第34卷 第18期 计 算 机 工 程 2008年9月
Vol.34 No.18 Computer Engineering September 2008
·软件技术与数据库· 文章编号：1000—3428(2008)18—0072—03 文献标识码：A 中图分类号：TP311.12

基于SAX的XML数据结构聚簇存储方法

杨 治[1]，鞠时光[2]
(1. 江苏大学工商管理学院，镇江 212013；2. 江苏大学计算机科学与通信工程学院，镇江 212013)

摘 要：目前在存储XML数据时没有考虑数据之间的结构关系，但对其进行操作时往往需要涉及这些关系。该文在分析XML数据操作的特点后，提出基于SAX的编码解析算法，利用该算法遍历XML文档一次即可得到XML数据的<start,end,level>三元组编码，以该编码为基础，提出2种XML数据的结构聚簇存储算法，通过实验对算法进行了分析和比较。

关键词：XML数据；编码；结构聚簇

Structural Clustering Storage Method of XML Data Based on SAX

YANG Zhi[1], JU Shi-guang[2]
(1. School of Business Administration, Jiangsu University, Zhenjiang 212013;
2. School of Computer Science and Telecommunication Engineering, Jiangsu University, Zhenjiang 212013)

【Abstract】 The structure relations between XML data are always regardless while the data are stored, but they are needed during operation. After analyzing the operation characteristics of XML data, this paper designs the coding and parsing arithmetic based on SAX. It travels XML document only once to get a triad of <start, end, level> for coding XML data by using this arithmetic. An XML data structural clustering arithmetic based on the arithmetic is proposed. Analysis and comparion are done according to experiment.

【Key words】 XML data; coding; structural clustering

1 概述

XML是Web上信息表示和数据交换的事实标准，由于具有开放性、灵活性、易读性和平台无关性等特点，因此应用越来越广泛。而如何有效地存储XML数据成为目前的研究热点。XML存储的方法[1]主要有：文件存储方式，支持XML的传统数据库(XEDB)以及原生的XML数据库(NXD)。其中，NXD存储方式是以专有的格式将XML数据存储在原生XML数据库中，它以自然的方式处理XML数据，针对XML的数据存储和查询特点，专门设计适用的数据模型和处理方法。其内部模型基于XML，保持了物理存储模型与逻辑模型的一致。但是目前的NXD存储方法没有考虑根据XML数据的半结构化特点对其进行存储。

文献[2]讨论了基于DOM的结构聚簇存储，但是没有考虑应用更广的SAX模型，也没有考虑如何对XML数据进行编码，而编码对提高XML数据的查询效率很重要。本文则提出了一种基于SAX的XML数据结构聚簇的存储策略。

2 基于SAX的XML数据存储

为了存储XML数据，首先需要对如下XML文档进行解析，得到XML数据(即节点)：

```
<books>
  <book>
<tile>designing XML Databases</tile>
<price>49</pbice>
  </book>
  <book>
<title>XML and Database</title>
<price>25<price>
<book>
</books>
```

目前的解析模式主要有DOM[3]和SAX。DOM是基于对象的模型，通过在内存中建立一棵文档对象树，与应用程序进行交互，其中包含了XML文档的所有元素，即这个对象树是对XML文档中元素树的精确映射。而SAX是基于事件的模型，它的基本原理是由接口的使用者提供符合定义的处理器，如果分析XML时遇到特定的事件，就调用处理器中特定事件的处理函数。与DOM相比，SAX更适用于处理大数据量，特别是有特定数据结构或以数据库形式存储XML数据的情况。因此，本文讨论的存储方式是基于SAX模型的。

SAX的核心基于2个接口：XMLReader接口表示分析器，ContentHander接口从分析器接收数据。XMLReader读取文档时，调用ContentHander中的方法：XMLReader读取开始标志时，调用startElement()方法；读取文本内容时，调用characters()方法；读取结束标志时，调用endElement()方法。

