利用视图重写技术优化 XML 查询

郑永清　洪晓光

（山东大学计算机科学与技术学院　济南250061）

摘　要　规则路径查询，其长度是任意的，这就意味着对数据库的任意多次访问，这样的代价是很昂贵的。我们采用视图重写技术，通过对某些经常使用的路径查询定义视图，从而减少了对某些高频使用的路径查询的重复搜索，最终提高了查询效率。我们的视图不但可以对路径查询进行重写，而且还可以对树查询进行混合重写。我们设计了一个使用动态规划策略实现的视图重写算法。

关键词　XML，视图重写，分支查询，XMLDT，混合重写

Apply the Technology of Rewriting View to Optimize XML Query

ZHENG Yong-Qing　HONG Xiao-Guang

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

Abstract　With the advent of XML standard, XML is widely used in exchanging data on the Web. More and more people is taking up with optimizing XML query. In fact, the common feature of the query languages for semi-structure data is the use of regular path expressions to query the data, the length of the paths is arbitrary, and in other words, it means frequently accessing the database. In this paper, we propose a technology of rewriting view. Many queries are used repeatly, rewrting view can improve XML query efficiency by reducing the times of scaning the documents repeatly.

Keywords　XML, Rewriting view, Embranchment query, XMLDT, Mixed rewriting

1　介绍

由于 XML 半结构化的特点，它在表达诸如 Web 信息、网络信息等方面较之关系和面向对象的数据模型有着得天独厚的优势。随着 XML 应用的不断增多，对 XML 文档查询的挑战就不断增加。在这篇论文里，我们使用视图重写技术对查询进行了优化。

目前大多数针对于半结构化的查询语言都通过规则表达式（规则路径查询），向用户提供查询数据库的能力。在文[1]中讨论了对这种查询的优化问题。但是，作为具有半结构化特点的 XML 文档来说，仅有路径查询是远远不够的。比如，对于图1所示的一个显示超市商品的 XML 文档结构，顾客很有可能要求查询某种商品的名字和价格。在这里，我们增加了对树查询的处理能力。

使用 XML 来定义半结构化的数据是非常容易的。我们可以使用很多方法，比如 DTD，SCHEMA，或者不使用任何模式。图1 给出了一个 超市 XML 文档的例子，作为一个超市它所销售的商品多种多样，而且经常可能发生变化，所以为了减少元素的数量我们并不为每一种商品定义一个标签，而是对所有商品使用同样的 GOOD 标签，对于不同的商品可能会出现不同的子元素组合，但它们都应该包含一些必要的属性和元素比如说商品编码、名称、价格等。我们的文档结构就是基于这样的假设和考虑的。目前已出现一些查询语言支持对半结构化的 XML 文档的查询。图2给出了几个查询的树结构形式。

