第 35 卷 第 8 期 计 算 机 工 程 2009 年 4 月
Vol.35 No.8 Computer Engineering April 2009
·软件技术与数据库· 文章编号：1000—3428(2009)08—0107—03 文献标识码：A 中图分类号：TP311

XML 数据流中的后兄弟查询算法

汪万根
(江西师范大学数学与信息科学学院，南昌 330022)

摘 要：针对在 XML 文档树模型中进行后兄弟节点查询时内存消耗大、匹配效率低等缺陷，提出一种基于 XML 数据流与栈的后兄弟查询算法。采用 SAX 解析器与结构连接方法，对 XML 文档中所有已知节点与后兄弟节点进行精确匹配并输出。结果表明，该算法具有适用范围广、占用系统资源少、匹配效率高等优势。

关键词：SAX 解析器；XML 数据流；后兄弟

Algorithm for Querying Following-sibling on XML Data Stream

WANG Wan-gen
(School of Mathematics & Information Science, Jiangxi Normal University, Nanchang 330022)

【Abstract】There are some disadvantages, consuming a large amount of memories, matching with low efficiency, to query following-sibling nodes in XML document tree model. This paper provides an algorithm based on XML data stream and stack for querying following-sibling. The algorithm uses Simple API for XML(SAX) parser and structural join, matches and outputs all following-sibling nodes of the given nodes. Results show that the algorithm has the advantage of wide applicability, low system resource occupancy and high efficiency matching.

【Key words】Simple API for XML(SAX) parser; XML data stream; following-sibling

1 概述

作为 Web 数据交换与发布标准的 XML，具有形式与内容分离、嵌套、自描述、自相容、良好扩展等特点，已得到广泛应用。因此，在大型 XML 文档中对已知元素节点的祖先节点、后裔节点、兄弟节点等的高效查询，成为数据库领域研究的热点。现在已知的兄弟关系匹配算法[1-3]都必须用解析器对 XML 源文档进行解析，并将它们分裂为元素列表进行存储，然后基于有序的元素列表进行结构连接以实现兄弟关系匹配。

SAX(Simple API for XML)解析器提供了一种顺序访问 XML 文档的模式，读写数据快速，所有节点只须按前序遍历 1 次。SAX 每次总是处理单个节点，因此，须将必要的数据有效地缓存，以便返回结果。利用上述特点，本文提出基于 XML 数据流与栈的后兄弟关系匹配算法——SSFSQ。

2 XML 树模型样例

当前数据库领域研究的重要问题之一是在 XML 文档的树模型中进行兄弟关系匹配。基本的 XPath 语法类似于在一个文件系统中定位文件，如果路径以斜线/开始，那么该路径就表示到一个元素的绝对路径。如果路径以双斜线//开始，则表示选择文档中满足双斜线//之后规则的所有元素(无论层级关系)。XML 树模型样例如图 1 所示。

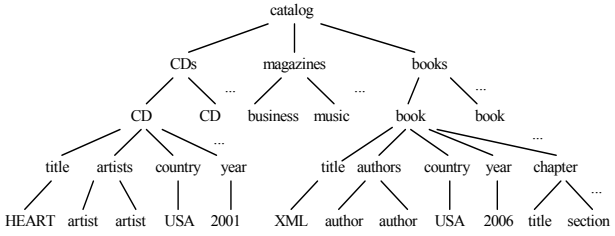

图 1 XML 树模型样例

在图 1 中进行以下查询：

(1)//catalog//CDs/following-sibling::*；

(2)//book/chapter/preceding-sibling::year；

(3)//CD//title/following-sibling:: country。

在(1)中可找到与//catalog//CDs 节点同层次的 magazines, books 等后兄弟节点。在(2)中可找到满足条件的前兄弟节点 year。

3 基于 XML 数据流与栈的后兄弟查询算法

3.1 XML 文档处理及相关符号

XML 文档通常以树形结构的方式表示数据，其中的节点代表元素或元素的属性，兄弟节点关系表达了元素之间的层次平行结构。在 XML 数据流中查询兄弟节点时，逻辑上是以前序遍历访问文档树中的节点，且先访问前兄弟节点，接着访问后兄弟节点。因此，与 XML 文档相应的 XML 树模型的数据流可表示为[4]连续且有序的节点 $x_1, x_2,\cdots, x_n$ 的序列。记作 $ST=(x_1, x_2,\cdots, x_n)$，其中，$x_i \in T$, $i=1, 2,\cdots, n$。

