利用DTD模式优化基于路径的XML的查询

许丰娟　洪晓光

（山东大学计算机科学与技术学院　济南250061）

摘　要　XML作为一种数据的表示形式，正在数据库及网络中数据传输的领域被广泛使用，提高对它查询的速度也成为一个我们研究的方向，最近提出了利用DTD来提高查询效率的方法，基于这一思想，为了更高效地利用DTD，这篇文章提出了一种更高效利用DTD的方法，首先介绍了扫描DTD树的结果，即真路径的存储方式，然后给出了扫描DTD树的算法，我们还给出了怎样利用真路径对文档树进行扫描的算法，最后给出了例子，并分析了我们提高效率的原理。

关键词　XML，DTD，真路径

Use DTD to Optimize XML Query Based Path

XU Feng-Juan　HONG Xiao-Guang

(College of Computer Science and Technology, Shandong University, Jinan 250061)

Abstract　XML is suitable for data representation, and it is being used widly in the area of DataBase and of exchangeing data over the Web. It is essential to find an effective XML query language. To improve XML query efficiency, it is proposed to use DTD to optimize XML query. In this paper, we propose a mode of saving for the result of scanning DTD to make the best of DTD. We call the result true paths. And we also propose several algorithms for scanning the tree of DTD and scanning XML document using true paths. At last, our analysis for this method shows that the proposed algorithms and mode can improve XML query efficiency.

Keywords　XML, DTD, True paths

1　背景

随着XML作为一种网络语言的广泛使用，产生一种有效的XML查询语言是目前研究的热点，关于这方面的研究很多，如现在正在发展的查询语言Xpath[2]、Xquery[4]等，尽管XML表达数据时经常以文本的格式出现，但它却能很清晰地表示出数据之间的层次关系，每个XML文档都对应着一棵树状结构，基于它的树形结构的查询也很多，比如关于利用XML文档结构对XML文档的查询[1,4~7]，它们都讨论了基于结构的递归的查询，在查询文档时调用了递归函数。我们所用的很多查询，大多都是以路径的方式输入的查询条件，但路径中的元素只是父子关系及子孙关系，并不是文档中找到该元素的详细路径，如：liberary//title，liberary只是title的祖先，查询//title，就是无论title的结构是怎样的，无论它在文档的何处，所有文档中的title元素的内容，都是我们想要的结果，Xquery[4]讨论了利用XML的结构来做查询的方法，其是按深度优先扫描XML文档的，因为每个XML文档都有一个DTD与之对应，我们可利用DTD对XML文档进行查询，Xquery[4]和Structural Function Inling[1]的做法都是将一个深度优先扫描XML文档的递归函数化简成一个嵌入式的简单的函数，每化简一步，都要扫描一遍DTD树中的结点。基于这一点，我们这篇文章中提出了一种高效利用DTD的方法，提出了一种对DTD扫描的方法及怎样存储其扫描结果，然后在扫描XML文档时利用了第一步扫描DTD树所得到的结点，并充分利用了它的存储结构，扫描DTD树的结果是所有有用的路径集合，由于知道结点的顺序，因此在扫描XML文档时，只要将文档中的结点与路径中相应结点直接找到其对应的DTD树中的结点，而不必再从DTD树的根节点慢慢往下找，这样只要扫描一遍DTD树即可，从而可大大提高查询的效率，并给出了利用DTD扫描的结果对XML文档树查询的方法。

本文第2部分中介绍了几个与文章相关的几个概念；第3部分描述了怎样存储扫描DTD得到的路径及扫描DTD的算法；第4部分描述了怎样利用由第3部分的结果来层次遍历XML文档；最后讨论了方法的应用情况，并对它的效率做了叙述。