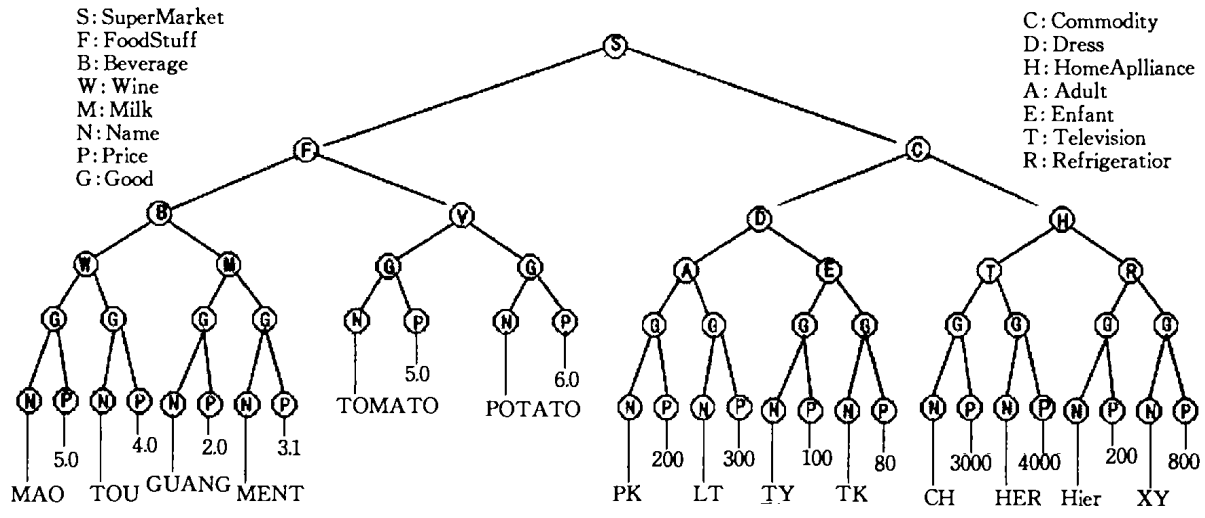

图1　XML 文档树

郑永清　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。

本质上查询是一个图模式，而查询的结果则是满足查询结构的数据图的子图。所以，对 XML 数据的查询就等价于在数据图中查找匹配查询结构的子结构。为了实现使用视图对查询的混合重写，我引入了文[2]中的 Structure-Encoded Sequence（结构化编码序列）和 Vist（虚拟后缀索引树）的概念，使用一个序列表达式来表示 XML 数据和 XML 查询。由此，XML 查询就等价于查找子序列的匹配问题。这样视图重写问题也就最终等价于查找子序列的匹配问题。Vist 的最大好处就是对于分支查询（例如 Q2）不必将分支结构分解成若干个子查询，然后使用这些子查询作为单独的路径在数据库中进行查询，然后将子查询的结果作连接从而得到最终结果，而我们使用它的唯一目的也就是给每一个元素分配一个唯一的标识符。最后，我们设计了一个动态规划的算法，实现了视图对查询的重写。

2 知识背景

我们把 XML 文档看成是一个树形结构，称之为 XML 文档树（XMLDT），如图1所示。其中的每一个节点都是 XML 文档的一个属性或者是元素。假设存在一个有限的字符集 Δ，我们称之为 XML 文档字符集。Δ^* 表示 Δ 上的所有组合形式。

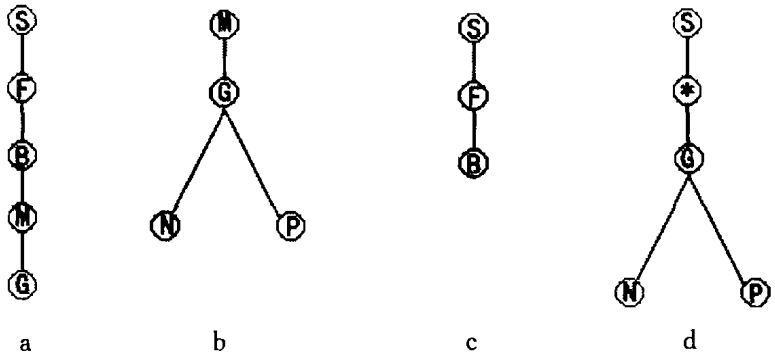

图2 XML 查询

一个 XML 查询就是在文档字符集 Δ 上的一个有限的或者是无限的表达式（如图2所示）。

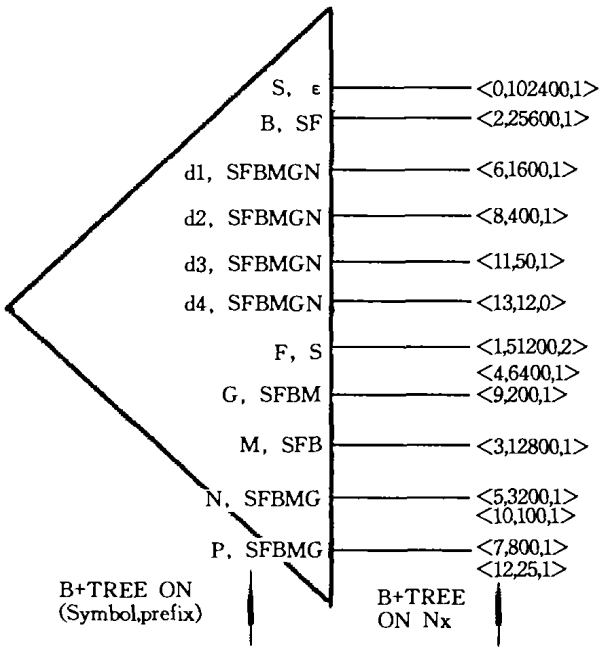

图3 索引结构

使用 STRUCTURE_ENCODED SEQUENCE 来编码 XML 文档的主要原因，是因为这样可以避免对树查询处理过程中不必要的连接操作。同时使用 STRUCTURE_ENCODED SEQUENCE 而不是使用节点和路径作为基本的查询单元，可以把整个查询结构当成一个整体，而不必将复杂的分支查询先分解成子路径查询，然后再将各个子查询的结果结合形成最终结果。这一特性使得我们可以使用视图对复杂的分支查询进行混合重写（混合重写可以同时写是指使用路径结构和树结构的视图对查询进行重写）。

