

关于处理大型 XML 数据的 NXD 方法研究

李鹏飞, 吴洁, 丁秋林

(南京航空航天大学 计算机应用研究所, 江苏 南京 210016)

摘要: XML 作为 SGML 标记语言的一个子集, 由于它能很好地表示结构化和半结构化数据, 而逐渐成为 Internet 上或应用程序间数据交换和信息表示的标准。分析和处理 XML 文档的场合也越来越多, 其方法和工具也有很多, 然而, 对于很大的文档, 传统的处理方法存在着很多的缺点和不足之处。文中提出了一种新的分析处理 XML 文档的方法, 即利用 Native XML Database(NXD), 以提高分析处理的性能。

关键词: XML 数据; 查询; native XML 数据库

中图分类号: TP311.138

文献标识码: A

文章编号: 1005-3751(2006)03-0179-03

Research of Processing Large XML Data Using NXD

LI Peng-fei, WU Jie, DING Qiu-lin

(Computer Application Institute, Nanjing University of Aeronautics & Astronautics, Nanjing 210016, China)

Abstract: As a subset of SGML markup language, XML can represent structured and semi-structured data very well, so it has been the standard of data exchange and information represent on Internet or among applications. It is more and more important to parse and process XML document, and there are more and more methods and tools to do this job. However, for a large XML document, traditional processing methods have various drawbacks. A new method using native XML database to parse and process XML document, especially for large document, is proposed in this article to improve the performance.

Key words: XML data; query; native XML database

0 引言

目前, Web 上的大量应用程序都使用 XML 作为表示结构化或半结构化数据的中间格式和数据交换的标准, 很多软件尤其是数据库厂商都对他们的产品增加了对 XML 的支持, 分析处理 XML 文档的方法、工具及途径也很多^[1]。

传统的存取 XML 文档中信息的方法主要有如下几种: 第一种方法是直接以文本文件的形式进行存储, 并利用 DOM, SAX 或 JDOM 对 XML 文档进行分析处理, 然后用 XPath 对其进行查询。另外一种方式是将 XML 文档以一定的方式映射并储存到关系数据库中, 对 XML 的查询转换为对关系数据库的 SQL 查询, 最后再将查询的结果转换为 XML 文档的形式呈现给用户^[2]。第三种方法是将 XML 数据映射为对象-关系^[2]或面向对象数据库^[3]。这种方法和第二种方法并无本质上的区别, 但它更受人们的青睐, 其原因如下: 首先, 很多数据库厂商都将他们的产品在关系数据库的基础上集成了对象-关系特

点, 使其成为对象-关系数据库, 这表明对象-关系数据库具有纯关系数据库的所有优点; 其次, 与纯关系数据库相比, 对象-关系数据库具有更为强大的数据类型系统; 最后, 对象-关系数据库支持层次结构, 能更好地实现与 XML 数据的映射。最后一种方法是使用 Native XML Database 来保存和查询 XML 数据。

在上述几种方法中, 第一种方法比较直接, 但是由于 DOM 是将 XML 数据完全载入到内存并构建成树的形式来表示 XML 信息, 所以其需要较大的内存空间和大量的 CPU 时间, 效率很低。SAX 不需要将整个文档载入内存, 但是编程比较复杂, 尤其是当文档的层次结构比较复杂时^[4]。文中将探讨采用 Native XML Database 存取 XML 文档, 并进行查询分析的方法, 并通过实践证明, 同其他方法相比, 它有很多的优点, 尤其是对于处理大文档, 它能很好地提高性能。

1 Native XML Database(NXD)及其优点

1.1 Native XML Database

Native XML Database 是一种特殊的数据库, 专用于存取 XML 数据, 其记录格式是以 XML 数据项进行存储的。XML:DB Initiative 对其定义如下^[5]:

* 为 XML 文档定义了一种逻辑模型, 并根据该模型

收稿日期: 2005-06-04

作者简介: 李鹏飞(1982-), 男, 安徽桐城人, 硕士研究生, 主要从事计算机应用研究; 吴洁, 副教授, 主要从事需求管理与系统架构研究; 丁秋林, 博士生导师, 研究方向为企业信息化、系统集成、CIM、CAD、CAM、MIS。

对文档进行存储和查询。该模型至少应包含元素,属性,PCDATA 等。

* 以 XML 文档作为其基本(逻辑)存储单位。正如关系数据库的基本存储单位是数据表格中的一行。

* 不需要任何底层的特定物理存储模型。例如,它可以建立在关系、层次或对象-关系数据库的基础上。

目前,Native XML Database 产品很多,主要有:Sleepycat's DB XML, Xindice, eXist 等。文中将以 eXist 为例。

1.2 基本概念

* 驱动器(Driver):

类似于 ODBC, JDBC 中的数据库驱动器,它封装了所有的数据库访问逻辑。驱动是由 Database 接口实现的,并由 DatabaseManager 类进行管理^[5]。

* 集合(Collection):

在 Native XML Database 中, collection 是用于保存 XML 文档的容器。与关系数据库相比, collection 类似于数据表格^[5]。

* 服务(Service):

Native XML Database 在设计时充分考虑到了其灵活性和可扩充性。这是通过 service 来实现的。事实上,你可以不使用任何服务而可以做很多有用的工作。使用得最广泛的服务是 XPathQueryService,它是用来在数据库中执行 XPath 查询的。其他的服务包括 XUpdateQueryService, CollectionManagementService 等^[5]。

1.3 Native XML Database 的优点

与其它的存取分析方法相比, Native XML Database 具有如下的优点^[5]:

* XML 数据可以直接存入数据库,不需进行任何操作或将数据从文档中提取出来。

* 将 XML 文档插入数据库时,文档的绝大多数特征,包括空格,都能得以完整地保存下来。

* 查询返回的结果是 XML 文档或文档片断,这就意味着 XML 信息的层次结构得以保存。同时,所返回的 XML 文档片断,既可以由其它应用程序进行后续处理,也可以通过 CSS 或 XSL 转换,以友好的方式呈现给查询的用户。

2 分析处理的方法

使用 Native XML Database 进行分析处理文档的基本思路是:先将

XML 文档保存到 Native XML Database 中,然后利用 XPath 进行查询分析,最后将查询的结果(XML 文档片断)进行格式化或转换以呈现给用户,或者直接提交给应用程序进行后续处理^[4]。

下面将以文档 switchstates.xml(见图 1)为例,介绍上

述方法在“扬州宝军公司的汽车行驶记录仪软件开发”项目中的应用。文档 switchstates.xml 用于记录车辆在行驶过程中开关量的变化。

2.1 存储 XML 文档

首先,导入必要的包:

```
import org.xmldb.api.*;
import org.xmldb.api.base.*;
import org.xmldb.api.modules.*;
```

然后注册所需的数据库驱动器,这与用 JDBC 访问关系数据库比较类似^[4]:

```
String driver="org.exist.xmldb.DatabaseImpl";
Class cl=Class.forName(driver);
Database database=(Database)cl.newInstance();
DatabaseManager.registerDatabase(database);
```

最后创建数据库实例,并将 XML 文档保存在其中:

```
Collection root=DatabaseManager.getCollection("URI"+"db");
```

```
CollectionManagementService mgtService=(CollectionManagementService)root.getService("CollectionManagementService","1.0");
```

```
Collection col=mgtService.createCollection("switchstates");
```

```
XMLResource doc=(XMLResource)col.createResource(null,"XMLResource");
```

```
File f=new File("switchstates.xml");
```

```
doc.setContent(f);
```

```
System.out.println("storing document "+doc.getId()+".....");
```

```
col.storeResource(doc);
```

