一种简单高效的XML与关系数据库信息交换的方法

蔡 飞 贝 佳 潘金贵

(南京大学计算机软件新技术国家重点实验室 南京大学计算机科学与技术系 南京210093)

摘 要 关系数据库是应用广泛且成熟的数据库技术。随着Internet的飞速发展,XML作为数据存储和交换的手段也越来越受到重视。在很多Web服务和应用中,需要在XML和关系数据库之间进行数据交换,目前这种数据交换的实现主要依赖于XSLT(XSL Transformations),这种实现存在书写复杂、效率较低等缺点。针对这些缺点,设计了一种不同于XSLT的新型映射规则,并且通过程序实现和大量的测试比较,证明它能够更加简单高效地实现这种数据交换。

关键词 XML,关系数据库,信息交换

A Simple and Effective Method of Data-Exchange between XML and Relational Database

CAI Fei BEI Jia PAN Jing-Gui

(The State Key Laboratory for Novel Softwave Technilogy,Department of Computer Science and Technology, NanJing University,NanJing 210093)

Abstract Relational database is a kind of developed database technology and widely used. XML has also become the standard of data storage and exchange. In many web services and applications, data-exchange between XML and relational database is needed, which is implemented by XSLT currently. XSLT is a kind of complex low-performance transformation. This paper offers a new kind of mapping rule. After the design work of mapping rule, we implement the system and test it. Results show that the data-exchange can be implemented more simply and effectively in this way.

Keywords XML,Relational database,Data-exchange

1 引言

在基于网络的商业服务中用XML对数据进行标记、表示和传输得到越来越多的应用,因此XML数据的分析和管理也就成为网络应用的研究热点。关系数据库经过数十年的发展,以其成熟稳定的优点在现代信息系统中发挥着极其重要的作用。在很多Web服务和应用中,为了利用关系数据库在查询、索引、事务控制和并发控制等方面的优点,往往使用关系数据库作为数据存储的载体,而XML只是作为数据交换的载体,因此,XML和关系数据库之间的信息交换便是一个关键的过程。

在将XML与数据库结合并交换数据的过程中,主要存在以下需要研究的问题和克服的困难:XML数据模型和关系数据模型之间的映射[1];如何将XML存入数据库;如何从数据库中查询出XML。

现有的商业数据库,例如Oracle,正在试图解决这个问题,在原有的数据库基础上通过自定义XML与数据库的映射模式提供了XML扩展的功能,并且提供了相应的工具,即XSU。经过研究,我们发现,包括XSU在内的这类工具的处理方法都存在使用繁琐、映射规则定义复杂、效率低下等缺点,因此,有必要研究更为简单高效的操作模式。

2 目前常用的处理方式

XSU是Oracle提供的工具,用于在XML和关系数据库之间进行数据交换,即使用关系数据库存储XML的数据,或者使用XML的形式表达数据库中存储的内容是具有代表性的一个工具。下面将使用这个工具解释目前常用的处理方法。