为了加速在 STRUCTURED_ENCODED SEQUENCE 中的查询速度，我们使用 B+树作为索引结构，同时引入文[2]中的 Vist，一种动态分配元素标示 ID 的方法。它使得在插入或删除一个元素时不会影响其他的元素或属性的标示符，因此也就不必重新构建索引树。我们把图1中一部分作为 XML 文档，并给出了它的视图结构。

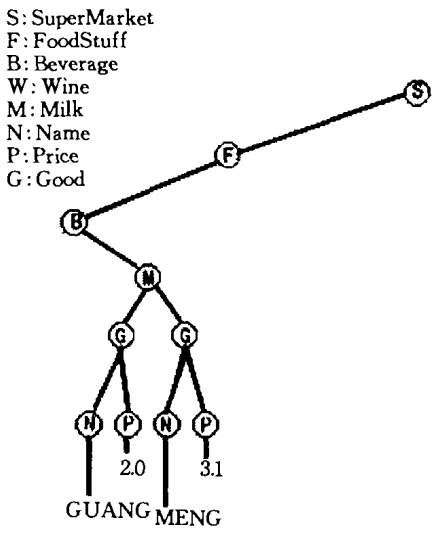

图4 文档结构

图3中 D={(S,ε)(F,S)(B,SF)(M,SFB)(G,SFBM)(N,SFBMG)(d1, SFBMGN) (P,SFBMG)(d2,SFBMGP) (G,SFBM)(N,SFBMG)(d3, SFBMGN) (P,SFBMG)(d4, SFBMGP)}（其中 di 表示字符数据）

首先为根节点分配一个空间，我们以102400为例，在实际应用中我们可以位置分配一个足够大的空间值 如2,32，这个值仅仅是一个标示而不必为之分配实际的空间。这样根节点的 Vist 编号为(0,102400,1)，其所有子孙节点的编号 N_x 都应该满足 $1 \leqslant N_x \leqslant 102400$。在图5中 l 表示祖先节点可能出现的不同元素的个数（一个预估计值），对于孩子的空间分布我们采用这样的策略，第一个孩子节点分配 scope/l 大小的空间，第二个节点分配 scope/l 平方，依此类推。对于比较大的

XML 文档，我们可以采取分块的策略避免出现过长的嵌套使得 scope/l 的 n 次方无法分配子空间。

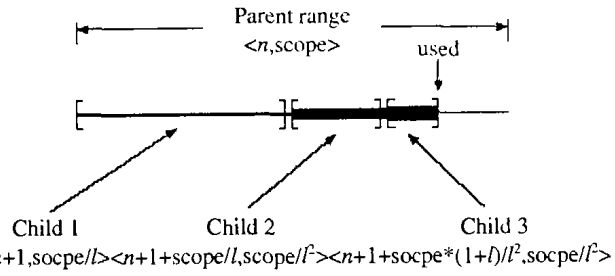

图5

3 算法实现

这一部分将介绍如何使用视图重写路径查询和树查询。下面，先介绍一下视图和重写的基本概念。首先定义一个视图集合 $V=\{V_1,V_2,V_3,\cdots,V_n\}$，其中 V_i 代表一个有限或者是无限的 XML 查询表达式。

与每一个视图定义对应着有一个视图的名字我们称之为 v_i，与之对应的集合定义为 $\Omega=\{v_1,v_2,v_3,\cdots,v_n\}$；对于每一个 $v_i\in\Omega$，我们定义 $def(v_i)=V_i$。

给定一个 XML 文档树(XMLDT)，如图1所示，它的每一个节点代表一个元素或者是属性。我们定义新的虚拟视图文档为 Vdoc，它由下面的公式导出：

$$U\{(a,V_i,[b_1,b_2,\cdots,b_n]):(a,[b_1,b_2,\cdots,b_n])\in Answer(V_i,XMLDT)\}\quad i\in(1,\cdots,n)$$

