苏州大学学位论文使用授权声明

本人完全了解苏州大学关于收集、保存和使用学位论文的规定，即：学位论文著作权归属苏州大学。本学位论文电子文档的内容和纸质论文的内容相一致。苏州大学有权向国家图书馆、中国社科院文献信息情报中心、中国科学技术信息研究所（含万方数据电子出版社）、中国学术期刊（光盘版）电子杂志社送交本学位论文的复印件和电子文档，允许论文被查阅和借阅，可以采用影印、缩印或其他复制手段保存和汇编学位论文，可以将学位论文的全部或部分内容编入有关数据库进行检索。

涉密论文□

本学位论文属　　　　　　在____年___月解密后适用本规定。

非涉密论文□

论文作者签名：赵圣维　日　期：2010.6.17

导　师　签　名：　日　期：2010.6.17

一种改进的XML数据管理方案

摘　要

随着互联网技术的飞速发展，基于网络的诸多服务如电子商务、电子图书等在生活中起着越来越重要的作用，如何利用Internet上的大量信息成为函待解决的问题。XML以其简单、可扩展和跨平台等诸多优点，已经成为数据表示、数据存储和数据交互的事实标准。如何有效地管理XML数据，如对XML数据进行存储、查询、更新、发布等已成为当今数据库领域中一个重要的研究课题。本文在分析了相关研究现状的基础上，开展了以下的研究：

首先，根据当前存储方案不能有效支持文档更新的现状，本文提出了一种具有更新功能的XML存储方案XSC。通过设计若干个关系表来存储XML文档树中的结点信息和结构信息。无论XML文档是否具有DTD，都可以将XML文档映射到存储模式中。当涉及插入、删除结点操作时，只需要对其余的结点进行少量的重新编码就可以正确地实现XML文档的发布、查询。

其次，针对目前小枝模式查询效率不高的特点，本文提出了一种非归并的小枝模式匹配算法TwigWM。TwigWM算法利用索引将XML文档中的结点组织成标签流，使用部分栈和链表的数据结构实现查询。与许多小枝模式查询算法不同，TwigWM算法的执行是一个输出整体结果的非归并过程。

最后，本文在Office数据源上构建了一个XML数据管理应用的实例，实现了上述提出的XML数据管理方案。本实例通过友好的用户界面，可以实现任何以数据为中心的XML文档的存储、更新和查询。

本文对XML数据管理技术的研究具有一定的现实意义。它不仅提出了对XML文档有效更新的存储方案，还进一步研究了在XML查询中小枝模式非归并匹配的问题，可以提高XML查询的效率。另外，本文的实例验证也对相关的实际应用具有一定的参考价值。

关键词： XML、存储映射、动态更新、小枝模式、查询算法

作　　者：赵圣猛

指导教师：钱培德

An Improved Schema for XML Data Management

Abstract

many web-based services such as e-business are playing an increasingly important role in life. How to use the large amount of information has become the problem which need to be solved urgently. XML has become a new standard for data representation, data exchange on the Internet-known for its simplicity, extensibility and numerous other benefits. How to effectively manage XML data has become an important research topic in database field, such as XML storage, XML query and so on. In this paper, the main research work as follows:

Firstly, this paper provides a storage schema named XSC, which effectively supports XML update. Through designing a number of tables, XSC stores node information and structural information. Regardless of whether XML document has DTD, the XML document can be mapped into XSC. When coming to insert nodes and delete nodes, the XML document can be correctly published without much re-encoding on the remaining nodes.

Secondly, in the case of inefficient twig pattern query, the paper proposes an algorithm of twig pattern query named TwigWM. TwigWM organizes nodes into label stream using index, and achieves querying with the data structure of stack and linked list. Compared with other twig query algorithms, the execution of TwigWM algorithm is the non-merging process which provides the overall results.

Finally, based on the data of Office application, this paper builds an experimental system, which adopts the above proposed XML data management schema. Through friendly user interface, the system can manage any data-centric XML documents.

The work on XML data management has certain practical significance. On one hand, when storing XML documents, it solves the issue of updating XML documents. On the other hand, this paper studies the algorithm of twig pattern query without merging, which improves the efficiency of XML query. Besides, experimental system provides some reference for relative researches.

Keywords: XML,Storage Mapping,Dynamic Update,Twig Pattern,Query Algorithm

Written by Zhao Shengmeng

Supervised by Qian Peide

目　录

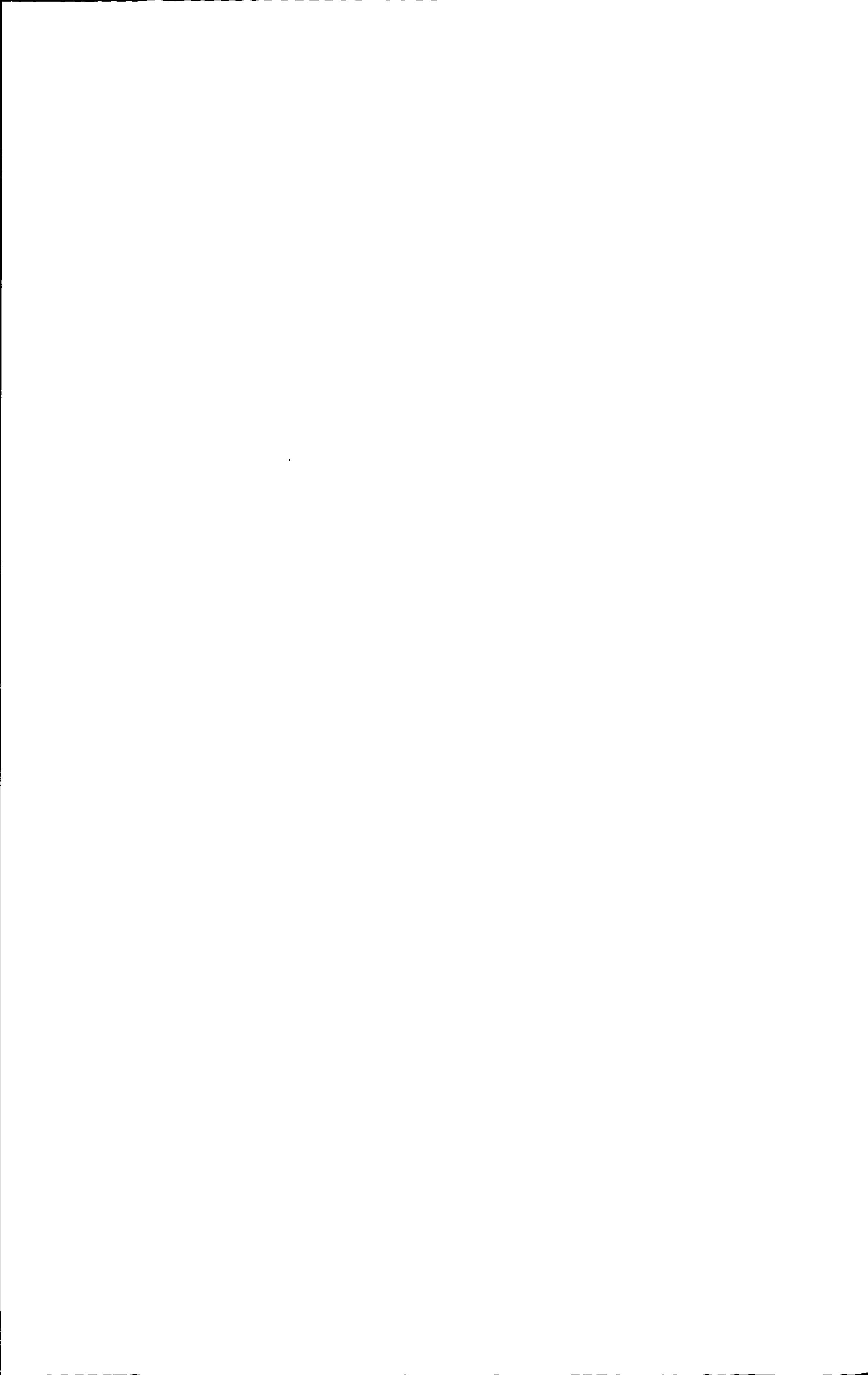
第一章　绪论 …… 1
1.1　课题背景 …… 1
1.2　课题研究现状 …… 2
1.3　课题研究内容 …… 4
1.4　课题研究意义 …… 5
1.5　文章组织结构 …… 5

