# Native XML 数据库的文档编码机制研究

张 鹏[1],冯建华[1],韩秀峰[2]
ZHANG Peng[1],FENG Jian-hua[1],HAN Xiu-feng[2]

1.清华大学 计算机科学与技术系,北京 100084
2.首都经济贸易大学 金融学院,北京 100070
1.Department of Computer Science & Technology,Tsinghua University,Beijing 100084,China
2.Financial College,Capital University of Economics and Business,Beijing 100070,China
E-mail:zhangp@tsinghua.org.cn

**ZHANG Peng,FENG Jian-hua,HAN Xiu-feng.New XML document coding scheme in Native XML database.Computer Engineering and Applications,2008,44(12):147-150.**

**Abstract:** Structure join operation is the main solution to native XML database query.Structure join operation depends on XML documents' coding,in order to quickly determinate ancestor-descendant relationships between the nodes of the XML document tree.In this paper,a new coding scheme is proposed,which is named as prefix division (PDIV) coding scheme.The scheme is easy to realize and only one positive integer is needed to express the position of the node in XML tree.The scheme may identify the ancestor-descendant relationships in constant time-bounding.It also supports XML document update.In the scheme,the length of the code is short and it is about $o(\ln(n))$.

**Key words:** Native XML database;coding scheme;XML query

**摘 要:** Native XML 数据库快速查询的实现,可以采用基于 XML 文档编码的结构连接算法。而结构连接算法的实现需要对 XML 文档进行编码,以便于快速判断 XML 文档树结点之间的祖先后裔关系。在对现有编码机制进行综述的前提下,提出一种新的 XML 文档编码机制——前缀整除编码(PDIV)机制。该机制编码形式简单,只需要一个正整数即可充分表示结点在 XML 文档树中的位置信息;可以实现祖先后裔关系的快速查询;支持 XML 文档的更新操作;编码长度较短,编码长度约为 $o(\ln(n))$。

**关键词:** Native XML 数据库;编码机制;XML 查询
**文章编号:** 1002-8331(2008)12-0147-04 **文献标识码:** A **中图分类号:** TP311

## 1 引言

互联网是人类历史上最伟大的发明之一。因特网的出现和发展彻底地改变了人们的生活方式和思维方式。相应地,与之相关的技术也得到飞速的发展。HTML(Hyper Text Markup Language,超文本标记语言)技术和 SGML(Standard Generalized Markup Language,标准通用标记语言)技术是伴随着因特网产生的重要技术。虽然 HTML 功能非常简单,但是其无法处理大量的结构化信息。虽然 SGML 功能完善,但是其非常复杂。HTML 技术和 SGML 技术所存在的上述不足使得人们重新思索,意图解决上述问题。基于此,XML 技术得以出现,其旨在克服 HTML 技术和 SGML 技术所存在的不足。XML 的全称是 eXtensible Markup Language(扩展标记语言),它是一种专门为因特网所设计的标记语言。XML 的重点是管理信息的数据本身,数据的显示交给其他技术解决。可以说,XML 技术从诞生之日就预示了它辉煌的未来,特别是随着近些年 Web Service 的蓬勃发展,XML 越来越多地活跃在数据交换和存储领域。

大量 XML 文档的出现促进了 Native XML 数据库的研究,Native XML 数据库已经成为当前数据库理论研究的前沿之一。Native XML 数据库是以自然的方式处理 XML 数据,以 XML 文档作为基本的逻辑存储单位,针对 XML 的数据存储和查询特点专门设计适用的数据模型和处理方法[1]。由于查询是数据库最为频繁的操作,因此在 Native XML 数据库的研究中,XML 查询处理器是研究的热点。XML 查询处理器的主要功能在于查询分解、查询优化和查询执行等,主要目的在于查询求解。目前 Native XML 数据库的查询求解方法包括:基于 XML 文档索引或者结构索引的导航遍历算法;基于 XML 文档编码的结构连接算法;基于 XML 文档序列表示的序列匹配算法等。在上述方法中,利用结点编码的方法是主流技术之一。因此,XML 文档的结点编码机制是实现 Native XML 数据库高效查询的重要条件,对 XML 文档结点编码机制的研究具有非常重要的意义。

本文的内容结构如下:第 2 章对 XML 文档编码机制的相

**基金项目:** 浙江省自然科学基金(the Natural Science Foundation of Zhejiang Province of China under Grant No.Y105230);清华大学基础研究基金(JCqn2005022)。

