第1章 Oracle与 XML

扩展标记语言（ XML）是一种元标记语言。就像在广域网协会（ World Wide Web Consortium，W3C）的XML 1.0规范中所说明的，XML可以让用户定义自己的标记语言，从而可以在XML文件中描述并封装数据。这些文件可以在类似于 Netscape Navigator或Microsoft Internet Explorer的浏览器中显示，并通过因特网在应用之间或业务之间交换，存储到数据库中或从数据库中取出。XML具有简单性，它是开放标准的一部分，加入了用户自定义的标记。

XML起源于通用标记语言（Standard Generalized Markup Language，SGML），后者于1986年由国际标准化组织（International Standards Organization，ISO）批准，基于通用标记语言的超文本标记语言（Hypertext Markup Language，HTML）诞生于1990年。虽然在文档世界中SGML仍然是广为使用的标准，而HTML作为广域网上数以百万计的Web页面的基础依然用途广泛，但XML正在获得广泛的接受，因为它与现有的标记语言相比，在数据的交换、存储、描述等方面都具有优点。自从W3C在1998年2月出版XML 1.0版的规范以来，在相当广泛的范围内，许多人都认为它将成为电子商务在语言和数据交换方面的选择。

1.1 XML基本概念和术语

类似于所有的标准，XML也有数不清的概念和技术术语需要解释。由于开发 XML是用来传送数据的，因此，介绍一个例子，看一下标准的数据库中书籍列表的数据记录。一个复杂的SQL查询典型情况下将会返回下述格式的数据：

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

如果XML用作输出表单，则该记录的每个数据项都将具有附加的上下文环境，如下所示：

```
<book>
     <title>History of Interviews</title>
     <author>
     <first name>Juan</first name>
          <last name>Smith</last name>
     </author>
     <ISBN>99999-99999</ISBN>
     <publisher>Oracle Press</publisher>
     <publish year>2000</publish year>
     <price type="US">1.00</price>
</book>
```

本例中某些值得注意的项将在稍后进行探讨。可以注意到文件具有对称性，且每个数据块都被其上下文所包围，形式类似于<context>...</context>。尖括弧及其内部的文字称为标记（tag），每组标记及其包围的数据称为元素。这种关系可以认为类似于数据库中表的一列，其中标记的文字相当于列标题，标记之间的文字相当于该列中某一行的数据。在前一例中，title应该是列名，

而History of Interviews应当是某一行中的数据。

还可以注意到，有几个标记包含的并非数据而是标记。这是 XML一个很重要的特性，它允许数据的嵌套以便更好的定义关系。返回到数据库的意义上， <author>标记可以用表来模拟，而该表的列则是<first name>和<last name>。在XML的术语中，这些列标记被称为表名标记，即父标记Author的子标记。

现在注意<price>标记，你可以看到它以 “名字=‘值’”的形式包含了文字。这些“名—值”对叫做属性，开始标记中可以包括一个或多个属性。属性出现在结束标记中是非法的（例如，</tag name=“foo”>）。

下面是最后一个值得注意的术语，显然整个 XML实例是被<book>...</book>所包围的。这种标记定义为文档的根，在整个文档中只能有一个。若 XML文档遵循这样的规则：只有一个根标记并且所有打开的标记都正确地关闭，则称该文档是构造良好（ well formed）的，也称良构。

XML的基本概念和术语是简单的，且已经形式化为开放的因特网标准。像 W3C的XML 1.0规范中所说的，“ XML文档由称为实体的存储单元组成，实体中包含分析过或未分析的数据。分析过的数据由字符组成，其中一些形成字符数据而另一些形成标记。标记对文档的存储布局和逻辑结构进行描述。”XML文档同时具有物理结构和逻辑结构。 XML文档的物理结构简单地指向XML文件本身及可能引入的其他文件，而 XML文档的逻辑结构则指向文档的序言（ prolog）和主体（body）。

图书列表的XML实例显示了XML文档的主体，但它缺失了有助于识别其性质的重要信息。该信息在文档的序言中，将在下一节定义。

1.1.1 序言

序言由XML的声明组成，包括：版本号、可能的语言编码、其他属性（形如“名字 =‘值’”的对）、可选的文档类型定义（Document Type Definition，DTD）等，DTD可以是内部的，即包含在XML文档中，也可以是外部的，即指向另一个文件。例如：

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

请注意，形如<?...?>的行是XML中的处理指令（processing instruction，PI），将在本章中稍后讨论。在这个例子里， XML处理指令的名字就是 xml。另外，本例所支持的字符集编码是Unicode的压缩版本，称为 UTF-8。最后，standalone属性是指处理程序是否需要包含或引入其他的外部文件。

序言的第二行使用了DOCTYPE。该文档在这里声明数据模型。它为什么如此重要？请记住，XML文件同时具有物理和逻辑描述。在某些应用中，即使不知道信息是否缺失，也可能足以处理XML。但在大多数情况下，应用需要验证其收到的 XML文档的合法性。要这样做，应该需要知道 哪些元素是必需的，哪些可以有子标记，哪些可以有属性，等等。用 XML的术语来说，数据模型称为文档类型定义。文档类型定义可以包含在 XML文件中，也可以简单地顺次引用，这样处理程序就可以对其进行定位。

1.1.2 文档类型定义

文档类型定义（Document Type Definition，DTD）是以SGML的形式加入到XML中的，它并不是XML语法的一部分。由W3C XML模式工作组所做的工作定义了一种利用 XML语法的新型模型定义方法，该方法扩展了 DTD的功能来支持数据类型。前一例中， DTD包含在外部文件中。它也可通过统资源定位器（ Uniform Resource Locator，URL）的形式指出，类似于 http://www.foobar.com/book.dtd。DTD还可以在XML文件内部声明，与下面的代码很相似。

```
<!-- DTD bookcatalog may have a 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|EURO) #REQUIRED>
]>
```

根元素的声明紧跟着 DTD的DOCTYPE声明，是<!ELEMENT>与bookcatalog的组合。元素的组成很简单，有开始标记，例如 <foo>；对应的结束标记，如 </foo>；以及二者之间的文字。在一个XML文档中，只可能存在一个根元素。根元素标记着文档的开始，并被视为所有其他元素的父元素，那些元素都嵌套在它的开始标记和结束标记之间。相应于上面的 DTD，如果一个XML文档被认为是有效的，开始其主体部分的第一个元素必须是根元素 bookcatalog。