第二章　XML数据管理概述 …… 7
2.1　XML数据模型 …… 7
2.2　XML查询语言 …… 9
2.3　XML文档编码方案 …… 10
2.3.1　基于区间的编码 …… 10
2.3.2　基于路径的编码 …… 12
2.3.3　其它编码 …… 12
2.4　XML数据索引技术 …… 12
2.4.1　结构概要类索引 …… 13
2.4.2　结点记录类索引 …… 14
2.5　XML数据存储映射 …… 15
2.5.1　模型映射方案 …… 15
2.5.2　结构映射方案 …… 16
2.6　XML结构连接算法 …… 17
2.6.1　包含关系的结构连接 …… 18
2.6.2　小枝模式的结构连接 …… 19

第三章　数据管理方案设计 …… 21
3.1　问题提出 …… 21
3.2　设计目标 …… 23

3.3 模型方案 …… 24
3.4 小枝查询方案 …… 25
第四章 可支持更新的XML存储方案XSC …… 27
4.1 引言 …… 27
4.2 XSC映射实现 …… 28
4.2.1 XRel映射思想 …… 28
4.2.2 XSC映射策略 …… 28
4.2.3 XSC映射算法 …… 30
4.3 XML文档重组 …… 32
4.4 XML查询处理 …… 34
4.5 动态更新XML文档 …… 35
4.6 实验分析 …… 36
4.6.1 XML存储性能分析 …… 36
4.6.2 XML查询性能分析 …… 37
第五章 基于非归并的小枝模式查询方案TwigWM …… 39
5.1 引言 …… 39
5.2 相关概念及说明 …… 40
5.2.1 结点编码 …… 40
5.2.2 相关定义 …… 41
5.3 查询匹配算法TwigWM …… 43
5.3.1 数据结构及说明 …… 43
5.3.2 算法思想 …… 44
5.3.3 具体算法描述 …… 46
5.4 算法分析 …… 50
5.5 实验分析 …… 51
5.5.1 实验环境 …… 51
5.5.2 实验结果 …… 52

第六章 XML数据管理在Office数据源实例中的应用 …… 54
6.1 实例背景 …… 54
6.2 实例设计 …… 55
6.2.1 概要设计 …… 55
6.2.2 详细设计 …… 56
6.3 实例实现 …… 58
6.3.1 实现环境及数据源 …… 58
6.3.2 结果展示 …… 58
第七章 总结与展望 …… 62
7.1 总结 …… 62
7.2 展望 …… 63
参考文献 …… 64
攻读硕士学位期间发表的论文 …… 70
致谢 …… 71



第一章　绪论

1.1　课题背景

随着互联网技术的发展，特别是Web技术飞速发展，其丰富的资源给人们的生活带来了极大的便利。特别是应运而生的HTML（Hyper Text Markup Language，超文本标记语言），以其简单易学、灵活通用的特性，使人们发布、检索、交流信息变得非常简单，从而使Web成了最大的环球信息资源库。然而基于网络的诸多服务和应用如电子商务、电子图书等使Web数据变得更加复杂和多样化，从而使得作为Web使用最广的HTML语言的局限性越发明显。由于Internet上的数据多以无结构的形式出现，XML（eXtensible Markup Language，可扩展标记语言）[1]以其简单、可扩展和跨平台等诸多优点，已经成为数据表示、数据存储和数据交互的事实上的标准。XML与HTML相比具有许多优点：

(1)XML简单、自我描述且易于解析。XML对数据的语义描述和数据内容本身都包含在XML文档中，一个应用可以按照各种方式解析、过滤、重构XML文档。

(2)HTML中的标记是固定的，不能扩展，而XML中的标记是由用户定义的，可以任意地扩展。XML的嵌套结构可以表示现实世界中各种复杂的对象，各种格式的数据都可以比较容易地转化为XML数据。

(3)HTML中的标记表示数据的显示格式，没有任何语义，而XML的标记则明确指出了数据的含义，使得细粒度的XML数据处理成为可能。

(4)XML实现了内容和表现二者的分离。文档类型定义DTD(Document Type Definition)描述了文档中元素和子元素间的嵌套结构，不同的用户可以通过XSL按不同的显示方式显示全部或部分的文档内容。

(5)XML具有开放的国际化标准。XML支持文档对象模型标准、可扩展类型语言标准、可扩展链接语言标准和XML指针语言标准。使用XML数据可以在不同类型的计算机系统间交换信息。

由于XML同HTML兼容，且与平台无关，同时又是一种真正的扩展语言，受到了业界的广泛关注。比如微软、IBM、Oracle等参加了XML各项标准的制定，微软的Windows、Office都使用XML文件格式，IE6.0、IE7.0已经全面实现了对XML的

支持。在网络传输协议方面，由W3C、微软、IBM和SAP共同制定的SOAP（Simple Object Access Protocol简单对象访问协议）同样也是以XML为核心。

随着XML应用的日益普及，互联网上XML的信息量也随之急剧增长，很难想象面对成千上万的数据文件，如果仅仅通过文件系统来管理，那么无论是文件的搜索还是文件的调用都将是不可能。随之而来的问题就是：如何有效地存储这些海量的XML数据，又如何查询这些XML数据。在存储和查询数据这一领域，目前大致主要分为两种方法：

第一种方式就是为XML数据本身量身定做的数据库即纯数据库(Native XML Database)，它为XML文档定义了一个逻辑上的模型，XML数据的存储和查询都是基于这个模型。这个模型至少要包含元素，属性等，并保持文档间的顺序。将XML文档作为逻辑存储的基本单位，正如关系数据库系统将元组作为存储的基本单位一样，不要求只能使用某一特定的底层物理模型或某种专有的存储格式。因此，纯XML数据库充分考虑XML数据的特点，以一种自然的方式来处理XML数据，能够从各个方面很好地支持XML存储和查询，并且能够达到较好的效果。但是，纯XML数据库才刚起步，要和关系数据库那样成熟还有很长的路要走。

另一种方式是在已有的关系数据库系统或面向对象数据库系统的基础上扩充相应的功能，使其能够胜任XML数据的处理要求，这种数据库又称为使能数据库(XML Enable Database)。目前，XML使能数据库的研究主要是基于关系数据库系统，此种数据库方法的优点是可以利用非常成熟的关系数据库技术，集成现有的大量存储在关系数据库中的商用数据。

1.2 课题研究现状

XML作为一种通用的格式进行数据的表示、交换、存储和访问，这对数据库系统提出了许多新的挑战。在XML数据管理领域研究中，关于XML数据的存储和查询是两个重要方面，受到了广泛的关注。

目前国内对XML数据管理的研究尚处于探索阶段，特别是XML查询方面。复旦大学的VXMLR系统 [2]把文档存储到关系数据库中。VXMLR先把XML模式映射为关系模式，然后把文档数据存入关系数据库中。查询时，先把XML查询语言重写为SQL语句，最后把查询结果重构为文档。中国人民大学孟小峰教授等研究开发的OrientX系统 [3]是国内第一个纯XML数据库系统，它的记录级别是文档级的，按

