基于分层结构的前缀编码方案研究

徐 娟[1] 李战怀[1] 柯希林[2]

(西北工业大学计算机学院 西安 710072)[1] (武汉大学资源与环境科学学院 武汉 430079)[2]

摘 要 在分析现有XML文档树前缀编码[1-4]存储空间特性的基础上,提出了一种新的基于分层结构的前缀编码方法。本编码方案具有较小的平均编码长度,且编码长度不随XML文档中结点深度的增加而加大;给出了查询轴关系计算的算法,由于编码长度较小,在查询轴关系计算时比较次数较少,因此可以提高计算效率,加速查询过程。充分的理论分析和试验结果证明,基于分层结构的前缀编码方案是一种加速查询和节约编码存储空间的较好的编码方案。

关键词 XML,前缀编码,分层结构,子树

中图法分类号 TP311.13 **文献标识码** A

Novel Prefix Encoding Scheme Based on Layered Structure

XU Juan[1] LI Zhan-huai[1] KE Xi-lin[2]

(School of Computer Science and Technology, Northwestern Polytechnical University, Xi'an 710072, China)[1]

(School of Resource and Environment Science, Wuhan University, Wuhan 430079, China)[2]

Abstract Most of the XML query strategies are based on some prefix schemes[1-4]. By analyzing the current prefix schemes, we proposed a novel prefix encoding scheme with layered structure. The new prefix encoding scheme has relatively smaller mean coding length, and the code length is not increased with the depth increment of XML document. Another advantage this scheme brings out is, the query process was accelerated because component code comparisons for Xpath query axis computation become fewer with smaller code length. Extensive theoretic analysis and experimental results show that this scheme is a better one which accelerates query process and saves the store space for the codes.

Keywords XML, Prefix encoding, Layered structure, Subtree

XML已经成为互联网上数据表示和交换的标准,有效地存储XML数据并提供高效的XML数据查询,成为当今急需解决的问题。目前,大部分有关XML数据的索引和查询技术都是基于某种对XML文档树的编码方法。XML编码就是按照一定规则给XML树中每一个节点分配唯一的编码。通过编码,可以在不遍历XML文档树的前提下,直接判断两个节点之间的j结构关系。目前,前缀编码是主流的编码方式之一。前缀编码保存了元素的路径信息,任何元素的编码是其后裔元素编码的前缀,这样元素间祖先后裔关系的确定就等价于基于前缀字符串包含关系的判断。除结构化查询之外,前缀编码能够有效地支持对XML文档的更新,当对XML文档进行更新时,有着比区间编码更小的更新代价。另外,XML关键字查询是一个新的研究热点[5]。大多数XML关键字查询中求解多个检索关键字之间最小公共祖先(LCA)时,均使用前缀编码方案对XML文档进行编码。因此,对前缀编码的研究,特别是如何降低编码存储空间、提高查询效率,对于解决XML研究领域的许多问题很有意义。本文对现有的前缀编码方案进行分析比较。为了不降低编码性能和查询性能,同时大大降低编码长度,提出一种基于分层结构的前缀编码方案。理论分析和试验结果证明,这种方案在大规模静态XML数据库中具有非常大的优势。

1 相关研究

Tatarinov等人[1]使用Dewey ID对XML文档树进行编码,XML文档中的每一个元素表示为一个向量,其中,(1)根节点用空串标记;(2)当非根节点 u 是 s 的第 n 个孩子时,label(u)=label(s).n。label(s)表示 u 的父节点 s 的编码,整数 n 表示节点 u 的自编码(self_label),"."是连接父节点编码和自编码的分隔符。Dewey ID支持元素间有效的结构连接,即当且仅当label(u)是label(s)的前缀时,元素 u 是 s 的祖先。Dewey编码的结构关系判断条件简单,查询效率较高。

Cohen等人[2]提出了简单前缀(Simple Prefix,SP)编码方案,使用二进制字符串标记XML树中的节点,将树根编码为空串。根的第一个孩子label是0,第二个是10,第三个是110,第四个是1110,依此类推。SP编码的码长很大,需要很大的编码空间存储,因此编码效率和查询效率较低。

