第32卷 第24期 计 算 机 工 程 2006年12月
Vol.32 No.24 Computer Engineering December 2006
·软件技术与数据库· 文章编号：1000—3428(2006)24—0075—03 文献标识码：A 中图分类号：TP311.56

协同编辑系统中多版本 XML 文档的存储与查询

邵伟峰，杨 洋
(苏州大学计算机科学与技术学院，苏州 215006)

摘 要：在软件配置管理和协同工作领域中，多版本 XML 文档的管理是一个非常重要的应用点和价值点。该文为多版本 XML 文档的存储与查询提供了一种有效的技术，利用 XML 文档的 SPaR 模型以及在文档节点加上时间戳 LifeSpan 来达到多版本 XML 文档的增量存储与查询。

关键词：协同工作；SPaR 模型；SAX；HIBERNATE

Storage and Query of Multiversion XML Documents in Cooperative Editing Systems

SHAO Weifeng, YANG Yang
(School of Computer Science and Technology, Soochow University, Suzhou 215006)

【Abstract】The management of multipleversions of XML documents represents an key problem for many traditional applications, such as software configuration control and computer supported cooperative work(CSCW). this paper presents a novel approach to achieve storage and query of multiversion XML documents by a scheme based on Sparse Preorder and Range and timestamps for the elements of XML documents.

【Key words】CSCW; Sparse Preorder and Range(SPaR); SAX; HIBERNATE

XML 即可扩展标记语言，是一种采用开放的自我描述方式定义的数据格式，它具有开放性、灵活性、易读性和平台无关性等特点。该标准发布后，得到了 IBM、SUN、Oracle 和 Microsoft 等大公司的支持，并被广泛应用，尤其是在软件配置管理与协同工作领域的应用。

我们开发的协同编辑系统 Z-OFFICE 是一个具有群体性、交互性、分布性和协作性的人机网络工作环境，对于那些物理上分散的作者对一篇文档进行共同编著，允许多个物理上分布的用户并发地浏览和编辑一篇共享文档。在该项目中，我们定义了一种专门的 Z-OFFICE 文档格式化协同编辑中处理的文档，Z-OFFICE 文档是对 XML 文档的扩展，我们的系统就要求对这些文档进行增量存储及查询。

对于管理和存储，目前已有数种不同存储 XML 文档的方法可供选择，例如 XML 数据库。但是这些方法都无法保留使用者对文档所做的一连串修改痕迹，这就是文档版本管理问题。文档版本管理要点在于如何有效地存储连续修改的文档，减少存储成本，并且能快速获取文档版本。

1 Z-OFFICE 文档的格式和组织结构

常规文档是连续的线性结构，文档在组织结构上没有划分手段。在 Z-OFFICE 中考虑使用 XML，它是组织大型和复杂文档的理想格式，其基于文档的结构可进行分工协作和分布管理，比常规文档更适合于协同编著工作和版本管理。在协同编辑系统中，协作成员的工作方式是基于分工协作的，即把一个大的编辑任务划分为多个子任务，每个协作成员分配一个或多个子任务。于是将整篇文档划分成若干的块(block)，这样文档的主体结构就有了层次，分为文档(document)、块(block)二层。因此将整篇协同编著文档称为全局文档版本，局部文档版本则是对应 block，全局版本并不对应实际内容，它只记录组成它的各个局部版本，局部版本才对应实际内容。于是，block 成为系统要处理的单位文档，单位文档具有树型结构，具有块(block)、段落(paragraph)和句子(sentence)3 个主层次，如图 1 所示。

```
<document name="测试文档">
  <block id="block1">
      <paragraph>
          <sentence>  体育新闻</sentence>
      </paragraph>
      <paragraph>
          <sentence>  乌巴之战的结果原来是乌兹别克 10 比 0 获胜，</sentence>
      </paragraph>
      ……
  </block>
</document>
```

图 1 Z-OFFICE 文档组织结构

2 文档增量存储的核心——SPaR[1,2]

XML 文档可以被看作是一棵有序树，其中每一个树节点代表 XML 文档中的一个元素(element)。树的先序遍历节点编码可以唯一地表示 XML 文档元素，先序编码很容易计算，但该编码方式有致命缺陷，比如当文档变化发生插入、删除节点时将改变文档树节点的先序编码，因此需要一种持久节点编码来表示文档各个元素，就是在先序节点编码的基础上