然后是元素的声明，要确保嵌套在根元素 bookcatalog之中的子元素必须符合根元素的内容模型。注意到bookcatalog的所有子元素都在它的声明中显式列出，而且 author有后缀“ + ”。这是用来描述内容模型的，称为扩展的巴科斯 -诺尔范式（Extended Backus-Naur Format，EBNF）。可用的后缀有：

- ？ 出现零次或一次。
- * 出现零次或多次。
- + 出现一次或多次。

还可以注意到：用 #PCDATA来声明的元素中文字是没有标记的，另外，价格的必需的属性均进行了显式声明。 CDATA与PCDATA的区别在于：解析器将简单地略过 CDATA组成的段，并不进行良构性检查，因此， CDATA可视为未分析的字符数据。

这样，用于合法性验证的 XML解析器，通过对照在 DTD中指定的规则，尽力确定该文档是否符合其DTD，也就是DTD中结构化的关系与文档中的元素序列是否是一致的。

1.1.3 文档的主体

序言处于根元素之前，后者包含了 XML文档的剩余部分。这一部分由元素、处理指令、内

容、属性、注释、实体引用等等组成。正如前面所提到的，元素必须有开始标记和对应的结束标记，而且嵌套的顺序要正确，否则XML文档就不是良构的，XML解析器有可能因此而报错。元素也可以有属性，或称为‘名—值’对，例如<author firstname=“Juan” lastname=“Smith”>。还有由XML 1.0规范所定义的内部属性，例如xml:space=“preserve”表示元素之间的数据用于表示预留的空格。实体引用类似于在实体中定义过一次的宏，然后引用它们就可以了，例如&nameofentity，可以在其整个定义域中使用。举例来说，可以声明 <!ENTITY Copyright “Copyright 2000 by Smith,Jones,andDoe—All rights reserved”>；这样，&Copyright实体就可以作为快捷方式在XML文档中到处使用。XML解析器必须能够识别在DTD中定义的实体，即使为此要关掉合法性检查。再强调一次，有XML1.0规范内建的实体也是存在的，例如“&”符号实体——&amp、省略号——&apos、小于号——&lt等等。而注释则通过包围在形如 <!-- -- >的结构中来识别。

一些额外的与XML解析器应用编程接口相关的基本术语可以分类到以下的领域：

- 文档对象模型（Document Objec+ Model，DOM）。
- XML简化编程接口（Simple API for XML，SAX）。
- 名字空间。
- 解析器。
- 可扩展格式页语言变换（Extensible Stylesheet Language Transformation，XSLT）。

其中最大的一部分，即有关API的部分受到W3C规范的约束，所以应用开发者可以使用标准的编程接口。请注意，如果有些功能未指定，则厂家可能会决定实现其自己的对规范的增强。

1.1.4 文档对象模型API

由于开始标记均有对应的结束标记，因此，XML文档是结构化的，并且这些标记以有序的方式嵌套。由于其结构性，XML文档可视为树结构，其节点由标记以及处于标记之间和对应于标记的信息组成。实际上，当XML解析器分析XML文档时，在内存中会形成该文档的语法树表示，称之为文档对象模型（Document Odject Model，DOM）。

W3C已给出一些DOM API可用于存取及访问该树。它的组成包括文档的根元素、子结点及兄弟结点、元素、属性、表示元素或属性文字内容的文字结点、用来标识文字块结束而易被认为是标记的字符数据（character data，CDATA）数据段、注释、实体引用、处理指令等等。提供所有DOM API的XML解析器称为与W3C的DOM推荐标准相容。

下面的代码实例演示了对解析器和DOM API的简单使用。输入到应用程序的XML文档被分析，然后文档中的元素节点和属性值被打印出来。最后，又演示了对设置解析器选项的使用。

```
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)
    {
        File f = new File(fileName);
        try
        {
            String path = f.getAbsolutePath();
            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.println("Cannot create url for:" + fileName);
            System.exit(0);
          }
       }
       return url;
    }
}
```

1.1.5 XML简化编程接口API

XML简化编程接口(Simple API for XML,SAX)是基于事件的API，其意义在于，分析XML文档时遇到的某些事件和数据的通知，可以通过回调函数报告给应用程序。对于这些事件的通知，应用程序必须进行处理。例如，应用程序可以拥有含有回调事件处理程序的数据结构。最后，回调函数返回通知或信息，返回的类型大体上是元素的开始和结束标记或与元素的内容有关的信息，如字符数据（CDATA)、处理指令、和/或子元素。

SAX最初是由David Megginson开发的，现已成为W3C XML标准。相对于DOM，使用SAX分析的优点在于不必在内存中建立语法树，这样就节省了内存空间，而且对于某些类型的操作性能更好，例如搜索。另一方面，进行修改、更新或执行其他的结构化操作，使用 DOM解析器可能更有效。

下面的代码实例演示了解析器和 SAX API的简单使用。输入到解析器的 XML文件被分析后打印了一些有关文件内容的信息。还提供了各种有用的接口实例代码。

```
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);
    }

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

    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;
    }

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

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

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

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

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

    public void error (SAXParseException e)
            throws SAXException
    {
        throw new SAXException(e.getMessage());
    }

    public void fatalError (SAXParseException e)
            throws SAXException
    {
```

```
            System.out.println("Fatal error");
            throw new SAXException(e.getMessage());
        }
    }
```

1.1.6 名字空间API

这些接口给出XML文档的名字空间的有关信息，名字空间是由统资源标识符（Uniform Resource Identifiers，URI）引用所标识的，URI引用可修饰元素和属性名或用于定位可能位于不同的计算机或文档中的资源等。考虑到同名的问题，名字空间用URI来修饰元素或属性名以区分这些名字。例如foo:hello，称之为受限名，其中名字空间前缀foo映射到URI，www.oracle.com，本地部分为hello。请注意，URI引用中可以包含在名字中所不允许的字符，这也是为什么用foo来代替上面的URI的原因。