张剑妹等人[3]提出了一种适用于顺序XML树的前缀编码方法。在这种编码方法中,节点编码由自编码、前缀码和自编码的长度码3部分组成。每个节点的第一个孩子节点的自编码为"1",第二个孩子节点的自编码为"2"。当节点的顺序

到稿日期:2008-09-28 返修日期:2009-01-13 本文受国家自然科学基金项目(60720106001)资助。

徐 娟 博士生,主要研究方向为XML数据管理,E-mail:xuj@mail.nwpu.edu.cn;李战怀 博士,教授,博士生导师,主要研究方向为数据库理论与技术。

码全为“9”时，在自编码后补加“0”。这样，第九个孩子的自编码为“90”，第十个孩子的自编码为“91”，依此类推。每个节点编码都继承其父节点的自编码和前缀码作为其自编码的前缀。为了便于从节点编码中提取节点自编码，在每个节点编码的前面用固定长度的编码存储节点顺序码的长度。这种编码对顺序 XML 文档查询能够较好地支持，也具有较小的更新代价，但是其最大的不足是编码长度较大。

ORDPATH 编码[4]是一种扩展的 Dewey 编码，它使用奇数进行初始编码。当 XML 树进行更新时，它使用两个奇数之间的偶数连接一个奇数作为新的编码。ORDPATH 编码能够部分解决更新操作造成的重新编码问题。但是它浪费了一半编码空间，使得编码的规模很大。由于 ORDPATH 需要花费很多的时间基于奇数或者偶数来判断前缀的层次，它的查询效率低下，而且会出现编码溢出，不能完全解决 XML 文档的动态更新。对于 XML 数据库而言，为了解决动态更新问题，使用 ORDPATH 便大大降低了查询性能，使用这种编码方式的代价太大。

图 1 是上述前缀编码方案示例。上述编码在节点深度较大时，编码长度非常大，这样对系统的存储空间要求很高，尤其当深度较大节点具有较大扇出，所需存储空间迅速膨胀；同时，由于编码长度较大，在结构连接时，需要进行多次分量码的判断，降低了查询效率。

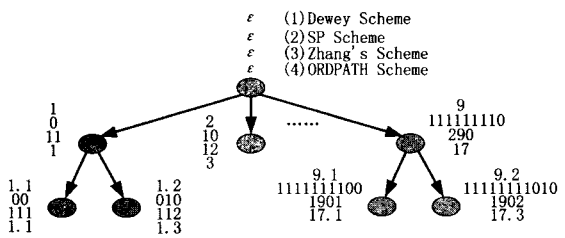

图 1 XML 文档树的各种前缀编码方案

2 平均编码长度分析

下面我们定量分析一个特定的 XML 文档树平均编码长度。考虑 XML 文档树 T 是深度为 D 的满 F 叉树($F\neq1$)，采用 Dewey 编码。对于第 l 层，存在以下关系：节点个数 F^{l-1}，编码长度($l-1$)。因此，

树 T 的结点总数：

$$\sum_{l=1}^{D}F^{l-1}$$

平均编码长度：

$$\sum_{l=1}^{D}(l-1)F^{l-1}/\sum_{l=1}^{D}F^{l-1}=\frac{[1-(D+1)F^{D}+DF^{D+1}]}{(1-F^{D})(1-F)}-1$$

