. Пример 6: запись XML-данных в таблицу базы данных

Приведенная в листинге 15 программа на языке Java вставляет XML-данные, взятые из файла emp.xml, в таблицу EMP. Пример построен в предположении, что структура XML-документа соответствует структуре таблицы EMP.

Отметим, что в случае, если XML-данные не соответствуют структуре таблицы базы данных, в которую они сохраняются, необходима трансформация документа до его записи в таблицу.

7. XSQL Servlet

XSQL Servlet представляет собой средство, обрабатывающее SQL-запросы и поставляющее результирующие наборы данных как XML-документы. Он берет в качестве входного XML-файл, содержащий встроенный SQL-запрос, и использует программу-анализатор XML-документов.

Можно применять XSQL Servel совместно с любым Web-сервером, который поддерживает сервлеты Java. На рис. 7 показана схема работы пользователя с данными с применением сервлета.

Цифрами на рисунке обозначена последовательность действий.

1. Пользователь, работая с навигатором, вводит URL, который интерпретируется и передается через Java Web Server компоненту XSQL Servelet. URL содержит имя целевого XSQL-файла (.xsql) и, возможно, некоторые параметры, такие как значения и имя стиля. В то же время, пользователь может вызвать XSQL Servlet из командной строки.

2. Сервлет передает XSQL-файл программе-анализатору XML-документов для Java, которая, в свою очередь, разбирает XML и предоставляет программный интерфейс для доступа к содержанию документа.

3. Процессор страниц (page processor), один из компонентов сервлета, использует предоставленный программный интерфейс для передачи XML-параметров и запросов на языке SQL (которые помещаются между тэгами <tag></tag>) компоненту XML SQL Utility. Процессор страниц также передает любые операторы XLS-процессинга XSLT-процессору.

4. Компонент XML SQL Utility направляет SQL-запрос базе данных под управлением Oracle8i, сервер баз данных возвращает результирующий набор данных компоненту XML SQL Unility.

5. Компонент XML SQL Utility возвращает результат запроса XSLT-процессору в виде текста на языке XML. Этот текст помещается на то место в исходном файле, которое было помечено тэгом **<query>**.

6. Если это необходимо, результат запроса и любые другие XML-данные трансформируются XSLT-процессором с использованием заданных

```
http://localhost/xsql/demo/emp.xsql?find=T&sort=EMPNO
```

Листинг 18. URL для примера 7.

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7654	Martin	Salesman	7698	1981-09-28	1250	1400	30
7788	Scott	Analyst	7566	1987-04-19	3000		20
7369	Smith	Clerk	7902	1980-12-17	800		20
7844	Turner	Salesman	7698	1981-09-08	1500	0	30

Рис. 8. Результирующая HTML-страница для примера 7.