现在我们就可以通过查询视图而不是整个 XML 文档树来回答 XML 查询了。为此，我们有如下定义

定义1(重写) 对于 $Q'\in\Omega^*$，如果 $def(Q')$包含在 Q 中，称 Q' 是 Q 的一个重写；如果 $def(Q')=Q$，称这样的重写为准确的。

定理1 如过 Q' 是 Q 的一个重写，并且 Q' 是准确的，那么 $Answer(Q',XMLDT)=Answer(Q,XMLDT)$

定义2(部分重写) 如果 Q' 是 $\Omega\cup\Delta$ 上的一个混合表达式，并且 $def(Q')\subset Q$，称 Q' 是 Q 的一个部分重写；如果 $def(Q')=Q$，则称它是准确的。

定理2 $NSWERv(Q',Vdoc,XMLDT)=Answer(Q,XMLDT)$

由此我们引入一个偏序关系 $\leqslant_v^?$。Q_1、Q_2 是 Q 的重写，我们说 $Q_1\leqslant_v^?Q_2$，就意味着 Q_2 比 Q_1 包含着更多的视图符号。例如：$Q_1\leqslant_v^?Q_2\leqslant_v^?Q_3$。但是有时包含的视图多并不能代表它就是最好的。

为了得到最优的重写，我们定义了一个最大部分重写，记为 $MPRv(Q)$。我们规定，如果一个重写是 $MPRv(Q)$，则它的长度应该是所有重写中最短的，而且它要满足 $def(MPRv(Q))=Q$。在这里就应该是 Q_5。显然，$def(MPRv(Q))=Q$，也就是说 $MPRv(Q)$是准确的。

重写问题虽然在文[1]中都有讨论，但所讨论的重写都仅仅只是针对规则路径查询的。为了能够进行复杂的树查询，我们引入了第二部分提到的结构编码序列。下面的讨论中都将以这种结构为基础，而不再是树形结构。我们将 XML 文档、XML 查询、视图都转换成这种结构编码序列，如此一来这样的编码序列而不是节点/路径就成为 XML 查询的基本单元，这样就避免了将复杂的树查询分解成多个简单的路径查询然后再将各个结果合并的高昂代价。与此同时将 XML 文档、XML 查询、视图转换成结构编码序列使得查询的重写问题就转化为子序列的匹配问题。下面我们就给出一个使用动态规划策略求解结构编码序列上的最大部分重写问题的算法。

算法1

```
给定视图集合 V={V1,V2,V3,…,Vm}
输入:一个结构编码序列 Q
输出:MPRv(Q)
假定 Q={(a1,p1),(a1,p1),(a2,p2),…,(an,pn)}
令 f(k)表示 Q 的子序列 Q[1..k]的使用最少个 V 的元素表示的元素的个数
状态变量 k:k 表示当前位置处在查询 Q 的第几个元素上
状态转移方程:k=k+1
指标函数:f(k+1)=min{f(j)+1} :k=0..n-1, j=0..k
        f(k+1)=min{f(j)+1}这里 j 取遍所有可能使得 Q[j+1..k+1]使 V 中的元素,如果 Q[j+1..k+1]不是 V 中的元素,则将加上∞,例如 f(j)+∞
初始化令
        f(0)=0;k=1;
while(k<n)
{
   for(j=0;j<k;j++)
   {
      /*在视图集合中查找与 Q[j+1..k+1]匹配的视图*/
      if 找到
        d←f(j)+1
      break
   }
   f(k)=min{f(k-1)+1,f(j)+1}
}
输出 f(n),由 f(n)可以找出 MPRv(Q)
```

算法1可以在多项式时间内完成，但是在视图中进行子序列的匹配仍然很复杂。为了提高使用视图重写的效率，我们并不简单地将查询和视图转换成结构编码序列。我们在其中加入其叶节点的个数以及叶节点在查询或视图中相对位置信息。我们先证明结构编码序列有如下性质：