**作者简介:** 张鹏(1981-),男,硕士,国家知识产权局审查员,主要研究领域为 XML 数据库,Ad hoc 网络和移动通信;冯建华(1967-),男,博士,副教授,主要研究领域为数据库和信息处理;韩秀峰(1981-),女,硕士生,主要研究领域为金融数据分析和信息处理。

**收稿日期:** 2007-08-14 **修回日期:** 2007-11-13

关研究进行综述;第 3 章详细描述本文提出的 XML 文档编码机制(PDIV 编码),并且对编码性质进行分析;第 4 章对本文所提出的 PDIV 编码与其他编码方法进行性能比较;第 5 章进行总结。

## 2 相关研究

为了能高效地查询求解,已经提出多种 XML 文档编码机制。所谓 XML 文档编码机制,是指为 XML 文档树的每个结点赋予一个特定的编码,以便于能够通过编码直接判断 XML 文档树中的结点之间的祖先后裔关系,而不需要对原 XML 文档进行导航遍历。依据编码的原理,可以分为基于区间的编码机制和基于路径的编码机制。基于区间的编码机制将每个结点编码为一个区间,而基于路径的编码机制则是依据结点的路径将每个结点编码为一个数字。

### 2.1 基于区间的编码机制

有关文档编码的早期研究多集中在区间(range-based)编码法上。其主要思想在于为 XML 文档树中的每一个结点赋予一对正整数 *begin* 和 *end*,编码形式为[*begin*,*end*]。所有结点的编码区间都是其祖先结点的编码区间的子集。也就是说,结点 *a* 是结点 *d* 的祖先的充分必要条件是 $a.begin \leqslant d.begin$ 并且 $a.end \geqslant d.end$。

Li-Moon(XISS)编码机制[2]。其主要思想在于:XML 文档树 *T* 中的每个结点被赋予二元组<*order*,*size*>,*order* 是结点的扩展先序遍历序号,*size* 是结点的后裔范围。因此,文档树中结点 *a* 和结点 *d* 存在祖先-后裔关系,当且仅当 $a.order<d.order$ 且 $d.order+d.size \leqslant a.order+a.size$。另外,每个结点被赋予一个大于或者等于零的值 *depth*,其表示该结点在文档树中的层次。

Zhang 编码机制[3]。其主要思想在于:XML 文档树 *T* 中的每个结点被赋予二元组<*begin*,*end*>,*begin* 是结点在 XML 文档中的开始位置,*end* 是结点在 XML 文档中的结束位置。因此,文档树中结点 *a* 和结点 *d* 存在祖先-后裔关系,当且仅当 $a.begin<a.end$ 且 $a.end>d.end$。另外,每个结点被赋予一个大于或者等于零的值 *depth*,其表示该结点在文档树中的层次。

Wan(ApproXQL)编码机制[4]。其主要思想在于:XML 文档树 *T* 中的每个结点被赋予二元组<*order*,*maxOrder*>,*order* 是结点的扩展先序遍历序号,*maxOrder* 是结点的后裔编码中扩展先序遍历序号的最大值。因此,文档树中结点 *a* 和结点 *d* 存在祖先-后裔关系,当且仅当 $a.order<d.order$ 且 $d.maxOrder \leqslant a.maxOrder$。

Dietz 编码机制[5]。其主要思想在于:XML 文档树 *T* 中的每个结点被赋予二元组<*pre*,*post*>,*pre* 是结点的先序遍历序号,*post* 是结点的后序遍历序号。因此,文档树中结点 *a* 和结点 *d* 存在祖先-后裔关系,当且仅当 $a.pre<d.pre$ 且 $d.post<a.post$。

基于预留算法的编码机制[6]。其主要思想在于:XML 文档树 *T* 中的每个结点被赋予二元组<*order*,*size*>,*order* 是结点的扩展先序遍历序号,*size* 是预留的编码空间。基于数据模式和更新模式进行一定运算,从而获得预留的编码空间。

区间编码机制根据区间判断结点关系,其优点是编码简单,结点编码的平均长度为 $o(\log(n))$;缺点是使用非等值比较进行结点关系判断,并且需要单独提供层次信息。

