第 33 卷 第 11 期 计 算 机 工 程 2007 年 6 月
Vol.33 No.11 Computer Engineering June 2007
·软件技术与数据库· 文章编号：1000—3428(2007)11—0070—03 文献标识码：A 中图分类号：TP311

XPath 中的文本查询研究

王竞原，胡运发，葛家翔
（复旦大学计算机与信息技术系，上海 200433）

摘 要：介绍了一种能够统一地索引全文数据与 XML 树型结构数据的模型——互关联后继树，提出了后继模式树的概念，使用后继模式树有效地处理了 XPath 中的正则查询问题。在后继模式树的基础上提出了 XPath 中节点与文本的联合查询方法。结果表明该方法能够有效地提高 XPath 中节点与文本联合查询的效率。

关键词：XPath；XML 索引；互关联后继树；后继模式树；联合查询

Research on Full-text Search in XPath

WANG Jingyuan, HU Yunfa, GE Jiaxiang
(Department of Computer and Information Technology, Fudan University, Shanghai 200433)

【Abstract】 An indexing model called inner-relative successive trees which can unify the index of full-text data and XML-tree data is presented. A notion named successive pattern tree is proposed to effectively process the regular expression search in XPath. Based on successive pattern tree, a united method to search nodes and text in XPath is introduced. The experiment results indicate that this method improves the efficiency of the united search of nodes and text in XPath.

【Key words】 XPath; XML index; Inner-relative successive trees; Successive pattern tree; United search

随着全球信息化的进一步发展，数据大量产生，XML 为各种异构系统之间的信息交换提供了一种便捷的方式，成为了各个系统之间数据交换格式的新标准。从 XML 中快速地抽取信息成为了各种信息系统的迫切需要，因此对 XML 文档的高效索引与快速查询成为了当今数据库领域的一个研究热点。

目前对 XML 索引和查询的研究主要针对 XML 文档树形结构进行。较有影响的研究成果是对 XML 树型结构的节点进行相对位置的编码，然后用 B+树对编码节点进行存储的方法。比如 XISS 用前序遍历和后序遍历组成的区间来判断两个节点的祖先——后代关系，然后用 B+树对节点进行存储；而整枝连接技术(Holistic Twig Join)则采用文档序与节点深度组成的编码来判断这个关系，然后用 B+树的一种变形 XB 树存储节点。

这种索引模型在 XML 树型结构上的查询效率较高，但是不适合用于 XML 文本节点的字符串，所以它通常将文本节点也作为一个树节点进行编码存储，用模式匹配法对字符串进行查询。这种方法在文本节点较小时整体性能较好，因而被广泛采用，但是在文本节点中的字符串很长时，模式匹配方法的查询效率很低。因此，用全文索引模型来索引 XML 文档的方法被提出，这种方法虽然能够快速地在文本节点的索引中进行查询，但是在无结构全文索引和查询中表示树型结构的语义是比较复杂的，因此其在树型结构上的查询效率通常不高。

1 联合索引的统一模型——互关联区间后继树

定义 1(前驱与后继)：设∑是构成文本的基本符号单元的集合：$\{\alpha_1, \alpha_2, \dots, \alpha_N\}$，$\alpha_i \in \sum(i=1,2,\dots,N)$是∑中的一些基本符号，它们的有序组合便可构成一个文本。对于任意文本T中的任意字符序列a_1a_2，称a_1是a_2的前驱，a_2是a_1的后继。

定义 2(后继树)：设有全文I由一字符串$a_1a_2\dots a_n$#组成，下标i指明a_i在字符串中的位置。若其中$a_{i1}= a_{i2}=\dots= a_{im}$为相同的字符，不妨记为α，$a_{i1+1}, a_{i2+1},\dots, a_{im+1}$分别是其后继，而$a_{i1+1}, a_{i2+1},\dots, a_{im+1}$的后继又分别为$a_{i1+2}, a_{i2+2},\dots, a_{im+2}$，记为$a_{i1+1}[pos_1], a_{i2+1}[pos_2],\dots, a_{im+1}[pos_m]$，其中$a_{i\ j+2}$是$a_{i\ j+1}$的所有后继节点中的第$pos_j$个，那么称$\{(a_{i1+1}, pos_1),(a_{i2+1}, pos_2),\dots,(a_{im+1}, pos_m)\}$为α的后继表达式，可以用一棵后继树来描述此表达式，如图 1 所示。

