第 33 卷 第 8 期 计 算 机 工 程 2007 年 4 月
Vol.33 No.8 Computer Engineering April 2007
·软件技术与数据库· 文章编号：1000—3428(2007)08—0077—03 文献标识码：A 中图分类号：TP311

一种基于路径分块索引的 XML 查询方法

陈冬霞，吉根林，肖 袁
(南京师范大学计算机科学系，南京 210097)

摘 要：针对 XML 的相对路径查询及引用路径查询问题，提出了一种面向 XML 数据的路径分块索引 KI。探讨了 KI 索引构造方法、索引节点分裂算法和相关查询处理的算法，并用 VC++实现，利用 Shakespeare 和 Xorder 数据集进行了 XML 查询测试，实验结果表明，提出的 KI 索引能有效地提高 XML 查询效率。

关键词：XML 技术；查询；索引

XML Query Method Based on Path Fragment Index

CHEN Dongxia, JI Genlin, XIAO Yuan
(Department of Computer Science, Nanjing Normal University, Nanjing 210097)

【Abstract】 Focused on the problems of XML relative path query and XML references query, a path fragment index named KI is put forward. The algorithms about the index constructed and XML data query based on the index are proposed. In order to ensure the stability of XML index query, the index node's splitting conditions and algorithms are studied. All algorithms proposed are implemented by VC++. Performances of them are studied by experiments. The experiment results show that the query algorithm based KI is effective and efficient.

【Key words】 XML technology; Query; Index

XML查询技术是目前XML研究领域的重要课题，为提高XML查询效率，人们提出了各种各样的XML索引，包括：DataGuides[1]，T-index[2]，A(k)-index[3]，D(k)-index[4]，XISS[5]，SphinX[6]，Index Fabric[7]，ApproXQL[8]等。DataGuides适合处理基于绝对路径的查询；T-index只能查找特定路径；A(k)-index、D(k)-index无法查询包含引用属性语义的路径；XISS、ApproXQL将复杂的路径表达式分解成若干SPE，每个SPE产生一个中间结果，通过连接合并各中间结果得到最终查询结果，而大量的连接合并操作影响查询效率；SphinX利用有限自动机进行相对路径到绝对路径的转化，路径转化效率较低；Index Fabric使用层次化的Patricia树作为索引结构，处理相对路径查询的效率不高。

针对如何高效处理相对路径查询和含有 IDREF 的查询问题，本文提出了路径分块索引 KI，并探讨了 KI 索引的构造算法、节点分裂算法以及相关查询处理算法。

1 KI索引

1.1 相关概念

为叙述方便，设有一 XML 文档对应的 DOM 树如图 1 所示，下面给出相关定义。

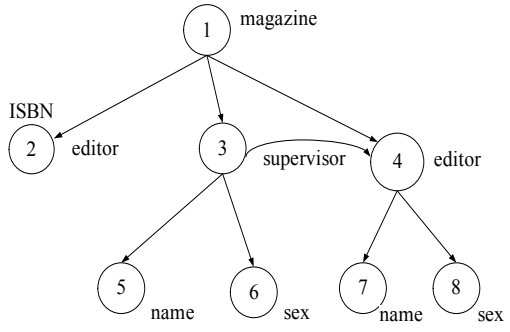

图 1DOM 树示例

定义 1 等价标签集是在入度上有相同标签的节点集合，记为 Leq。例如图 1 中标签 name 的 Leq＝{5, 7}。

定义 2 索引KI中的节点可表示成三元组(Nid，P，O)，Nid对应DOM树所有的相异节点标识；P对应从根节点到此节点的路径集合；O为Nid的Leq。对于KI中的索引节点ki，它的识别标签为N_{id}。ki节点由hash表组织。图 1 对应的KI结构如图 2。