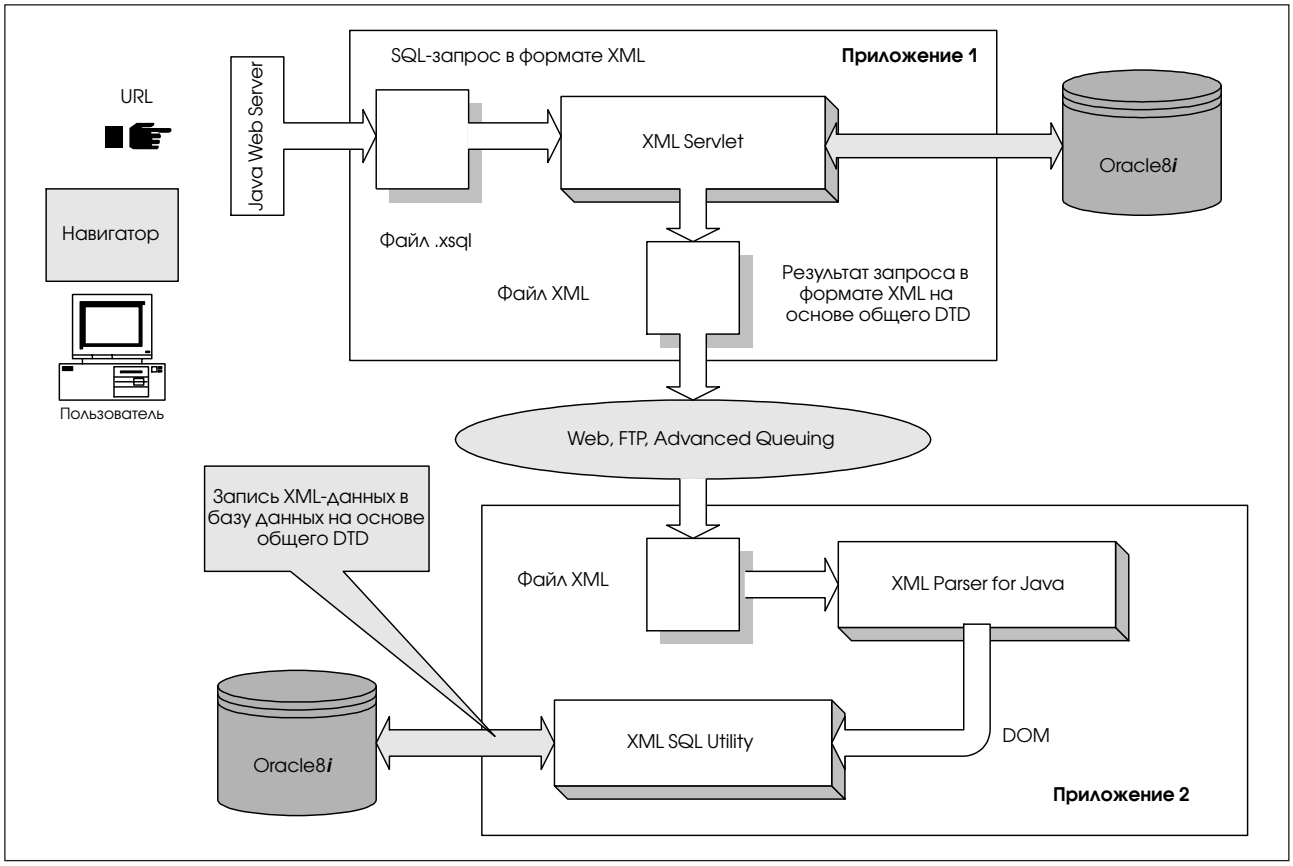


Рис. 9. Пример обмена XML-данными с использованием общего DTD.