下面的代码实例演示了解析器和对DOM API的名字空间扩展的简单使用。XML文件输入到该应用程序，打印出元素和属性。

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

      // Get an instance of the parser
      Class cls = Class.forName("oracle.xml.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("*");
         XMLElement nsElement;

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

         System.out.println("The elements are: ");
         for (int i=0; i < nl.getLength(); i++)
         {
            nsElement = (XMLElement)nl.item(i);

// Use the methods getQualifiedName(), getLocalName(), getNamespace()
// and getExpandedName() in NSName interface 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.getExpandedName();
            System.out.println("  ELEMENT Expanded Name :" + expName);
         }

         System.out.println();
      }

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

```
    XMLAttr nsAttr;

    String attrname;
    String attrval;
    String attrqname;

    NamedNodeMap 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.getAttributes();

       if (nnm != null)
       {
          for (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();
    }
}

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();
```

```
            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.println("Cannot create url for: " + fileName);
            System.exit(0);
         }
      }
      return url;
   }
}
```

1.1.7 解析器API

应用程序调用解析器API来读取XML文档，并利用DOM或SAX API来访问其内容或结构。通常，初始和结束函数必须和分析函数协同调用。请注意，各种标志（如取出空格符或打开合法性验证选项,）都可以在应用程序调用分析函数之前用一些初始化函数来设置。例如，有些XML解析器不支持合法性验证，这意味着它们无法检查XML文档是否符合DTD；而另外的一些解析器（如由Oracle所提供的）有可选的合法性验证选项，这意味着用户可以在调用分析函数之前指定是否需要合法性验证。另外，用于分析的函数必须能够接受在XML文档中指定的各种不同的语言编码。

1.1.8 扩展格式页语言变换API

如需要转换或格式化XML文档，W3C推荐使用扩展格式页语言变换（Extensible Stylesheet Language Transformation，XSLT），它是对格式页的应用，由用户定义的XML或XSLT标记/函数写成，由XSLT处理程序加到XML文档中。实际上，XSLT处理程序所做的就是：把格式页中的样式关联到XML文档中，然后使用命令，例如HTML标记把发现的数据包围起来，再把结果作为独立的XML文档输出。继续前面的例子，使用XSLT处理程序，<book title>标记可以通过格式页函数被“匹配”而被表示黑体的HTML标记包围，例如，把<B>和</B>应用到数据上，产生另外一个XML文档，其中书标题部分的数据被黑体HTML标记所包围。这些把格式页应用到XML文件的API都相对较为简单，有些XML解析器（如Oracle所提供的）集成了XSLT处理程序，从而使得变换的API相当少。

1.2 为何使用XML

XML文档是易于阅读和理解的。与数据所关联的标记，例如 <author frist name>Juan</author first name>，标记使数据具有了含义,所以 XML文档很容易理解。其结果是：在因特网上如果

Web页是XML格式的，则搜索会更加高效。不仅可以搜索数据，而且可以在搜索中加入与数据相关联的上下文信息，这样就形成了更精确的搜索机制。另外，如果与数据相关联的标记有足够的与之关联的语义信息，那么10年后的最终用户可能会毫不困难地理解XML文档，而不会像其他文档格式那样难理解。

XML另一个巨大的优势在于它基于由W3C制定的开放标准。这样，没有一家公司可以垄断标准而强制其他公司接受它所制定的东西。如果其他的公司需要某项功能，则业界公司的成员代表组成的W3C工作组委员会就召开会议并讨论相关问题然后加入所需的功能。如果公司不想等到W3C采取行动，它们可以自行实现当前未定义的功能。这样做的公司是冒风险的，由于W3C没有引入该功能，最终用户可能会不采用该扩展。这些公司需要逐渐修改该功能，最后正式把扩展引入W3C，这样所有人都可以各取所需，从中受益。

由于XML生成简单，易于理解，并基于开放标准，可以想象得到当前正在建设的基于XML的应用的广泛性。这些应用程序处理基于XML的电子书籍、电子杂志或任何种类需要存储并与其他应用交换的文档。它们也处理在线业务，包括抽取数据和把数据以一般格式转发给其他在线业务。例如，所谓业务型电子商务（business-to-business，B2B），就涉及到把存储在关系数据库中的数据转换到XML等等。实际上，以文档为中心和以数据为中心的领域都会从XML中受益，当前的应用就反映了这一点。

业界公司允许自由地下载和使用通用的XML功能，类似于XML解析器和XSLT处理程序，这使得应用开发者也可以从中受益。由于这些功能可以在因特网上免费得到，而且即使公司需要进行内部发行，也可以在短时间内重写，因此，在因特网时代应用开发者可以快速的完成其程序。另外，由于这些组件基于开放的标准，可以想到，开发者很容易由一个生产商转到另一个，替换一下功能，程序就又可以工作了。很自然的，在使用类似的免费软件之前，性能、内存占用、功能的完整性、技术支持等都是开发者所必须考虑的。即使如此，如果实现的数量不断增长并可以广泛地得到，那么应用开发者就应该能够很快开发出基于XML的产品。

最后，除了已提到的优点之外，健康、金融及其他一些行业特定的DTD已经制定了用于统一在因特网上的业务之间交换的XML文件的格式。明显的好处是通过把XML作为交换和存储的数据格式，因特网上的业务到业务型电子商务就变得容易进行。只要某一行业关于一种特定的XML格式达成协议，该行业的商家就不必再为把文件由一种格式转换到另一种格式而烦恼，例如汽车零件行业。最终，把数据和文档转换到XML后，在因特网上进行商务活动的顾客和商家都将直接从中受益。

1.3 Oracle的XML战略

回溯到1997和1998年，Oracle并没有参加制定XML1.0标准的工作。事实上，当时搜索Oracle全球企业网找不到任何XML的命中。不到24个月后，同样的搜索产生了数以千计的命中。对XML技术的快速接受是由Oracle公司的Architechural Review Board将Oracle公司范围内所有的开发组都可用的XML基层组件的开发任务交给核心开发组而开始的。

与此同时，ARB把Oracle的XML战略整合为简洁的一句话：“向开发者提供最好的平台以使他们高效的建立和部署可靠、可伸缩的利用XML的互联网应用。”这句话重要之处在它提出了一