对于上文所示的XML文档，SAX解析生成的事件流如图1所示。

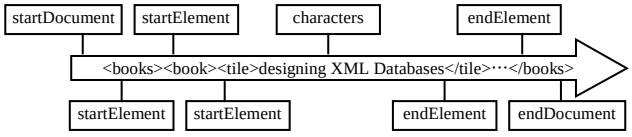

图1 SAX解析生成事件流

存储经由SAX解析得到的XML数据时，一般利用SAX解析器的处理特征，随startElement()事件流生成相应的XML数据节点，随startElement()事件把该节点写到物理页面上。这

作者简介：杨 治(1977－)，男，讲师，主研方向：数据库技术；鞠时光，教授、博士生导师

收稿日期：2007-10-10 **E-mail：**yangzhi@ujs.edu.cn

种方法虽然简单，但没有考虑XML数据之间的父子或兄弟结构关系。由于XML数据本身的特点，在对其进行操作(如遍历)时往往需要涉及这些数据之间的结构关系[4]。如XPATH路径表达式：/book/title，就需要访问book和title节点，而book和title是具有父子关系的。如果将book和title存储在一起(如同一页面)就能减少I/O，提高操作效率。所以，为了提高操作效率，需要在存储XML数据时考虑它们之间的结构关系。

3 XML 数据结构聚簇存储算法

根据前面的分析可以知道，如果将具有父子关系的数据尽量存储在同一物理页面中，能减少 I/O 频率，提高相应的父子操作效率。同理，尽量将具有兄弟关系的数据存储在同一物理页面中进行兄弟结构聚簇存储，可以提高相应的兄弟操作效率。这就是 XML 数据结构聚簇存储的基本思想。本文利用 XML 数据结构聚簇存储算法实现该思想。

下面先介绍 XML 解析算法，该算法在解析的同时实现了对这些节点的编码。通过该算法只须对 XML 文档遍历一次即可得到解析后经过编码的 XML 数据。

3.1 解析及编码算法

利用SAX可以将XML文档解析为元素、文本等节点所表示的XML数据[5]。为了支持XML数据的结构查询[6]，本文对这些节点采用三元组<start, end, level>编码，其中，Start是对XML文档进行深度遍历，进入该节点时得到的编号；end是访问完该节点全部后裔后，结束时得到的编号；level是该节点的层次编号。每个解析后得到的节点都由该三元组和XML数据组成。此结构能同时表示节点的结构信息和值信息，支持XPATH的全部轴，有效实现对XML数据的结构查询，同时支持对节点的值查询[7]。另外，这些编码也是进行聚簇存储的基础。

在 SAX 的解析过程中需要为相应的事件设计处理方法，以下是对 XML 文档的解析和编码算法。限于篇幅，本算法仅给出常见的元素节点以及文本节点相关的事件。

为了处理 XML 数据及其三元组编码，首先定义一个 Node 类:

```
class Node{
  string Value;
  int Start,end,level;}
```

算法 1 解析及编码(PC)算法

输入 XML 文档。

输出 经过解析和编码的节点数组 NodeArray，该数组为 Node 类型。

```
start=end=0;
开始利用 SAX 对 XML 文档进行解析
If 触发元素开始事件(startElemnet())
  New node;
  Node.value=元素值;
        Node.start=++start;
        Node.level=++level;
        Node 压栈;
Endif
If 触发元素结束事件(endElement())
  将栈顶 Node 出栈;
  Node.end=Node.start+1;
  NodeArray[start]=Node;
Endif
If 触发文本事件(characters())
  New node;
  Node.value=文本值;
Node.start=++start;
  Nnode.end=Node.start;
        Node.level=++level;
Endif
解析结束
```

利用 PC 算法能将 XML 文档转换为三元组编码的 XML 数据节点。为了实现对这些 XML 数据结构的聚簇存储，需要考虑如何将这些节点根据父子或兄弟结构关系聚簇并且存储到物理页面中。