平均编码长度与深度 D、扇出 F 的关系如图 2 所示。

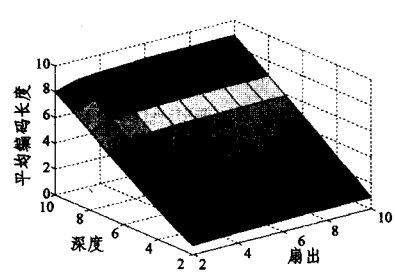

图 2 平均编码长度与深度 D、扇出 F 的关系

由图 2 可以看出，扇出的增大对平均编码长度影响不大。而随着深度的增大，平均编码长度迅速增大，需要更多的空间来存储节点编码，这是一项很大的系统开销。

3 分层前缀编码方案

由第 2 节分析可知，XML 文档树使用前缀编码时，随着其深度的增加，平均编码长度迅速增加。为了降低编码长度，我们采用分层结构编码的思想，即将 XML 文档树先根据深度划分成若干个子树，然后对各个子树的根进行编码，最后进行子树内的前缀编码。

在具体描述编码方案之前，给出几个相关定义。

定义 1 将 XML 文档树 T 按照深度间隔 d 划分子树，深度为 $kd+1$(k 为非负整数)的节点成为第 k 级子树的根节点，定义 d 为划分粒度。

定义 2 将 XML 文档树 T 按照划分粒度 d 划分子树，各级子树的根节点构成一个新的树 T'，对树 T' 中节点进行编号，定义该编号为子树根序号。

定义 3 对于子树 ST 中节点，根节点编码为该子树根节点对应的子树根序号，其它节点按照前缀编码规则进行编码，符合这种编码方案的编码定义为分层前缀编码。

定义 4 设子树 ST_k 中第 d(划分粒度)层节点的节点 u 是子树 ST_{k+1} 的根节点，节点 u 在子树 ST_k 中的编码为 $c_k(u)$，而作为子树 ST_{k+1} 的根节点对应的子树根序号为 $n(u)$，定义 $c_k(u)$ 是子树 ST_{k+1} 的根节点的子树根编码。

通过子树根编码和子树根序号的映射，可以将分层前缀编码还原成该节点在整个文档树中的前缀编码。

定义 5 节点在整个文档树中的前缀编码，定义为完全前缀编码。

采用分层编码方案得到的节点编码可以表示为 $SrN.prefix.self$，其中，SrN 为子树根序号，$prefix$ 为在子树中节点父节点的前缀编码，$self$ 为节点的自编码。

设划分粒度为 d，下面给出具体编码方案。

① 将 XML 文档树 T 按照划分粒度 d 划分子树，各级子树的根节点构成一个新的树 T'，对树 T' 中节点进行先序遍历，为了和分层前缀编码进行区别，用负的先序遍历序号表示各子树根的序号。

② 对子树 ST 进行 Dewey 编码，根节点的编码为子树根序号。

③ 对于子树 ST_{k+1} 的根节点，同时属于子树 ST_k 的一个深度为 d 的节点，将子树根序号和子树根编码用一线性表 LT 存储。存储格式为(子树根序号，子树根编码)。

根据上述的编码方案，设计递归编码算法如下。

Algorithm 1 EncodingSubTree(pn,SrN)

Input：pn 为当前节点，c(pn)为节点 pn 的编码，SrN 表示当前已使用的子树根序号

Output：以 pn 为根节点的子树中所有节点的前缀编码

Description：

```
1：if 节点 pn 没有孩子
2：     return SrN
3：else
4：     while 遍历节点 pn 所有的孩子
5：       pn 第 i 个孩子 vi
6：       c(vi)= c(pn).i
7：       if c(vi)的编码长度等于划分粒度 d
```

```
8:        节点 v_i 为新的子树的根,子树根序号为(SrN-1)
9:        将(SrN-1,C(v_i))保存到线形表 LT 中
10:       重新设置节点 v_i 的编码 C(v_i)=SrN-1
11:       SrN=SrN-1
12:     end
13:     return EncodingSubTree(v_i,SrN)
14:   end
15: end
```

Algorithm 2 Prefix_Encoding(T)

Input:T 是 XML 文档树

Output:文档树 T 中所有节点的前缀编码

Description:

1: 令树 T 的根节点 R 的子树根序号为-1,子树根编码为 ε,分层前缀编码为-1

2: 将(-1,ε) 保存到线形表 LT 中

3: EncodingSubTree(R,-1)

