第35卷 第18期 计 算 机 工 程 2009年9月
Vol.35 No.18 Computer Engineering September 2009
·软件技术与数据库· 文章编号：1000—3428(2009)18—0051—03 文献标识码：A 中图分类号：TP311.13

基于DTD的XML索引查询技术

魏东平，宗德君，孙华国
(中国石油大学(华东)计算机与通信工程学院，东营 257061)

摘 要：针对XML文档索引查询中非法路径查询响应时间过长的问题，提出一种利用DTD模式进行预处理的索引方法。建立索引DWBI，采用新的基于区域编码方式，有效地支持祖先-后代判断。查询时利用DTD模式对查询进行预处理，再查询带有DTD信息的XML索引树，从而提高查询的效率。

关键词：路径查询；路径索引；模式；区域编码

XML Index Query Technology Based on DTD

WEI Dong-ping, ZONG De-jun, SUN Hua-guo
(Institute of Computer and Communication Engineering, China University of Petroleum, Dongying 257061)

【Abstract】 Aiming at the problem that the response time of illegal path query is too long, this paper proposes a new indexing method which proceeds pretreatment with DTD schema of XML document. The new index DTD and WAN-Based Indexing(DWBI) adopts a new region-based numbering scheme which can determine the ancestors-descendant relationship between a pair of objects effectively. It uses the DTD to pretreat the querying and queries XML index tree which has been appended DTD information, so that the efficiency of the query is improved.

【Key words】 path query; path index; schema; region-based numbering

1 概述

路径索引是对一类具有相同特征的索引结构的总称。大体上来说，就是支持路径表达式查询的索引结构。以下是几个有代表性的索引：

(1)DataGuide[1]和1-Index[2]是一样的，利用简化树描述所有从根开始的路径。DataGuide虽然减少了遍历路径查询时所需要的部分节点，但不适用于从任意节点开始的查询。

(2)Index Fabric[3]是基于Patricia tree的一种半结构化数据索引结构，对XML文档中从根到叶的路径进行编码形成字符串，由于所采用的编码的局限性，因此在处理带“//”效率不高。

(3)SphinX[4]是一种利用DTD对XML文档进行索引的方法。SphinX利用B-树建立索引，但XML文档只有叶子节点携带了相应的DTD结构信息，只有条件路径在XML文档中的定位可以利用DTD的结构，对于目标路径依然需要对XML文档树遍历。

这些方法的缺点之一是对XML文档中根本不存在的路径也会试着查找。

索引的建立离不开对XML文档结构的描述，而DTD本身就具有这一功能，如能充分利用DTD的结构信息，将有利于改善查询效率。

本文基于WAN编码[5]提出了新的基于区域编码方案，并利用DTD模式信息建立索引DWBI(DTD and WAN-Based Indexing)，从而提高了查询的效率。

2 索引的编码方法

本文提出的新编码方案是基于WAN编码的。首先，将DTD与XML文档都映射为树，分别编码。

原理：为树的每个叶子节点添加一个有序的虚拟节点，相应树中每个节点编码为(minOrder, maxOrder, level)，其中，minOrder是以此节点为根节点的子树的最左下角叶子节点的虚拟节点的序号；maxOrder为子树最右下角叶子节点下方的虚拟节点的序号；level为节点所在层数。

如图1所示，树T中任意2个节点u和v具有祖先/后裔关系，当且仅当 minOrder(u)≤minOrder(v)∧maxOrder(v)≤maxOrder(u)∧level(u)<level(v)，即祖先节点 u 的编码区间[minOrder(u), maxOrder(u)]包含后裔节点 v 的编码区间[minOrder(v), maxOrder(v)]。显然判断父/子关系时只需要判断是否满足条件：level(u)-level(v)=1。