给定一组查询的结构编码序列，如果可以给出某一查询的长度和它所有的叶子节点信息，则可以唯一确定该查询。

对于如何区分路径查询和树查询，我们可以简单地查看其叶节点个数即可，叶节点数为1就是路径查询，>1则为树查询。下面得出算法1的优化算法。

算法2

```
给定视图集合 V={V1,V2,V3,…,Vm}
输入:一个结构编码序列 Q
输出:MPRv(Q)
假定 Q={(a1,p1),(a1,p1),(a2,p2),…,(an,pn)}
令 f(k)表示 Q 的子序列 Q[1..k]的使用最少个 V 的元素表示的元素的个数
状态变量 k:k 表示当前位置处在查询 Q 的第几个元素上
状态转移方程:k=k+1
指标函数:f(k+1)=min{f(j)+1} :k=0..n-1, j=0..k
    f(k+1)=min{f(j)+1}这里 j 取遍所有可能使得 Q[j+1..k+1]使 V 中的元素
初始化 令 f(0)=0;k=1;
  if Q 是一个路径查询
       while(k<=n)
       {
          1. 在索引结构中以当前元素和长度 k 为键进行查找,若存在则选择长度≤k 的最长的视图(其长度用 j 表示)替换查询中的子序列,保存 f(k-j)+1
          2. f(k)=min{f(k-1)+1,f(j)+1}/*f(k-1)为上一步的结果*/
       }
else
    while(k<n)
    {/*与路径查询处理方法相同*/}
    if(k==n)
    {
       1. 在索引结构中以当前元素和长度 k 为键进行查找,若存在则返回所有长度 Qx≤L≤k(Qx 为 Q 中记录的叶节点的个数)的视图→V[m](V[m]中的元素按叶节点个数的多少由大到小排序)
       2. While(I=0;I<=m;I++)
          {if(Vx[i]==Qx)/*Vx[i]表示视图中的叶节点的个数*/
              {
                 if(视图中的所有叶节点与查询子序列匹配)
                   {
```

(下转第124页)

算法，自动完成系统的不断优化。所以说这两种方法都具有复杂自适应系统的演化特征。但是MASM中系统的参数的选择对系统性能的优劣影响很大，目前只能通过实验-反馈-调整参数-实验-反馈-调整参数这样循环反复的过程来优化系统的性能，需要人为地判断优劣，从这一层意义上来说，LCS具有更强的适应性。

4. 知识表示：LCS的本质是学习一些规则，这些规则最终用于行动选择。从最终的结果来看，LCS能够概括总结行动选择的规律或本质，并且以直观的描述表现出来（通过规则），而MASM采用网络式结构，系统的行动选择完全体现在网络结点之间的连接和系统参数上，系统的整体性行动选择规律不能概括、总结出来。从这一点上来看，LCS优于MASM。

5. 最优性：MASM不能保证行动选择的全局最优，而且关于MASM的理论研究比较难以入手。XCS方法可以保证系统找到最优解——系统找到最少的分类器形成问题的解[16]。不过，XCS所找到的最优解是针对全局的，但是，在某个环境状态下，系统采用分类器所建议的行动是否就是该环境状态下的最佳行动，XCS并不能保证。这时，XCS能否保证选择最佳行动，需要其首先在学习过程中遍历所有的环境状态和行动组合，而这一点通常是做不到的。

6. 可扩展性：XCS结构之上，可以包涵自适应主体的认知结构[16]，比如说规划。而MASM只能通过调整系统参数，从而在规划和反应式之间进行权衡。也就是说，LCS是一个通用的行动选择方法，而系统的结构可以自行设计，系统是反应式还是慎思式体现在设计者的设计方法中，而MASM网络结构本身蕴含了这些特点。

7. 通用性：LCS方法不只是行动选择方法，还可以用于其他领域，比如数据挖掘、问题求解等。而MASM的则局限于行动选择领域。

总结 本文介绍了两种典型的行动选择理论MASM和LCS，并且进一步对比分析，讨论了它们的优缺点。从我们的分析中可以总结出，最根本上，两种方法的区别体现在学习上。抛开学习功能，LCS在系统的适应性、知识表示、可扩展性、通用性方面都优于MASM。所以，MASM的应用具有很大的局限性，目前的研究，MASM只是专门应用于动物的行动选择问题，而LCS则应用领域广泛，比如说问题求解、数据挖掘、机器人领域。为了改进MASM的缺陷，拓展MASM的应用，笔者认为MASM首先应该解决的问题是全局参数如何选取的问题，只有这个问题解决了，MASM才可能真正地便于应用。这个问题可以从两个方面着手：加入学习的功能，或者从模型论的角度给出全局参数间的制约关系。尽管LCS具有很好的特性，目前对LCS的很多研究还是从"小问题"（比如网格世界）研究LCS本身的特性，为了检验LCS在复杂行动选择问题环境中的真正效果，需要采用复杂模拟实验的方法检验其有效性。