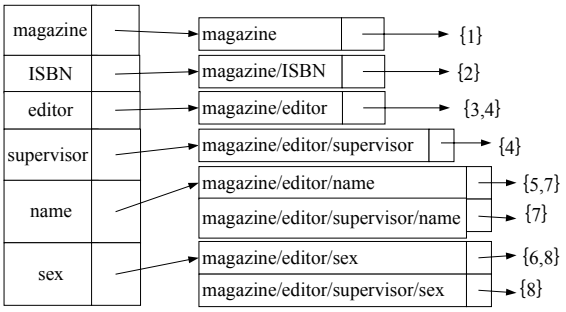

图 2 KI 结构示例

推论 1 KI 中的节点识别标签互不相同且这些标签的集合等于 DOM 树中的节点标签集。

推论 2 对于 DOM 树中的每个 Path，在其 KI 索引中必存在一个节点 ki 包含之;同理，若某 Path 存在于 DOM 树中，则其必存在于该 DOM 树对应的 KI 索引的某个节点 ki 中。

1.2 索引 KI 的建立算法

算法 1 描述了 KI 的建立过程，深度优先遍历 DOM，同

基金项目：江苏省高校自然科学基金资助项目(04KJB520075)
作者简介：陈冬霞(1978－)，女，助教，主研方向：数据库与XML技术；吉根林，教授；肖 袁，硕士生
收稿日期：2006-05-16 **E-mail：**chendongxia@njnu.edu.cn

时生成 KI。

算法 1 creatKI(root)

```
{index_root=creatIndexNode( );
 index_root.label=root.label;
 path.pushlabel(root.label);
 append(path,index_root);
 append(root,{eqnode}); //将根节点存入等价标签集
 recMake(index_root，{root});}
recMake(node，co_extent)
//参数是最近访问的索引节点和最近访问的数据节点集
 {label_set=ϕ;
  for(each object o in co_extent)
  label_set=label_set∪{ni}; //{ni}是o的出边节点集
  for(each label l in label_set) //对节点集中相异的节点
   {next_node=Gh.lookup(l);
//Gh 为全局 hash 表，存 DOM 中所有不同标签
if(next_node==nil)
{next_node=creatIndexNode( );
    next_node.label=l;
    Gh.Insert(l,next_node);}
    path.pushlabel(l);
    append(path,eset1);
//eset1 为最近访问的数据节点沿 l 到达的节点集
    recMake(next_node,eset1-eset2);
//eset2 是 next_node 分块中的所有节点
    path.poplabel( ); }}
```

1.3 索引节点的分裂算法

KI 建好后，查询时找到对应的标识标签就找到了相应的索引节点，根据查询直接去访问即可。存在的问题是：DOM 树/DG 往往不平衡，若倾斜严重，必导致某些索引节点存在大量相异路径，导致查询低效。为使分块合理化，令 KI 节点大小均匀，可将较大的节点分裂。

定义 3 对于 DOM 树中的标签 l，num(l)表示路径的尾标签为 l(即标识标签为 l)的路径条数。Path(l)代表标识为 l 的路径集合。对于标签序列 ls，num(ls)代表以该序列标签为结尾的路径条数。

定义 4 对于长度为k的路径$l_1/l_2/\ldots/l_k$，anchor(m)表示l_{k-m}的标签，特别地，anchor(1)表示l_{k-1}，称为锚，是这条路径的锚标签。在一个索引节点中的路径集合中，锚标签I的大小num(I)是这部分路径中所有锚标签为I的路径条数。

定义 5 DOM树中有n个不同标签$l_1, l_2, \ldots, l_n$，则KI中有n个节点，对应的路径集合大小分别为：$num(l_1), num(l_2), \ldots, num(l_n)$。其平均值记为AVG，需要分裂的索引节点应在Q集中，$Q=\{P(l_m)|AVG\text{-}num(l_m)>\xi, 1\leqq m\leqq n\}$，ξ为可调节分裂节点个数的阈值。