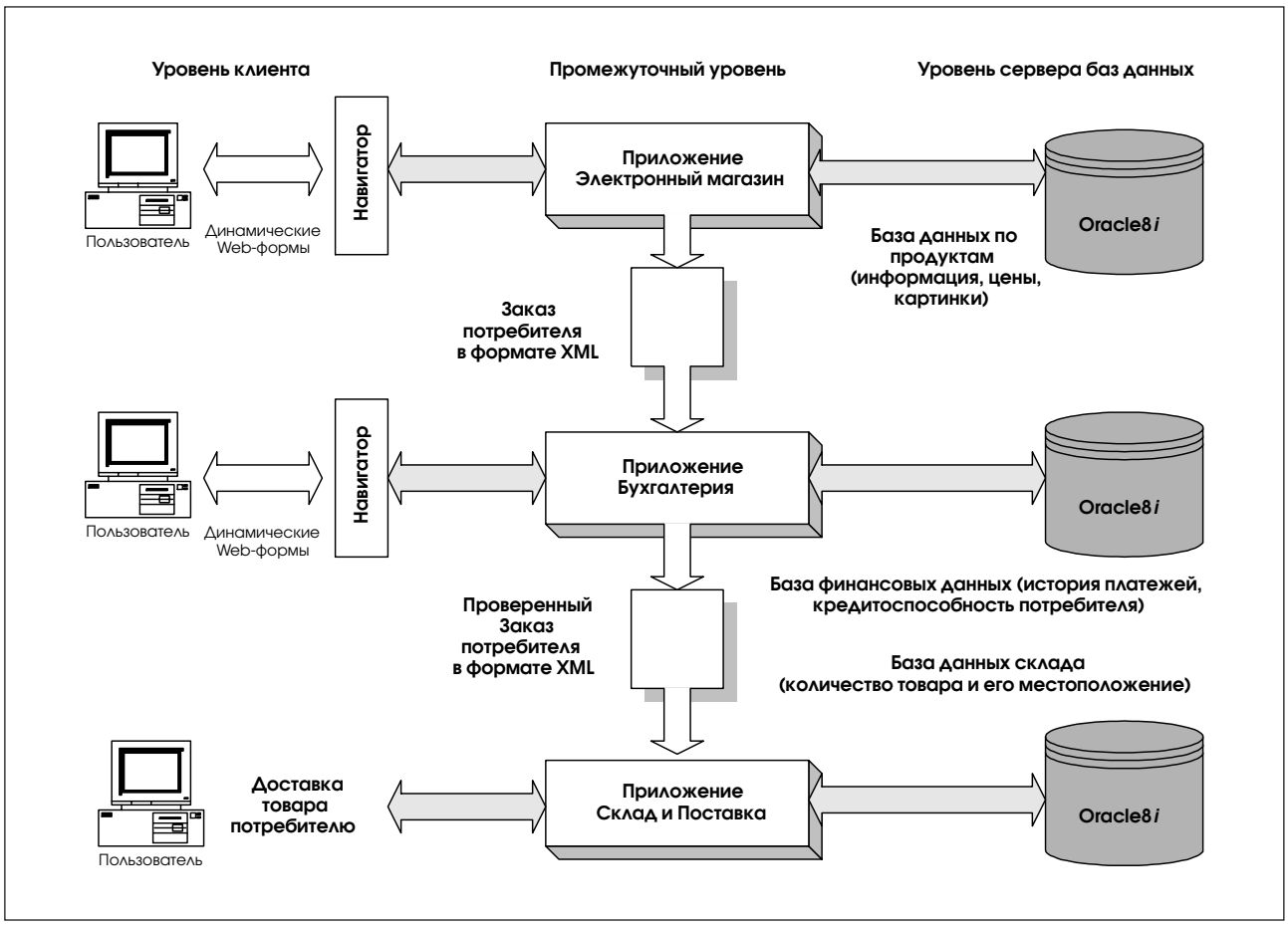


Рис. 10. Альтернативный вариант обмена XML-данными с использованием общего DTD.

стилей. Данные могут быть преобразованы в HTML или любые другие форматы, определенные стилем. XSLT-процессор может выборочно применять различные стили, имея в виду тип клиента, который сделал исходный URL-запрос.

7. Наконец, XSLT-процессор возвращает сформированный документ навигатору, и тот отображает его для пользователя.

7.1. Пример 7: XSQL Servlet

В листинге 16 приведен пример XSQL-файла, который запрашивает таблицу EMP сотрудников. Поведение запроса по умолчанию состоит в том, чтобы вернуть все записи о сотрудниках, имеющиеся в таблице. Можно сузить область поиска, добавив параметр URL find= при вызове компонента XSQL Servlet из навигатора. Например, задав букву 'T' в этом параметре, мы добьемся того, что из всего множества строк таблицы EMP будут выбраны и возвращены в результирующем наборе только те, в которых значение в столбце ENAME содержит эту букву. Задав параметр URL sort=, мы вызовем сортировку записей в результирующем наборе и т.д.

В XSQL-файле необходимо также определить, с помощью какого стиля должен быть обработан результирующий набор данных. Стиль для примера 7 приведен в листинге 17.

На рис. 8 показана HTML-страница, которая будет сгенерирована компонентом XSQL Servlet с использованием файлов emp.sql и стиля rowcol.xsl. Необходимый сервлет вызывается с помощью URL из листинга 18.

8. Обмен документами между приложениями

Наибольший интерес вызывает возможность организации обмена электронными документами между приложениями при помощи стандарта XML. Такой обмен получается простым, гибким и надежным.