物理块划分，使记录之间联系的指针得以减少，提高了存储效率。OrientX系统 [4] 将语义相近的记录尽量按聚集存储，这是一种按逻辑意义存储策略，判断语义相近的方法是根据记录类型是否相同进行的。在XML查询处理方面，王静等 [5]提出了以目标结点为导向对路径表达式进行查询处理，该方法利用扩展基本操作来减少连接操作的数目。万常选等 [6]将编码方案、逆序列表和路径索引的思想相结合，提出了一种改进的索引结构，最多只需对参与连接的两个列表分别进行一次扫描，即可实现查询处理。

国外有不少研究机构在数据管理方面做了大量工作，如XML数据的高效存储、索引机制、查询处理和优化等。德国Software AG软件公司的Tamino [7]是第一个商用的纯XML数据库系统，它的主要创新在于索引结构和接口。密歇根大学的Timber [8]是一种采用面向对象数据库Shore来对XML进行存储的XML数据库，它借用了Shore系统中良好的存储管理、并发机制等。由于其存储粒度是结点级的，因此，在XML文档重组时可以避免冗余的转换。

在具体存储方面，已提出的XRel映射 [9]、XParent映射 [10]等属于模型映射。模型映射方法对某些应用来说提供了一定的灵活性，可以实现异构系统间数据的无缝集成，而不需要考虑文档的模式信息，即使源文档的结构发生了一定的变化也不会影响系统间数据的交换。然而，这种灵活性也是需要付出一定代价的，因为在模型映射方法中，每个数据项都要重复其模式信息，因此增加了系统的磁盘开销。DTD方法 [11]、STORED方法 [12]和P-Schema方法 [13]属于结构映射，它们需要将XML模式（或DTD）映射为关系模式，关系模式用来表示目标XML文档的逻辑结构，因此它是XML模式（或DTD）相关的。从XML文档的DTD或Schema推断XML元素应该怎样映射到关系中，这种方法能够利用关系数据库的特性，如查询优化和并发性控制等。文献 [14,15]中提出的方法属于约束映射方法，在存储XML文档时，考虑XML模式中的语义约束，如：键、函数依赖等，在此基础上推导出的关系模式可以进一步保持语义信息。

XML查询处理方面，集中关注如何对路径表达式进行匹配。已提出的方法主要有两种。第一种方法是把一个复杂的路径表达式分解成几个简单路径表达式，而简单路径表达式的计算采用逐步结构连接法，这样对XML文档的结构查询通常被转化为两个结点列表之间的包含关系或文档位置关系的结构连接操作。目前已提出的结构连接算法有EE/EA-JOIN [16]、MPMGJN [17]等。但是，这种计算方法会产生大量的中间结果，从而影响了查询的效率。第二种方法是把一个路径表达式表示成一

个小枝模式Twig，然后通过小枝模式匹配的方法来计算路径表达式的值。在小枝模式匹配方法方面，具有代表性的算法有TwigStack [18]、TJFast [19]等，它们只需扫描一遍输入结点列表就可输出匹配结果。此类算法对只包含祖先后代边的小枝模式查询可以保证为最优的（即所有的中间匹配结果都是可归并连接的），但是在包含父子边的小枝模式查询方面，这些算法并不是最优的，因为它们会产生许多无用的中间结果路径。为了解决包含父子边的小枝模式查询的效率问题，目前已提出了Twig2Stack [20]、TwigList [21] 算法，其分别采用层次栈和队列的结构而不需要归并，因此性能优于前述算法。但是他们需要遍历文档树中的每个结点，并且算法也较复杂。文献 [22,23]也针对Twig查询进行了相关地研究。

1.3 课题研究内容

XML作为数据交换的国际标准，已经贯穿在Internet应用的各个领域之中，从而带来了大量的XML格式的数据。如何存储和查询XML数据是一个重要的研究课题。

本课题以XML数据管理为研究的切入点，在提出一种编码方案的基础上研究XML数据存储，此存储方案考虑文档更新情况，当在XML文档中插入、删除结点时能及时地反映给用户。在XML查询方面，主要研究小枝模式匹配算法，利用索引确定哪些文档结点不满足输出要求，事先将其舍弃，用非归并的思想输出最终结果。具体研究步骤如下：

(1)提出一种XML编码方案

正确的编码方案是XML数据管理的基础，已存在的XML编码主要有区间编码和路径串编码，但在支持XML文档更新的情况下并不理想。本课题将先序编号与路径串相结合对文档树中的结点进行编码，并且在后期查询方面将编码中的路径串转换成数字串以利于模式匹配。

(2)具有更新功能的XML存储方案

XRel映射属于结点模型映射，它通过设计若干个关系来存储XML文档树中的结点信息和结构信息，与XPath标准紧密结合，能够对基于XPath的查询给予相当好的性能支持。但是它的PATH信息具有很大的冗余，修改一个结点的名称都需要相当复杂的操作。本课题借鉴了XRel方法的部分思想并提出了一种基于不同结构的XML文档存储映射，无论XML文档是否具有DTD，都可以将XML文档映射到存储模式中。当涉及插入、删除结点操作时，只需要对其余的结点进行少量的重新编码，就可以

正确的实现XML文档的发布、查询。

(3)非归并的小枝模式查询方案

小枝模式查询是XML查询中重要的方面，已有的算法如TwigStack和TJFast算法等被提出，但是他们都是基于归并的，不能避免大量的不必要的路径归并。本文提出了一种新的算法，利用索引将XML文档中的结点组织成标签流，使用部分栈和链表的数据结构实现查询。在将元素结点进入链表之前先判断其是否对整个路径解有贡献，若有贡献则进入链表，否则丢弃。整个过程是不断输出结果的非归并过程。

(4)本文将上述的管理方案合成一个整体的流程，构建了一个XML数据管理在Office数据源上的应用实例。

1.4 课题研究意义

XML为Web提供了一致的数据模型和描述语言，使得从语法上规范化地表示Web数据成为可能。通过研究XML数据的管理技术，可以为基于Web的数据管理提供新的途径和方法。

本课题主要研究了XML数据管理的相关技术，并改进了其中的部分算法。尽管这些算法并不一定是最优算法，还有待于进一步的研究与完善，但是本课题所做的工作还是具有一定的现实意义，体现在如下方面：

(1)本文在对XML存储方案做了深入研究的基础上分析了其优缺点。针对目前存储映射中更新的难题，设计了一种编码方案并实现了XML模式的映射。经实验验证，该方案具有跨平台性，给XML数据管理中映射方面的研究提供了借鉴价值。

(2)本文在小枝模式查询方面提出了一种新的匹配算法，实验结果比较表明，此算法在时间和空间方面有较好的效率，对部分XML查询算法的改进具有参考价值。

(3)本课题将XML数据管理的理论运用于实际，在Office应用方面进行了有益的尝试，并取得了较为满意的结果，促进了XML数据管理的研究。

综上所述，课题研究的内容对于XML数据管理具有一定的现实意义和参考价值，对XML的进一步应用有一定的推动作用。

1.5 文章组织结构

全文的组织如下：

第一章简要介绍了课题提出的背景、课题的研究内容、课题的研究现状和课题的研究意义。

第二章介绍了XML数据管理的相关知识，重点介绍了XML数据模型、查询语言、XML的常用编码、XML索引、存储映射方案、查询中的结构连接算法。

第三章介绍了数据管理方案的设计，其中对方案设计的由来和目标进行了阐述，同时还对存储方案和查询方案进行了介绍。

第四章针对目前XML存储方案不能有效地支持更新，提出了一种映射方案XSC，并介绍了如何用XSC方案来更新和发布XML，最后将XSC方案与别的方案进行了性能上的对比。