2　相关知识

以下我们给出几个与我们的文章及其背景相关

许丰娟　硕士生，主要研究方向：数据库优化、XML和半结构化数据库系统。洪晓光　博士，教授，主要研究数据库优化、网络应用、分布式数据库、并行数据库、Web数据库。

的基本概念。

·符合 DTD 定义的 XML 文档，叫合法的 XML 文档。

·XML 文档所对应的树状结构，我们称之为文档树。

·DTD 所对应的树状结构，称之为 DTD 树。

·文档树或 DTD 树中的一结点，它所对应的内容是所要查询的结果，我们称为目标结点。

·在 DTD 树中，从根节点到目标结点所经过的所有结点及其顺序构成的路径，我们称为真路径，在文档树中若要找到目标结点，其路径也必为某条真路径。真路径上结点的顺序对我们也是非常重要的，所以我们要记的是真路径，而不是一大堆的结点。

```
<!ELEMENT bib(book+,cd+)>
!ELEMENT book(title,author+)
! ELEMENT cd(title,price)
! ELEMENT title(#PCDATA)
! ELEMENT author(first,last)
! ELEMENT price(#PCDATA)
! ELEMENT first(#PCDATA)
! ELEMENT last(#PCDATA)>
```

(a) DTD 定义

```
<bib>
    <book>
      <title>DB</title>
      <author>
        <first>Li</first>
        <last>Ping</last>
      </author>
</book>
<cd>
  <title>Songs</title>
  <price>50</price>
  </cd>
<bib>
```

(b)DTD 定义的 XML 文档

图1

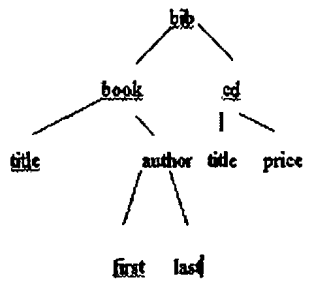

图2 图1(a)对应的 DTD 树

对于图2，若查询路径为//title，目标结点为 title，真路径有两条(bib，book，title)，(bib，cd，title)。

3 遍历 DTD 树

我们在本节中将详细描述怎样遍历 DTD 树，找到我们所需要的真路径。

3.1 真路径的存储结构

对于怎样利用我们得到的真路径查询文档树，真路径的存储结构也是非常重要的，如下：

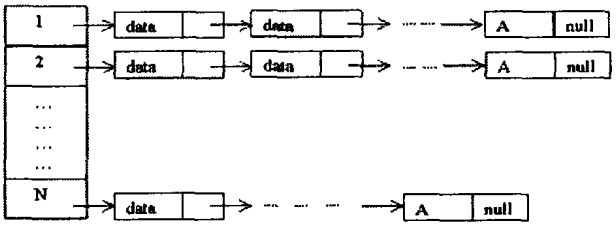

图3 真路径的存储结构

总的是以数组存储的，数组的每个元素是可能找到目标结点 A 的一条真路径；每条真路径是用链表存储的，即链表中的元素加上链表中元素的顺序就成为一条真路径(如图3)。链表中每个元素的 data 为 DTD 树中查到目标结点 A 所经过的结点。

3.2 扫描 DTD 的基本思想

深度扫描整棵 DTD 树，将每个结点压栈，直到遇到目标结点 A，然后将栈顶元素出栈，并将其放入真路径中；以后每个元素出栈时，看一下已出栈的真路径部分中是否有以自己的孩子结点为开头的，若有，就将其加在相应真路径的前面。若我们的目标结点为图2中的 title，开始将 bib，book 依次压栈，找到 title(1)，将 title(1)放入 road 中，road 为存储真路径的数组，其结构如图3，然后将 last 压栈，first 出栈，真路径的开头元素中无其孩子结点，然后将 last 压栈，依次将 last，author 出栈，都不做其他操作，因为它们出栈时，road 中没有以其孩子结点开头的真路径。然后 book 出栈，因为 road 中的一条真路径的开头元素为 title，为 book 的孩子结点，所以应将 book 加在此条真路径的开头；然后 cd 入栈，这时又遇到 title(2)，将 title(2)放入 road 中，产生第二条真路径，然后 price 入栈、出栈，什么都不做，然后 cd 出栈，这是 road 中有两条真路径，分别以 book 和 title(2)开头，所以将其加在以 title(2)开头的那条真路径中，然后 bib 出栈，因为 book，cd 都是其孩子结点，所以将 bib 加在两条真路径中的开头，扫描过程中，road 的变化如图4，(a)是扫描完 title(1)后的情况；(b)是扫描完 book 子树后的情况；(c)是扫描完 title(2)后的情况；(d)是扫描完 cd 子树后的结果；(e)是扫描完 bib 树后的结果。