由算法 1、2 可以看出,基于分层结构的编码方案实现十分简单,每个节点的编码长度均小于 d。但是,需要一些额外的存储空间来保存子树根序号及其编码之间的映射关系。与总编码长度相比,这个开销很小。因此,可以认为平均编码长度近似为 d。

4 性能分析

本节将分析采用新的分层前缀编码方案对平均编码长度和位置关系计算复杂度的影响。

4.1 平均编码长度分析

与第 2 节假设相同,同样考虑满 F 叉树的 XML 文档树(扇出 F,深度 D,划分粒度 d)。

① 子树的级数 $STLevel=\lfloor(D-1)/(d-1)\rfloor+1$,其中,$\lfloor x\rfloor$表示不大于 x 的最大整数。

② 第 0 级子树有 1 个,第 1 级子树有 F^{d-1},依次类推,第 k 级子树有 $F^{k(d-1)}$,子树的个数为 $STNum=\sum_{k=0}^{STLevel-1}F^{k(d-1)}$。

③ 总编码长度计算。

第 0 级到第 $STLevel-2$ 级子树,对于任意子树,编码长度为 $\sum_{l=1}^{d-1}l\times F^{l-1}$,子树个数为 $\sum_{k=0}^{STLevel-2}F^{k(d-1)}$,则第 0 级到第 $STLevel-2$ 级子树总的编码长度为:

$$A=\sum_{k=0}^{STLevel-2}F^{k(d-1)}\times\sum_{l=1}^{d-1}l\times F^{l-1}$$

第 $STLevel-1$ 级子树的深度可能不足 $d-1$,需要单独考虑。该级子树的深度为 $d'=D-(STLevel-1)\times(d-1)$,对于该级任意子树,编码长度为 $\sum_{l=1}^{d'}l\times F^{l-1}$,子树个数为 $F^{(STLevel-1)(d-1)}$,则第 $STLevel-1$ 级子树总的编码长度为:

$$B=\sum_{l=1}^{D-(STLevel-1)\times(d-1)}l\times F^{(l-1)}\times F^{(STLevel-1)(d-1)}$$

总的编码长度为 $TotalLength=A+B$。

④ 线性表所需的额外空间。除树 T 的根节点外,其它子树根节点的前缀编码的长度为 d,存储子树根节点序号需要 1 个存储单元。因此,线性表 LT 所需空间为 $LTLength=(d+1)\times STNum$。这里,为了方便计算,存储树 T 根节点前缀编码 ε 也认为需要 d 个存储单元。

综上所述,平均编码长度为 $(TotalLength+LTLength)/\sum_{l=1}^{D}F^{l-1}$。采用分层前缀编码和 Dewey 编码对应的平均编码长度比较如图 3 所示。

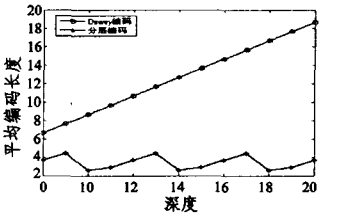

图 3 分层前缀编码算法的平均编码长度($F=4,d=5$)

由图 3 可以看出,采用传统 Dewey 编码,平均编码长度随着深度直线增长;而采用本文提出的分层编码方案,平均编码长度近似为 4($=d-1$),随深度增加变化不大。在对大深度、大数据量 XML 文档进行编码时,可以节约编码的存储空间。

4.2 查询轴的关系计算

采用本文提出的分层前缀编码方案,可以降低编码存储空间。同时,平均编码长度的减小,使得比较分量码的次数减小,这样可以提高位置关系计算效率。

使用前缀编码时,前缀关系的判断是一项重要的任务。这里我们先给出采用分层前缀编码时判断前缀关系的算法。

Algorithm 3 PrefixDecide(cu,cv,dLevel)

Input:cu 和 cv 为 2 个分层前缀编码

Output:当 cu 是 cv 的前缀,返回 TRUE;否则,返回 FALSE。dLevel 是返回值,当 *cu* 是 *cv* 的前缀时,该变量表示 2 编码的长度差。