Существует несколько сценариев организации обмена XML-данными, которые рассматриваются ниже.

8.1. Обмен XML-данными с использованием общего DTD

На рис. 9 приведен пример обмена XML-данными с использованием общего DTD. Здесь пользователь вводит запрос посредством Web-формы. XML-данные генерируются компонентом XSQL Servlet. XML-документ структурирован в соответствии с некоторым DTD, который далее рассматривается как разделяемый несколькими приложениями. Приложение-приемник получает XML-документ, выполняет его анализ с привлечением программы-анализатора для Java и записывает XML-данные в свою базу данных с помощью XML SQL Utility.

Возможен иной вариант развития событий, представленный на рис. 10. Некий заказчик оформляет заказ на покупку, обращаясь к Web-странице. Ввод заказов обеспечивает приложение Электронный магазин. Введенный заказ передает-

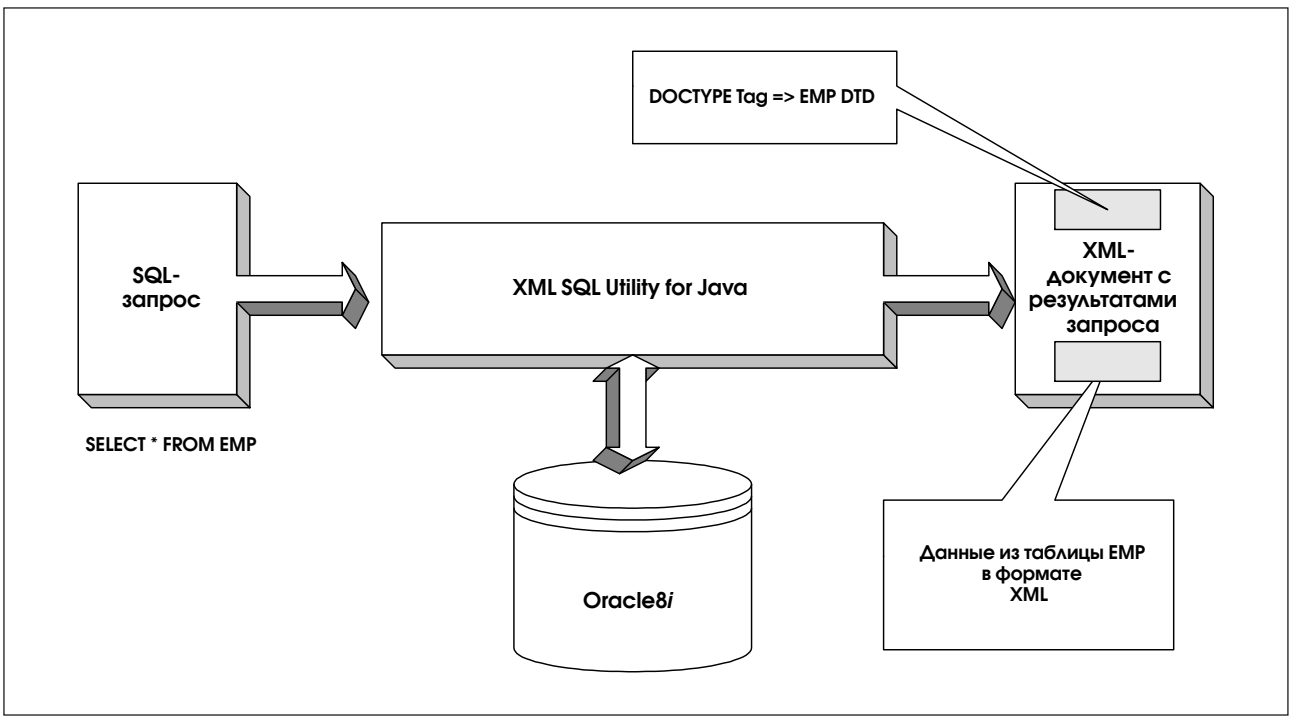


Рис. 11. Обмен XML-документами при отсутствии общего DTD.