### 2.2 位向量编码机制

位向量编码机制[7]的主要思想在于:XML 文档树 *T* 中的每个结点与一个 *n* 位向量 *v* 对应,*n* 是树 *T* 中结点的数目。在向量的某个位置 *i* 上值"1"惟一标识了第 *i* 个结点,并且每一个后裔结点继承了其祖先的所有"1"位。若结点 *a* 是 *d* 的祖先,当且仅当 $a.v \ \& \ d.v=a.v$;若结点 *d* 是 *a* 的后裔,当且仅当 $d.v|a.v=d.v$ 位向量编码将结点与 *n* 位向量对应,其优点是容易判断结点间的层次关系;缺点是不支持更新操作,编码空间较大。

### 2.3 PBiTree 编码机制

PBiTree 编码机制[8]的主要思想在于:将 XML 文档树 *T* 转化为完全二叉树,然后按照"自底向上,自左至右"的顺序为每个结点进行编号。PBiTree 编码按照结点在文档树中的位置进行编码,其优点在于编码简单,结点编码的平均长度为 $o(n)$;并且使用等值比较进行结点关系判断。缺点是不支持更新操作,需要单独提供层次信息。

### 2.4 Dewey 编码机制(前缀编码机制)

Dewey 编码机制[9]的主要思想在于:将 XML 文档树中父结点的编码直接作为其子结点编码的前缀,也称为前缀编码。因此,文档树中结点 *a* 和结点 *d* 存在祖先-后裔关系,当且仅当 *a.code* 是 *d.code* 的前缀。Dewey 编码将 XML 文档树中父结点的编码直接作为其子结点编码的前缀,其优点是编码简单,结点编码的平均长度为 $o(n)$;支持更新操作。缺点是前缀操作比算术操作运算速度慢,因此该编码方法效率较低。

### 2.5 素数编码机制

素数编码机制[10]的主要思想在于:在自底向上素数编码方法中,将每个 XML 文档树的叶结点按顺序赋予不同素数,将父结点编码为子结点编码的乘积;在自顶向下素数编码方法中,将根结点编码为 1,将每个结点编码为来自父亲的"父标记(parent-label)"和来自素数列的"个人标记(self-label)"的乘积。另外,当文档树过高的时候,可以对树进行"分解(tree decomposition)"。在自底向上素数编码方法中,文档树中结点 *a* 和结点 *d* 存在祖先-后裔关系,当且仅当 $(a.code) \bmod (d.code)==0$;在自顶向下素数编码方法中,文档树中结点 *a* 和结点 *d* 存在祖先-后裔关系,当且仅当 $(d.code) \bmod (a.code)==0$。素数编码采用整除运算判断祖先-后裔关系,其突出的优点是较好地支持更新操作,并且对于 XML 树的扇出(fan-out)不敏感;整除判断运算速度较快。其缺点是编码长度较大。

### 2.6 BBT 编码机制

BBT 编码机制[11]的主要思想在于:将根结点编码为 1,将某结点的最左儿子结点编码为其父结点编码的 2 倍,其他儿子结点的编码为其最近左兄弟结点编码乘以 2 再加 1。该文献还提出了支持更新的 BBT 编码方法,其主要思想在于:提供额外的信息来表示一个结点在其兄弟结点中的位置序号,而不能利用 BBT 编码本身所包含的兄弟结点的位置信息。另外,为了节省存储空间,BBT 编码可以采用分块存储的方法。BBT 编码机制的优点是祖先-后裔关系的判断采用计算机常用的移位操作,具有很高的运算速度;较好地支持更新操作,仅在删除非最左叶结点或者添加结点到非最左位置时会导致同一层兄弟结点之间的顺序错误,为了更好地支持更新,可以增加分量记录结点在其所在层中的顺序。其存在的问题在于编码长度较长,对于结点数为 *n* 的文档树而言,BBT 编码的长度为 $o(n)$。

综合上述分析,区间编码机制和位向量编码机制对于更新操作支持较差,运算效率较低。另外,一项理想的编码机制应当满足如下技术指标:支持祖先/后裔关系的快速查询;支持更新;编码尽可能短。因为数据库查询是数据库中频率最高的操

作,所以第一项指标最为重要;对于第三项指标,一般可以通过压缩技术和分块技术加以解决,因此重要性最低。

## 3　前缀整除编码机制

鉴于现有编码机制存在的上述缺陷,本文提出一种基于前缀整除关系的编码机制,并简称为 PDIV(prefix division)编码机制,该机制具有如下优点:第一,编码形式简单,只需要一个正整数即可充分表示结点在 XML 文档树中的位置信息;第二,可以实现祖先后裔关系的快速查询;第三,支持 XML 文档的更新操作;第四,编码长度较短,编码长度约为 $o(\ln(n))$。