索引节点分裂算法描述如下：

算法 2 SplitNode(IndexNode *node*)

```
{size=0;
  a1=a2=∅;
  for each a in anchor_set
// anchor_set 按锚标签大小升序排列
    {size+=size of a;
     if(size<=node.ki.size/2)
     insert a into a1;//a1 中是无需分裂的
     else insert a into a2; }//a2 中是需要分裂的
  n1=creatIndexNode();
  n2=creatIndexNode();
  for each path p in KI//根据锚标签分裂成两部分
 {anchor=the anchor of p;
  if(anchor∈a1)
insert p into n1.path;
  else
insert p into n2.path;}
for each parent pa of node//处理节点的入度
{remove edge from pa to node;
if(pa.label∈a1)
        add an edge from pa to n1;
   else
add an edge from pa to n2;}
 for each child c of node
  {if(∃c.path.anchor(2)∈a1)
    add an edge from n1 to c
  if(∃c.path.anchor(2)∈a2)
    add an edge from n2 to c}
  delete node from index;}
```

2 查询处理

2.1 查询算法

在查询处理时根据用户查询的具体情况进行处理，利用 KI 索引进行查询，查询算法先判断是 SPE 查询还是 RPE 查询，然后再分别处理。

算法 3 QXMLI(qe)

```
输入 qe 查询表达式
输出 查询文档结果集
 {result_set=∅; //查询结果集置空
  for each sig-root in XML_DB
    l=test(qe); //判断 qe 是否为 SPE 表达式
    if(l)
     result_set=Qspe(qe); //处理 SPE 查询，返回查询结果集
    else
     result_set=Qrpe(qe); //处理 RPE 查询，返回查询结果集
    endif;
outxml(result_set);}
```

2.2 处理 SPE 查询

SPE 查询含绝对路径查询(形如:/book/editor/name)和相对路径查询(形如:lib/book/ISBN)。处理 SPE 查询的算法见算法 4。

用签名方法进行预处理，缩小文档搜索范围。取查询 spe 的 LastLabel;访问 KI 的 Ghash 从中找到与 LastLabel 匹配的节点;将 spe 与找到的 KI 节点的路径进行匹配;若匹配，则将此 KI 指向的 Leq 存入结果集 r_set。

算法 4 QSPE(spe)

```
输入 spe 查询表达式
输出 答案节点集
 {r_set=∅; //答案节点集初始化
  ql=Get_lastlabel(spe); //取查询路径的尾标签
  ki_set=Search_KI_L(ql);
//从 Gh 中找标识标签为 ql 的索引节点
  for each ki in ki_set
   if ( cmp_path (ki.path, spe) ) //若 KI 的路径符合 spe
    r_set +=ki_leq; //找到 KI 指向存放的 leq，加入查询答案集
return( r_set);}
```

2.3 处理 RPE 查询

RPE 查询亦含绝对路径查询(形如:/set/#/title/procedure)和相对路径查询(形如:#*/title/book)，处理的难点在于正则运算符的处理。相应的查询处理算法如下：

算法 5 QRPE(rpe)
输入 rpe 查询表达式。
输出 答案节点集。
(1)for each RPE expression(#/e1/e2 or e1/e2/#)
//消除开头和结尾的“#”号
replace(#/e1/e2) with (e1/e2);
r_set=QSPE(e1/e2);
replace(e1/e2/#) with (e1/e2);
r_set=QSPE(e1/e2/allchild);
(2)for each RPE expression(e1/#/e2) //消除中间的“#”号
replace it with (#/e2);
a_set=QRPE(e2);
r_set=cover(sig_el, a_set);
(3)replace each e？with e | ; //消除“？”号
(4)for each RPE expression((e1)*/e2) //消除“*”号
replace it with (#/e2);
r_set=QSPE(e2);
(5)for each RPE expression((e1)+/e2) //消除“+”号
r_set=QSPE(e1/e2) ∪ QSPE(e1/#/e2);
(6)for each RPE expression like(e1/value>A or e1/value>A etc)
//处理等值或范围查询
r_set=QRPE(e1);
for each rr in r_set
r_set=find_value(rr.vl, A);