相关符号说明：*Type*(*x*), *Name*(*x*), *Level*(*x*), *Order*(*x*)分别表示 XML 树中节点 *x* 的类型、名称、层次及序号。若节点 x_1 为节点 x_2 的后兄弟，当且仅当：$Level(x_1)=Level(x_2)$, $Order(x_1)>Order(x_2)$且 x_1 与 x_2 有相同的双亲节点。*element* 表示元素节点类型；*GN*, *QN*, *PN* 分别表示查询中已知节点名称、待匹配的后兄弟节点名称、已知节点与其后兄弟节点的双亲节点名称；*PthNL* 表示查询中已知节点与其后兄弟节点的路径节点名称列表；*GS*, *QS*, *PS*, *PthS* 分别表示查询中已知节点栈、待匹配的后兄弟节点栈、兄弟节点的双亲节点栈、

基金项目：江西师范大学青年成长基金资助项目([2007]1982)

作者简介：汪万根(1974－)，男，讲师、硕士，主研方向：Web 数据管理，系统分析与设计

收稿日期：2008-09-04 **E-mail：**wangwangen@126.com

路径节点栈；*push*(*M*)表示将元素节点 *M* 的相关信息都压入栈中，即压入栈中的元素包含节点类型、名称、层次、序号等信息；*pop*()表示将栈顶元素的相关信息出栈。

对于XML树模型样例中的(3)，*GN*为title；*QN*为country；*PN*为任何元素节点；*PthNL*为CD；*GS*, *QS*, *PthS*分别存储名称为title, country, CD的元素节点信息；*PS*存储任何元素节点信息。

3.2 SSFSQ算法步骤

SSFSQ算法对XML树进行1次前序遍历，运用SAX解析以产生数据流，且在访问节点时确定其类型、名称、层次及序号。将之与栈顶元素或列表的最后元素进行比较，找出所有满足查询条件的已知节点与其后兄弟节点，分别压入各栈中。根据已知节点与后兄弟节点的层次关系和序号关系，匹配并同时输出已知节点-后兄弟节点对。

假设*ST*表示按前序遍历XML文档的所有节点的顺序排列，SSFSQ算法的基本步骤如下：

(1)初始化堆栈*GS*, *QS*, *PS*, *PthS*，以存储已知节点、后兄弟节点、双亲节点、路径节点及相关信息。

(2)用SAX解析数据流*ST*中第1个节点M_0，确定其类型、名称、层次、序号等信息。

(3)若数据流*ST*非空，则重复以下步骤：

1)若节点M_0的层次(即*Level*(*M*))小于栈*GS*, *PS*, *PthS*中任何一个栈顶元素的层次，则执行：调用过程*PrintFSib*，输出栈*QS*, *GS*中层次大于*Level*(*M*)且匹配的所有兄弟节点对；调用过程*PopSTop*，分别弹出栈*PthS*, *PS*, *QS*, *GS*中不满足条件的所有节点；

2)若节点M_0的类型(即*Type*(*M*))为*element*，则执行3)；否则，执行7)；

3)若*PthNL*非空，则调用*PthSProc*，将满足条件的*M*压入栈*PthS*中；

4)调用过程*PSProc*，将满足条件的*M*压入栈*PS*中；

5)调用过程*GSProc*，将满足条件的*M*压入栈*GS*中；

6)调用过程*QSProc*，将满足条件的*M*压入栈*QS*中；

7)将M_0取为数据流*ST*中的下一节点，并解析；返回3)。

在算法SSFSQ中，调用过程*PrintFSib*与*PopSTop*的顺序不能颠倒，否则会遗漏输出已经在栈*QS*, *GS*中匹配的兄弟节点对；为提高算法效率，调用过程*PSProc*, *GSProc*, *QSProc*的顺序最好不要颠倒，以减少不必要的出入栈操作。

SSFSQ算法代码如下：

```
Algorithm SSFSQ (ST, GN, QN, PN, PthNL)
  initStack(GS, QS, PS, PthS);
  M0 = ST→FirstNode;
  M = Parse(M0);
  While (ST != NULL) {
    If (Level(M) < Level(GS→top) or Level(M) < Level(PS→top)
        or Level(M) < Level(PthS→top)){
      PrintFSib (GS, QS, M);
      PopSTop (PthS, PS, GS, QS, M);
    }
    If (Type(M) == 'element') {
      If (PthNL != NULL){
        PthSProc (PthS, M); }
      PSProc (PS, PthS, PN, M);
      GSProc (GS, PS, GN, M);
      QSProc (GS, PS, QS, QN, M);
    }
    M0 = ST→NextNode;
    M = Parse(M0);
  }
```