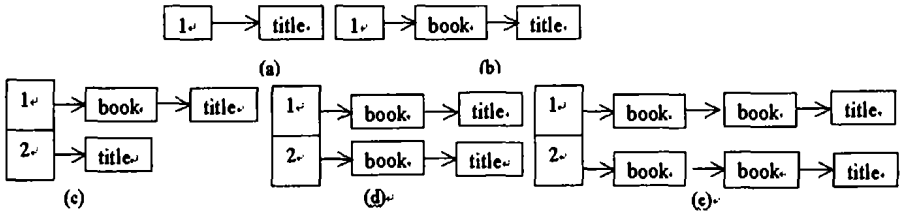

图4

3.3 扫描 DTD 的算法

算法1

if judge(Root)then return road

算法2

```
1. judge(root){
2. if root is A then
3. (road. add(A);
4. return true)
5. else
6. (vi=1
7. putin=0
8. while a=root child(vi)()null do
9. (if judge(a) then putin=1
10. vi++)
11. if putin=1 then
12. (road addsub(root)
13. return true)
14. else
15. return false
16. }//end if root
17. //end judge
```

Root 为 DTD 树的根结点，函数 judge 是用来判断 Root 是否在某条真路径中，因为 Root 为根结点，所以 Root 必被写在每条真路径的开头元素，若 Root 不在任何一条真路径中，则说明 DTD 树中不存在真路径；road 为存储所有可能找到目标结点 A 的所有路径的数组。

函数 judge(如算法2)，若元素 root 被写入某条真路径中，则返回 true，否则返回 false，若 root 为 A，则写入真路径中，即插入数组 road 的最后一元素的下一条真路径中(line 2 to 4)，putin(line 7)表示 root 能否加入 road 的某条真路径中，若 root 的某个孩子加入到 road 的某个路径中，则 root 也要加入相应的路径中，并将其加在相应真路径的最前面(line 9 to 13)，若 root 的孩子结点没有在任何真路径中，且 root 也不是目标结点 A，则返回 false (line 14. 15)；其中 road、addsub(root)作用是查找所有以 root 的子结点开头的真路径，并将 root 加在其前面，因为这些路径的第一个结点为 root 的孩子结点，所以在 DTD 树中若想找到这些路径的开头，必先找到 root 才行。

4 遍历文档树

本节中将要用到前面得到的真路径 road，我们用分支限界法层次遍历这棵文档树。由于 DTD 是 XML 文档结构的规范，因此在 XML 文档树中所有能找到目标结点 A 的路径必为真路径的子集。这也是以下遍历文档树中限界的主要思想。Q 为遍历中用到的队列，Q 中每个元素的结构如图5，node 为文档树中的结点，layer 为 node 在文档树中的层次(1，2…)；i，j，分别表示 node 可能是 road 中第 i 条和第 j 条真路径经过的结点。

设第 i 条真路径称为第 i 种找到目标结点的方法，则要看 node 可能是第几种找到 A 的方法中的结点，就要先看它的父亲结点是第几条真路径中的结点，假设它的父亲是第 m 条真路径经过的结点，则只要将第 m 条路径的第 layer 个结点的 data(即第 m 条真路径在 DTD 中经过的第 layer 个结点)与 node 比较即可，若相同，则 node 有可能为第 m 种找到目标结点 A 的方法所经过的结点，若 node 不是任何一种方法中经过的结点，则就没必要再验证它的子孙结点了，这也是限界的条件。