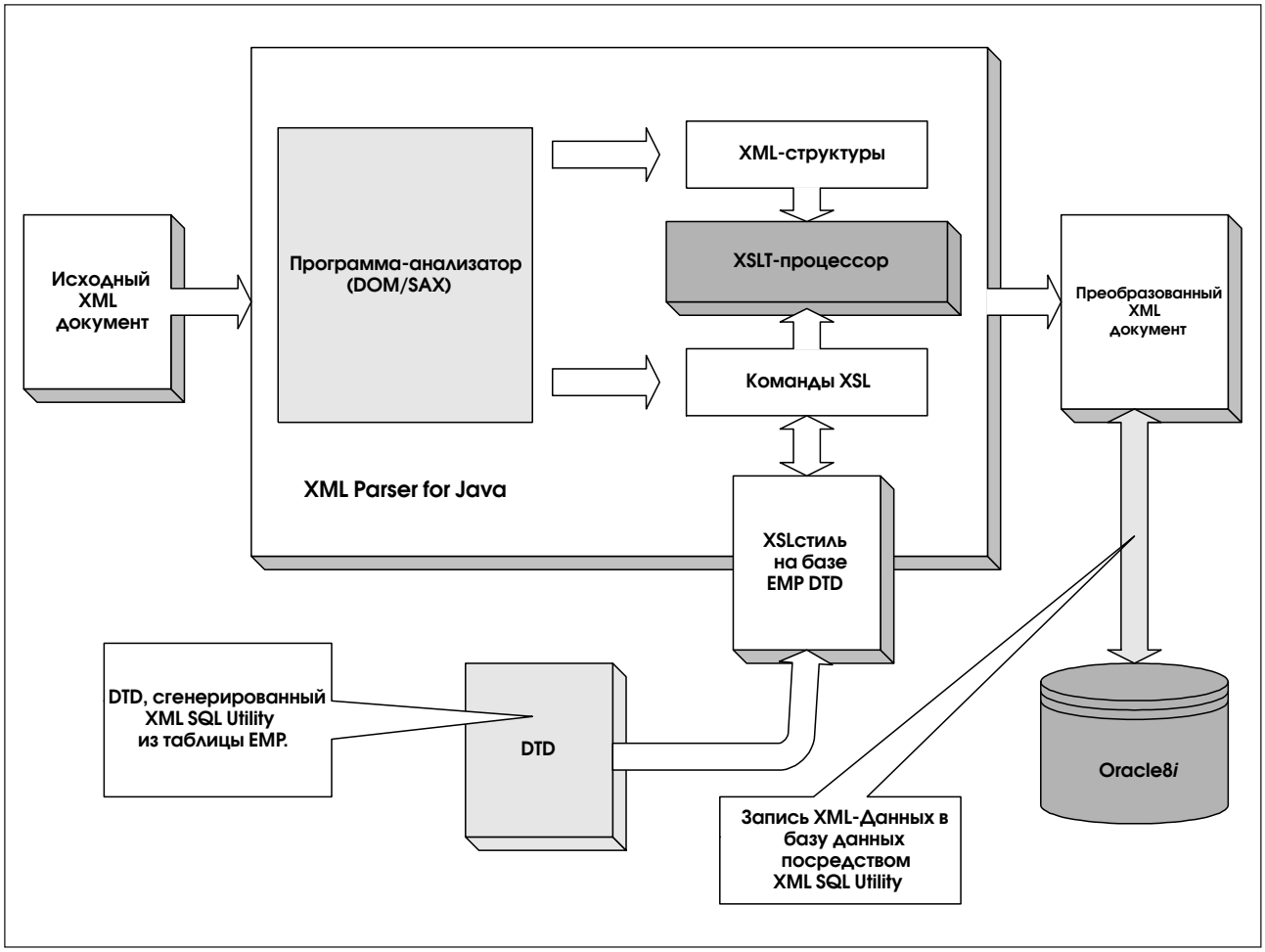


Рис. 12. Использование DTD для разработки стилей.