个新的思想，而不是产品或引人注意的焦点。从整体上来说这是具有一致性的，因为 XML并不是应用，在这一点上它与其他的标记语言不同。 XML是一种允许启用的技术，正因为如此，从具体的解决方案中才能看到其重要性。

一旦这项XML开发战略开始传播和运作，几乎每一项 Oracle软件产品都对在某些方面使用XML技术感兴趣。最近的统计显示，有 40个以上的开发组在其产品中的某些方面运用了 XML技术。

不仅仅是Oracle内部的开发组对XML感兴趣，合作者以及由Oracle的客户群体所获得的信息提出了把XML技术集成到第三方系统中的需求，其中许多需求是由专营行业所提出的。 XML及其伙伴XSL非常适于提供链接这些系统的基础。

由于具有XML功能的应用与服务器之间在技术上的约定比较简单，所以互联网及其对通过防火墙和各种有线或无线的通信媒体对各种系统的分布式访问的需求，就成为了传输 XML文档和数据的理想环境。 XML已迅速成为所有数据的集成点，无论是数据仓库、自助式销售还是B2B电子商务、内容管理或后端集成。因此，为支持这种需求， Oracle的对象 —— 关系数据库很自然就具有了XML功能。

1.4 Oracle在XML工业技术上的努力

Oracle大量的参与了 XML，对于Oracle来说XML不仅仅是应用开发可以使用的一项核心技术，而且也是其标准和相关的其他技术的驱动力。以下是 Oracle协同诸如广域网协会、 Sun Java Community Process以及Open Application Group等组织在XML开发和标准等方面做出的努力。

1.4.1 Oracle在W3C工作组委员会中

尽管Oracle最初并未参与 W3C XML工作组，但由于大量公司对 XML标准的采纳，迫使Oracle不仅加入了所有的XML标准，而且也推动了它们的进步。 Oracle是下列W3C XML工作组的成员：

- XML核心工作组　负责制定所有其他XML工作组均要用到的XML基本技术。例如Xbase，用于指定基于XML的URI的格式和行为特性；以及 Xinclude，用于指定把其他XML文件包含到XML文档中的方式和语法。
- XSL工作组　负责定义扩展格式页语言的两种实现。 XSL:T定义了转换一篇XML文档到另一篇所用的格式页语法和功能，而 XSL:FO定义了把XML文档转换到电子出版格式所用的格式页语法。
- XML模式工作组　负责定义在XML文档内简单的数据特性和复杂的结构的语法和使用。它也提供了模式定义XML文件格式，以取代DTD并加入了对数据和结构类型验证的支持。
- XML链接工作组　负责定义怎样在XML文档内引用XML资源。Xlinks包含了HTML 链接的功能，还允许引用多个文档，而不管它们是在一行中还是超出一行，它也允许定义行为特性。
- DOM工作组　负责用于访问XML文档的DOM API。这些API提供了对XML文档中的元素

和节点对应的实际内存中的结构的可编程的访问和处理。

- XML查询工作组　负责制定查询XML文档的语法和句法。由于XML文档是层次性的树结构，因此可以周游树，然后查询返回的结果数据集。

1.4.2 Oracle的XML开发包

前面提到，数据库部门的核心与XML开发组负责开发Oracle范围内的开发组都可用的XML基层组件。任何处理XML的软件产品都必须能够确定文档是否合法，即符合指定的DTD，以及是否是良构的，即开始标记必须有结束标记并且嵌套顺序必须正确。这项工作由XML解析器来完成。因为不同的顾客对Oracle的开发组有不同的需求，所以XML解析器以及附带的XSLT处理程序，其编码有四种不同的语言版本：C，C++，Java以及PL/SQL。另外，XML解析器的不同实现都是可获得的，其中包括命令行可执行程序、库以及jar文件。可以获得几个Java Bean来使可视化开发更容易，它们可用来察看、编辑、转换XML文档。未来对于W3C XML模式、DOM level 2、SAX level 2等等的支持，都将加入到Oracle XDK中。

Oracle XML解析器的Java版本是使用Java写成的，可以在Oracle8i的Java虚拟机上运行。它有可选的合法性验证选项及DTD缓存功能、完全的国际字符集支持、完全的DOM和SAX支持和集成的XSLT处理程序，而且与其他的Java XML解析器相比十分快速。Solaris、Linux、HP-UX以及Windows NT平台的C和C++语言版本的XML解析器也是可获得的，提供了完全的DOM和SAX接口支持、完全的国际字符集支持、集成的XSLT处理程序，而且与其他的C和C++语言的XML解析器相比也十分快速，可以与Oracle8i发行版本2一同得到。PL/SQL版本的XML解析器是基于Java版本的，并允许PL/SQL API调用XML解析器接口，也可以和Oracle8i发行版本2一同得到。

XML TransViewer组件是对一些XML/XSLT类的封装，特别的，有XSLT转换器组件，XML DOM解析器组件，XML源程序察看/编辑组件，以及XML树形视图组件。所有的组件安装后均出现在Jdeveloper 3.1中的XML面板上。最后，还有Java和C++的类生成器，它把DTD作为输入而后生成可以编程调用的代码来产生符合该DTD的XML文档。

XSQL Servlet是一种可以在Web服务器中运行的Java小型服务程序，它可以基于一个或多个SQL查询产生动态的XML文档。它也可以利用XSLT对服务器或客户端的结果XML文档进行可选的变换。它使用声明性的.xsql XML文件来放置指令和变换。实际上，XSQL Servlet建立起到数据库的连接，先剥去标记，再把SQL语句传递给数据库，读出结果集数据并按照标记与数据库表和列的对应关系嵌入到XML中，然后再对XML应用可选的格式页。

1.5 Oracle技术网络与XML链接

XML组件的发布版本可以通过Oracle技术网络得到，其位置在http://technet.oracle.com/tech/xml。Oracle开发组使用该站点的一个映像服务器而无须注册。其Web页面看上去如图1-1和图1-2所示。