Description:

```
1: 令 dLevel 为 0
2: 令 lu 和 stu 为 cu 的长度和第一个分量码
3: while TRUE
4:     令 lv 和 stv 为 cv 的长度和第一个分量码
5:     if stu < stv // case ①
6.       return FALSE
7:     else if stu > stv // case ②
8:       在线性表 LT 中查找 stv 对应的子树根编码 stc,
9:       并替换 cu 中第一分量码形成新的编码
10:    else // case ③ stu == stv
11.      if lu ≥ lv
12:        return FALSE
13:      end
14:      // 比较第 2~lu 个分量码
15:      for i=2:lu
16:        if cu 和 cv 的第 i 个分量不相等
17:          return FALSE
18:        end
19:      end
20:      令 dLevel 为 lv-lu
21:      return TRUE
22:    end
23:end
```

由于子树根节点序号采用负数先序编码方式,因此当 *cu* 是 *cv* 的前缀,*cv* 的第 1 个分量码不可能大于 *cu* 的第 1 个分量码。基于这个事实,可以迅速判断这种情况下的前缀关系,对应于算法 3 中的 case ①,这样可以提高判断速度。例如,*cu*=-3.1.2 对应的完全前缀编码为 1.1.1.1.1.2,*cv*=-2.2.3

对应的完全前缀编码为 1.1.1.2.3。如果采用算法 3，只需比较 1 次就能否定前缀关系。而采用传统的前缀编码前缀关系判断方法，需要比较 4 次才能否定前缀关系，这一点体现了本文编码方法的优势。

算法 3 中的 case ②表示 cu 和 cv 对应的 2 个节点存在 AD 或者 Following 关系，还不能直接判断前缀关系。因此，需要根据 cv 的子树根序号回溯线性表 LT，直至 cu 和 cv 第一个分量码满足大于关系（对应于 Following 关系），或者相等关系（对应于 AD 关系，即前缀关系成立）。最坏的情况需要回溯到根节点。而一般情况下，判断也可能很快结束，没必要比较完全前缀编码所有的分量码。

算法 3 中的 case ③表示 cu 和 cv 对应的 2 个节点具有相同的祖先节点，判断前缀编码关系至多比较 lu（小于划分粒度 d）次。例如，$cu=-10.2$ 对应的完全前缀编码为 1.1.1.1.1.1.1.1.2，$cv=-10.2.3$ 对应的完全前缀编码为 1.1.1.1.1.1.1.1.2.3。如果采用算法 2，只需比较 2 次就能确定前缀关系。而采用传统的前缀编码前缀关系判断方法，需要比较 8 次才能确定前缀关系，这种情况下大大减小比较次数。

借助于前缀关系判断算法，我们可以很方便地计算查询轴的关系。设节点 u 和节点 v 的分层前缀编码为 $c(u)$ 和 $c(v)$。

· 祖先-后裔关系（AD 关系）

将 $c(u)$ 和 $c(v)$ 带入算法 3 中，若 PrefixDecide 返回 TRUE，则表示节点 u 是节点 v 的祖先，满足祖先-后裔关系；否则，该关系不成立。

· 父子关系（PC 关系）

将 $c(u)$ 和 $c(v)$ 带入算法 3 中，若 PrefixDecide 返回 TRUE，且深度差 $dLevel$ 为 1，则表示节点 u 是节点 v 的父亲，满足父关系；否则，该关系不成立。

· 兄弟关系（Sibling 关系）

采用前缀编码方案，节点的兄弟关系判断十分简单。具体步骤可以描述如下：

① 比较 $c(u)$ 和 $c(v)$ 的长度 lu 和 lv 是否相同。若相同，继续；否则，兄弟关系不成立；

② 判断 $lu(=lv)$ 是否等于 1，即判断节点 u 和 v 是否为子树根节点。若 $lu=lv=1$，查找线性表 LT，得到节点 u 和 v 子树根节点编码，并替换 $c(u)$ 和 $c(v)$，重新计算长度 lu 和 lv；