ся некоторому приложению Бухгалтерия и, после обработки, направляется приложению Склад и поставка. Каждое приложение в цепочке читает и обрабатывает XML-данные так, как это предписано их логикой, и записывает некоторые из этих данных в собственные базы, передавая следующему исходный или модифицированный в процессе обработки XML-документ.

Ниже детально расписаны роли каждого из приложений.

Приложение Электронный магазин:

- генерирует форму заказа;
- заказчик использует форму для запроса к базе данных для поиска доступных товаров, вводит необходимые для оформления заказа данные и подтверждает заказ;
- приложение получает заказ и на его основе генерирует XML-документ, используя общий DTD;
- приложение направляет DTD и XML-документ в Бухгалтерию.

Приложение Бухгалтерия:

- получает и обрабатывает заказ (XML-документ), полученный из Электронного магазина;
- преобразует XML с использованием соответствующего стиля в представление заказа для навигатора, с которым работает бухгалтер;
- бухгалтер запрашивает базу данных о заказчиках, проверяет кредитную информацию по заказчику, подтверждает или отвергает заказ;
- приложение обновляет соответствующие записи в таблицах базы, используя для этого данные, полученные из XML-документа;
- приложение на основе общего DTD генерирует новый XML-документ (модифицированный заказ), содержащий все привнесенные бухгалтерией значения и передает его следующему в цепочке приложению.

Приложение Склад и поставка:

- получает и обрабатывает XML-документ, используя для этого DTD, полученный из бухгалтерии;
- обновляет записи в базе данных поставок, используя для этого данные о заказчике и заказе, взятые из XML-документа;
- генерирует внешнее представление XML-документа для навигатора, который используется кладовщиком;
- кладовщик отпускает товар заказчику.