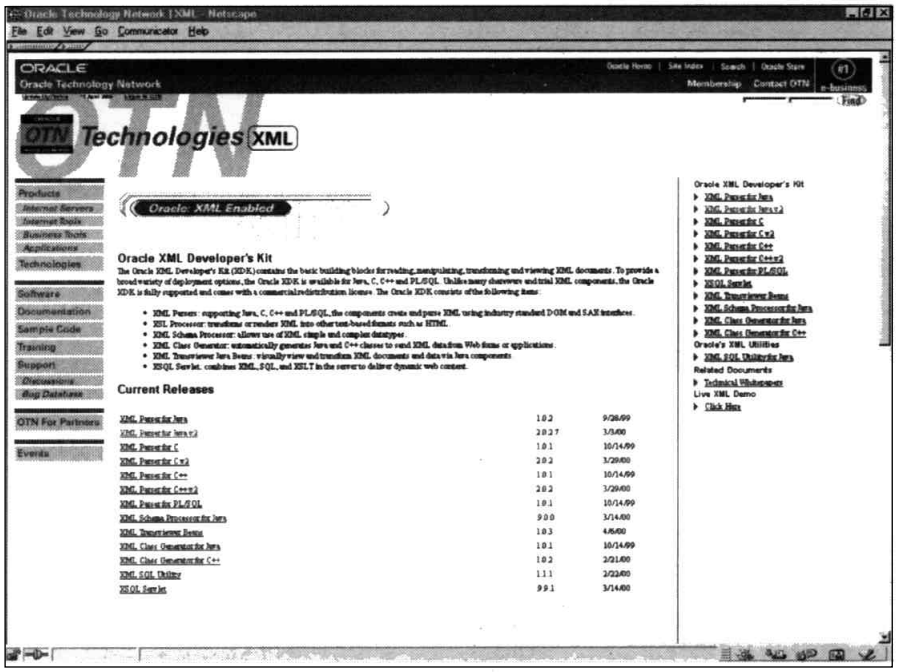

图1-1 XML组件发布版的Web页面

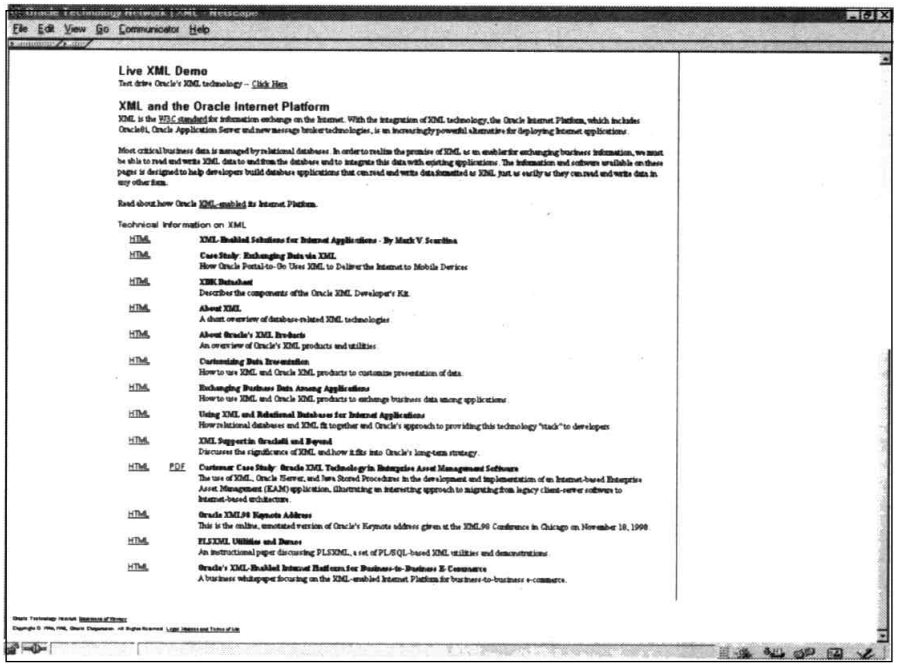

图1-2 XML组件发布版的Web页面（续）

每个XML组件的发布版本都有一个演示区域，包括库 /动态链接库、可执行文件、文档、实例，这些都可以免费地下载、使用以及与其他应用集成。例如， Java的XML解析器版本2的Web页看起来类似于图1-3。