以上程序段将文档 switchstates.xml 保存在数据库的 switchstates 集合中。

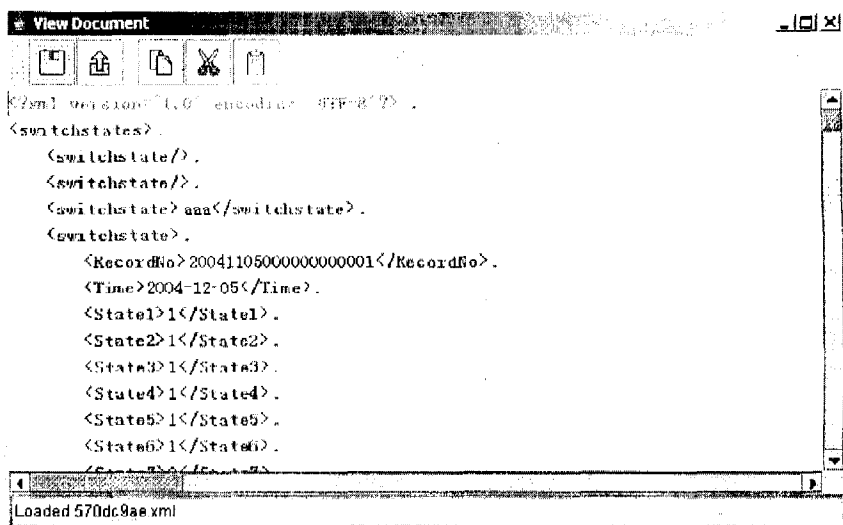


图 1 存储的 XML 文档: switchstates.xml

2.2 进行查询和处理

下面将查询出 2004 年 12 月 6 日的所有记录,即 Time 属性为 2004-12-06 的 switchstate 元素。

```
Collection col=DatabaseManager.getCollection("xml:db:exist://localhost:8080/exist/xmlrpc/db/switchstates");
```

```

XPathQueryService service =
    (XPathQueryService) col. getService("XPathQueryService",
    "1.0");
service.setProperty("indent", "yes");
ResourceSet result = service.query("//switchstates/switch-
state[Time='2004-12-06']");
ResourceIterator i = result.getIterator();
while(i.hasMoreResources())
{
    Resource r = i.nextResource();
    System.out.println((String)r.getContent());
}

```

查询所得的结果为(部分):

```

<switchstate>
  <RecordNo>20041105000000000001</RecordNo>
  <Time>2004-12-06</Time>
  <State1>1</State1>
  <State2>0</State2>
  <State3>1</State3>
  <State4>1</State4>
  <State5>1</State5>
  <State6>1</State6>
  <State7>1</State7>
  <State8>0</State8>
</switchstate>
<switchstate>
  <RecordNo>20041105000000000001</RecordNo>
  <Time>2004-12-06</Time>
  <State1>1</State1>
  <State2>0</State2>
  <State3>1</State3>
  <State4>1</State4>
  <State5>1</State5>
  <State6>1</State6>
  <State7>0</State7>
  <State8>0</State8>
</switchstate>

```

可见,Native XML Database 可以有效地实现对 XML 文档的查询分析。同时可以看出,Native XML Database 的一个重要特点是,直接将 XML 文档存入数据库,并可以直接从数据库得到所需的文档片断,在 XML 文档和数据库之间不需要任何的转换工作。

3 与传统方法的比较

与传统的分析处理 XML 文档的方法(如 DOM)相比,Native XML Database 主要解决了下面两个问题:

1)它避免了将大文档载入内存然后进行查询处理所需的内存需求和 CPU 处理时间。使用 DOM 时可能会遇到这个问题,例如处理一个 100MB 的文档可能需要大约 1.2GB 的内存。

2)要想访问文档中的某一个部分,必须先要对整个文档进行分析处理,才能得到所需的文档片断。对于一个较小的 XML 文档来说,这可能不是什么问题,但对于较大的文档尤其是需要通过网络传输时,会造成对网络带宽和处理器时间的极大浪费。而 Native XML Database 可以不经任何处理,就可以得到所存储的文档中的任意片断。