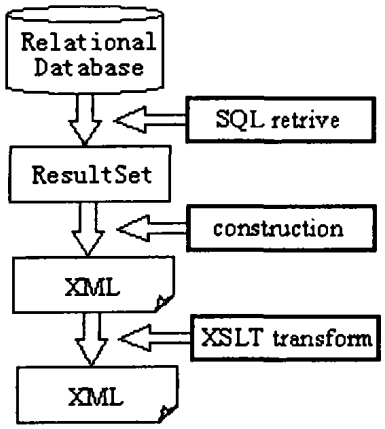

图1 XSU中关系数据库→XML的流程图

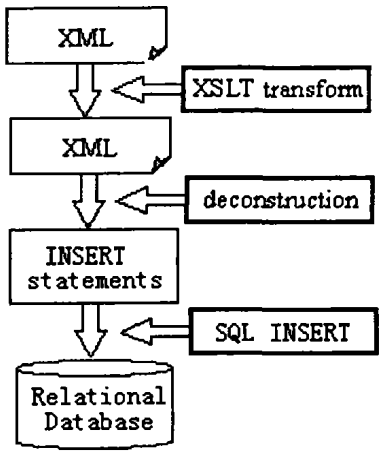

图2 XSU中XML→关系数据库的流程图

蔡 飞 硕士生,主要研究领域为XML数据管理。

在实现以上两个功能时，XSU 的做法分别是：

(1) 用 XML 的形式表达数据库中存储的内容(图1)：a. 使用 SQL 语句查询关系数据库，获取 ResultSet；b. 书写 XSLT；c. 使用 ResultSet 生成 flat XML；d. 使用 XSLT[2]转换 XML 文档结构，将 flat XML 转换成目标 XML 文档。

(2)使用关系数据库存储 XML 的数据(图2)：a. 书写 XSLT；b. 使用 XSLT 转换 XML 文档结构，将 XML 文档转换成 flat XML；b. 从 flat XML 中提取记录的数据；c. 使用 SQL 语句将记录插入数据库。

使用这种作法有以下两个缺点：(1) XSLT 的书写比较复杂；(2) XSLT 的执行效率比较低。针对现有技术的以上两个缺点，我们提供了一种更加简单、高效的信息交换的方法。

3 新的映射机制和处理方法

在 XML 与数据库之间的信息交换中，需要一种完善的映射机制。现有的做法就是使用 XSLT 实现这样的映射，但是 XSLT 实现的不是数据库和 XML 文档的映射，而是 XML 文档之间的映射，我们设计了一种易于书写的映射规则集，并且能够实现数据库和 XML 文档的映射，省略了生成 flat XML 的中间过程，从而达到简单、高效的目的。在将 XML 映射到数据库时，有两种基本方法：(1)将 XML 的一个元素或属性映射到表的一列；(2)将整个 XML 或 XML 片段映射到表的一列。

映射规则集的设计是这种新型处理模式的关键内容，它是使用 XML 表述的一组规则，分为两个部分，分别用于使用 XML 的形式表达数据库中存储的内容和使用关系数据库存储 XML 的数据，我们将这两个部分命名为 XMLGenerator 和 XMLSaver。

3.1 XMLGenerator

3.1.1 *处理流程* 图3为 XMLGenerator 的模块示意图，从图中可以看出，XMLGenerator 模块不涉及从数据库中取数据的操作，只负责实现从关系数据即 ResultSet 到 XML 的转换。

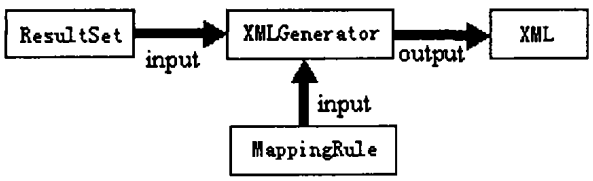

图3 XML Generator 模块示意图

使用 XMLGenerator 主要有以下步骤：

(1)用户使用 SQL 查询语句从数据库中获取查询结果；

(2)书写映射规则，如果不需要转换 XML 的结构，只需要 flat XML，可以省略这一步；

(3)XMLGenerator 接受 ResultSet 和映射规则作为输入，根据映射规则生成特定结构的 XML。

3.1.2 *映射规则集* XMLGenerator 的映射规则集的基本规则是：(1) 数据库的每一行转化为一个 XML；(2) 根据一定规则，某些行合并为一个 XML。它表达了：

·所生成的 XML 的结构

·二维表中的列与 XML 的元素或属性之间的对应关系

·合并行的规则

·XML 的格式

文[3,4]中详细介绍了 XMLGenerator 模块的映射规则集的设计和书写，在这里我们不作重复的讨论。

3.2 XMLSaver

3.2.1 *处理流程* 图4为 XMLSaver 的模块示意图，从图中可以看出，XMLSaver 模块不仅实现从 XML 到关系模型的分解，而且要生成相应的 INSERT 语句，将数据插入到数据库的表中。