*PrintFSib*过程代码如下：

```
Procedure PrintFSib (GS, QS, M)
  If (QS != NULL){
    Sqn = QS→top;
    While (Level(Sqn)>Level(M)) {
      Sgn = GS→top;
      While (Level(Sgn)>Level(M)) {
        If (Level(Sgn) == Level(Sqn) and Order(Sgn) < Order
(Sqn)){
          OutPut(Sgn, Sqn); }
        Sgn = GS→NextNode;   /*指针从栈顶向下移并指向下
一元素*/
      }
      Sqn = QS→NextNode;   /*指针从栈顶向下移并指向下一
元素*/
    }
  }
```

在解析*M*时，及时输出栈*GS*与*QS*中匹配的且层次大于*Level*(*M*)的部分(并非所有)兄弟节点对。具体步骤为：若*QS*非空，则获取*QS*, *GS*栈顶元素并分别赋予*Sqn*, *Sgn*，然后比较*M*与*Sqn*, *Sgn*的层次，以判别是否进入相应的循环；在内外层循环中，每一次循环结束后都须将*QS*, *GS*的指针从栈顶向下移以获取下一元素；进入内层循环后，对*Sqn*, *Sgn*的层次与序号分别进行比较，若两者满足兄弟关系，则调用过程*OutPut*(*Sgn*, *Sqn*)将兄弟节点对输出。

*PopSTop*过程代码如下：

```
Procedure PopSTop (PthS, PS, GS, QS, M)
  While (PthS != NULL or Level(M) <= Level(PthS→top)){
    PthS→pop(); }
  While (PS != NULL or Level(M) <= Level(PS→top)){
    PS→pop(); }
  While (QS != NULL or Level(M) < Level(QS→top)){
    QS→pop(); }
  While (GS != NULL or Level(M) < Level(GS→top)){
    GS→pop(); }
```

根据解析*M*的情况，在栈*PthS*, *PS*, *GS*, *QS*分别不为空时，通过将*M*的层次分别与各栈顶元素的层次进行比较，弹出各栈中不满足条件的所有元素。解析到流*ST*中节点M_0时，如果M_0的层次，即*Level*(*M*)不大于栈*PthS*, *PS*的栈顶元素层次，那么即使这2个栈顶的元素分别是查询表达式中描述的路径节点、双亲节点，因为已不可能满足M_0及其后节点的后兄弟匹配条件，所以出栈。对于栈*GS*, *QS*，由于*PrintFSib*已将栈中层次大于*Level*(*M*)且匹配的所有兄弟节点对输出，因此将这些元素出栈。

过程*PthSProc*则将可能的路径节点添加到*PthS*中。解析到新节点M_0时：如果*Name*(*M*)与*PthS*→*top*元素所期待的且在*PthNL*列表中的元素名称相同，则将*M*压入栈*PthS*中。

*PSProc*过程代码如下：

```
Procedure PSProc (PS, PthS, PN, M)
  If (PthNL != NULL){
    If (PN != NULL) {
      If (Name(M) == PN and Name(PthS→top) ==
LastName(PthNL)){
```

```
            PS→push(M); }
        }Else{
          If (Name(PthS→top) == LastName(PthNL)) { PS→push(M); }
        }
      }Else{
        If (PN != NULL) {
          If (Name(M) == PN) { PS→push(M); }
        }Else{
          PS→push(M); }
      }
```

PSProc 过程根据 *PthNL* 与 *PN* 将可能的双亲节点压入栈 *PS* 中：

(1)如果栈 *PthNL* 非空，那么：

1)如果 *PN* 非空，则只要 *Name*(*M*)与 *PN* 相同且 *PthS* 栈顶元素与 *PthNL* 列表最后节点名称相同，就将 *M* 压入栈 *PS* 中；