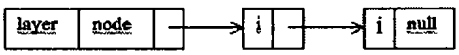

图5 Q 中每个元素的结构

算法3

```
1 new(Root)
2 Root. layer=1
3 Root. node=root
4 For i=1 to road. lengty()
5   Root. addsub(i)
6 Q. putin(Root)
7 While Q is not empty do
8   {H=Q. out()
9   if H. node=A then printout A
10  else
11    {vi=1
12    while b=H. node. child(vi)()null do
13    {bin flag=0
14    r=H. next
15    while r()null do
16    {for I=1 to (H. layer+1)
17      x=road[r. data]. next
18    if x. data=b then
19       {if bin flag=0 then
20          {new (B)
21          B. layer=H. layer=1
22          B. node=b
23          bin flag=1}
24       B. addsub(r. data)
25       }//end if data=b
26    r=r. next
27    }//end while r()null
28 }//end while b()null
29 }//end if H. node=A
30 }//end while Q is not empty
```

算法3中，首先将 root 进队列 Q，并在其后追加经过 root 的真路径的序号(line1 to 6)，然后依次将 Q 中头元素 H 出队列(line 8)，若为 A 则输出(line 9)，否则依次让 b 为 H. node 的孩子结点；r 依次取 H 后的真路径的序号，以为 H. node 经过第 r. data 条路径，b 在文档树中的层次为(H. layer+1)，所以，b 只要与第 r. data 条真路径中的第(H. layer+1)个元素相比即可(line 16 to 27)。若比较结果为相等，则将 b 入队列，并将 r 追加在其后(line 19 to 24)。

5 算法的应用及其高效性

图6为图1(b)所对应的文档树。如上例，由 DTD 树得到的真路径 road 有两条：(1)(bib，book，title)；(2)(bib，cd，title)。

通常我们用到的文档很大，而 DTD 树与其对应的 XML 文档树相比，结点却少得多，利用 DTD

(下转第133页)

中的部分数据冲突问题。从整体构架上，考虑了系统的通用性和人机交互友好性。今后的研究方向主要集中于采用更多更好的方法解决数据集成中的数据冲突问题，如采用基于本体的集成[3]等。

参考文献

1 Calvanese D, degiacomo G, Lenzerini M, Nardi D, Rosati R. Data Integration in Data Warehousing. Internation Journal of Cooperative Information Systems, 2001, 10(3): 237～271

2 Building L L Y, Ozsu T, Liu L. Accessing heterogeneous data through homogenization and integration mediators. In: Proc. 2nd IFCIS Conf. on Cooperative Information Systems (CoopIS-97), 1997. 130～139

3 Buccella A, Cechich A. An Ontology Approach to Data Integration. JCS&T, 3(2)

(上接第111页)

来优化文档查询，只要扫描一遍规模较小的 DTD 树，即可省去大量的对文档树所做的无用的遍历。平均来说，文档树的结点数与 DTD 树的结点数的比越大，我们方法的效率越高。