基金项目：苏州大学“211”工程重点建设学科资助项目(x2118004)
作者简介：邵伟峰(1980－)，男，硕士生，主研方向：数据库应用；杨 洋，硕士
收稿日期：2005-12-21 **E-mail：**wfshao@zhhz.org

改进，将先序编码改成不连续、之间存在间隔的先序节点编码；Sparse Preorder and Range(SPaR)就是为解决该问题提出的一种方案。

2.1 编码方案

SPaR 编码方案使文档的每个节点包含两个属性：

(1)Durable Node Number(DNN:节点序列的稀疏编码)；

(2)Range(保存父子关系)。

注意：如果 DNN 采用先序遍历法，Range 就用来表示该节点的子孙数目。

举个例子，假设 A、B 分别为 XML 文档中的两个节点，存在如下关系：

dnn(A) ≤ dnn(B) ≤ dnn(A) + range(A) iff a node B is descendant of a node A

很明显，interval[dnn(X),dnn(X)+range(X)]只与节点相关，由此可以考虑将这两个属性整合到一起，以区间 SPaR(start,end)表示。由此当文档中节点被修改，它们的 SPaR 值并不改变，改变的只是节点内容。当有新的节点插入时，分别给它们指定 SPaR 值，如果 DNN(SPaR 上界)采用连续先序编码方案，指定的 SPaR 值必将与相邻的文档节点产生冲突，从而使文档节点之间关系产生混乱，反之如果 DNN 采用稀疏的编码方式就不存在该问题，不会影响相邻的文档节点。

2.2 版本化模型

在 SPaR 版本化模型中，XML 文档的每个节点被扩展拥有两种属性：SPaR(SPaRstart,SPaRend)以及时间戳 lifespan(Vstart,Vend)，如图 2 所示。

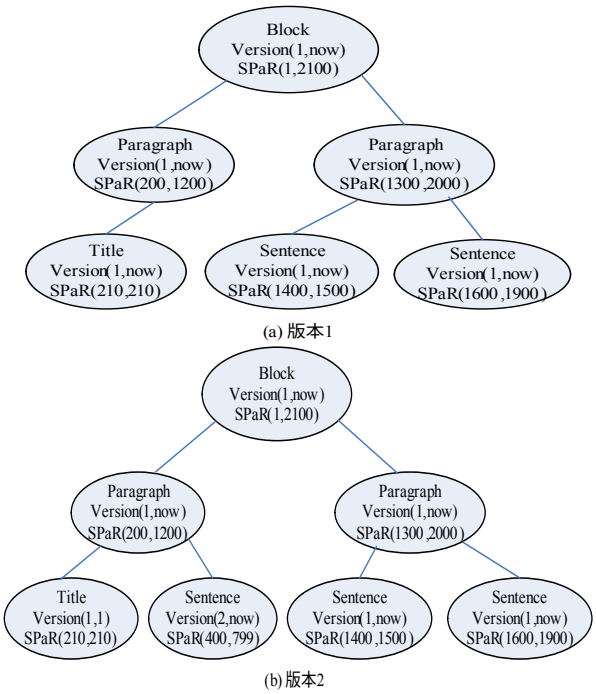

图 2 版本化模型

图 2 显示了不同时间的文档版本：版本 1 和版本 2。版本 2 的产生是由版本 1 删除节点 Title 和新增节点 Sentence。注意，在该版本化模型中，节点的删除只是逻辑上的删除，节点信息被保留，但要将 Vend 改为节点被删除前的版本号；节点的插入是新增一个文档节点，生成新插入节点信息如 SPaRstart、SPaRend、以及 Vstart、Vend，节点的修改为节点的逻辑删除加上节点的插入，新节点信息继承原节点的其他属性以及 SPaRstart，SPaRend，同时产生新的 Vstart、Vend，形成一个文档新节点。

2.2.1 文档的解析

文档初始版本通过 XML 解析器将 Z-OFFICE 文档解析成文档树结构，解析器、文档树以及文档节点都是自行定义设计的，并分别以 3 个 Java 类(Zparser,Ztree,Znode)实现。文档存储的实质就是对文档节点进行保存，采用的技术实现方法是 O/RM-对象关系映射(Hibernate3.0)，将文档节点对象映射到关系型数据库存储。