### 3.1　前缀整除编码原理

前缀整除编码机制的核心思想是:给 XML 文档树 $T$ 中的每个结点赋予一个大于等于 1 的正整数 *code*,结点的编码形式为<*code*>,它能完全表示结点在树 $T$ 中的位置信息。

#### 3.1.1　基本定义

**定义** 1　前缀整除关系与前缀约数。两个正整数 $x$、$y$ 之间存在前缀整除关系当且仅当:

(1)$x|y$ 且 $x\neq y$;

(2)若 $z=y/x$,则 $z$ 的任意素数因子 $a$ 和 $x$ 的任意素数因子 $b$ 之间存在 $a\geqslant b$ 的关系。

此时,$x$、$y$ 之间存在前缀整除关系,$x$ 称为 $y$ 的前缀约数。

#### 3.1.2　前缀整除编码机制(PDIV 编码机制)

设结点的编码为 *code*,则:

**步骤** 1　定义素数数列$\{p_n\}$,即 $P_1=2,P_2=3,\cdots$;

**步骤** 2　给 XML 文档树 $T$ 的根结点的编码 *code* 赋值为 1;

**步骤** 3　给 XML 文档树 $T$ 的第一层子结点的编码 *code* 自左至右赋值为 $P_1,P_2,P_3,\cdots$;

**步骤** 4　给 XML 文档树 $T$ 的其他结点的编码 *order* 赋值:设某结点 $P$ 的最大素数因子为 $P_k$,则其子结点自左至右依次为 $P.code\times P_k$,$P.code\times P_k+1$,$P.code\times P_k+2,\cdots$。

根据上述编码机制,可以用下面的算法实现 PDIV 编码机制:

**算法** 1　*Coding_XML_Tree*($T$)

输入:XML 文档树 $T$,素数数组 $P[n]$,其中 $P[1]=2,P[2]=3\cdots$

输出:XML 文档树 $T$ 中每个结点 $N$ 的编码 *Code*[1..*N*]

```
BEGIN
    if N is the root of T
        then   code[N]=1;
    else if (N.level==1)
        code[N]=P[n-1];
    else
        获得 N.parent.code 的最大素因子 *p;
        temp=N;
        While(temp.parent is N.parent)
        {
            code[temp]=code[N.parent]*(*p);
            temp=temp.rightbrother;
            p++;
        }
            for(each child NC of N)
                Coding_XML_Tree(T)   //深度优先递归调用
END
```

利用算法 1 只需要对于 XML 文档树 $T$ 进行一遍扫描就可以得到文档树 $T$ 中所有结点的编码值。图 1 是一个示例性的 XML 文档 productset.xml,以及对该 XML 文档编码后得到的结果。

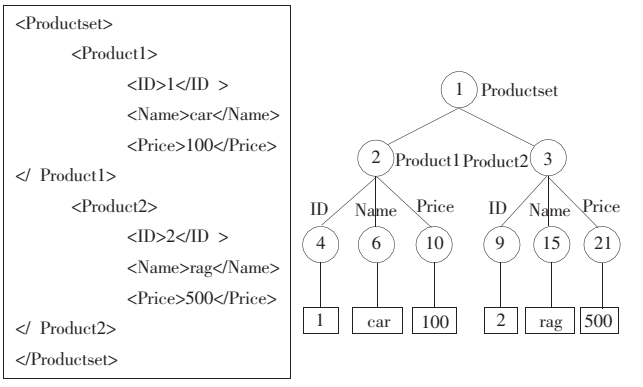


**图 1**　XML 文档及其对应的编码文档树

### 3.2　前缀整除编码性质分析

#### 3.2.1　祖先后裔关系的快速查询

结点 $A$ 与结点 $D$ 之间存在祖先后裔关系当且仅当 *A.code* 是 *D.code* 的前缀约数。

**证明**　当结点 $A$、$D$ 相差仅为一层的时候,上述命题显然成立。当结点 $A$、$D$ 相差多层的时候,如果 *A.code* 是 *D.code* 的前缀约数,设 $P$ 为 *D.code*/*A.code* 的最小素数因子,那么编码为 *A.code***P* 的结点是 $A$ 的子女结点;并且编码为 *A.code***P* 的结点是 $D$ 的祖先结点,因此,结点 $A$ 与结点 $D$ 之间存在祖先后裔关系。