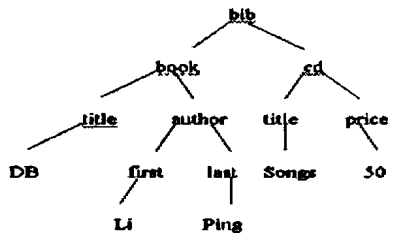

图6 文档树

我们来看一下它们的 I/O 情况，不用 DTD，直接对 XML 文档树进行扫描查询，则要扫描一遍 XML 文档，假设内存还有 s 个缓冲区，XML 文档有 d 个块，这时扫描一遍 XML 文档树的 I/O 为[d/s]，且每次查询时都要有[d/s]次 I/O；若用 DTD 树对 XML 文档树进行扫描，一般的查询方法是在扫描文档树的同时，参考 DTD 树。这样，每扫描一条路径，都要相应地扫描一次 DTD 树，假设 DTD 树有 n 个块，若利用回溯法，假设 XML 有 r 个叶子结点，即相应地有 r 条路径，设利用回溯法需扫描的路径有 r1条(r1＜＝r)，所以，这时在这棵 XML 树上做一次查询扫描 DTD 树共需要 r1[n/s]次 I/O，扫描文档树中的 r1条路径需[r1d/r/s]次 I/O，所以共需 I/O 次数为 r1[n/s]＋[r1d/r/s]；通常 DTD 的规模远远小于 XML 文档树的规模，从以上分析，容易看出，利用 DTD 模式查询 XML 树与直接查询 XML 文档树相比较，通常会大大地减少 I/O 次数，并且 r1越小，即根据 DTD 树在扫描 XML 文档树时去掉的分支越多，前者的方法会更有效；而本篇文章中提出的方法，在扫描一遍 DTD 树得到真路径时，I/O 次数为[n/s]，得到的真路径数组中只记着真路径中经过的结点的地址，所以占很少的空间，如果经常有同样的查询(指能用到找出的真路径)，可以让真路径数组常驻内存，可直接用。设真路径中结点有 trn 个块，因为它们只为 DTD 树中的部分结点，所以 trn＜＝n，通常情况下，trn≪n，扫描真路径时，是与数组中的有用的真路径的第几个结点比的，所以可直接从真路径数组中直接到外存中找到所需 DTD 树中结点的地址，可直接将其取到内存中，并且，往往不是每条真路径都有用，所以真正用到的真路径的结点还不是全从外存中取一遍，即在对 XML 树做查询的过程中并不是再全部访问一遍 DTD 树中真路径所经过的结点，而只是其中的一部分，所以查询过程中，扫描真路径的 I/O 次数要小于等于[trn/s]，所以据我们的方法对 XML 文档树做一次查询需要的 I/O 次数要小于等于[n/s]＋[trn/s]＋[r1d/r/s]，r1为在限界的条件下扫描的 XML 文档树中的路径条数，所以通常 r1＞2，并且 trn＜n，所以有：[n/s]＋[trn/s]＋[r1d/r/s]＜2[n/s]＋[r1d/r/s]＜r1[n/s]＋[r1d/r/s]，所以我们的方法更能减少 I/O 次数，更能提高效率。

参考文献

1 Structural Function Inlining Technique for Structurally Recursive XML Queries

2 clark J, DeRose S. XML Path Language(XPath) Version 1.0 w3c Recommendation. Nov. 1999. http://ww. w3. org/TR/1999/REC-xpath-19991116

3 Bacon D F, Graham S L, Sharp O J. Compiler transformations for high-performance computing. ACM Computing Surveys, Dem. 1994, 26(4): 345～420

4 Boag S, Chamberlin D, Fernandz M, Florescu D, Robie J, Simeon J, Stefanescu M. Xquery 1.0: An XML query language. Working Draft. http://www, w3. org/TR/2001/WD-xquery＝20011220. Dec. 2001

5 Buneman P, Fernandez M, Suciu D. UnQL: a query language and algebra for semistructured data based on structural recursion. The VLDB Journal, 2000, 9: 76～110

6 Chamberlin D, Fankhauser P, Marchiori M, Robie J. XML query use cases. Working Draft. http://www. w3. org/TR/2001/WD-xml query-use-cases-10011220. Dec. 2001

7 Fankhauser P, Fernandez M, Malhotra A, Rys M, Simeon J, Wadler P. The Xquery 1.0 formal semantics. Working Draft. http://www. w3. org/TR/2001/WD-query-semantics-20010607. June 2001