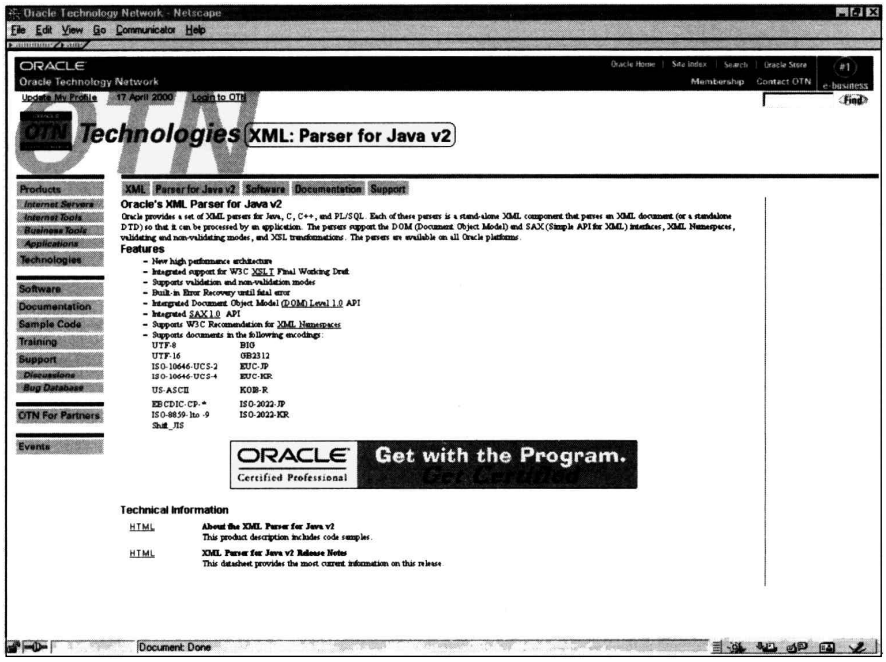

图1-3 OTN上的Java XML解析器版本2的Web页

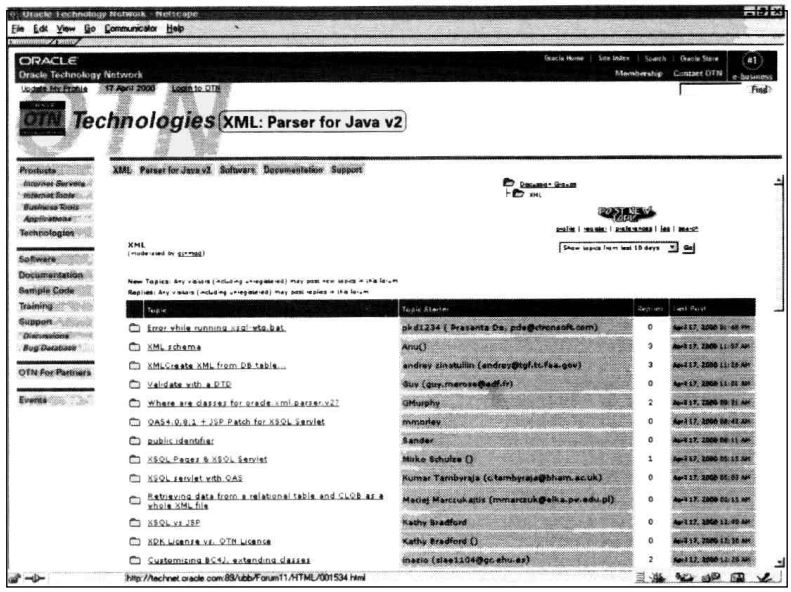

图1-4 XML论坛的Web页面

最后，除了白皮书和其他附件，还有一个供用户群体发表信息、 bug等等的XML讨论论坛。XML讨论区的Web页看上去 如图1-4到图1-6所示

图1-5 XML论坛的Web页面（续）

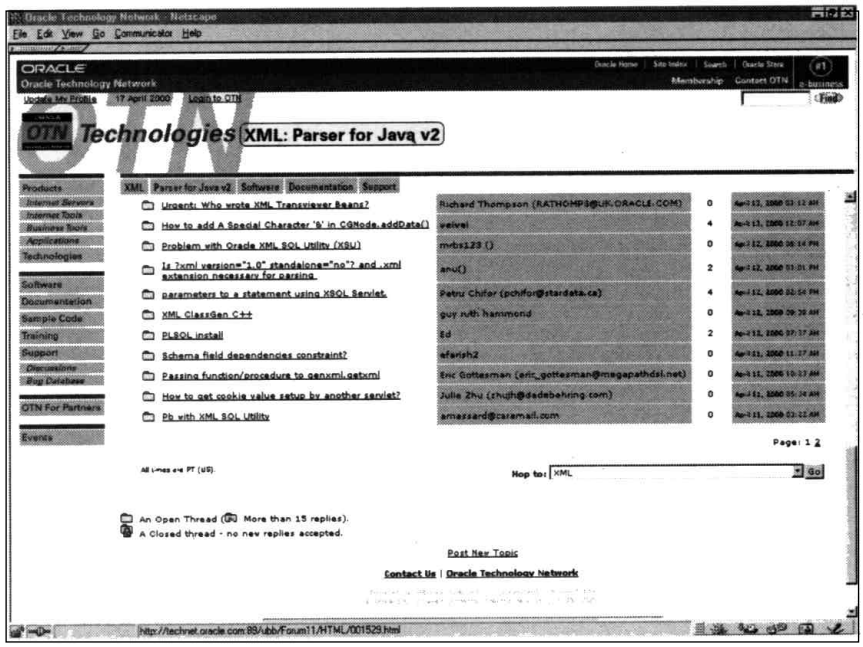

图1-6 XML论坛的Web页面（续）

另外，这些解析器与其他因特网产品一起打包在因特网平台 ISV包（the Internet Platform ISV kit）中。最后，类似于Oracle8i，Jdeveloper 3.1，Applications11i，以及OAS均在其Java虚拟机或库中预先装载了XML解析器，所以通过这些产品应用开发者可以直接访问解析器 API。

1.6 具有XML功能的Oracle产品概览

有40个以上的Oracle内部开发组，涉及到服务器、工具、应用等部门，在其开发的产品中使用了XML组件。他们使用C、C++、Java、PL/SQL语言版本的XML/XSLT解析器来分析和变换XML文档，在C++、Java中使用类生成器来辅助应用开发，利用 XSQL Servlet来提供文档出版和包装服务。这些Oracle开发组和产品包括提供XML API的产品、使用XML进行数据交换的产品、使用XML进行配置的产品、以及使用 XML进行内容管理和出版的产品。

1.6.1 提供XML API的Oracle产品

Oracle已经增强了它的一些服务器端产品和工具以提供可被应用使用的外部 XML API。这些API主要是由上文提到的XML组件提供的，这些组件在各种 Oracle服务器中运行。

1) Oracle 8i Jserver：从Oracle 8i开始，开发者可以使用在数据库内部运行的Java虚拟机。Jserver用PL/SQL加载存储过程的类似方法来加载 Java类。当执行SQL查询时，这些Java类使用内存中的JDBC连接。Jserver最初的发行版本是与JDK 1.1兼容的，而在Oracle8i发行2中的版本是与JDK 1.2兼容的。XDK的Java组件可以加载到Jserver中，这时应用就可以从高性能的数据库访问中受益了。

2) JDeveloper：JDeveloper是Oracle首要的集成开发环境（IDE），从版本3.0开始，已经为内部使用和应用开发集成了 XML功能。通过 JDeveloper可以使用 XDK中所有的 Java组件，而且 XML TransViewer组件可以在工具条面板上安装。从版本 3.1开始，JDeveloper在XML方面进行了重要的增强。除了前面提到的 TransViewer组件面板以外，还增加了额外的数据组件，可以生成XML格式化的数据。编辑器现在可以接受 .xml和.xsl类型的文件并提供语法高亮显示以及语法检查。最后，JDeveloper的Java业务组件对于业务规则和数据使用了 XML格式。