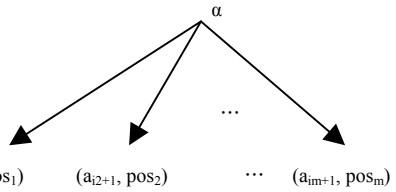

图 1 后继树的结构

定义 3(互关联后继树)：由一个索引库对应的全文 I 的所有后继树组成的森林，叫做 I 的互关联后继树。

因为互关联后继树将全文中各个字符的位置信息映射成了后继以及后继编号信息，并且这个映射是一一对应的，所以由互关联后继树能够准确地生成原文。同时，由于互关联后继树提供了每个字符的查询入口，即树根节点，因此在其上的查询具有很快的速度。

定义 4(互关联区间后继树)：在α的后继$a_{i1+1}, a_{i2+1},\dots, a_{im+1}$中，若$a_{i1+j1}= a_{i1+j2}=\dots= a_{i1+jK}= \beta_j$，即a的后继节点中有相同的，

基金项目：国家自然科学基金资助项目(60473070)
作者简介：王竞原(1982－)，男，硕士生，主研方向：XML 数据库；胡运发，博导；葛家翔，教授
收稿日期：2006-06-20 **E-mail：**042021137@fudan.edu.cn

那么它们可以被合并在一起，后继编号则组成了一个集合，将它们连续地存放在一起偏移量就形成了一个区间，把全文的所有后继树下相同值的节点合并之后并按节点值进行排序，就形成了互关联区间后继树，其后继表达式形式为：{ (β_1, [pos_{j1}, pos_{j2}, …,pos_{jK}]), …,(β_m, [pos_{s1}, pos_{s2}, …,pos_{sM}]) }，其中$\beta_1<\beta_2<\ldots<\beta_m$。利用各个区间的端点值可以加快在后继树上的查询速度。

互关联区间后继树作为一种存储全文数据的树型结构，有其特定的模式，就如同关系数据库有概念模式、外模式和内模式，XML 有 DTD 和 Schema 两种数据模式一样。互关联区间后继树的模式描述了每个后继树的根节点下有哪些字符值，每个字符值有多少个字符数，每个字符值对应的后继编号在文件中的存储范围。一棵区间后继树的模式可以用树型结构来表示。

定义 5(后继模式树)：后继模式树的根节点为一棵区间后继树的根节点，其叶节点集由该后继树的所有叶节点值组成。叶节点按红黑树的形式组织，每个叶节点对应了与其有相同值的所有字符的存储区间的起始位置和当前位置。红黑树是一种平衡的二分查找树，其插入、查找时间复杂度均为 O(log *n*)，其证明见文献[5]。若当前某后继树的根节点值为‘R’，对应的 ASCII 编码为 52 (十六进制，下同)，有 5 个不同的叶节点值：‘A’，‘B’，‘C’，‘D’，‘E’，代表了该后继树的后继模式，它们对应的 ASCII 编码为：41，42，43，44，45。图 2 显示了存储这棵后继树的后继信息的后继模式树，后继模式树的节点是叶节点字符的编码值，对应了该叶节点值在后继编号数组中的起始位置和当前位置，形成了一组键——值对。

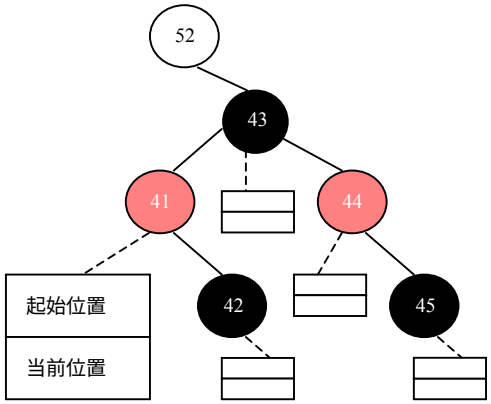

图 2 存储一棵后继树的后继位置信息的后继模式树

在具体实现时将所有后继模式树的叶节点组成的红黑树合并为一个红黑树数组，将根节点值作为数组下标，这样可以充分利用现有的数据结构来简化实现。下文中由于后继模式树的根节点与红黑树构成了一组键——值对，因此用 STrees[α]表示具有根节点值 α 的后继模式树对应的红黑树；而红黑树中各个节点与该节点的[起始位置，当前位置]又构成了另一组键——值对，所以用 STrees[α][β]表示红黑树中 β 节点对应的值，值中的 position 与 offset 分别对应图 2 中的后继值 suc 的起始位置和当前插入位置。

2 XML 树型结构与文本的联合索引

本文使用全文索引模型——互关联区间后继树来索引 XML 文档，统一了树型结构索引与文本索引，不仅索引 XML 树型结构，还同时索引 XML 文档中的文本数据。因为互关联后继树本身就是一种树型结构，所以用于索引 XML 树型结构和处理 XPath 查询时，不需要进行树结构语义转换，能够大大提高索引创建和查询的效率。