第五章介绍了小枝模式查询的处理过程，分析了当前小枝模式匹配算法的不足，在此基础上提出了一种非归并的小枝模式查询算法TwigWM，并将TwigWM算法与几种常用算法进行了比较。

第六章介绍了此文提出的XML数据管理方案在Office数据源上的应用实例。

第七章对所做的工作进行总结，并对未来工作进行展望。

第二章　XML数据管理概述

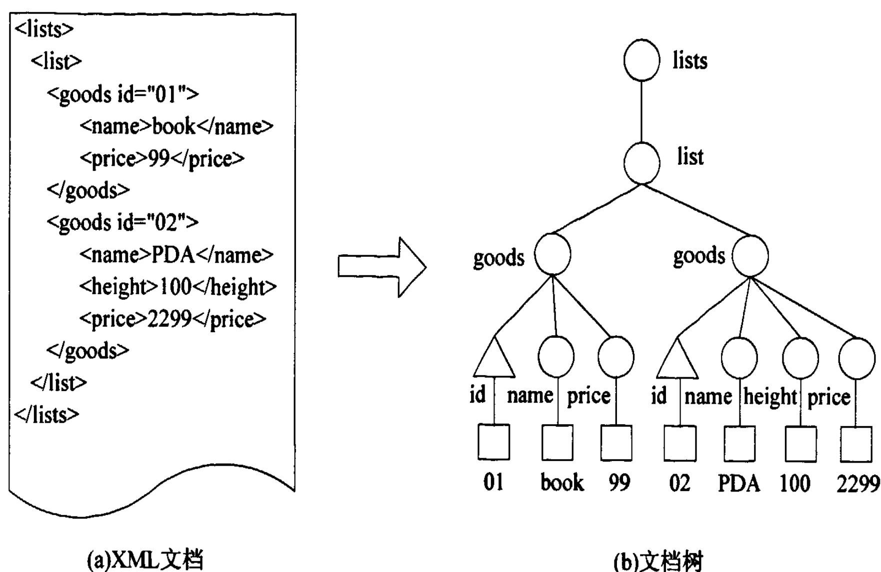
本章将介绍XML数据管理的相关知识：XML数据模型、XML查询语言、XML编码技术、XML索引技术、XML存储映射技术和XML查询技术。XML数据模型和XML查询语言是XML数据管理知识中的基础。不同的编码方案对XML存储、查询有很大的影响。由于索引的设计主要是为了提高查询效率，因此，设计索引时的主要考虑因素应该是查询。底层的存储表示形式对上层的查询和优化有着重要的影响，如何有效地存储XML文档已经是一个重要的问题。关于查询方面，目前的XML查询执行策略主要有两种：基于导航的XML查询执行策略和基于连接的XML查询执行策略。由于基于导航的查询策略重点在于结构概要类索引的设计，因此，本文重点介绍基于连接的查询策略。下面进行详细介绍。

2.1　XML数据模型

对XML数据进行有效地建模，正确、完全地反映XML数据的固有特征是对XML数据进行有效管理的基础。当前的研究工作大都建立于图模型或者树模型之上。图模型将XML数据看作一个复杂的图状结构，XML数据中元素之间、元素和属性之间的联系在图模型中被抽象成有向边。树模型将XML数据看作一棵有向的标签树，XML数据上元素之间、元素和属性之间的联系被抽象成树上的有向边。这两种模型都可以形象地表示数据，其中，树模型是描述XML数据最常使用的，因此本文的研究也是基于该模型。图2.1表示一个XML文档实例及其对应的树模型。 在树模型中，结点表示文档元素、属性和文本数据等，边表示两个结点之间的父子关系。一个结点的路径是从文档根结点开始到该结点所经历的标签序列，标签之间用“/”分隔。例如在图2.1中，height结点的路径是/lists/list/goods/height。

结点是树模型中最基本的概念，在树模型中共有七种类型的结点，分别是文档结点、元素结点、属性结点、文本结点、名字空间结点、处理指令结点和注释结点。一个XML文档有一个唯一的文档结点，称为根结点。元素结点依次指向其它的元素结点、属性结点、文本结点等。

为了得到有效的XML文档，还要确保文件中信息遵守的结构，即需要一种用来描述XML文档中信息结构的机制，这种机制定义了XML文档中元素的顺序、元



(a)XML文档　　　　　　　　(b)文档树

图 2.1 XML文档及其对应的树模型

素的嵌套关系和内容模型，并建立了文档数据的数据类型。DTD（Document Type Definition，文档类型定义）可以满足上述要求，它定义了XML文档的逻辑结构，可以用直接写入或以外部链接的方式与XML文档相结合。利用外部链接的方式，可以让多个XML文档共同使用一个DTD。DTD列出了可用在文档中的元素、属性、实体和符号表示法，以及这些内容之间可能的相互关系。例如，DTD可以确切的规定每个goods元素只有一个name子元素，只有一个price子元素。这些规则都是在DTD中定义的。图2.2表示图2.1(a)中XML文档对应的DTD。

```
<!DOCTYPE lists[
  <!ELEMENT lists (list+)>
  <!ELEMENT list(goods+)>
  <!ELEMENT goods(name, height*, price)>
  <!ELEMENT name (#PCDATA)>
  <!ELEMENT price (#PCDATA)>
  <!ELEMENT height (#PCDATA)>
  <!ATTLIST  goods id CDATA #REQUIRED>
  ]>
```

图 2.2 XML DTD示例

2.2 XML查询语言

现已提出的XML查询语言有很多种，比如XPath [24]、XQuery [25]和XML-QL [26]等，这些查询语言有一个共同点就是通过路径表达式来实现对文档的查询。本文主要介绍XPath语言 [24]。

XPath是由W3C创建的，它使用类似于URL的路径表示法，在一个XML文档中进行导航。XPath的主要部件是表达式，其中最重要的表达式是定位路径表达式，亦称为路径表达式，每个路径表达式都是由一个或多个路径步组成。路径表达式有绝对路径表达式和相对路径表达式两种：绝对路径表达式以正斜杠(“/”)开始，它是从文档树的根结点开始定位路径；相对路径表达式则直接从某个定位步开始定位路径。

以绝对路径表达式为例，XPath的语法为：

$$Path ::= /Step_1/Step_2/\cdots/Step_n \tag{2.1}$$

式2.1中路径步$Step$被定义为：

$$Step ::= Axis :: NodeTest[Predicate*] \tag{2.2}$$

式2.2中 “*Axis*” 轴指定了向前或向后的方向。每个轴都有一个基本结点类型：对于属性轴，基本结点类型是属性；对于命名空间轴，基本结点类型是命名空间；对于其它轴，基本结点类型是元素。在XPath中最常用的轴有：*child*、*descendant*、*parent*、*ancestor*。“*NodeTest*” 结点测试用来对轴中的结点进行测试：如果给定结点的测试值为true，则保留在结果结点集中；否则将它从结果结点集中删除。可以使用结点名称或结点类型进行结点测试。“*Predicate*” 谓词对原结果结点集进行筛选并生成新结果结点集。

路径表达式可以使用缩减句法，实际上缩减句法比完整句法更常用，因为它们更简洁。例如，地址路径goods/name是 *child* ::goods/ *child* ::name的缩写；地址路径goods[@id= “03”]是 *child* ::goods[*attribute* ::id= “03”]的缩写；地址路径lists//name是lists/ $descendant-or-self :: node()$ / *child* ::name的缩写。

实际使用中的XPath查询只用到XPath语言的一个子集或称之为片段，其中一个较常用的XPath片段包含：孩子轴(/)、后裔轴(//)、谓词和结点测试。下面举几个简单的例子说明XPath表达式的含义：