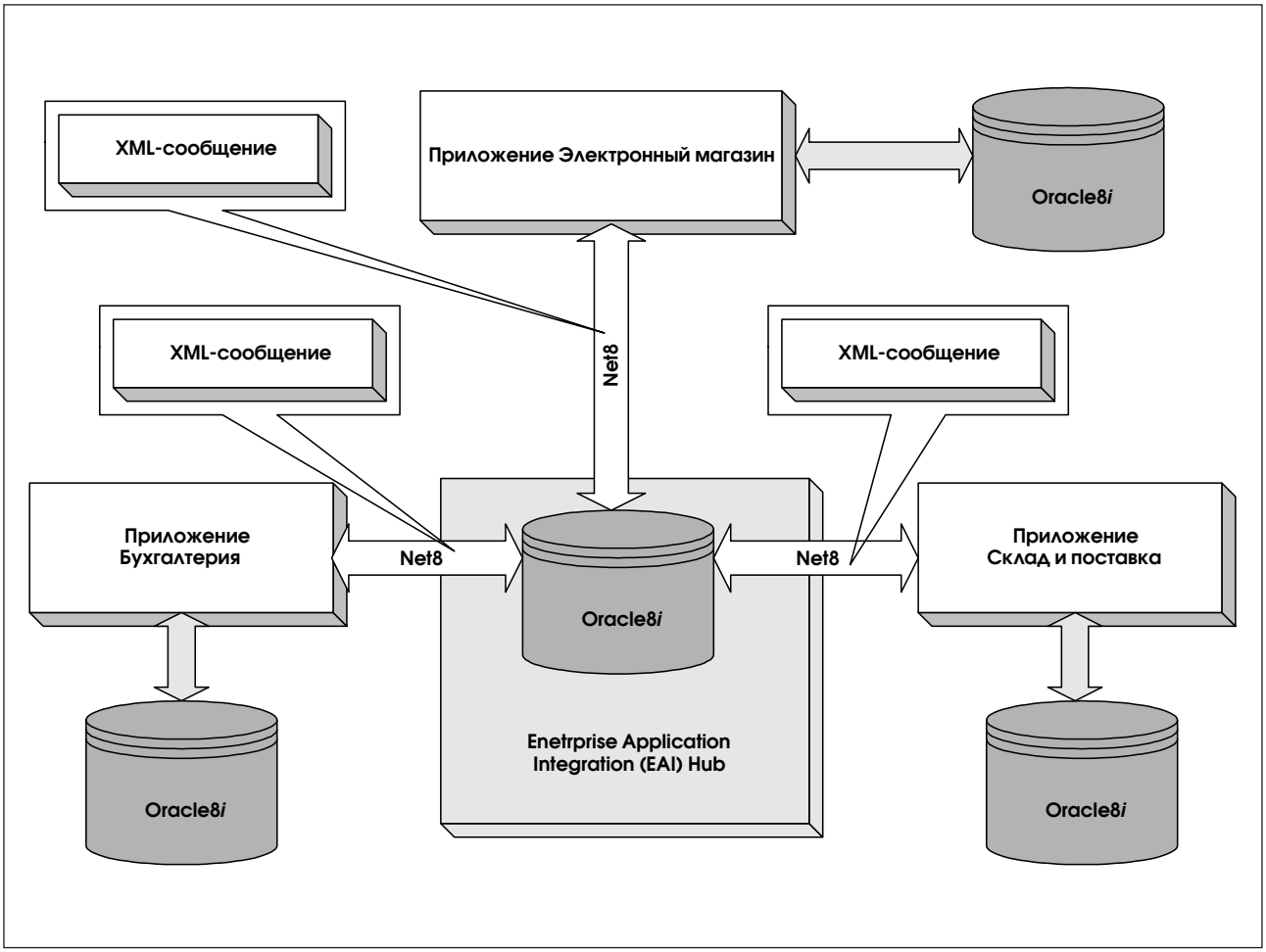


Рис. 13. Архитектура системы с асинхронной передачей стандартизованных сообщений.

8.2. Обмен XML-документами без использования общего DTD

При отсутствии общего DTD требуется выполнение некоторых дополнительных действий. Например, возможна ситуация, когда мы хотим записать содержание XML-документа в базу данных, однако, его структура не соответствует структуре таблицы (или таблиц) поддерживающей БД. Следовательно, необходимо выполнить преобразование XML-документа.

Хорошим способом будет использование стилей для преобразования исходного документа в новый, структура которого соответствует структуре таблицы поддерживающей БД. Нужный XML-документ можно получить, используя XML SQL Utility. Таким же образом можно получить и «локальный» DTD, а затем использовать его для «оформления» приходящих «внешних» по отношению к этой базе данных документов. Общая схема описанного процесса представлена на рис. 11.

XML SQL Utility создает DTD в отдельном файле или добавляет его к сгенерированному XML-документу в тэге DOCTYPE. DTD можно использовать для разработки стилей, которые будут использованы для преобразования исходного XML-документа до записи в базу данных. Эта возможность проиллюстрирована рисунком 12.

Очевидны естественные ограничения DTD. Основная неприятность состоит в том, что DTD не содержит информации о типах данных. Собственно, основным и единственным типом данных, описываемым DTD, является строка символов. Ясно, что при извлечении данных из базы будет потеряна такая важная характеристика, как тип. Это означает, что приложение, использующее DTD, должно само присваивать типы данным, опираясь на контекст, то есть выводя типы из элементов, которым они приписаны.

Как добиться того, чтобы данные, введенные посредством заполнения полей Web-формы, были адекватно отображены в таблицу (таблицы) поддерживающей базы? Ответ дает приведенная ниже последовательность действий:

- Java-приложение использует XML SQL Utility для генерации DTD, который отражает структуру таблиц поддерживающей базы данных;
- приложение передает DTD генератору XML-классов для Java, создающему классы, которые будут использованы для построения Web-формы, предоставляемой для работы пользователя;

- форма динамически создается компонентами Java Server Page, Java servlet и т.д. с использованием сгенерированных классов;
- когда пользователь заполняет форму и отправляет ее на обработку, сервлет отображает данные формы в соответствующие структуры XML, а компонент XML SQL Utility записывает данные в БД.