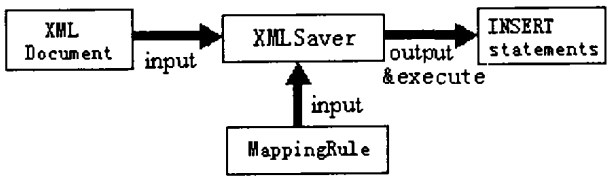

图4 XMLSaver 模块示意图

使用 XMLSaver 主要有以下步骤：

(1)指定 XML 数据源，可以是一个文件、DOM 树，或者字符串；

(2)书写映射规则；

(3)XMLSaver 接受 XML 数据源和映射规则作为输入，根据映射规则生成 INSERT 语句并执行，向数据库的表插入数据。

3.2.2 *映射规则集* XMLSaver 的映射规则集的基本规则是：(1)XML 的元素或属性转化为二维表的一列；(2)据一定规则，将 XML 的重复结构转化为表的多行。它表达了：

·XML 的元素或属性与二维表中的列的对应关系

·将重复的结构转化为多行的规则

XMLSaver 的映射规则集应该考虑到数据对应、重复结构的定义，针对模型映射的需要，XMLSaver 定义了两种映射规则：

(1)targetRule：表示 XML 中的元素和数据库的表之间对应关系的规则，包括 id，table，column 和 path 等域。

(a) id：规则的标识，在规则集中唯一。

(b) table：表示对应数据库中表的名称。

(c) column：表示对应的列名。

(d) path：表示数据在 XML 中的位置，以 XPath[5]的形式给出。

(2)repetitionRule：表示一条重复结构的定义，以 path 定义的结构为单位对应到 table 定义的表中的一行，包括 id，table 和 path 等域。

(a) id，table：作用同 TargetRule。

(b) path：用 XPath 定义 XML 的重复结构，重复结构出现的次数对应表中的行数。

3.2.3 *应用示例* 以下的文档1和文档2分别是有重复结构和没有重复结构的 XML 文件的典型情况。

```
〈?xml versin="1.0"?〉
〈PERSON〉
  〈NUM〉001〈/NUM〉
  〈NAME〉GOLD〈/NAME〉
  〈JOB〉KING〈/JOB〉
  〈DETAIL〉
    〈LOCINFO〉
      〈LOCNUM〉A01〈/LOCNUM〉
      〈LOC〉TOKYO〈/LOC〉
    〈/LOCINFO〉
  〈/DETAIL〉
〈/PERSON〉
```

文档1

```
〈?xml〉version="1.0"?〉
```

```
〈LIST〉
  〈PERSON〉
    〈NUM〉001〈/NUM〉
    〈NAME〉GOLD〈/NAME〉
    〈JOB〉KING〈/JOB〉
    〈LOCNUM〉A01〈/LOCNUM〉
    〈DETAIL〉
      〈LOC code="A01"〉TOKYO〈/LOC〉
    〈/DETAIL〉
    〈DETAIL〉
      〈LOC code="A01/〉YOKOHAMA〈/LOC〉
    〈/DETAIL〉
  〈/PERSON〉
    〈NUM〉002〈/NUM〉
    〈NAME〉SILVER〈/NAME〉
    〈JOB〉BISHOP〈/JBO〉
    〈LOCNUM〉A02〈/LOCNUM〉
    〈DETAIL〉
      〈LOC code="A02"〉NEWYORK〈/LOC〉
    〈/DETAIL〉
  〈/PERSON〉
〈LIST〉
```

文档2

(1)处理没有重复结构的 XML。对以上的文档1,要插入下列表:

PERSON

A	B	C	D
001	GOLD	KING	A01

LOCATION

E	F
A01	TOKYO

此时需要的数据对应关系如下:

Column	XPath of the Value
PERSON. A	/PERSON/NUM
PERSON. B	/PERSON/NAME
PERSON. C	/PERSON/JOB
PERSON. D	/PERSON/DETAIL/LOCINFO/LOCNUM
LOCATION. E	/PERSON/DETAIL/LOCINFO/LOCNUM
LOCATION. F	/PERSON/DETAIL/LOCINFO /LOC