(1) //name 选择所有的name元素，不论name处在哪个层次。

(2) //goods[@id= "01"]/price 选择所有父元素是goods的price元素，并且goods元素的id属性值为01。

2.3 XML文档编码方案

根据编码之间的关系可以将编码方案主要分为两类：基于区间的编码和基于路径的编码。基于区间的编码方案利用XML文档的有序特点，通过某种访问顺序给文档中的每个结点赋予一个编码；基于路径的编码方案则利用XML文档的嵌套特点，给从文档根结点开始所能到达的每个路径上的结点赋予一个编码。

常见的编码方案设计在以下几个方面是不同的 [27]：

(1)所支持的结构关系，例如包含关系、文档位置关系。

(2)码的最大长度或平均长度。

(3)编码后的查询执行时间。

(4)插入操作导致的重新编码代价。

下面详细介绍几种编码方案。

2.3.1 基于区间的编码

区间编码方案的思想：树T中的每一个结点被赋予一个区间编码$< start, end >$，一个结点的区间编码包含它的后裔结点的区间编码，即若结点u是结点v的祖先，当且仅当：

$$start(u) < start(v) \wedge end(v) < end(u) \tag{2.3}$$

两个结点的区间编码之间关系：要么完全包含，要么完全不相交，不可能出现其它情况。

第一种区间编码方案是Dietz编码 [28]，其编码规则：树T 中的每一个结点被赋予一个由先序遍历序号和后序遍历序号组成的二元组$< pre, post >$。由于树T 中的一个祖先结点u在先序遍历中必然出现在它的后裔结点v之前，因此若u是v的祖先，则

$$pre(u) < pre(v) \wedge post(v) < post(u) \tag{2.4}$$

树T 中的任意两个结点之间的包含检测能够在常数时间内被计算。对于该编码方案，pre或$post$ 均可作为结点的唯一标识。一个Dietz编码示例如图2.3所示：

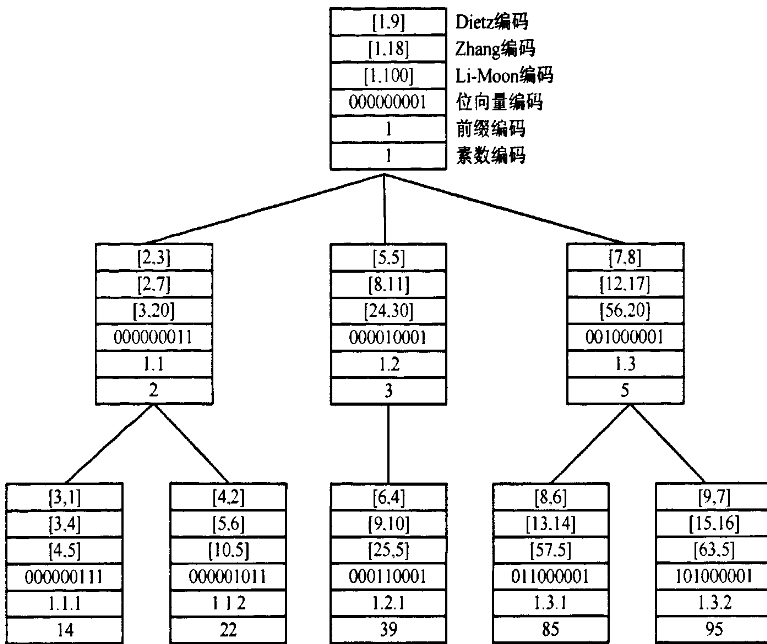


图 2.3 XML数据的编码方案

第二种区间编码方案是Li-Moon编码 [16]，它的编码规则：XML 文档树T 中的每一个结点被赋予一个二元组$< order, size >$，其中，*order* 是结点的扩展先序遍历序号，它的取值是非连续的，为结点的插入预留序号空间；*size* 为结点的后裔范围，对于该编码方案。$< order, size >$必须满足如下两点：

(1)结点v 和其父亲结点u，有

$$order(u) < order(v) \land order(v) + size(v) \le order(u) + size(u) \tag{2.5}$$

(2)兄弟结点u 和v，若在先序遍历中结点v 是结点u 的右兄弟，则

$$order(u) + size(u) < order(v) \tag{2.6}$$

对于树T中的结点u，一定满足

$$size(u) \ge \sum_{v} size(v) \tag{2.7}$$

式2.7中v 是结点u的所有直接孩子结点。因为事先预留了空间，所以Li-Moon 编码比Dietz 编码能够更好地支持文档的修改操作。对于该编码方案，*order* 作为结点的唯一标识。一个Li-Moon 编码的示例如图2.3所示。

2.3.2 基于路径的编码

基于路径编码方案则是利用XML文档的树形结构，给从文档根结点开始所能到达每个路径上的结点赋予一个编码。典型代表是前缀编码（Dewey编码）[29]，它直接将一个结点的父亲结点编码作为该结点编码的前缀。若文档树T中一个结点u的前缀编码为$d(u)$，则结点u的孩子结点v的前缀编码为

$$d(v) = d(u).x \tag{2.8}$$

式2.8中x是结点v在结点u的所有孩子结点中的序号。图2.3给出了一个前缀编码的示例。

对于前缀编码，要判断一个结点u是否是另一个结点v的祖先结点，只需判断字符串$d(u)$是否是字符串$d(v)$的前缀。前缀编码的另一个重要性质是字典有序：任意一个结点u，它的前缀编码$d(u)$大于（小于）它的左兄弟子树（右兄弟子树）中所有结点的前缀编码。因此，前缀编码不仅能有效地支持包含关系的计算，而且还能够有效地支持文档位置关系的计算。

2.3.3 其它编码

文献[30]提出了位向量编码方案，它的编码规则：树T中的每个结点被译码为一个n位向量，n是T中的结点数量，在某个位置i上的一个“1”位唯一地标识第i个结点；并且在一个自顶向下的编码方案中，每一个结点继承标识它祖先的所有位上的“1”。图2.3给出了一个继承祖先的位向量编码的示例。

文献[31]根据素数的性质提出用素数对XML结点进行编码。其编码规则：让中间结点的值为素数，而叶子结点的值为积。此时根据结点之间是否存在整除关系，可以判定祖先、后裔关系。但是由于每个结点只有一个父亲，因此每个结点关联两个值：一个是自身值，它是一个唯一的素数值；另一个则是结点的编码值，其值是父亲结点的编码值与自身素数值的乘积。图2.3给出了一个素数编码的示例。

2.4 XML数据索引技术

通过对XML文档树的周游遍历找出目标结点集，完全可以响应XPath表达式对XML文档的查询，但是这种方式效率比较低下，主要体现在：

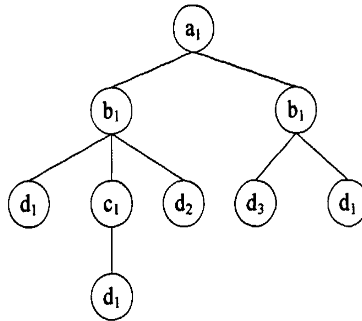


图 2.4 一个XML文档树示例

(1)访问许多无关的结点。如在图2.4 所示的文档树中查询$a_1//d_1$，按周游遍历的方式需要访问b_1，c_1结点，而这些结点并没有在表达式中出现，所以是无用的结点。

(2)重复访问相同的标记路径。如在图2.4 所示的文档树中查询$a_1/b_1/d_1$时，需要访问标记路径完全相同的两条结点路径，这种重复的扫描导致了效率的低下。