3.2 结构聚簇存储算法

下面介绍将 NodeArray 数组中存储的 XML 数据根据父子或兄弟结构关系组织在物理页面上。

算法 2 父子结构聚簇存储(FC)算法

输入 SAX 解析和编码得到的节点数组。

输出 父子结构聚簇存储的页面。

```
将 NodeArrqy 数组按照 start 顺序排序;
i=1;
  While  NOT(读取到 NodeArray 数组的最后一个元素)
读取 NodeArray[i];
IF (当前页面有空间可以存放)
  将读取的元素存储在该页面;
Else
  新申请一个页面;
  元素存储在该新页面上;
Endif
i++;
  Endwhile
```

FC 算法实际就是根据 start 顺序依次读取数组中的元素并将其存储在物理页面上。由于 start 顺序代表的是 XML 数据在 XML 文档中出现的先后顺序，因此 start 顺序较好地表示了数据之间的父子结构关系，即 FC 算法实现了 XML 数据的父子结构聚簇存储。

由于通过 PC 算法得到的 NodeArry 数组就是按 start 顺序排列的，因此 FC 算法中的第一句排序过程可以省略。

算法 3 兄弟结构聚簇存储(SC)算法

输入 SAX 解析和编码得到的节点数组。

输出 兄弟结构聚簇存储的页面。

```
将 NodeArrqy 数组按照 level 顺序排序;
i=1;
While NOT(读取到 NodeArray 数组的最后一个元素)
读取 NodeArray[i];
IF (当前页面有空间可以存放)
  将读取的元素存储在该页面;
Else    新申请一个页面;
        元素存储在该新页面上;
Endif
i++;
Endwhile
```

SC 算法首先对 NodeArray 数组按照 level 顺序进行排序，再将 XML 数据依此顺序依次存储在物理页面上。level 表示的是 XML 数据在文档中的层次位置关系。因为兄弟节点必定在同一层次上，所以 level 顺序表示了数据之间的兄弟结构关系，即 SC 算法实现了 XML 数据的兄弟结构聚簇存储。

4 实验分析

测试使用的平台为：Pentium4 1.7 GHz，256 MB内存，Windows 2000 Professional操作系统。用Java分别实现上述

2 种存储策略，性能评价测试标准采用XMark[8]，对其中涉及大量结构关系操作的查询Q13~Q16 进行实验。实验结果如图 2 所示，其中，FC表示父子结构聚簇方法；SC表示兄弟结构聚簇方法；Q13 和Q14 主要利用广度优先遍历方法进行查询，需要大量涉及数据之间的兄弟关系；Q15 和Q16 利用广度优先遍历进行查询，需要用到数据之间的父子关系。

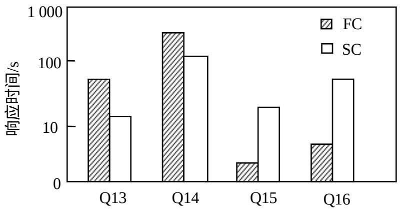

图 2　实验结果

从图 3 中可以看出，对于 Q13 和 Q14，SC 比 FC 的性能好，对于 Q15 和 Q16，FC 比 SC 的性能好，因此得出如下结论：父子结构聚簇(FC)算法由于尽量保持了节点之间的父子关系，能提高涉及父子关系操作的效率。而兄弟结构聚簇(SC)算法则考虑了节点之间的兄弟关系，有效地提高涉及兄弟关系的操作效率。因此，使用何种结构聚簇存储方法应该根据操作的特点进行选择。

5　结束语

本文针对 XML 数据的特点，提出了一种基于 SAX 的 XML 数据结构聚簇存储方法，并以此为基础设计了 2 种 XML 数据的结构聚簇存储方法：基于父子关系的 FC 存储算法和基于兄弟关系的 SC 算法。通过实验可知，FC 方法适合于涉及父子关系的操作，而 SC 方法更适合涉及兄弟关系的操作。

参考文献

[1] Bourret R. XML and Databases[EB/OL]. (2003-11-20). http://www.rpbourret.com/xml/XMLAndDatabases.htm.

[2] 乔百友, 王国仁, 韩东红, 等. XML 数据聚簇技术研究[J]. 东北大学学报: 自然科学版, 2005, 26(6): 538-541.