3) Oracle应用服务器：Oracle应用服务器（OAS）是Oracle的中间层服务器产品。 OAS同时提供了Web服务器与应用服务器，还集成了 XDK组件。Java XDK组件可以在 OAS的Java虚拟机中运行，向 Web与Java应用提供XML接口。 PL/SQL组件也可在 OAS的PL/SQL运行环境下运行。对于那些有扩展的 XSL处理或需要支持数以千计用户的应用来说，这种中间层的能力是特别重要的。

4) XML SQL实用程序：XML SQL实用程序是用Java写的，利用了一个已有的 PL/SQL包。它使得开发者可以用 JDBC发出SQL查询并接收结果，结果是包裹在模式的 XML格式中的。除提供了文本流输出外，XML SQL实用程序还能输出DOM结果集对象、DTD、相应于所查询数据库模式的Schema定义。与XDK Java组件相仿，XML SQL实用程序可以在Oracle 8i中作为存储过程加载，可以与任何具有 JDBC功能的关系数据库一同使用。

1.6.2 使用XML用于数据交换的Oracle产品

许多Oracle的应用使用数据库表来交换数据。随着 XML标准的出现，这些应用已采用 XML

作为其数据消息发送和报表格式。

1) Oracle集成服务器：Oracle集成服务器（Oracle Integration Server）使用XML作为与应用之间接受、发送消息的格式。Oracle消息代理器（Oracle Message Broker）把这些XML消息包裹在Java消息服务（Java Message Service）包中，送至已连接的应用。

2) Oracle Reports 6i：Oracle Reports加入了对XML格式的报表、以及为B2B数据交换服务的目录出版的支持。它也支持把XSL格式页应用到报表上，所以报表可以在Internet浏览器中显示。

3) Oracle Application Release 11i：Oracle的企业级业务应用程序套件包括客户关系管理（Customer Relationship Management）、生产/供应链（Manufacturing/Supply Chain）、财政（Financials）、Internet预购（Internet Procurement）、人力资源（Human Resources）以及计划（Projects）。为进行集成，这些应用把XML作为交换数据和发送消息的格式。

4) 自助式购物：自助式购物（Self-Service Purchasing）使用XML文档作为“购物车”在买卖双方之间交换数据。它也使用XML来格式化目录信息，并在线提供给顾客。

1.6.3 使用XML进行配置的Oracle产品

应用的配置文件过去的格式是平面文件，需要用定制的分析器来读取数据。在使用XML格式后，应用就可以利用XML易于扩展的结构，并建立基于开放标准的分析器来读取数据。

1) Oracle Designer 6i：这是一个快速应用开发工具，它利用组件库来辅助开发。这个库使用扩展元数据交换格式（Extensible Metadata Interchange，XMI）来描述存储于其中的组件的行为和属性。XMI是XML的标准子集，可以使组件在Web上开放交换。

2) Internal Release Application：Internal Release Application是Oracle内部的一个应用，用Java设计，使用XML格式的文件来进行配置和表示产品状态。

3) Oracle Configurator Developer：该产品为Oracle Application Release 11i提供了配置模拟环境，后者是为开发销售模型提供拖放接口的。XML用于格式化配置规则，这些规则是用来在诸如对象状态、数量型建模参数、产品兼容性及相关性等领域对模型进行限制，以及用于设计图表绘制。

1.6.4 使用XML进行内容管理和出版的Oracle产品

内容管理和出版是XML影响最大的主要的应用领域。Oracle有若干产品支持该应用领域，并利用XML使它们可以读，写，搜索XML文档，并用XSL对它们进行包装。

1) Oracle Internet文件系统（iFS）：Oracle Internet文件系统是Java API的集合，它以灵活的方式对需要在Oracle 8i中存储文档的应用开发者提供了文件级的接口。包括XML在内的很大范围内的文档类型，不仅可以完整的存储为大对象（LOB），而且也可以被分析后拆分成部分以分布式的方式存储到模式中。这种映射可以在.typ类型的文件中指定。

2) interMeidia Text：interMedia Text是在Oracle8i中用的文本搜索引擎。它在XML方面提供的支持包括：索引存储在字符大对象中的XML文档和通过SQL查询来发现元素和属性的能力。

3) Portal-to-Go：Portal-to-Go是设计用来向很大范围内的有线或无线设备提供信息服务的服务器，包括蜂窝电话（cell phones），PDA，记事本（pagers），及掌上机（Palmtops）等。它使用XML进行信息的供给和输出，Portal-to-Go中包含了与设备相关的网关，来把经过 XSL转换的XML文档转换到合适的接收设备格式。

4) Oracle Discover 3i：本产品为使用XML完全重新设计，不仅进行数据传输，而且也对组件的行为和属性进行封装。Discover与XSL一起很容易的提供了定制的Web报表和用户界面。

1.7 Oracle XML组件的使用概览

Oracle的XML组件不仅在其内部使用，前面已提到过，而且在许多应用中也会用到，这些应用通常要求数据在应用间或企业间具有高度的可移植性。下面是这些应用中的一些的列表和简单描述，在你通读全书时，请把它们记在脑子里。

1. 文档生成与出版

一个公司有一个或更多的文档库，由 XML或SGML写的标记文字片断组成，并且想动态地出版整合后的文档。这可以使用 XSL格式来组装片断，然后以电子形式把整合的文档发送给用户。

2. 个性化信息发送服务

一个公司从各种数据源接受输入，使用 XSL来使之规范化，然后存入数据库。这种信息存储过程还可以用来作为一个或更多的 Web站点或门户的后端，而它们可以从各种有线或无线客户接受http请求。利用XSL格式页和XSQL Servlet，就可以向请求的设备动态地发送合适的外观。

3. 易于定制的数据驱动应用

建立一个用于向瘦客户端发送数据并提供交互性的应用。数据由数据库中查询得来，与用户界面组件一起，通过一个或多个 XSL格式页进行包装后发送给客户应用。存储形式可以是由XML具体化而来的关系型数据，也可是存储在大对象中的 XML。

4. 使用XML购物车的电子商务

一个电子商务Web站点把顾客的订单编辑成 XML格式然后将内容发送给一个或多个生产厂家的数据库进行处理。XSL用来转换和划分购物车以进行兼容的传输。数据由 XML具体化而来，以关系型存储。