Zparser 使用 SAX(Simple API for XML)解析数据，一次扫描文档，生成文档节点对象 Znode，并将这些节点组织形成 Ztree 的数据结构形式。

文档节点 Znode 具有如下有属性结构：

(DOCID,BLOCKID,SPARSTART,SPAREND,VSTART,VEND,ISLEAF,ISATTRI,FLAG,ELEMENTVALUE)

(1)DOCID、BLOCKID 为文档标识符和块标识符，用来唯一标识拥护操作的单位文档。对于同一文档块中的节点而言，DOCID 与 BLOCKID 是相同的，在系统中分别为文档标记与文档块标记设置一个累加器，相应生成 DOCID 和 BLOCKID(正整数即可)。

(2)SPARSTART、SPAREND、ISLEAF(是否为叶子节点)、ISATTRI(是否属性节点)、FLAG(节点标志位，0:insert、1：delete、2：update，查询使用)、ELEMENTVALUE(若节点是叶子节点，节点信息就是文本信息，若非叶子节点，节点信息就是标签名)的生成。假设文档解析器扫描到某一文档节点 node，查找节点堆栈，若堆栈空，则节点 node 为文档树根节点，SPARSTART 赋 1，SPAREND 暂时为未知数(退栈时回填)，同时为节点其他属性(ISLEAF、ISATTRI、FLAG、ELEMENTVALUE)赋值，节点 node 进堆栈；若堆栈不空，假设栈顶节点为 node＇,则有 node. SPARSTART=node＇. SPARSTART+1,node 的 SPAREND 暂时为未知数(退栈时回填)。如果 node 为非叶子节点，则继续扫描，如果 node 为叶子节点，则 node.SPAREND=node.SPARSTART。下面的工作就是自堆栈顶向下，逐个回填堆栈中节点的 SPAREND 值，方法是依次增加 1，直到堆栈底。然后，再自底向上，逐一输出堆栈中各元素节点。

(3)对于 VSTART、VEND，在存储文档版本 n 时，需要比较版本 n-1。若版本节点 node 为新插入节点，则节点的 VSTART 为 n，VEND 为 now ;若 node 为被删除节点，VSTART 保持不变，VEND 赋值为 n-1。

(4)既然是稀疏编码，可是以上产生的 SPaR 值却是连续的，为此就要做一些调整，在应用程序中对节点链表顺序访问一遍，为 SPaR 值乘上一个系数，达到稀疏编码的目的，为之后增量式版本存储节点的插入、更新以及删除提供支持，尽可能保证节点的操作不会影响邻近节点。

2.2.2 文档的增量版本化

作为增量式文档版本存储，提交新版本是在之前最新版本(Vn)上进行一系列编辑操作。

一些基本编辑操作如下:

(1)DELETE:更新被删除文档节点以及所有它的子孙节点的 Vend 为 Vn；释放被删除节点的 SPaR 以便重新使用；

(2)INSERT：为新插入节点创建记录，lifespan(Vn,now),分配一个未使用的 SPaR 区间值；

(3)UPDATE：更新被节点属性 Vend 为 Vn，然后创建记录，lifespan(Vn,now),SPaR 保持不变(节点在文档中的位置没

发生改变)。

以上 3 种是最基本的编辑操作的一般性定义，在实际应用中对文档的操作不仅仅只是单一节点删除、插入以及更新，针对子树的操作反而更为频繁，为此自行设计并实现了Operation类，实现了各种简单单一节点操作以及复杂子树操作，这些操作(也就是deltas)通过Xdiff[5]文档变化比较算法 产生。

2.3 文档的版本重构

2.3.1 文档重构步骤

文档版本重构可以归纳成 3 个主要步骤:

(1)鉴别出指定文档指定版本的所有节点记录；

(2)文档节点根据 SPaR 值排序；

(3)利用文档重构算法形成文档有序树结构。

ZNodeBean 类完成文档节点的获取与排序(步骤 1、步骤 2)，节点只要符合版本起始值 VSTART<=要获取的版本<=版本终止值 VEND。

2.3.2 文档重构核心算法

文档重构算法大致如下(其中 SORTED_LIST 是根据 SPaR 值递增排序的文档节点链表)：

```
VersionReconstruction(SORTED_LIST) {
    Initialize ANCESTOR_STACK as empty;
    Assign the first element of SORTED_LIST as ROOT
    and remove it from SORTED_LIST;
    Push ROOT into ANCESTOR_STACK;
    current_node = ROOT;
    For (each element,E,in SORTED_LIST from the beginning)
    {
     if (SPaR(current_node) contains SPaR(E))
     Insert E as the first direct child of current_node;
     Push E into the ANCESTOR_STACK;
     else {
          do {
          Pop the top element, A, from ANCESTOR_STACK and
          compare SPaR(A) with SPaR(E);
          if (SPaR(A) contains SPaR(E)) {
               Insert E as the next direct child of A;
               Push A back into ANCESTOR_STACK;
               Push E into the path_stack
          }
        } while (the parent of A is found);
     }
     Set E as the current_node for the next run;
     }
}
```

ZVersion 类实现了该文档版本重构算法，给定 DOCID、BLOCKID 以及版本号能够返回相应的文档版本。

3 多版本 XML 文档(Z-OFFICE 文档)查询

对于查询，尽管 XML 数据表现形式灵活，可以描述相当复杂的结构，但 XML 数据本质上是一种自描述的半结构化数据，其数据模型不同于以往的层次、关系、面向对象数据模型，是树状模型。针对这种树状数据的查询，学者们已提出了多种 XML 查询语言，例如 XPath 和 XQuery 等，但它们目前仍处于发展和完善阶段，这些查询语言都不支持多版本 XML 文档的查询。

项目采用了 SPaR 版本化模型的存储数据结构，该模型在文档存储的过程中保留了一系列文档修改痕迹，为多版本 Z-OFFICE 文档查询提供了一切前提，在一般 XML 文档版本管理中，当想要查询某个版本的内容时，必须先恢复其原有版本，再进行查询。而采用了 SPaR 版本化模型的存储数据结构的文档查询，就不需要恢复文档版本就能直接进行查询，原因是 ZNode 节点拥有空间位置和时间位置的信息，可以保证不恢复文档版本就能进行直接且快速的查询，只要对节点元素增加版本判断就能达到目的。

基于关键字查询是通过检索数据库获取符合关键字查询条件的所有文档节点，统计和计算得到关键字在各个版本文档中出现的次数以及各个版本文档，并以表单形式出现，表单每一行内容包括版本号、关键字出现次数以及查询关键字，通过双击选中表单记录弹出文档显示窗口，以便用户查看，见图 3。

图 3 界面图

4 结束语

本文分析了 XML 文档多版本的成因、XML 文档版本管理技术——SPaR 模型，并在该模型上扩展时间戳属性(即文档节点版本属性)，并在系统中将文档节点对象使用对象映射技术(OR/M)——HIBERNATE3.0 存储到关系型数据库，对于完善协同编著系统中文档存储模块，增强文档存储模块使用性有一定的意义。

参考文献

1 Chien Shu-Yao, Tsotras V J, Zaniolo C, et al. Efficient Complex Query Support for Multiversion XML Documents[C]. Proceedings of the 8th International Conference on Extending Database Technology,2002.

2 Chien Shu-Yao, Tsotras V J, Zaniolo C, et al. Storing and Querying Multiversion XML Documents Using Durabl Node Numbers[C]. Proceedings of the 2nd International Conference on Web Information Systems Engineering, Kyoto, 2001-12.

3 Al-Khalifa S, Jagedish H V, Koudas N, et al. Structural Joins: A Primitive for Efficient XML Query Pattern Matching. Proceedings of the 18th International Conference on Data Engineering, 2002.

4 Cobena G, Abiteboul S, Marian A. Detecting Changes in XML Documents[C]. Proc. of ICDE, 2002.

5 Wang Yuan, DeWitt D J, Cai Jinyi, et al. An Effective Change Detection Algorithm for XML Document[C]. Proceedings of the 19th International Conference on Data Engineering, 2003.

6 World Wide Web Consortium. Extensible Markup Language (XML) [EB/OL]. http://www.w3.org/XML.