与关系或对象-关系数据库相比,Native XML Database 具有如下的优势:

(1)由于 XML 数据是半结构化的,如果将其映射到关系数据库,结果是要么出现大量空值(null)的字段,要么表格的数量过多,浪费空间或效率低下。虽然半结构化的数据可存储到面向对象的或层次型的对象-关系数据库中,你还可以选择将它以 XML 文件的形式存储于 Native XML Database 中。

(2)读出速度。根据 XML 数据库存储数据的物理方式的不同,数据的读出速度可以做到比关系型数据库快得多。其原因是,Native XML Database 对整个文件一起进行物理存储,和表示文件各个部分的物理(而不是逻辑)指针可采用同一存储策略。这就可以不使用连接(joins)或只使用物理连接读取文件,无论哪种情况都比关系型数据库所用的逻辑连接要快。

4 相关工作

关于 XML 数据的存取,相关的工作主要集中于如何将 XML 文档映射并存储于关系或对象-关系数据库中,然后将对 XML 的查询、分析处理转化为对数据库的查询。如 Kanda Runapongsa 和 Jignesh 提出的 XORator 算法(XML to OR Translator),是使用 DTD 将 XML 文档映射为 ORDBMS 中的表格。该算法的关键部分是将 XML 文档中的片断赋予一种新的 XML 数据类型,称为 XADT(XML Abstract Data Type)^[3]。

5 结论

文中探讨了借助于 Native XML Database 对 XML 文档进行查询分析的方法,及其相对于其它传统方法所具有的优点。通过分析和实践证明,对于较大的 XML 文档,采用 Native XML Database 可以很好地提高查询分析的性能。

参考文献:

- [1] World Wide Web Consortium. Extensible Markup Language (XML) 1.0 [EB/OL]. <http://www.w3.org/TR/1998/REC-xml-19980210>, 1998-02.
- [2] Dayen I. Storing XML in Relational Databases [EB/OL]. <http://www.xml.com/pub.>, 2001-06-20.
- [3] Runapongsa K, Patel J M. Storing and Querying XML Data in Object-Relational DBMSs [M]. Heidelberg, Berlin: Springer-Verlag, 2002.

(下转第 184 页)

自动生成一个任务号为 nmcJobId;

3) PMIRP 将该请求下发给性能适配器 PMAAdapter;

4) 性能适配器根据 NMC 的 moc 参数发现定义的是交换的测量任务, 进一步将该请求下发给性能交换适配器 PMAAdapterMSS;

5) PMAAdapterMSS 解析该任务的 moi 参数, 获得需要定义任务的局号 (假设为下级局 100001), 并通过级联服务, 在指定的下级局 serverId (100001) 查找 CORBA 使用的 EJB 服务 CorbaUseEJB, 并通过 CorbaUseEJB 访问拓扑管理, 获得该下级局的网元类型 (MSC, HLR, MSCHLR 合为一局)。

6) PMAAdapterMSS 获得需要定义任务局的网元类型后, 根据网元类型与设备 Id 的对应关系, 可以获得需要定义任务局的设备 Id 为 deviceId。

7) PMAAdapterMSS 解析该任务, 分解出 CAF 任务和参数任务 (包括电路群、目的码、No7 链路等)。

8) 对于 CAF 任务, PMAAdapterMSS 通过 JNDI 查找到 CAF 任务管理 EJB PMTaskManager, 并将需创建的 CAF 任务信息 TaskInfo 和刚获得的该局的设备 Id 传给 PMTaskManager 来创建 CAF 任务。创建成功后返回 cafJobId。

9) 对于参数任务, PMAAdapterMSS 通过级联服务, 在指定的下级局 (100001) 查找 CORBA 使用的 EJB 服务 CorbaUseEJB, 由 CorbaUseEJB 访问交换性能 MAFEJB, 来完成参数任务的创建。创建成功后返回 paraJobId。