但是，XML 树型结构是分叉的，图 3 显示了一篇 XML 文档的树型结构。而互关联区间后继树模型同其它全文索引模型一样，遇到的最大问题是：树型结构有分叉，如果从树根开始向叶节点遍历，遇到路径分叉的节点时将所有分叉都作为一个后继，那么在外存上的索引结构就要区别不同长度的后继编号组，处理会变得很复杂。因此，本文采用了倒向的 XML 索引，有效地解决了这一问题。即建立的索引的方向是从叶节点指向根节点的，这样每个节点只有一个后继节点，即它的父节点。按照这个思想建立的互关联区间后继树如图 4 所示，图中‘#’表示 XML 文档的根节点‘/’，对应全文中的文档结束符。由于图 4 的后继编号只有 18 个，分别对应了 XML 树中 18 个节点的位置路径信息，因此倒向索引的膨胀比较低，又能保持树结构的语义。

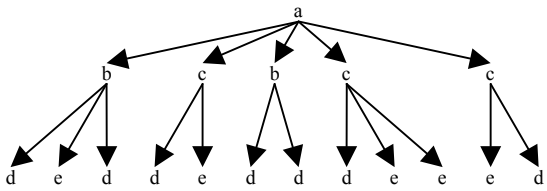

图 3 一篇 XML 文档的树型结构

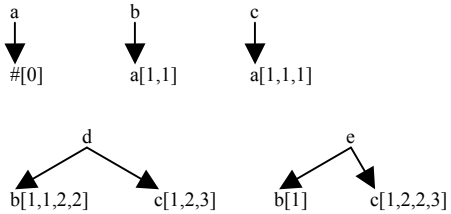

图 4 采用倒向索引的互关联区间后继树索引结构

将图 4 中后继树中所有的后继编号去掉，再加上起始位置和当前位置，就得到了后继模式树。在实现时，可以将索引看成后继模式树加上后继编号区间两部分组成。

因为互关联区间后继树用于全文索引，所以很自然地也可以用于 XML 中文本节点的索引，并且树型结构索引与文本索引的结构是相同的，只要在二者之间建立连接信息，就可以实现树型结构与文本的联合查询。因为一个文本节点只有一个父节点，所以在建立文本节点字符串的互关联区间后继树时，可以将最后一个字符的后继设置为该文本节点的父节点(元素节点或属性节点)，这样在 XPath 中的文本查询完成后，就可以利用这个后继值从文本索引部分进入树型结构索引部分进行查询。

3 基于后继模式树的 XPath 联合查询

由于各个后继模式树在索引建立完毕时，[起始位置，当前位置]表示了后继树叶节点对应的后继偏移量在外存文件中的存储范围，因此在查询时可以使用这些信息过滤掉无关节点。同时因为联合索引的 XML 树部分和文本部分的结构是相同的，所以查询时可以使用同一个查询算法，如下所示：

算法 1(基于后继模式树的区间过滤查询算法)

输入：区间后继树索引和查询路径$B = b_1b_2\ldots b_n$

输出：查询路径终止节点的后继编号集合 Offsets

方法：

01　令 Offsets 为全集;

```
02  for i = 1 to n – 1 do
03      position =STrees[bi][bi+1].position;      //起始位置
04      offset = STrees[bi][bi+1].offset;     //当前位置
05      在bi对应文件中偏移量从position到offset的数据取出，//
与Offsets集合求交，结果作为新的偏移量赋给Offsets集合。
06  endfor;
07  if offsets 为空  then
08      return null;     //查询结果为空
09  else
10      return Offsets;     //查询结果非空
11  endif。
```

算法分析：

算法 1 使用了基于区间后继树的区间过滤方法，该方法可以在搜索查询串中某一个字符的同时过滤掉不匹配的所有解是一种广度优先的查询方法。与通常的全文索引的深度优先查询方法相比，广度优先查询能够大大减少读取磁盘的次数，所以区间过滤法的查询效率很高，对于 GB 级的索引文件，其查询时间在秒级左右。