③ 比较 $c(u)$ 和 $c(v)$ 的前 $lu-1(=lv-1)$ 个分量码是否相同，若相同，兄弟关系成立；否则，兄弟关系不成立。

可以看出，兄弟关系判断至多需要比较 $(d-2)$ 次，即可确定。

· 之后关系（Following 关系）

由前面分析，子树根节点序号采用负数先序编码方式，之后关系的判断可以先根据子树根序号判断是否在同一子树、满足祖先-后裔或之后关系，然后针对不同情况进行计算。具体计算步骤描述如下：

① 比较 $c(u)$ 和 $c(v)$ 的子树根序号 stu 和 stv。若 $stu \geqslant stv$，满足在同一子树、祖先-后裔或之后关系，继续；否则，之后关系不成立。

② 分 $stu=stv$ 和 $stu>stv$ 进行判断：

A）若 $stu=stv$，表示节点 u 和 v 在同一子树下，$c(u)$ 和 $c(v)$ 的长度 lu 和 lv，令 $l=\min(lu,lv)$，从左到右依次比较第 2 到 $l-1$ 个分量码。如果其中 $c(u)$ 和 $c(v)$ 的第 $i(2\leqslant i\leqslant l)$ 个 cu_i、cv_i 满足 $cu_i<cv_i$，且前 $i-1$ 个分量码相同，则之后关系满足；否则，该关系不满足。

B）若 $stu>stv$，将 $c(u)$ 和 $c(v)$ 带入算法 3 中，若 PrefixDecide 返回 TRUE，则表示节点 u 是节点 v 的祖先，不满足之后关系；否则，之后关系成立。

可以看出，第①步只需比较 1 次就可以排除很多节点。在 A）中至多需要比较 $d-1$ 次就能确定关系。在 B）中，采用分层编码时的前缀关系的判断比较有效。因此，与采用传统前缀编码算法相比，本计算过程显示出较高的效率，能够快速地判断之后关系。

· 之前关系

与之后关系计算类似，之前关系的判断具体的计算步骤可以描述如下：

① 比较 $c(u)$ 和 $c(v)$ 的子树根序号 stu 和 stv。若 $stv \leqslant stu$，满足在同一子树、祖先-后裔或之前关系，继续；否则，之后关系不成立；

② 分 $stv=stu$ 和 $stv>stu$ 进行判断

A）若 $stv=stu$，表示节点 u 和 v 在同一子树下，$c(u)$ 和 $c(v)$ 的长度 lu 和 lv，令 $l=\min(lu,lv)$，从左到右依次比较 l 个分量码。如果其中 $c(u)$ 和 $c(v)$ 的第 $i(2\leqslant i\leqslant l)$ 个 cu_i、cv_i 满足 $cu_i>cv_i$，且前 $i-1$ 个分量码相同，则之前关系满足；否则，该关系不满足；

B）若 $stv>stu$，将 $c(v)$ 和 $c(u)$ 带入算法 3 中，若 PrefixDecide 返回 TRUE，则表示节点 v 是节点 u 的祖先，不满足之后关系；否则，之后关系成立。

从上面的各种关系计算可见，与采用传统前缀编码算法相比，采用分层前缀编码，能够减少比较次数，高效、快速计算查询轴关系。

5 仿真试验结果

这一节我们研究本文提出的前缀编码方案的编码长度、查询效率等性能，并将它们与文献[1,4]提出的编码方案进行比较。由于 Zhang 编码[3]的编码方式不适合具有大扇出的 XML 文档，因此本节的试验结果不与 Zhang 编码的性能进行比较。本文的测试程序是基于 Visual C++ 6.0 实现的，运行硬件参数为 Pentium D CPU，主频为 3.4 GHz，512M DDR 内存，160 G 的 SATA 硬盘，操作系统为 Windows XP Professional。仿真的数据集采用通用数据集 Treebank[6] 和 XMark[7]，相关参数如表 1 所列。对于本文提出的编码方案，根据参数集的特点，划分粒度选择 $d=5$。