9. Архитектура с асинхронной передачей стандартизованных сообщений

Существует несколько возможностей по передаче XML-документов между приложениями. Во-первых, их можно передавать попросту как файлы, используя для этой цели FTP, NFS, SMB либо другие известные протоколы передачи файлов. Во-вторых, можно использовать HTTP. В этом случае приложение, которому необходим XML-документ, запрашивает по HTTP сервлет. Третьей возможностью является использование Web-форм.

Наконец, можно использовать компонент Oracle8i Server под названием Advanced Queuing. Рассмотрим эту возможность более подробно.

Oracle8i Server может инициировать отправку XML-документа через Net8 и JDBC в качестве сообщения одному или нескольким приложениям-приемникам, используя для этой цели Oracle Advanced Queuing. Приложение-приемник извлекает XML-документ из входной очереди сообщений и обрабатывает его. Это как раз тот подход, который применяется Oracle для интеграции приложений. Здесь сообщения в формате XML направляются инициирующими приложениями некоторому серверу, который можно было бы назвать концентратором сообщений (AQ hub) — по отношению к тем приложениям, которые хотели бы получать сообщения, циркулирующие в системе. При этом может быть использован стандартный механизм взаимодействия «публикация/подписка», реализованный в Oracle8i.

На рис. 13 представлена архитектура системы, в которой различные приложения асинхронно взаимодействуют, направляя друг другу стандартизованные (XML) сообщения, извлекают из них данные, размещая их в локальных, принадлежащих им базах и генерируя сообщения на основе этих данных. Вся инфраструктура строится на основе продуктов Oracle, ядром системы является сервер Oracle8i Enterprise Edition. Использование только Net8 является некоторым ограничением архитектуры, не позволяя расширить ее до масштабов глобальной сети, однако, не будем забывать, что пересылку данных можно будет организовать и другими способами (FTP, HTTP).

10. Заключение

В настоящее время существует полный набор средств для конструирования среды обмена электронными документами. В данной статье были описаны следующие компоненты такого набора:

- XML — хорошо структурированный декларативный язык описания документов любой сложности;
- Java — система программирования, ставшая фактическим стандартом для создания приложений, обрабатывающих XML-документы;
- Oracle8i — сервер реляционных баз данных, завоевавший огромную популярность, в том числе в сфере электронного бизнеса, интегрированный с XML- и Java-средствами.

В комплект поставки Oracle8i включается весь перечисленный выше инструментарий по работе с XML, который носит название XML Developer's Toolkit. Инструментарий распространяется свободно, и разработчики могут скопировать его с web-сервера http://technet.us.oracle.com (и, при желании, проверить корректность приведенных в статье примеров).

Воспользовавшись Java, XML и Oracle8i, можно построить целостную среду обмена электронными документами, опирающуюся на стандарты, гибкую и масштабируемую. Стандартизованность XML, его высокая популярность гарантируют, что созданная таким образом система будет развиваться в общем технологическом потоке.

11. Литература

1. Дуров И. Современное состояние языков и средств разметки документов. — Jet Info, 2000, 1.
2. Wait B. Using XML in Oracle Database Application. White paper. — Oracle Corporation, November 1999.
3. Таранов А., Цишевский В. Java в три года. — Jet Info, 1998, 11-12.
4. Галатенко В., Ладыженский Г. Объектные технологии в продуктах Oracle. — Jet Info, 1998, 9-10.

Jet Info
ИНФОРМАЦИОННЫЙ БЮЛЛЕТЕНЬ
Издается с 1995 года
Издатель: компания Джет Инфо Паблишер

Главный редактор: Галатенко В.А. *(galat@jet.msk.su)*
Технический редактор: Антонов А.Н. *(silver@jet.msk.su)*
Россия, 103006, Москва, Краснопролетарская, 6
тел. (095) 972 11 82, 972 13 32
факс (095) 972 07 91
e-mail: JetInfo@jet.msk.su, http://www.jetinfo.ru
Подписной индекс по каталогу Роспечати 32555