[3] Robie J, Hors A. Document Object Model Level 2[EB/OL]. (2000-03-14). http://www.w3.org/TR/2000/REC-DOM2.

[4] Paparizos S, Al-Khalifa S, Jagadish H V, et al. Grouping in XML[C]//Proc. of XMLDM’02. Berlin, Germany: Springer, 2002.

[5] Jagadish H V, Al-Khalifa S, Chapman A, et al. TIMBER: A Native XML Database[J]. The VLDB Journal, 2002, 11(4): 274-291.

[6] Mchugh J, Widom J. Query Optimization for XML[C]//Proceedings of the 25th VLDB Conference. Edinburgh, Scotland: [s. n.], 1999.

[7] Fiebig T, Helmer S, Kanne C C, et al. Anatomy of A Native XML-based Management System[J]. The VLDB Journal, 2002, 11(4): 292-314.

[8] Schmidt A R, Waas F, Kersten M L, et al. XMark: A Benchmark for XML Data Management[C]//Proc. of the 28th VLDB Conference. Hong Kong, China: [s. n.], 2002.

~~~~~~

(上接第 71 页)

IRP_MJ_QUERY_ INFORMATION 请求时，检测到运行的 A 进程被 Rootkit 保护。

若检测系统在 Hacker Defender 之前启动，可以检测出注册表键的隐藏情况。具体检测对象如下：

HKLM\SYSTEM\ControlSet001\Control\SafeBoot\Minimal\HackerDefender100

HKLM\SYSTEM\ControlSet001\Control\SafeBoot\Network\HackerDefender100

HKLM\SYSTEM\ControlSet001\Services\HackerDefender100\ImagePath

3.3　检测效果分析

该检测技术对于使用钩挂技术的 Rootkit 可以取得很好的效果，如：Urbin, Mersting, Aphex 等。对于使用底层技术的 Rootkit(比如过滤驱动、文件系统驱动等)，检测的效果取决于：Rootkit 的具体实现，监控驱动与 Rootkit 驱动在系统堆栈中的相对位置。若监控驱动相对更为底层，则能获得准确的请求信息，检测会成功；反之，如果 Rootkit 驱动没有对请求进行重定向而只修改返回状态，检测仍会成功，例如使用文件系统内联钩挂技术的 CNNIC 中文上网(它集成了 Rootkit 功能)。

该文检测的行为是“隐藏的资源被访问”，包括隐藏的进程被执行、隐藏的文件被访问等。该驱动可以监控到恶意程序对资源的访问，但验证程序不能访问该资源，这样的矛盾能够准确证明系统内有 Rootkit 存在。由于依靠行为检测 Rootkit 而非特征码，所以该文对未知 Rootkit 也有很好的检测效果。

Rootkit 的隐藏行为区别于一般软件，因此，这种检测方法误报率很低。在实际测试中，只会偶尔误报一些动态释放或删除模块的程序。

4　结束语

Windows Rootkit 出现较晚，现在正是其快速发展阶段。该文应用 cross-view 方法监控系统中的隐藏行为，与现有检测技术相比，能够及时、有效地判断系统是否被装入 Rootkit，并能检测出大部分的内核级 Rootkit。由于针对行为检测而非依赖于特征码，本方法对未知 Rootkit 也有较好的效果。

参考文献

[1] 双世勇. Windows Rootkit 检测方法研究[D]. 郑州: 中国人民解放军信息工程大学, 2005.

[2] 郭　艳. Windows 2000 下 WDM 驱动程序的研究与开发[J]. 计算机工程, 2006, 32(22): 266-268

[3] Joanna R. Thoughts About Cross-view Based Rootkit Detection[EB/OL]. (2005-06-02). http://www.invisiblethings.org/papers/crossview_detection_thoughts.pdf.

[4] Wang Y M, Beck D. Detecting Stealth Software with Strider GhostBuster[C]//Proc. of International Conference on Dependable Systems and Networks. Yokohama, Japan: [s. n.], 2005.

[5] James B. Rootkits: Subverting the Windows Kernel[M]. [S. l.]: Addison-Wesley Professional Press, 2005.
~~~~~~