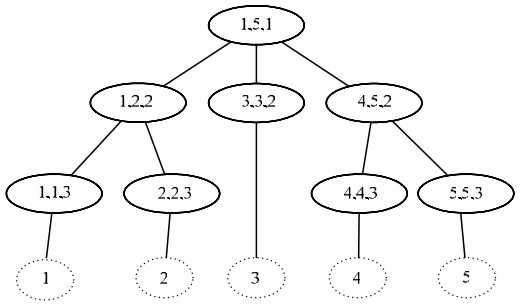

图1 修改后的WAN编码

为了防止节点增删操作影响其后节点的前序编码，对虚拟节点编码可以采用实数制。这样增删操作不会对其他虚拟节点产生影响，当然也就避免了树中节点的编码频繁更新。

2.1 DTD编码

DTD树中的节点 n 的编码为四元组(minOrder(n),

作者简介：魏东平(1965－)，男，副教授、硕士，主研方向：数据库应用技术，应用软件开发，网络系统；宗德君，助理工程师、硕士；孙华国，硕士

收稿日期：2009-03-16 **E-mail：**zdjss@163.com

maxOrder(*n*), level(*n*), EA)。若 EA 为 0，该节点为元素节点，为 1 则表示属性节点。

2.2 XML 编码

XML 树中的节点 *n* 由一个六元组(docid(*n*), dtdminOrder(*n*), minOrder(*n*), maxOrder(*n*), level(*n*), Value, EA)标识。其中，docid(*n*)是节点 *n* 所在的 XML 文档的 ID；dtdminOrder(*n*)是节点 *n* 在相应的 DTD 树中对应的节点，如图 2 所示(图中省略了 docid 和 Value)。

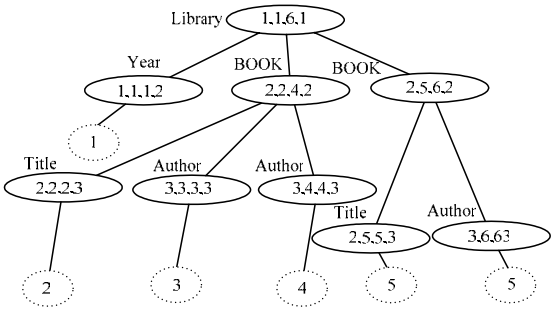

图 2 XML 文档的索引编码

对于 XML 文档，每个节点的编码中都携带了对应的 DTD 结构信息，用 dtdminOrder 表示，使索引能够精确定位参与结构连接运算的 XML 节点集。例如，XML 文档中路径 Book/Title 中的 Title 节点，不会参与路径 Magazine/Title 的结构连接运算。

3 索引结构

XML 索引见图 3。

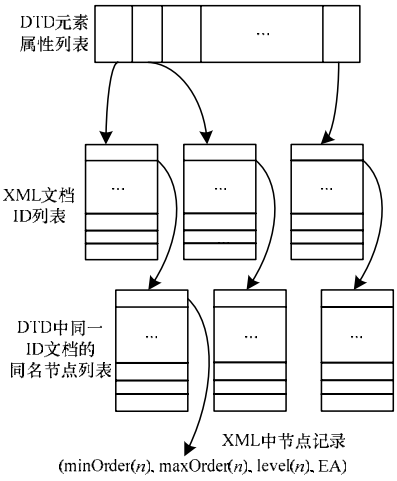

图 3 XML 索引

DTD 索引是一个倒排表，关键字是 DTD 中的元素/属性名，每个元素/属性名指向一组有相同名字的元素/属性的记录。

在 XML 索引中，左边第一列是 DTD 树中的元素/属性节点的列表，每个记录指向一组有相同名字的对应 DTD 树中同一元素/属性节点的 XML 元素/属性节点集。

4 路径查询的实现

同其他大部分处理多谓词的索引查询一样，索引分 2 步实现只有 1 个谓词约束的路径查询，即 DTD 树结构匹配和 XML 查询处理。

4.1 DTD 树结构匹配

(1)如果查询路径在 DTD 树中是不合法的

DTD 中没有与查询路径相匹配的结构，在相应的 XML 文档中也不存在与查询路径相匹配的结构。这时不用再对 XML 进行查询匹配，结束查询。

(2)如果查询路径在 DTD 树中是合法的

1)既含有目标路径又含有条件路径

例如，对于/Library[@Year>2000]//Title，条件路径为 Library/@Year，目标路径为/Library/Book/Title，分支点为 Library。分别求取 DTD 树中条件路径的叶子节点的 minOrder(Year)、层次 level(Year)，分支点的 minOrder (Library)、层次 level(Library)，还有目标路径的叶子节点的 minOrder(Title)。

2)只含条件路径

例如/Book[Author]，分别求取 DTD 树中条件路径的叶子节点的 minOrder(Author)、层次 level(Author)，还有目标路径的叶子节点的 level(Book)，求取 minOrder(Book)。