根据3.2.2中描述的规则各个域的功能,可以通过下述的规则集来表达 XML 与数据库之间的关系:

```
〈targetRule id="t1" table="PERSON" column="A" path="/
  PERSON/NUM"/〉
〈targetRule id="t2" table="PERSON" column="B" path="/
  PERSON/NAME"/〉
〈targetRule id="t3" table="PERSON" column="C" path="/
  PERSON/JOB"/〉
〈targetRule id="t4" table="PERSON" column="E" path="/
  PERSON/DETAIL/LOCINFO/LOCNUM"/〉
〈targetRule id="t5" table="LOCATION" column="E" path="/
  PERSON/DETAIL/LOCINFO/LOCNUM"/〉
〈targetRule id="t6" table="LOCATION" column="F" path="/
  PERSON/DETAIL/LOCINFO/LOC"/〉
```

(2)处理有重复结构的 XML。对以上的文档2,要插入下列表:

PERSON

A	B	C	D
001	GOLD	KING	A01
002	SILVER	BISHOP	A02

LOCATION

E	F
A01	TOKYO
A02	NEWYORK
A01	YOKOHAMA

从文档2中可以看出,此 XML 文档中包含2处重复结构,即/LIST/PERSON 和/LIST/PERSON/DETAIL,因此,规则集中需要用到 repetitionRule。其需要的数据对应关系与上例相似,这里不重复列出。需要说明的是,实现 XML 元素的一个属性和列之间的对应关系时,XPath 中是使用"@属性名"表示的。以下即满足要求的规则集:

```
〈targetRule id="t1" table="PERSON" column="A" path="/
  LIST/PERSON/NUM"/〉
〈targetRule id="t2" table="PERSON" column="B" path="/
  LIST/PERSON/NAME"/〉
〈targetRule id="t3" table="PERSON" column="C" path="/
  LIST/PERSON/JOB"/〉
〈targetRule id="t4" table="PERSON" column="D" path="/
  LIST/PERSON/LOCNUM"/〉
〈targetRule id="t5" table="LOCATION" column="E" path="/
  LIST/PERSON/DETAIL/LOC@code"/〉
〈targetRule id="t6" table="LOCATION" column="F" path="/
  LIST/PERSON/DETAIL/LOC"/〉
〈repetitionRule id="s1" table="PERSON" path="/PERSON"/〉
〈repetitionRule id="s2" table="LOCATION" path="/PERSON/
  DETAIL/LOCINFO"/〉
```

3.3 规则集的可扩展性

文[3,4]中详细介绍了 XMLGenerator 模块的映射规则集,并介绍了规则集的初步扩展,用于解决二进制数据、空值、字符集等问题的功能。同样,XMLSaver 中也可以通过在规则中添加特定的域实现一些扩展的功能,比如可以通过在 targetRule 中添加一个域将一个元素中的所有内容存到数据库的一列中。不管是 XMLGenerator 还是 XMLSaver,用户都可以根据自己的特定需求在规则集中定义新的功能域,当然,这都需要建立在合理设计的基础之上,新的功能域定义之后,还需要添加特定的处理模块。总之,映射规则集具有良好的可扩展性。