2)如果 *PN* 为空，则只要 *PthS* 栈顶元素与 *PthNL* 列表最后节点名称相同，就将 *M* 压入栈 *PS* 中。

(2)如果栈 *PthNL* 为空，那么：

1)如果 *PN* 非空，则只要 *Name*(*M*)与 *PN* 相同，就将 *M* 压入栈 *PS* 中；

2)如果 *PN* 为空，则将 *M* 压入栈 *PS* 中。

其中，函数 *LastName*(*PthNL*)表示获取 *PthNL* 列表中最后一个节点的名称。

过程 *GSProc* 的作用是将可能的路径节点添加到 *GS* 中：如果栈 *PS* 非空且 *Name*(*M*)与查询中已知节点的名称相同，则将 *M* 压入栈 *GS* 中。注意，由于过程 *PopSTop* 已将栈 *GS* 中满足 *Level*(*M*)<*Level*(*GS*→*top*)的元素出栈，因此解析新节点 M_0 时，压入栈 *GS* 中的元素 *M* 的层次一定不小于原栈顶元素的层次。

过程 *QSProc* 将可能的路径节点添加到 *QS* 中，如果 *QN* 非空，则只要 *Name*(*M*)与 *QN* 相同且 *Level*(*M*)与 *GS* 栈顶元素的层次相同，就将 *M* 压入栈 *QS* 中；如果 *QN* 为空，则只要 *Level*(*M*)与 *GS* 栈顶元素的层次相同且 *Level*(*M*)比 *PS* 栈顶元素的层次多 1，就将 *M* 压入栈 *QS* 中。

3.3 SSFSQ 算法分析

假设 XML 文档具有 *n* 个节点，SSFSQ 算法只需对 XML 数据流扫描 1 遍，遍历所有节点的时间复杂度为 $O(n)$。SAX 解析速度极快且运行良好[5]，因此，解析每个节点的代价可看作常数时间 d_1，则解析所有节点的时间代价为 nd_1。

另外，SSFSQ 算法中必须将符合查询条件的可能的已知节点、后兄弟节点、双亲节点、路径节点入栈与出栈，如果将入栈与出栈操作的代价分别看作常数时间 d_2, d_3，那么最坏情况(所有 *n* 个节点必须在 4 个栈中入栈与出栈)的时间代价不高于 $4\times(nd_2+nd_3)$。由此可知，在一般情况下 SSFSQ 算法的时间代价低于 $nd_1+4\times(nd_2+nd_3)$。

SSFSQ 算法的空间复杂度低的原因如下：

(1)采用 SAX 遍历源 XML 文档时，各栈中只需保存符合条件的已知节点、后兄弟节点与路径节点的信息，且每次访问数据流中的新节点时，极有可能释放栈中部分节点所占资源；

(2)对于不符合查询条件的节点，尽管必须被解析，但其占用的系统资源可立即或经过短暂时间后被释放，对查询影响不大；

(3)源 XML 文档经前序遍历后，全部释放所占资源。实际上，XML 流中任何被扫描过的节点都不再占用资源。若节点满足查询条件，则以其他方式被保存在相应的栈中。

4 结束语

本文提出一种基于 XML 流与栈的已知节点与后兄弟节点匹配算法，使用 SAX 技术解析 XML 文档，运用结构连接方法及时、有效地输出栈中匹配的节点对。算法为在半结构化数据 XML 中左兄弟、堂兄弟节点等节点的查询提供了一种解决思路，是当前节点查询研究的重要内容之一。

参考文献

[1] Wan changxuan, Liu Xiping. Structural Join and Staircase Join Algorithms of Sibling Relationship[J]. Journal of Computer Science and Technology, 2007, 22(2): 5-15.

[2] 王 钊, 耿 蓉, 王国仁. XPath 的轴连接查询技术研究[J]. 小型微型计算机系统, 2005, 26(11): 1942-1947.

[3] 王 钊, 周 博, 孙 冰, 等. RaP: 一种解决 XML 保序查询的编码方法[J]. 计算机科学, 2003, 30(增刊): 75-79.

[4] 李 智, 唐常杰, 栾 江, 等. 基于索引的XML数据流的变化检测[J]. 计算机科学, 2003, 30(10): 49-54.

[5] Katz H. SAX and Document Order[Z]. [2008-06-01]. http://www-900.ibm.com/developerWorks/cn/xml/tips/x-tipsaxdo/ index.shtml.

编辑 顾姣健

(上接第 106 页)

[4] Bernaes L. On-demand KM: A Two-tier Architecture[J]. IEEE IT Professional, 2002, 4(1): 27-33.

[5] Bowman B J. Building Knowledge Management System[J]. Information Systems Management, 2002, 19(3): 32-40.

[6] Lee S M. An Enterprise-wide Knowledge Management System Infrastructure[J]. Industrial Management & Data Systems, 2002, 102(1): 17-25.

编辑 陈 文