3)只含目标路径

例如/Library//title，分别求取 DTD 树中条件路径、目标路径的 minOrder(title), level(title), minOrder(Library), level (Library)。

4.2 XML 查询处理

(1)如果只有目标路径，则查询结果是含有 dtdminOrder 编码的 XML 元素/属性记录集合。

(2)如果查询路径中含有条件路径，则按照以下步骤实现查询：

1)从 XML 索引中找出所有对应 DTD 树的叶子节点的 dtdminOrder、符合条件约束的 XML 节点集，以 XML 文档 ID 进行分组。

2)如果只有 1 条只含谓词约束的条件路径，那么 XML 节点集 *C* 中节点的第(level(*c*)-level(*t*))代祖先节点集 *B* 就是所要查询的结果，查询也就结束了。

如果含有 2 条路径，那么求出上述祖先节点集 *B* 后按文档 ID 分组继续下面的步骤。

3)根据目标节点在 DTD 中的 minOrder，从 XML 索引中找出能够对应 DTD 树中节点编码 minOrder 的 XML 节点集 *T*。

4)对每一组具有相同 XML 文档 ID 的节点集 *B* 和 *C*，调用拥有关系结构连接算法 Hold-Join[6]，求出 *C* 中的节点在 *B* 中的祖先节点的集合 *B*_selected。

5)对每一组具有相同 XML 文档 ID 的节点集 *B*_selected 和 *T*，调用结构连接算法 Qeue-Tree-Anc[6]，求出 *T* 中所有 *B* 中节点的子孙节点集，其中的每个节点都是待求的 XML 节点。

4.3 算法 Hold-Join 条件节点与分支节点的连接算法

具体连接算法的流程如下：

输入 参与连接的祖先、后裔集合 *A* 和 *D*，用列表存储

输出 *A* 中满足拥有关系的元素节点序列

```
Hold-Join(A, D)
While(a 和 d 都不指向栈底或者 Stack 不空)
//a,d 表示当前记录；用 Sa 指向栈顶元素；
{If(Stack 非空，并且 Sa 拥有 d 或者 d 在 Sa 之后)then
If(d 在 Sa 之后)then
{将 Stack 中位于 d 之前的元素逐一出栈；
将出栈元素所指向的链表直接链接到栈中下一元素所指向的链表之后；
If(出栈的是栈底元素)then 输出栈底元素所指向链表中所有元素；
}
```

Else if(栈顶元素位于栈底)then

输出栈顶元素以及所指向链表中所有元素，并将栈顶出栈；

Else{栈顶元素出栈；将栈顶追加新栈顶元素所指向的链表；

将已出栈元素链表直接链接到新栈顶所指链表后面；

}

If(a 拥有 d) then

If(Stack 为空)then 输出 a；

Else 将 a 追加到栈顶元素所指向链表；

If(是双亲孩子拥有关系) then

a 指向列表 A 中 minOrder≥minOrder(d)的第 1 个记录；

else a 指向下一个记录；

else if(a 位于 d 之前) then a 指向列表 A 中 minOrder>maxOrder(a)的第 1 个记录；

else if(a 是 d 的祖先) then 将 a 入栈，并指向 A 中下一记录；

Else //d 是 a 的祖先或者 d 位于 a 之前

If(满足双亲/孩子关系或者栈已空)then d 指向 D 中 minOrder>min(maxOrder(d.parent))的首记录；

else d 指向 D 中 minOrder 大于 minOrder(a)的首记录；

}

5 索引的动态更新

为了更新方便将索引的存储结构作如图 4 所示的修改。

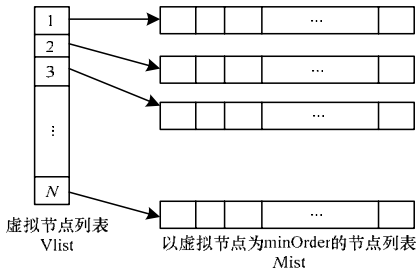

图 4 索引树的存储(按照 minOrder)

为了节省存储空间，用虚拟节点的存储序号代替 minOrder，所有以 *i* 为 minOrder 的节点都存在同一个列表中，这样每个节点的 minOrder 就不用存储了，对节点 minOrder 就演变成了节点记录在 *N*list 中的移动。