4 与 XSU 的性能比较

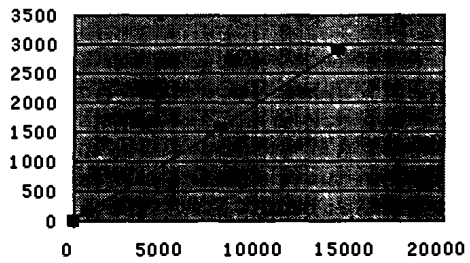

图5 关系数据库→flat XML

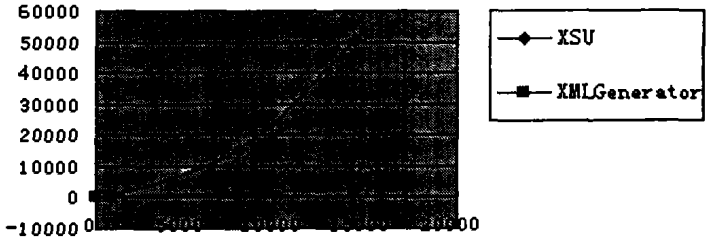

图6 关系数据库→复杂结构的 XML

本文所提出的处理方法的优点是没有采用书写复杂的 XSLT 转换 XML 的结构,而是采用了以上介绍的映射规则,从以上2个例子可以看出,规则集的编写确实比 XSLT 的编写要简单得多,而整体性能的提高主要体现在映射规则的处理效率上。为了验证性能的提高,我们用 JAVA 语言编程实现了以上描述的处理流程,并通过大量的实验数据,与 Oracle 的 XSU(版本号9.2.0.5.0)进行了详细的比较。以下是不同

情况下 XMLGenerator 和 XMLSaver 与 XSU 的性能比较数据，测试平台为 Windows2000，JDK 版本为1.4.1，测试机配置为 P41.6G，内存256M，数据库为 Oracle 8i。

XSU 和 XMLGenerator 的性能比较

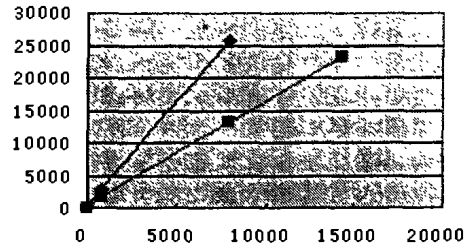

图7 flat XML→关系数据库

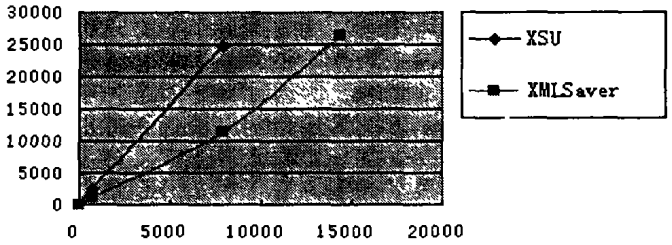

图8 复杂结构的 XML→关系数据库

以上图5～图8的横轴为记录的个数，纵轴为运行时间(ms)，这个数据是多次测试的平均值。从图中可以看出：

(1) 从关系模型向 XML 模型转换时，如果只是简单的 flat XML，不涉及到复杂的映射机制，这种情况下二者的性能相当；如果生成的 XML 中有重复结构，XMLGenerator 的处理时间并没有大幅的提高，而 XSU 中由于 XSLT 处理复杂结构效率的低下，使得整体效率与 XMLGenerator 相比显得差距很大。

(2)从 XML 模型向关系模型转换时，对于 flat XML 和复杂结构的 XML，XMLSaver 的性能优势基本相同，因为不清楚 XSU 内部的处理机制，所以这里不能从底层处理机制的角度进行解释，但是从测试结果看，XMLSaver 的性能优势还是很明显的。需要说明的是，XSU 在处理数据量大的时候，在图中反映为数据记录为10000多行时，就发生了溢出错误，而 XMLSaver 表现得更为稳定。

为了保证测试结果的正确性及通用性，我们使用了不同的数据库，例如 MySQL、Access、PostGreSQL，并且在 Solaris 及 Linux 等平台下做了同样的测试，都得到了相同的结果。

结语 本文介绍了一种 XML 与关系数据库信息交换的方法，利用自身定义的映射规则集和映射方法，实现关系模型和 XML 之间双向的转换。与目前通用的方法相比，本方法的优点是没有采用书写复杂的 XSLT 转换 XML 的结构，而是采用了自身定义的映射规则，比 XSLT 的编写要简单得多；并且将从数据库生成 flat XML 以及从 flat XML 到目标 XML 的转换过程合并为一个过程，因此简化了处理流程，从而达到提高性能的目的。整个系统的核心是映射规则集的设计，对于特别复杂的结构，映射机制仍然不是很完善，今后工作的重点仍然在映射机制的研究和规则的扩展等方面，设计出一个完善的转换系统。