因此,判断结点之间的祖先后裔关系只需要判断两结点的编码之间是否存在前缀整除关系。从前缀整除关系的定义来看,上述判断的时间复杂度为 $o(n)$。

#### 3.2.2　更新支持

和传统的关系数据库的更新操作相同,Native XML 数据库的更新操作主要包括修改、删除和插入。下面探讨 PDIV 编码对于 Native XML 数据库的更新操作的支持。

对 Native XML 数据库的修改主要是指在不改变文档树结构的前提下,修改文档树中结点的内容、标签名称等。因为上述修改并不影响文档树的结构,所以也不会影响结点的编码。对 Native XML 数据库的删除主要是指删除文档中某个结点或者某个子树。虽然删除操作改变了文档树的结构,但是没有改变文档树结点存在的上述前缀整除关系,因此,没有必要对文档树重新编码。对 Native XML 数据库的插入主要是指将新的结点增加到文档树中。该操作改变了文档树的结构。因为在对插入的新结点进行编码的时候引入了兄弟结点没有的素数因子,所以插入操作没有改变文档树结点存在的上述前缀整除关系,因此,没有必要对文档树重新编码。

需要说明的是,删除或者插入操作可能导致部分结点的编码不能正确表示其在兄弟结点中的位置信息。如果需要正确表示结点在其兄弟结点中的位置序号,可以在 PDIV 编码中增加补充信息。也就是说,将 PDIV 编码形式重新定义为<*code*, *sibling_code*>,其中 *code* 的含义与原 PDIV 编码的含义相同,*sibling_code* 表示结点在其兄弟结点中的位置序号,这种编码成为扩展的 PDIV 编码。在扩展到 PDIV 编码机制下,用 *sibling_code* 表示结点在其兄弟结点中的位置序号,更有利于提取结点在文档树中的位置信息;更便于 XML 文档的更新。

#### 3.2.3　编码长度

PDIV 编码的长度约为 $o(\ln(N))$,下面进行证明。

**证明**　下面对编码长度最大的深度为 $n$ 的满 $k$ 叉树 $T$ 进行

分析。该树 $T$ 中编码值最大的结点是第 $n$ 层最右端的结点，其编码为：$code=P_k\ P_{2k-1}\ P_{3k-2}\cdots P_{(n-1)k-(n-2)}$。基于数论，如果自然数中第 $t$ 个素数为 $P_t$，那么：

$$\lim_{t\to\infty}\frac{P_t}{t\ln t}=1$$

从而：

$$code\leqslant\prod_{i=1}^{n-1}p_{ik}=o\left(\prod_{i=1}^{n-1}ik\ln(ik)\right)=o((n-1)!\ k^{n-1}(\ln(n-1)k)^{n-1})$$

取对数，可得：

$$\ln(code)=o(\ln(n!\ )+n\ln k)$$

设 $N=k^n$，则有：

$$n=\log_k N=\frac{\ln N}{\ln k}=o(\ln N)$$

并且 $n!\leqslant n^n$，从而：

$$\ln(code)\leqslant o(\ln N\ln\ln N+\ln N)$$

综上，可以看出：

$$code\leqslant o(\ln N)$$

因此，PDIV 编码机制的编码长度较短。并且，宽度 $k$ 对于该编码的编码长度的影响呈现对数级别，而深度 $n$ 的影响较低，因此编码长度主要取决于文档树宽度。