参考文献

1 Brooks, Rodney A. A robust layered control system for a mobile robot. IEEE Journal of Robotics and Automation, 1986, 2: 14～23
2 Brooks, Rodney A. Intelligence without Representation. Artificial Intelligence, 1991, 47: 139～160
3 Brooks, Rodney A. Intelligence without Reason. In: Proc. of the 12th Intl. Joint Conf. on Artificial Intelligence (IJCAI-91), 1991
4 Brooks, Rodney A. Coherent Behavior from Many Adaptive Processes. In: Proc. of the Third Intl. Conf. on Simulation of Adaptive Behavior, 1994
5 Humphrys, Mark. W-learning: Competition among selfish Q-learners: [technical report no. 362]. University of Cambridge, Computer Laboratory, 1995
6 Humphrys, Mark. Towards self-organising Action Selection. In: Papers of the 14th Workshop of the UK Planning and Scheduling Special Interest Group. 1995
7 Humphrys, Mark. Action Selection in a hypothetical house robot: Using those RL numbers. In: Proc. of the First Intl. ICSC Symposia on Intelligent Industrial Automation (IIA-96) and Soft Computing (SOCO-96), 1996
8 Humphrys, Mark. Action Selection methods using Reinforcement Learning: [PhD thesis (first version)]. University of Cambridge, Computer Laboratory. 1996
9 Maes, Pattie . How To Do the Right Thing. Connection Science, 1989, 1: 291～323
10 Maes, Pattie. The dynamics of action selection. In: Proc. of the 11th Intl. Joint Conf. on Artificial Intelligence (IJCAI-89), 1989
11 Tyrrell, Toby. Computational Mechanisms for Action Selection: [PhD thesis]. University of Edinburgh, Centre for Cognitive Science, 1993
12 Tyrrell, Toby. The Use of Hiecharchies for Action Selection. University of Edinburgh, Centre for Cognitive Science, 1993
13 约翰·H. 霍兰. 隐秩序---适应性造就复杂性. 上海科技教育出版社, 2000
14 Wilson S W. Classifier Fitness Based on Accuracy. Evolutionary omputation, 1995, 3(2): 149～175
15 Wilson S W. Generalization in the XCS classifier system. 1998. 665～674
16 Kovacs T. Evolving Optimal Populations with XCS Classifier Systems. Master's thesis, School of Computer Science, University of Birmingham, Birmingham, U. K., 1996
17 Wilson S W. A critical review of classifier systems. In: Proc. of the Third Intl. Conf. on Genetic Algorithms. San Mateo, CA: Morgan Kaufmann, 1992. 244～255

（上接第82页）

```
                保存 f(k-j)+1; /* j 表示当前视图的长度 */
                break; /* 退出循环 */
                }
            }
        }
    3. f(k)=min{f(k-1)+1, f(j)+1}/* f(k-1)为上一步的结果 */
}
end if
输出 f(n)，由 f(n)可以找出 MPRv(Q)
```

算法2对于路径查询时间复杂度为 $n\log(m)$，n 为查询的长度，m 为视图的个数。

对于树查询其最坏时间复杂度为 $(n-1)\log(m)+n*m*\log(m)$，这种情况只能出现在视图集合中的所有视图的最后一个元素都与当前查询的最后一个元素相同，且这些视图中没有一个与当前查询匹配，这种情况是很少见的。

总结 在这篇论文里，我们讨论了使用视图对查询进行优化，而且与前面提到的文章不同的是我们增加了对树查询的处理能力，使用动态规划的方法实现了视图重写，而且我们的算法可以在PTIME完成。通过重写一方面可以减少搜索整个文档数的次数，另一方面对于结构相同的查询我们不必进行重复搜索，从而有效地提高了查询效率。

参考文献

1 Wang Haixun, et al. ViST: A Dynamic Index Method for Querying XMLData by Tree Structures, 2003
2 Abiteboul S, et al. Incremental Maintenance for Materialized Views over Semistructured Data
3 Alstrup S, Rauhe T. Improved labeling scheme for ancestor queries. In: Proc. ACM-SIAM Symposium on Discrete Algorithms (SODA), 2002