由此可见，没有XML数据索引时，虽然可以采用周游遍历的方式执行查询，但是效率低下，而XML数据索引则可以在不同程度上提高查询响应的效率。下面简要介绍几种索引：

2.4.1 结构概要类索引

结构概要索引 [32]是以XML 树结构中结点的路径信息为基础，采取某种约简方式，使得约简后的树结构只维护不同的路径信息，而不会存在具有相同路径的多个结点。结构概要索引采取标签有向图的结构，当基于结构概要索引进行XML 查询处理时，可以避免重复访问相同的标记路径问题。

DataGuide [33]是较早的XML结构概要索引，其基本思想：源文档中的每个标记路径在DataGuide中出现一次且只出现一次，而且DataGuide中的每个标记路径都是源文档中的路径标记。DataGuide实际上是一个与源文档等价的确定自动机，但由于与一个自动机等价的确定自动机有多个，因此源文档的DataGuide可能有多个。于是文中又规定一个索引结点对应的所有数据结点的射入路径相同，据此提出了强DataGuide。强DataGuide实际上是按照幂集法确定化一个自动机得到的确定自动机，因此一个源文档只对应一个强DataGuide。图2.5 是强DataGuide索引示例。

强DataGuide可以很好地响应查询，但它可能很大，甚至比源文档图大指数级：一方面是由于幂集自动机的构造决定的；另一方面由于在强DataGuide中，不同索引

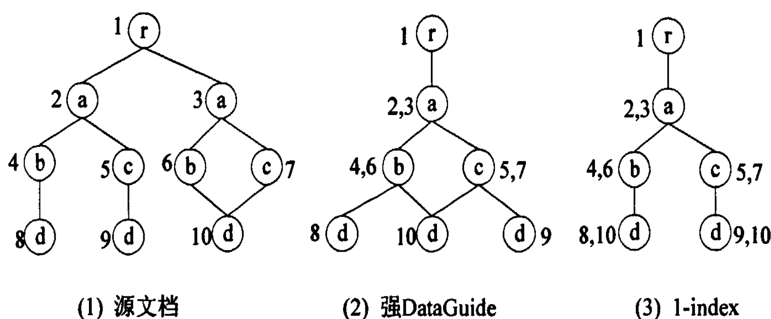


图 2.5 强DataGuide与1-index区别

结点的extent可能存在部分重叠。为了解决这种问题，1-index [34]提出了两结点“双拟”概念，其含义：若结点u，v具有相同的标签且结点u'是u的父结点，结点v'是v的父结点，则结点u'与v'双拟。

强DataGuide认为两个结点的某一条射入路径相同则它们是等价，而1-index则认为只有两个结点的射入路径集合相等才是等价（双拟）。从图2.5可以看出这一区别。对于图状数据库而言，1-index可能是不确定的，但是它的存储空间比强DataGuide小得多。

2.4.2 结点记录类索引

结点记录类索引本质上 [32]是将XML 数据分解为数据单元的记录集合，同时在记录中保存该单元在XML数据中的位置信息。主要分为两种：一种是结点序号索引，另一种是结点路径索引。

(1)结点序号类索引的基本思想：根据某种遍历策略得到由元素组成的序列，结点的标签在序列中具有唯一的标识，将序列与某指标集建立一一映射的关系，对应序列中某个标签就有唯一的序号；对任意两个具有结点序号信息的结点可以构建某种运算，该运算的结果可以表征结点间的结构关系。

根据映射指标集的不同,可分为赋以自然数 [35]、赋以局部编码 [36, 37]和赋以素数 [31]的方式. 在序列生成和结点序号赋值的基础上，可以构建不同形式的位置信息，近而形成不同的结点记录索引形式。本部分索引实际上与编码方案相关，详情见2.3节部分。

(2)结点的路径信息同样蕴含结点在XML数据中结构的信息：如果给定两结点的路径信息，同时预知两结点存在结构关系的情况下，就必然可以获知它们之间的结

构关系，即如果结点a的标签路径包含结点b的标签路径，那么在XML 树中a和b 之间一定具有祖先-子孙的结构关系，且b 是a 的祖先。所以基于路径信息获取结点的结构关系就成为另一种XML数据处理技术的思路来源，并演化出基于结点路径的XML索引。这类索引 [38–40]的核心技术是字符串的模式匹配。

文献 [38]提出了一种UB [41]多维树索引，其思想：先将XML文档映射到一个多维空间，维数是根据XML文档树中最长的路径中包含的边树来确定的；然后将XML文档的每个从根结点开始的路径映射为多维空间中的一个点，利用UB树来管理这些点。

2.5 XML数据存储映射

2.5.1 模型映射方案

对于模型映射，需要将XML文档模型（即文档树结构）映射为关系模式，关系模式表示XML文档模型的构造，对于所有XML文档都有固定的关系模式，因此它是XML模式（或DTD）无关的。具体来说，模型映射有如下两种：

1、边模型映射

边模型映射方法的代表有Edge方法 [42]和Monet方法 [43]。文献 [42]将XML文档看成是一个有序有向边标记图（称为XML图），用一个（或者若干个）关系表存储XML图的边信息和结点值。用来存储边信息的边表有三种方法：第一种是用一个表来存储图中所有边信息，这种方法称为Edge 方法；第二种是所有具有相同名称的边存放在一个边表中，这种方法称为Binary 方法；第三种是采用一个边表存储图中所有路径的边信息，该方法称为Universal 方法。

用来存储XML 文档值的值表有两种方法：第一种是不单独设计值表，将值和边存储在同一个表中，在边表中直接增加一个属性Value，用于存储叶结点的值，这种方法称为内联方法；第二种是为每一个可能的取值类型设计一个值表，该方法称为分离值表。

三种边表和两种值表方法合起来一共有六种存储模式，文中对边模型映射的六种基本的存储模式进行了分析，认为：Edge表方法优于Universal表方法，Binary表方法又优于Edge表方法；内联值表方法优于分离值表方法；Binary边表带内联值表的存储模式能获得最好的性能。

2、结点模型映射方法

通过设计若干个关系表来存储XML文档树的结点信息、结点值和结构信息（通过区间编码来译码结构信息，或者直接存储父亲／孩子结点对或祖先／后裔结点对）。典型的结点模型映射方法有XRel方法 [9]和XParent方法 [10]，文献 [44]提出的XISS/R方法也是基于结点模型映射。

XParent方法通过一个单独的Parent(*parentID, childID*)表来反映XML文档的模型结构，并根据内容和“结构与非结构”来划分边，同时将所有路径进行存储。因此XParent模式由四个关系表组成，如图2.6所示：

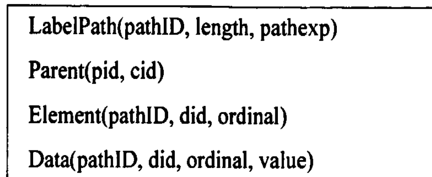

LabelPath(pathID, length, pathexp)
Parent(pid, cid)
Element(pathID, did, ordinal)
Data(pathID, did, ordinal, value)

图 2.6 XParent关系模式

Parent表存储的是父亲/孩子关系，因此，为了检查数据路径需要进行连接操作。为了加速处理，可以改用Ancestor表来存储祖先/后裔关系：

Ancestor(*did, ancestor, level*)

利用Ancestor表能快速检测结点之间的祖先/后裔关系，但是它比Parent表需要更多的空间，而且存在冗余信息，修改起来代价也更高。

XParent模式通过LabelPath和Parent表来支持标记路径和数据路径，因此XPrent模式既具有基于结点模型映射的特点，又具有基于边模型映射的特点。

2.5.2 结构映射方案

对于结构映射，需要将XML模式（或DTD）映射为关系模式，关系模式用来表示目标XML文档的逻辑结构（即XML模式或DTD），因此它是XML模式（或DTD）相关的。从XML文档的DTD推断XML元素应该怎样映射到关系中，这种方法能够利用到关系数据库的特性，如查询优化和并发性控制等。结构映射方法中比较有影响的研究有DTD方法 [11]、STORED方法 [12]、P-Schema [13]方法。