10) PMAAdapterMSS 得到 cafJobId 和 paraJobId 后, 通过访问 MSSCorbaEJB, 由 MSSCorbaEJB 将 nmcJobId, cafJobId, paraJobId, serverId 插入性能 CORBA 数据库的 nmc2OmcJob 表中。

11) PMAAdapterMSS 将任务创建的结果信息 CreateJobResult, 返回给适配器。

12) 适配器将结果返回 PMIRP。

13) 如任务创建成功, PMIRP 将任务信息记录入 PmMeasJob.xml 文件中。

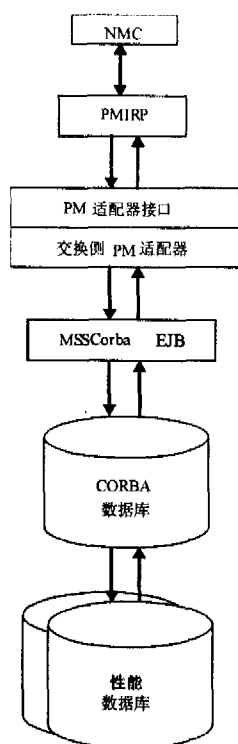


图 3 性能 CORBA 实现原理图

14) PMIRP 向 NMC 返回任务号。

15) 创建任务流程结束。

2.2.2 性能告警

性能告警包括以下几步:

1) PMIRP 扫描性能门限 XML 文件。

2) 性能告警判断时间到, PMIRP 将属于交换的性能门限信息 MonitorInfo, 发送给交换适配器 PMAAdapterMSS, 并询问是否有新的性能告警。

3) PMAAdapterMSS 向本级 MSSCorbaEJB 请求性能门限指标对应的当前数据。

4) MSSCorbaEJB 组织 SQL 语句, 从性能 CORBA 数据库结果表中查询出当前数据, 并返回给 PMAAdapterMSS。

5) PMAAdapterMSS 根据门限指标和当前值, 判断是否发生告警及告警级别。

6) PMAAdapterMSS 组织 NotifyAlarmInfo 列表, 并返回给 PMIRP。

7) PMIRP 遍历 NotifyAlarmInfo 列表, 判断其中是否有新的告警。

8) 如有, PMIRP 向 NMC 发送新的性能告警通知^[5]。

3 结束语

采用 CORBA 的分布式处理方法设计性能管理系统, 实现运营商要求的接口, 性能管理模块已经实现操作系统无关性、通信协议无关性、编程语言无关性、对象定位的透明性和服务器的透明性等优点。由于支持 CORBA 的多层处理构架, 保证了性能管理系统在不同的软硬件平台上均具有良好的移植性。目前基于 CORBA 的性能管理已经投入商用, 取得了良好的社会和商业效果, 运营商反映非常好, 希望能在其他方面继续扩展 CORBA 在 TMN 中的应用。

参考文献:

- [1] 薛君教. CORBA 系统结构、原理与规范[M]. 北京: 电子工业出版社, 2000.
- [2] 张国鸣, 唐树才, 薛刚逊. 网络管理实用技术[M]. 北京: 清华大学出版社, 2002.
- [3] Subramanian M. 网络管理原理与实践[M]. 王松, 周靖, 孟纯城译. 北京: 清华大学出版社, 2003.
- [4] 中国联通有限公司移动部, 北京邮电大学. CDMA95/1X 网络管理技术规范[M]. 北京: 中国联通有限公司, 2003.
- [5] Eckel B. Java 编程思想[M]. 京京工作室译. 北京: 机械工业出版社, 2000.

(上接第 181 页)

- [4] Wilcox M. Using Embedded XML Database to Process Large Document [EB/OL]. <http://www.xml.com/pub/a/2003/10/22/embed.html>, 2003-10-22.

- [5] Staken K. Introduction to Native XML Database [EB/OL]. <http://www.xml.com/pub/a/2001/10/31/nativexmlldb.html>, 2001-10-31.