(1)插入操作对 minOrder 的影响如下：

If(插入节点作为叶节点)then

{If(此处本来无节点)then

插入节点的 minOrder 与 maxOrder 同为虚拟节点序号，其他节点不用修改；

Else If(此处本来就有一个节点 A)then

{新插入的节点 B 作为 A 的右兄弟存储，Vlist 中 A 所在虚拟节点 NA 后面插入一个新虚拟节点 NB；

B 添加到新加入虚拟节点 NB 的 Nlist 中；

对以 A 为 maxOrder 的节点作修改，就是将原来在 A 的 Plist 中的祖先节点放到 B 的 Plist 中；

}

Else If(此处本来就有一个节点 A)then

新插入的节点作为这些节点的中间节点，插入的节点的 minOrder 与 maxOrder 为虚拟节点序号，其他节点不用修改；

}

Else If(插入的节点是路径中的节点)then

新插入节点插入其孩子节点所在列表，其他信息与孩子节点相同；

(2)删除操作对 minOrder 的影响如下：

If(删除的是叶节点)then

{If(该节点有左右兄弟)then

直接删除该节点，其他节点不用修改；

Else If(该节点只有左兄弟)then

将以它为 maxOrder 的祖先节点放到以其左兄弟为 maxOrder 的列表中；

Else If(只有右兄弟)then

将以它为 minOrder 的祖先节点放到以其右兄弟为 minOrder 的列表中；

}

对于 maxOrder 的更新同理。

6 实验测试及分析

由于 SpinX 也是利用 DTD 进行索引的，因此将索引 DWBI 同它进行查询效率比较。DWBI 的原型系统用 VC++ 实现(运行操作系统为 XP)，2 个系统运行的硬件环境为 Intel 800 MHz PIII, 512 MB 内存。实验数据集采用莎士比亚戏剧。

查询实例：

ACT//SPEEACH

/PLAY/TITLE

//PERSONAE[PGROUP/PERSONA=“AMIENS”]/TITLE

SPEECH//ACT

/PLAY /PERSONAE[PGROUP/PERSONA=“AMIENS”]/TITLE

6.1 索引建立效率

由表 1 中数据可以看出 DWBI 的索引建立效率较 SpinX 要低，原因是 DWBI 索引要建立 DTD 和 XML 2 个索引，同时在建立 XML 索引时要在节点中加入大量 DTD 节点信息，这个过程浪费了很多的时间。但是 SpinX 利用 DTD 索引定位相关节点集合后对 DTD 信息的利用就此结束，而 DWBI 却是将 DTD 信息嵌入每个 XML 节点中，在整个路径查询的过程中都不用费时地去遍历文档。再者 DWBI 索引的建立是在 XML 文档读入时，这种预处理的方式仅在查询的开始效率较慢，在索引建立后查询所用的时间效率是很可观的。

表 1 索引建立效率比较 s

数据集	SpinX 建立时间	SpinX 索引建立时间	DWBI XML 建立时间	DWBI DTD 索引建立时间
莎士比亚戏剧	9.0	9.7	16.1	15.5

6.2 响应时间比较

由图 5 明显可以看出 DWBI 的查询反应时间较 SpinX 要小很多。这是因为对于不合法的查询表达式，DWBI 将查询停止在对 DTD 索引查询的阶段，而 DTD 索引较 XML 索引要小很多，所以查询时间少，及时给出无匹配结果的响应，极大地提高了查询的响应时间。

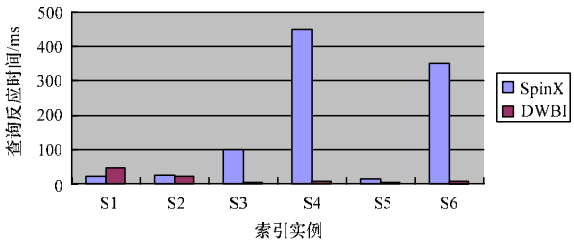

图 5 索引查询反应时间

同时查询通过利用 DTD 进行预处理，采用新的编码有效地利用区间范围判断任意两节点间的关系。处理拥有关系连接算法尽可能多地跳过了不需要参与连接的节点。结构连接时充分利用了结构连接算法的优点，有效处理嵌套、连接结果过大问题。

(下转第 56 页)