参 考 文 献

1 Zhang Xin, et al. The XQuery! --- An XML Algebra Optimization. Approach Workshop on Web Information and Data Management. ACM Press New York, NY, USA, Nov. 2002

2 XSL Transformations (XSLT) Version 1.0. W3C Recommendation, 16 November, 1999. http://www.w3.org/TR/xslt

3 蔡飞，等. 基于关系数据库的 XQuery 查询的实现. 计算机科学，2004(5)

4 陶列骏，等. XML 与数据库的存取技术：[硕士学位论文]. 南京：南京大学计算机系，2003

5 XML Path Language (XPath) Version 1.0. W3C Recommendation. 16 November 1999. http://www.w3.org/TR/xpath

(上接第60页)

一个完全的解决方法，作为编程人员应该清晰地认识到缓冲区溢出攻击的普遍性和危害性，增强安全意识，养成安全编程的思想，对软件进行全面、充分的测试，写出高质量的软件，切除缓冲区溢出的根源。

参 考 文 献

1 http://www.cert.org/stats/cert_stats.html

2 Wagner D, Foster J S, Brewer E A, Aiken A. A first step towards automated detection of buffer overrun vulnerabilities. In: Proc. of Network and Distributed System Security Symposium, Catamaran Resort Hotel, San Diego, California, Feb. 2000. 3～17

3 Spaford E. The Interner Worm Program : Analysis Computer Communication Review, Jan. 1989. 3～17

4 DilDog. The tao of Windows buffer overflow. http: //www.cultdeadcow.com/cDc_files/cDc-351/, April 1998

5 Conover C, w00w00 Security Team. w00w00 on heap overflows. http://www.w00w00.org/files/articles/heaptut.txt, Jan. 1999

6 Cowan C, et al. StackGuard: Automatic adaptive detection and prevention of buffer overflow attacks. In : Proc. of the 7th USENIX Security Conf. San Antonio, Texas, Jan. 1998. 63～78

7 Vendicator. Stack Shield technical info file v0.7 http://www.angelfire.com/sk/stackshiedl/, Jan. 2001

8 Baratloo A, Singh N, Tsai T. Libsafe: Protecting critical elements of stacks. White paper http://www.research.avayalabs.com/project/libsafe/, Dec. 1999

9 Fetzer C, Xiao Z. Detecting heap smashing attacks through fault containment wrappers. In: the Proc. of the 20th symposium on Reliable Distributed Systems, Oct. 2001

10 Viega J, Bloch J T, et al. ITS4: A static vlnerability scanner for C and C++ code. In: Proc. of the 16th Annual Computer Security Applications Conf. Dec. 2000

11 Larochelle D, Evans D. Statically detecting likely buffer overflow vulnerabilities. In: Proc. of the 2001 USENIX Security Symposium, Washington DC, USA, Aug. 2001

12 Cole E 著，苏雷，等译. 黑客一攻击透析与防范. 北京：电子工业出版社，2002

13 Axelsson S. A comparison of the security of windows NT and UNIX, 1998. http://www.securityfocus.com/data/library/nt-vs-unix.pdf

14 Levy E. Smashing the stack for fun and profit. Phrack Magazine, 1996, 49(14)

15 Dik C. Non_executable stack for solaris, posted to comp.secuity.unix Jan. 1997. http://x10.dejanews.ocm/

16 the Linux OpenWall Project. Nonexecutable stack patch for Linux. http://www.openwall.com/linux

17 Wojtczuk R. Defeating solar designer's nonexecutable stack patch. in Bugtraq, Jan. 1998

18 Dean D, Felten E W, Wallach D S. Java Security: from hotjava to netscape and beyond. In: Proc. of the IEEE Symposium on Security and Privacy, Oakland, CA, 1996

19 Kolishak A, BOWall. buffer overflow protection for Windows NT, 2000. http://developer.nizhny.ru/bo/eng/BOWall