5. 业务到业务互联网消息传送

一个复合的客户-服务器或服务器-服务器应用把数据资源或目录存储在库中，而该库可在企业间共享。资源的发放触发一个有效的 XML消息，该消息使用XSL将资源转换成多个与客户兼容的格式。反过来，当一个客户获得资源时将发消息给其他客户，通知已取走资源。消息存储在大对象中，数据由XML具体化而来，以关系型存储。

6. 通过XML消息发送的应用集成

若干应用需要进行通讯及共享数据，以进行业务的集成或处理。 XML在经XSL转换后，被包装并被路由，成为消息的有效负载。这些 XML消息存储在大对象中。

1.8 实例与应用

当使用Oracle XML开发包（XDK）时，能更快地写出应用程序，这些应用都利用 XML的优势来促进电子商务。开发者不需要使用编码来分析 XML文件，按对应的格式页转换 XML文件，从数据库表中获得数据并将其转换成 XML，而只需要针对由 XDK提供的开放标准的 API进行编码，很快就可以让应用程序作要做的事了。这样，如果用户从 Web站点上输入的数据需要封装成 XML格式并转送到其他站点，开发者使用 XDK很快就可以做出可用的应用程序。一个这样的例子将在本节描述，是一个可以使书籍销售在互联网上进行的应用程序。

Amazingly Enjoyable Books是一个Web站点，其所有者需要一个应用程序来捕获用户在 Web站点上的输入数据、把数据存储为 XML并把XML存储到数据库中，以及以合适的形式获得数据。作为程序员，你必须在一天内写出这个应用程序，因为 Web站点的所有者已经许诺，如果你可以在明天前把事情做完，让程序运行起来，她可以付给你三倍的报酬。想起来 Oracle技术网络的XML站点上有若干有用的XML组件和实用程序，然后你就去了http://technet.oracle.com/tech/xml，开始下载例如Java版本的XML/XSLT解析器和XSQL Servlet之类的组件。你可以从实例程序开始，然后就会看到可以多么快地为该 Web站点的所有者完成该程序。

在与店主的对话中，你意识到在 Web站点上由顾客所提供的数据是最小化的，这使你的工作颇为容易，然后你立刻写出了决定存储顾客数据的 XML文档结构的DTD：

```
<!-- DTD book customer list may have a number of book customers -->
<!DOCTYPE bookcustomerlist [
<!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,
               purchasedate)>
<!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)>
]>
```

你开始写出应用程序，捕获由用户输入的数据并将其存储在与你指定的 DTD相容的内存XML结构（或外部文件）中。例如，用户界面可能会类似于图 1-7中的表单。

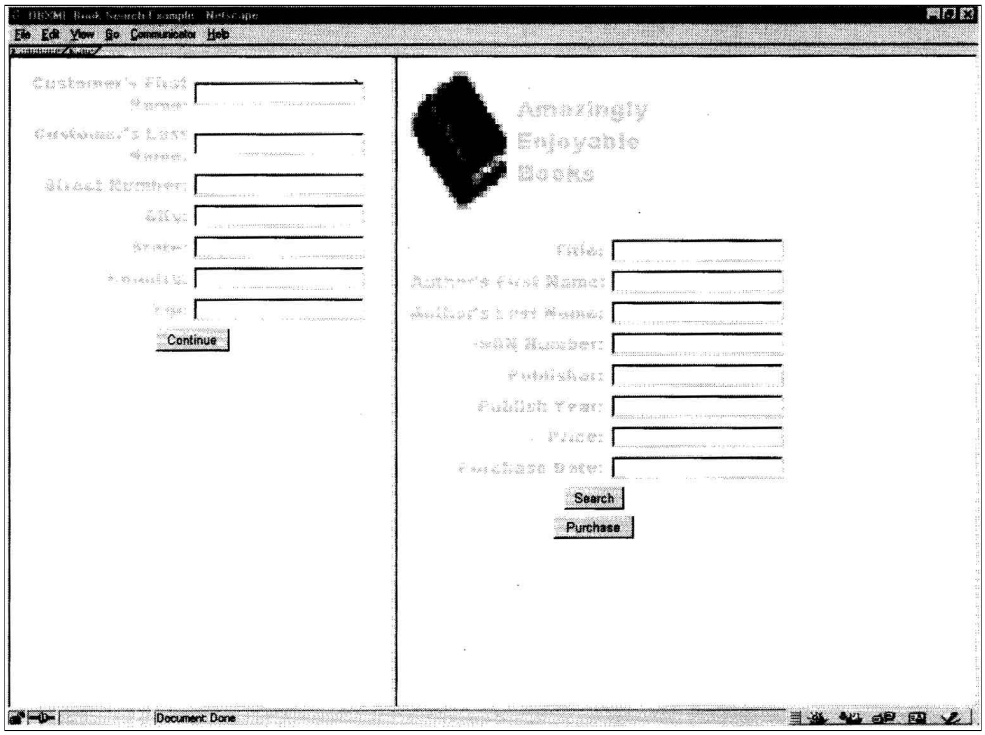

图1-7 书籍搜索表单示例

为在服务器上向数据库存储和获取信息，要写出 SQL语句，这样就需要让 XSQL Servlet对XML文件、可选的XSL文件、以及构造出的SQL语句进行操作。然后由在 Web服务器上运行的XSQL Servlet建立到数据库的连接，并在后端服务器上对相关的数据执行 SQL语句，然后可以获取来自数据库表的行集数据，转换为 XML，对其应用XSL格式页，如果有获取信息的请求就将其由Web服务器传送到客户端浏览器。如果处理数据的操作只是插入数据，那么获取信息的操作就没有必要了。

下面的XML文件可以作为一个入口，用于从数据库表中获取信息，将结果转换成 XML，再将它传送给其他应用或Web站点。

```
<!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>9999</ISBN>
     <publisher>Oracle Press</publisher>
     <publishyear>2000</publishyear>
     <price>1.00</price>
     <purchasedate>January 1, 2000</purchasedate>
</book>
          <totalsales>1000.00</totalsales>
     </bookcustomer>
</bookcustomerlist>
```

本书还会对该应用的细节进一步讨论，该应用是 XML最具影响的领域。