### 3.3 基于二叉树的前缀整除编码机制

树的存储多采用二叉树的形式，因此有必要对二叉树的前缀编码机制进行研究。基于二叉树的前缀整除编码机制充分利用了二叉树的特点，使得编码更为便利。

假设 $R$ 是二叉树 BT 的根结点，$A$ 和 $LD$、$RD$ 是 BT 中的任意结点，那么基于二叉树的前缀编码机制如下：

**步骤** 1 定义素数数列$\{p_n\}$，即 $P_1=2,P_2=3\cdots$；

**步骤** 2 给 XML 文档树 $T$ 的根结点的编码 $code$ 赋值为 1；

**步骤** 3 对于除根结点之外的其他任意结点 $A$，求得 $A.code$ 的最大素因子 $P_n$，则其左儿子 $LD$ 的编码 $code$ 为 $A.code\times P_n$；

**步骤** 4 对于除根结点之外的其他任意结点 $A$，求得 $A.code$ 的最大素因子 $P_n$，则其右儿子 $RD$ 的编码 $code$ 为 $A.code\div P_n\times P_{n+1}$。

## 4 性能分析

本部分主要采用标准的 Sharks 数据对 PDIV 编码进行性能评估，探讨 PDIV 编码机制与其他编码机制的性能比较，在对 XML 文档更新的支持程度方面 PDIV 编码机制与其他编码机制的比较。实验中的 Sharks 数据采用 Shakespeare2.00 标准[12]，执行实验程序的计算机的 CPU 是 Intel Core™2，内存为 512 MB，运行的操作系统是 Windows XP Professional，采用标准 C++程序设计语言编写实验程序。

### 4.1 时空性能比较

相对于区间编码机制而言，PDIV 编码机制有较好的空间性能，这主要是因为区间编码机制需要使用前后两个端点表示一个区间。相对于 BBT 编码机制而言，PDIV 编码机制也有较好的空间性能，这主要是因为 PDIV 编码机制的编码长度比 BBT 编码机制更短。图 2 显示了对 Sharks 数据中的 hen_v.xml，john.xml，lear.xml，m_wives.xml 分别进行编码后，结果编码所需要的存储空间。从图 2 可以看出，就存储空间的要求而言，PDIV 编码机制相对于其他编码机制有一定提高。

PDIV 编码机制比区间编码机制时间性能更好，这主要是因为区间编码机制至少需要扫描两遍 XML 文档树而 PDIV 编码机制只需要扫描一遍。PDIV 编码机制与 BBT 编码机制时间性能相当，由于 PDIV 编码机制编码过程中需要素数判断，时间性能比 BBT 编码机制略差。

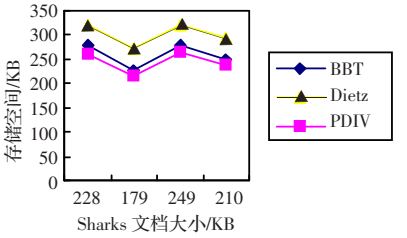


**图 2 对 Sharks 文档进行编码所需要的存储空间比较**

### 4.2 更新支持比较

图 3 显示了在结点的插入和删除操作中，PDIV 编码机制和以 Dietz 编码机制为代表的区间编码机制需要重新编码的结点的数目的比较。

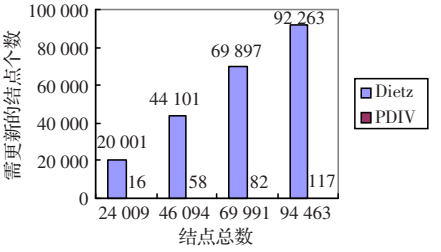


**图 3 更新支持性能比较**

从图 3 可以看出，在以 Dietz 编码机制为代表的区间编码机制下，当 XML 文档树中插入或者删除结点时，需要重新编码的结点的数目接近于 XML 文档树结点的总数。而 PDIV 编码机制下，当 XML 文档树中插入或者删除结点时，需要重新编码的结点的数目微乎其微。因此，PDIV 编码机制对 XML 文档更新的支持程度非常好。

## 5 结论和展望

本文集中讨论了 Native XML 数据库的文档编码机制，提出了一种新的文档编码机制前缀整除(PDIV)编码。理论分析和实验结果表明，该编码机制能够实现祖先后裔关系的快速查询，可以在常数复杂度的时间内实现结点之间祖先后裔关系的判断；该编码全面支持更新，插入或者删除结点时受影响的结点非常少；并且该编码长度较短，约为 $o(\ln N)$。这是在 Native XML 数据库文档编码机制上进行的有效尝试。

在本文的基础上，在 Native XML 数据库的文档编码机制的研究方面可以进一步展开其他研究工作。例如，本文提出的编码机制需要单独的素数表加以支持，如何减少由此带来的存储代价；在频繁进行插入或者删除操作的情况下，如何更为有效地提供兄弟结点关系信息。这些都是值得进一步研究的问题。

## 参考文献：

[1] 冯建华，钱乾，廖雨果，等.纯 XML 数据库研究综述[J].计算机应用研究，2006(6)：1-7.

[2] Li Quan-zhong，Moon Bongki.Indexing and querying XML Data for regular path expressions[C]//Apers P M G，Atzeni P.Proceedings of

(下转 181 页)