3 实验结果与性能分析

为研究KI索引对XML查询性能的作用，将本文提出的索引KI用VC++加以实现， 并将基于索引的XML查询算法QXMLI和基于签名的查询算法SXMLQ[9]进行查询性能比较。测试数据集为Shakespeare XML集和Xoder。Xoder是从虚拟的订单数据抽象出来的合成XML数据集。其部分文档片断如下：

```
<orders>
  <order>
    <customerid limit="10">
      0690801
    </customerid>
    <status>student</status>
    <item>
      <book>Abook</book>
      <book>Bbook</book>
    </item>
    <item>
      <CD>CD1</CD>
    </item>
  </order>
    …
</orders>
```

在两测试集中进行 SPE 和 RPE 查询，既有绝对路径查询又有相对路径查询。图 3 给出了一 RPE 查询实验样例。

图 3 RPE 查询样例

表 1 是在沙翁集中的一些查询实例。

表 1 Shakespeare XML 数据集中的查询

序号	查询表达式
q1	play/act/#*/speech
q2	play/#*/persona
q3	*.title
q4	play/act
q5	play/act/scene/speech/line/stagedir
q6	/scene/title
q7	/act/scene/speech
q8	#/speaker
q9	act/*/speaker
q10	#*/speech/#

每种查询进行 20 个以上实验，每个实验进行 3 次，取 3 次平均时间值，计时 ms 级。查询实验时间比较效果见图 4。

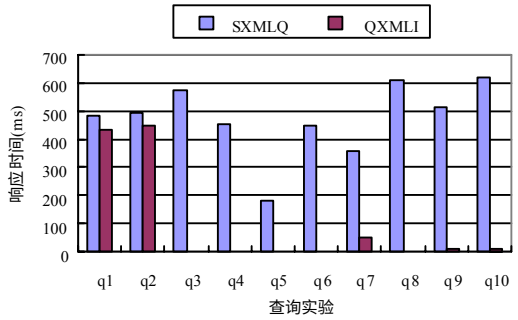

图 4 比较 SXMLQ 和 QXMLI 的运行时间

图 5 为两种方法在沙翁集不同数据量大小的情况下分别进行查询，比较平均查询时间的效果图。

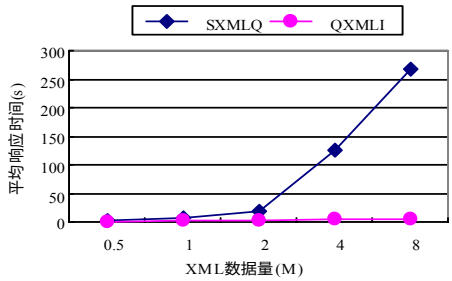

图 5 沙翁集查询性能

图 6 为 Xoder 集同样的实验性能比较。由实验结果可见，QXMLI 的性能优于 SXMLQ。SXMLQ 在沙翁集中逊于在 Xorder 中的查询性能，因沙翁戏剧中一些结点值中字符串较多，影响了签名查询效果。而 QXMLI 性能基本稳定，但其在 Xorder 中逊于在沙翁测试集中的查询性能，因 Xorder 元素种类少，相似的结点和路径很多，导致一些索引结点较大，影响了查询效果。

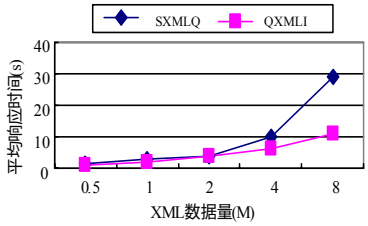

图 6 Xorder 集查询性能

4 结束语

XML 索引技术对 XML 数据库查询起着至关重要的作用，针对如何高效处理相对路径查询和含 IDREF 的查询问题，本文提出了路径分块索引 KI，实验结果表明索引 KI 能有效提高 XML 查询性能。 (下转第 82 页)