DTD方法根据DTD映射关系模式的存储策略：首先需要对DTD进行适当的简化，产生DTD图或元素图；然后再根据DTD图或元素图生成关系模式，有三种生成关系模式的方法，分别是基本内联法、共享内联法和综合内联法；最后将符合该DTD的XML文档数据装入关系数据库中。

STORED方法是较早用关系数据库来存储XML数据的。它结合了关系和半结构化存储技术，采用了启发式数据挖掘算法。从XML文档中提取有代表性而且置信度大于预先给定阈值的DTD，根据提取出的DTD自动生成关系模式，对于个别不符合所提取DTD结构的XML数据，STORED将它们另存在“OVERFLOW”表中，因而可以说STORED是一种无损存储方法，能实现数据的完整复原。但由于STORED用关系表和“OVERFLOW”表共同存储数据，当涉及结构变化较大的文档时，“OVERFLOW”表将变得很庞大，不但耗费大量的数据空间，而且查询操作也需要经常徘徊于关系表和“OVERFLOW”表之间，因而效率不高。

前两种方法都是根据XML数据的DTD转化为关系模式来对XML数据进行存储，但是，DTD有数据类型不丰富、不支持名域等缺点，在对关系数据的描述方面存在着不足。而XML Schema 有丰富的数据类型，并且支持用户对数据类型的扩展，基本上满足了关系模式在数据描述上的需要，P-Schema方法就是基于XML Schema的一种映射方法。

P-Schema方法的提出是为了解决XML Schema中重复数据的映射问题，提出了基于代价的XML关系存储技术LegoDB：首先根据XML Schema和XML文档实例提取统计信息，生成初始物理模式P-Schema，P-Schema可直接转换为关系模式；接着基于一组改写规则对初始物理模式进行不断的改写产生有限数量的物理模式集合，这对应于有限数量的关系存储方案；然后把XML数据统计信息映射成关系数据统计信息，XQuery查询映射成关系数据库的SQL查询后，对物理模式集合中的每个模式用传统的关系优化器进行查询代价估算，选择一个代价最小的物理模式，最后把它映射为相应的关系。

2.6 XML结构连接算法

目前，在结构连接方面提出了一系列有效的算法，大部分连接算法是基于归并的思想，充分利用XML数据的结构特点来减少连接的扫描代价；有些算法是基于非归并的思想，这部分算法主要是通过维护多个栈，将查询表达式看成是小枝模式树，整体上与给定的XML文档进行匹配。

已提出的结构连接算法主要有包含关系的结构连接算法，小枝关系的结构连接算法和文档位置关系的结构连接算法。本节主要讨论前两者：

2.6.1 包含关系的结构连接

包含关系的结构连接分三种：第一种是直接归并结构连接算法，有EE-Join/EA-Join算法 [16]、MPMGJN算法 [17]、IIMGJN算法 [45]和Tree-Merge算法 [46]等；第二种是基于缓存的归并结构连接算法，有Stack-Tree算法 [46]、XR-Stack算法 [47]、Hold-Join算法 [6]和Skip-Join算法 [48]等；第三种是基于区域划分的结构连接算法，有RangePartitioningJoin [5]。下面重点举例介绍前两种中的典型算法。

- MPMGJN算法

MPMGJN算法的基本思想：设参加连接的两个关系表为AList 和DList，则对外表AList 中的第一个结点a_1，首先在内表DList 中顺序搜索到可能与结点a_1进行连接的第一个结点（即$begin > a_1.begin$的第一个结点），称为扫描点，然后在内表DList 中从扫描点开始顺序扫描，对满足$begin < a_1.end$条件的所有结点d_j，再判断是否满足条件，若满足则产生连接结果结点对(a_1, d_j)；继续对外表AList 中的第二个结点a_2重复上面的步骤，直到外表AList 或内表DList 中的结点连接完毕。图2.7是MPMGJN算法内表操作示意图。

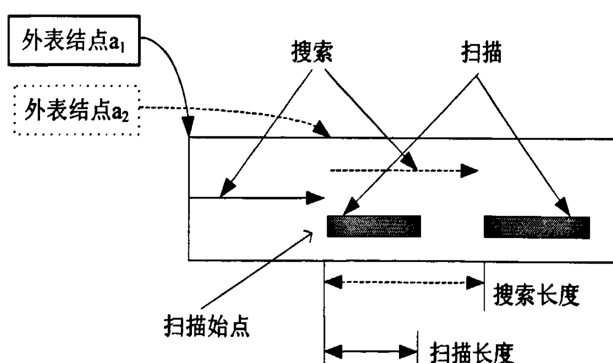


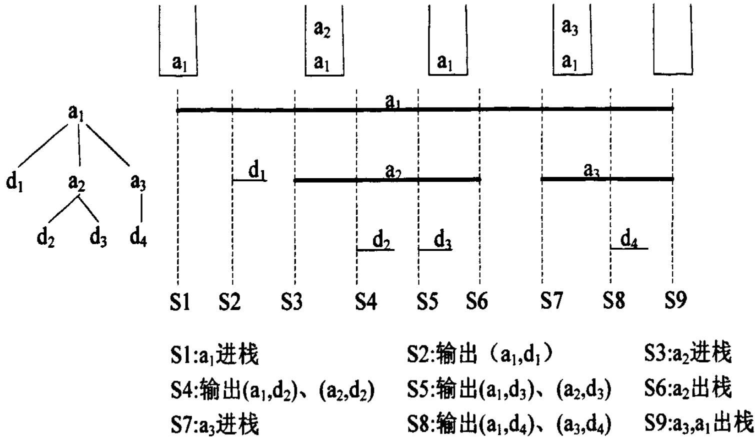
图 2.7 MPMGJN算法内表操作示意图

- Stack-Tree算法

Stack-Tree算法分为Stack-Tree-Desc（按后裔结点有序）和Stack-Tree-Anc（按祖先结点有序）两种，本文以Stack-Tree-Desc算法为代表。

Stack-Tree-Desc算法的基本思想：归并连接过程中，从AList中得到一个新结点a，如果它是目前栈顶结点后裔，则a被压入栈（若栈为空，a也被压入栈），从DList中得到的一个新结点d，如果它是目前栈顶结点的后裔，那么它也是目前栈中所有结点的

后裔，因此，结点d与栈中所有结点之间的连接结果可以被输出了；如果结点d不是目前栈顶结点的后裔，则可以将栈顶结点出栈了，并且判断d是否是新栈顶结点的后裔，直到栈为空。图2.8是一个Stack-Tree-Desc算法的执行过程，左边是一个XML文档实例，右边是算法执行过程，图中的查询语句是：a//d。



S1:a_1进栈　S2:输出（a_1,d_1）　S3:a_2进栈
S4:输出(a_1,d_2)、(a_2,d_2)　S5:输出(a_1,d_3)、(a_2,d_3)　S6:a_2出栈
S7:a_3进栈　S8:输出(a_1,d_4)、(a_3,d_4)　S9:a_3,a_1出栈

图 2.8 Stack-Tree-Desc算法执行步骤

2.6.2 小枝模式的结构连接

目前的研究将小枝模式看成一个整体进行模式匹配，使得算法的效率有大幅度的提高。根据是否需要归并操作，小枝模式匹配分为两种：第一种是基于归并的小枝模式匹配算法，有TwigStack算法 [18]、TSGenetic+算法 [49]、iTwigJoin算法 [50]、TwigStackList算法 [51]和TJFast算法 [19]等；第二种是非归并小枝模式匹配算法，有Twig2Stack算法 [20]、TwigList算法 [21]，TwigNM算法 [52]等。下面重点介绍Twig2Stack算法思想：