表 1 测试数据集

DataSet	Size of File	Max / average fanout	Max / average depth	Total of nodes
Treebank	84.065 MB	56,385/2.33	36/7.44	2,437,667
XMark1	11.597 MB	2,550/17.99	12/5.22	206,130
XMark2	23.265 MB	5,100/18.37	12/5.23	409,511

5.1 编码长度和时间

图 4、图 5 显示了采用不同编码方案对数据集 Treebank，XMark1 和 XMark2 进行编码时的平均编码长度和编码时间。从图 4 可以看出，采用本文提出的分层编码方案，平均编码长度较小，不随 XML 文档规模增加而变大。而对于 Dewey 编码和 ORDPATH 编码，平均编码长度较大，并随着 XML 文档树深度增加而变大。由图 5 可以看出，Dewey 编码和分层编码这 2 种方案，由于涉及到的操作都是简单的基本运算，编码时间基本相同；而 ORDPATH 编码由于编码时节

点之间的层次关系判断比较复杂，因此相应的编码时间也较长，编码时间性能较差。

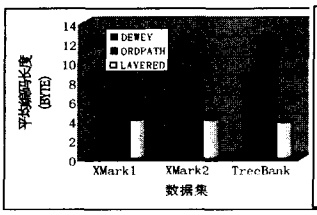

图 4 不同编码方案对应的平均编码长度

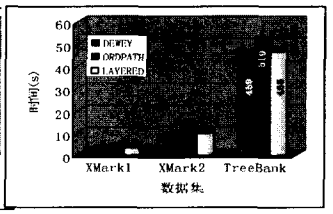

图 5 不同编码方案对应的编码时间

5.2 查询效率

为了测试各种编码方案的查询性能，设计查询正则路径表达式[8]（其中，Q3 是包含顺序的查询），如表 2 所列。图 6 显示了不同编码方案对查询 Q1－Q3 的响应时间。由图 6 可以看出，由于 Dewey 编码和本文提出的分层编码这 2 种方案，在比较两个编码的大小时层次判断简单，并且分量编码比较只是简单的数值比较，查询响应时间较快。但是由于采用分层编码，在关系判断时，可以较早确定结果，因此查询响应时间稍低于 Dewey 编码对应的查询时间。而 Ordpath 编码方案，编码比较时，同样由于层次关系判断比较复杂，且分量编码比较时，可能需要进行多次数值比较，因此相应的查询响应时间较长。

表 2 在数据集 XMark2 上的查询

	Queries	# of nodes retrieved
Q1	//item[location]/description//keyword	2,729
Q2	//people//person[. //address/zipcode]/profile/education	287
Q3	//open_auctions/open_auction/bidder[2]	887

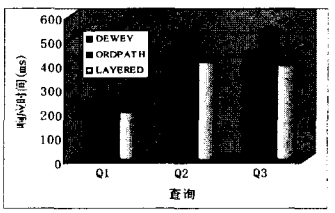

图 6 不同编码方案对应的查询响应时间

结束语 为了有效地支持以路径表达式为核心的 XML 查询，需要能够快速判断 XML 文档中元素间的相对关系，前缀编码是一种主流的 XML 文档编码方法。本文分析了现有的几种 XML 文档前缀编码方法具有编码长度随节点深度迅速增加的不足，提出了一种基于分层结构的前缀编码方式。将 XML 文档树按照划分粒度抽象成若干个子树，首先对子树根构成的新树进行先序遍历，给子树根赋予唯一序号，然后对子树内部进行前缀编码。同时，对子树根序号和根编码建立一个线性表，以还原该节点在整个文档树中的前缀编码。理论分析和仿真结果表明，本编码方案具有较小的平均长度，并且不随 XML 文档规模增大而变化；由于编码长度较小，在查询轴关系计算时比较次数较少，可以提高计算效率，加速查询过程。因此，本编码方法在不损失查询速度的前提下，可以较好地解决编码长度随节点深度增加迅速增大的问题，是一种较好的前缀编码。