算法 1 给出了在区间后继树上的一段已知路径的查询算法，在查询路径已知的情况下，效率很高，但是在查询路径未知时，只能利用深度优先的策略进行试探查询，因此降低了效率。对于 XPath 中的正则表达式查询，如/a//b [contains(., “Computer”)]，对“Computer”，可以利用‘r’的后继编号继续查找“Computer”的父元素节点 b，但是在树型结构部分由 a//b 并不知道 a 和 b 之间的路径，又因为没有进行树节点相对位置的编码，所以在原文档中，如果从 a 节点进行遍历查找 b，那么其时间复杂度将是指数级的，如果从 b 进行遍历查找 a，其时间复杂度是多项式级的，这在文献[4]中已有证明。为了解决路径未知的问题，本文使用了在后继模式树上的自底向上路径查找的方法。先将从 b 到 a 的所有路径找出，然后利用算法 1 进行区间过滤查询，由此大大提高了 XML 树型结构的区间后继树索引的查询速度。

算法 2(后继模式树的自底向上路径查找算法)

输入：路径的端点 a 和 b，其中 a 是 b 的祖先节点

输出：b 到 a 的所有路径集合 Q

方法：

```
01  Q = Φ;     //将路径集合置为空
02  分配 2 个栈：searchStack 和 pathStack;
    //存储查找节点和查询路径的栈
03  searchStack.push([b, 1]);
    //b 节点和出栈次数 1 进栈
04  while searchStack 非空  do
05      [node, time] = searchStack.pop();
        //当前查找节点和出栈次数出栈
06      pathStack.push(node);
        //记录已经经过的查询路径
07      if STrees 中有键 node then
08          foreach suc_node in STrees[node] do
            //将当前节点的所有父节点进栈
09              if suc_node 在 STrees[node]中值最大  then
10              searchStack.push([suc_node, time + 1]);
                //如果最后一个父节点出栈，因为所有父
                //节点都搜索完毕，所以要将子节点出栈，
                //如果子节点原来也是最后一个父节点那
                //么也要出栈，以此类推
11              else
12                  searchStack.push([suc_node, 1]);
                    //如果不是最后一个父节点，那么出栈时
                    //只出 1 次
13              endif;
14              if suc_node == a then     //到达 a 节点
15                  pathStach.push(a);
16                  将 pathStack 从栈顶到栈底的所有第一个值作为
                    一个路径加入 Q;
                    //找到一条路径
17              endif;
18          endfor;
19      else     //搜索路径终止，没有找到 a 节点
20          [node, time] = searchStack.peek();
21          for i = 1 to time do
            //当前搜索路径出栈 time 次
22              pathStack.pop();
23          endfor;
24      endif;
25  endwhile;
26  return Q。      //返回所有 a 到 b 的路径集合
```

算法分析：

因为 XML 树型结构索引部分的后继模式树可以从 DTD 或 Schema 映射得到，其大小只由 XML 文档的模式(DTD 或 Schema)大小决定，而与 XML 文档大小无关，所以其规模是 O(1)，因此后继模式树上的自底向上查找算法的时间复杂度也是 O(1)。

利用算法 1 和算法 2 就可以找出从 XPath 中文本查询串的第 1 个字符到树型结构根节点的所有路径。如果在图 3 的第 2 个 c 节点的 d 子节点下有一个文本节点，其内容是“Computer science”，那么对于 XPath 查询串：/a//d[contains (.,“Computer”)],利用算法 1 查询“Computer”可找到该 d 节点，利用算法 2 将所有从 d 到 a 的路径找出，即 d→b→a 和 d→c→a。然后利用算法 1 在这 2 条路径上进行区间过滤，可以排除掉 d→b→a，从而确定该 d 节点即为 XPath 查询的解。

4 实验分析

上述算法在 1GHz CPU、512MB 内存、Windows Server 2003 的计算机上用 C#实现，并与 SQL Server 2005 针对 XML 的全文索引法和 B+树索引法上的 XPath 查询处理进行了实验对比，XML 数据集为流行的 XML 测试数据集 DBLP，大小 131MB。表 1～表 3 显示了实验结果。

表 1 索引的膨胀比

索引模型	索引大小	膨胀比
SQL Server 2005 全文索引	713MB	5.44
SQL Server 2005 B+树索引	380MB	2.90
区间后继树倒向联合索引	237MB	1.81

表 2 针对 DBLP 数据集的 3 个查询

XPath 查询串	结果数
Q1: /dblp//school[contains(. , “Department of Computer Science”)]	15
Q2: //*[contains(.,"Database Workload ")]	6
Q3: //*[contains(@key,"w3c")]	67

表 3 3 个查询串查询时间的对比 (s)

索引模型	Q1	Q2	Q3
SQL Server 2005 全文索引	5.0	8.0	2.0
SQL Server 2005 B+树索引	2.0	7.0	2.0
区间后继树倒向联合索引	0.4	0.9	0.1

（下转第 75 页）