参 考 文 献

[1] Tatarinov I, Viglas S, Beyer K S, et al. Storing and querying ordered XML using a relational database system[C]// Proceedings of SIGMOD. 2002:204-215

[2] Cohen E, Kaplan H, Milo T. Labeling Dynamic XML Tree[C]// Proceedings of the 21st ACM Symposium on Principles of Database Systems. Madison, Wisconsin, USA, 2002:271-281

[3] 张剑妹，陶世群. 一种适用于顺序 XML 树的前缀编码方法[J]. 计算机应用，2005，25(12)：2879-2881

[4] O'Neil P, O'Neil E, Pal S, et al. ORDPATHs: Insert-friendly XML node labels[C]// Proceedings of SIGMOD. 2004:903-908

[5] Xu Yu, Papakonstantinou Y. Efficient Keyword Search for Smallest LCAs in XML Databases[C]// Proceedings of SIGMOD. 2005:527-538

[6] University of Washington XML Repository[OL]. http://www.cs.washington.edu/research/xmldatasets/

[7] The XML Benchmark Project[OL]. http://www.xml-benchmark.org

[8] Qin L, Yu J X, Ding B. Twiglist: Make twig pattern matching fast[C]// Proceedings of DASFAA. 2007, 4443:850-862

(上接第 116 页)

[6] Jézéquel J-M, Hussmann H. A Relational Approach to Defining Transformations in a Metamodel[C]// Cook S, ed. UML 2002, LNCS 2460. 2002:243-258

[7] 王学斌，王怀民，吴泉源，等. 一种模型转换的编织框架[J]. 软件学报，2006，17(6)：1423-1435

[8] 褚华，李青山，陈平，等. 一种基于 UML 序列图的状态图合成方法[J]. 系统工程与电子技术，2005，27(3)

[9] 刘辉，麻志毅，邵维忠. 模型转换中特性保持的描述与验证[J]. 软件学报，2007，18(10)：2369-2379

[10] 申小军，金益民. 状态图到扩展有限状态机转换技术研究与实现[J]. 现代电子技术，2004(12)

[11] Li Liu-ying, Qi Zhi-chang. Test selection from UML statecharts technology of object-oriented languages and systems [C]// 1999, TOOLS 31, Proceedings. Sept. 1999:2732279

[12] Bogdanov K, Holcombe M, Singh H. Automated test set generation for statecharts[C]// Volume 1641 of Lecture Notes in Computer Science. Springer Verlag, 1999:1072121

[13] 统一建模语言状态图的测试用例生成方法[J]. 计算机仿真，2007，24(8)

[14] 李留英，王戟，齐治昌，等. UML statecharts 的测试用例生成方法[J]. 计算机研究与发，2001(6)

[15] Chow T. Testing software designs modeled by finite-state machines[J]. IEEE Transactions on Software Engineering, 1978, SE-4(3):178-187

[16] Pimont S, Rault J C. A software reliability assessment based on a structural and behavioral analysis of programs[C]// Proceedings of the 2nd International Conference on Software Engineering(ICSE 1976). San Francisco, CA, USA, October 1976:486-491

[17] Bergstein P L. Object-Preserving class transformation [C]// Proc. of the ACM SIGPLAN Conf. on Object—oriented Programming, Systems, Languages, and Applications. New York: ACM Press, 1991:299-311. http://portal.acm.orgcitationcfm?id=117977&coll=portal&dl=ACM

[18] 缪淮扣，占学德，刘玲. 基于 UMLStatecharts 的测试用例生成[J]. 小型微型计算机系统，2005，26(4)

[19] 史耀馨，崔萌，李宣东，等. 基于 MDA 的 UML 模型转换技术——从顺序图到状态图[J]. 计算机工程与应用，2004，40(13)

[20] Mens T. A survey of software refactoring[J]. IEEE Trans. on Software Engineering, 2004, 30(2):126-139