- Twig2Stack算法

Twig2Stack算法的基本思想：后序遍历文档，对于元素e，当且仅当它满足以查询结点E为根的小枝时，将其压入层次栈$HS[E]$（E是与e对应的查询结点），由于是后序遍历，因此在访问e之前，已经访问过了e的后裔；在检查e是否匹配E的子小枝时，只需要检查一个查询步，即$E \to M$，其中M是E的子结点，在检查查询结点或者压入元素时，要注意维护层次栈的结构。

图2.9是一个层次栈构造示例，每个层次栈$HS[N]$都由一个栈树序列组成，

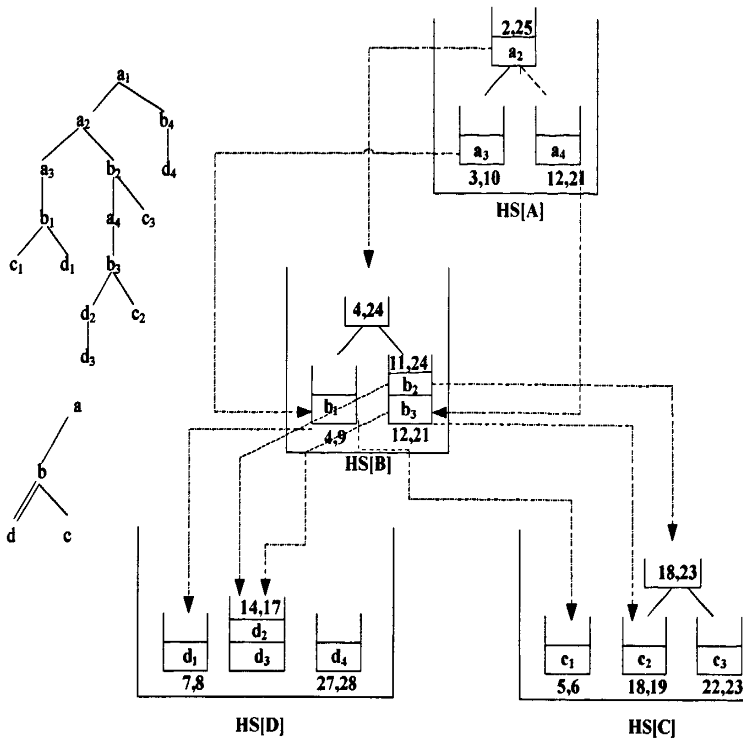


图 2.9 层次栈编码结构示例

一个栈树ST是一个有序树，其每一个结点是一个栈S，如$HS[A]$包含一个栈树，而$HS[D]$包含3个栈树。每个栈S包含0个或多个文档元素，栈中每个元素是其之下元素的父亲。

第三章 数据管理方案设计

本章主要介绍本课题涉及的XML数据管理方案，由于目前数据管理还有许多不完善的地方，本文就是从映射方案中不能有效支持更新的角度来阐述XML文档映射方案，针对结构连接中的小枝模式查询方面，本文在引入了非归并输出结果解的思想下，介绍了小枝查询方案。

3.1 问题提出

XML数据可能以两种形态存在：

(1)存在于一颗内存树中。XML文档被读入内存并建立一棵形如DOM树，然后就可以周游这棵树来处理查询。

(2)存在于磁盘上的若干表或者文件中。XML文档在解析时已被分解到若干表或文件中，查询XML时需要将数据文件和索引文件的某些数据块读入内存进行处理。

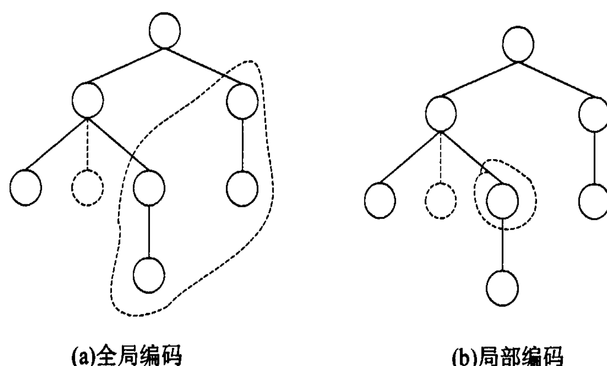
第一种方式虽然查询简单，但是严重依赖于内存大小，只能处理比较小的XML文档。对于一般的XML文档需要使用第二种，将XML文档分解到磁盘存储。

XML数据的存储技术可以分为原生XML数据存储和基于关系的XML数据存储。原生XML数据存储试图在物理存储格式上保持XML文档中独特的结构特征，XML数据无需经过映射，物理结构与逻辑结构一致，不会导致信息丢失，但在技术上还是非常不成熟的，研究的比较少。而基于关系的XML数据存储将XML数据分解到若干关系表中，在这种存储方式下，可以充分利用关系数据库的成熟技术来管理XML数据，XML查询操作转化为一系列关系查询操作，在技术上成熟可行。但由于XML数据大都是半结构化数据，而关系数据都是结构化数据，因此需要映射。

映射需要有编码方案作为基础，但映射之后，XML 数据也会面临插入、删除等更新问题。数据一旦更新，编码也要作相应的调整，才能保证基于这个编码的各种索引和查询算法的正确性。在编码的更新方面，针对性的研究还不多，主要是在研究编码方案的同时顺便考虑更新问题 [53–57]。可以支持更新的几种编码有：

Li-Moon编码作为全局编码，可以在一定程度上实现XML文档的更新。< *order*, *size* >中*order*的取值是非连续的，目的是为结点的插入预留序号空间，但是预留空间很大则造成存储浪费，相反容易导致过多的重新编码。

对结点的编码除了使用全局标识，还可以使用局部标识。使用局部标识，比如使用兄弟结点序号对XML文档树中的结点进行编码，其优缺点正好与全局编码相反。文献 [58]对其优缺点进行了详细了分析，文中给出了将实数作为序号的指标集的思路：利用实数可表示无穷精度的特点满足插入数据带来的结点序号的扩张，但是，由实数计算带来的结构关系计算的代价是提高查询处理性能所不容忽视的。图3.1表示当插入一个新结点时所引起结点编码的变化，虚线结点表示新插入的结点，虚线区域表示要重新编码的结点范围。



(a)全局编码 (b)局部编码

图 3.1 全局编码、局部编码更新文档示意图

本课题提出的编码方案优势在于可支持文档的动态更新，且具有局部编码更新的特点，在经过映射之后，当涉及文档的动态更新时，不需要对已存在的结点进行过多的编码就可以及时的反映给用户，这样节省了二次编码的时间，提高了映射方案中文档更新的效率。

XML查询处理中的连接是指以两个或多个元素列表为输入，输出满足一定结构关系的符合查询语义的结点对或者结点向量列表，与关系查询中的连接条件一般是关于值的约束不同，这里的连接条件一般是关于结构的约束，因而，此种连接常称为结构连接。目前关于结构连接的研究主要关注在多元的结构连接，即小枝模式的结构连接，小枝模式的结构连接将一个小枝模式视为整体，无需分解和逐步连接，省去了中间的许多开销。

但目前的小枝连接算法还主要集中在归并的结构连接，当参与连接的结点数目不多时，归并连接算法运行效率较好，但当参与连接的结点数目非常多时，归并连接要耗费很多的时间用于分支解的合并，以保证输出的整体解正确。例如图3.2中，查询树Q对应的查询表达式为$a[//b]//c[//d]$，在归并的连接算法中，(a_2,b_1)满

足$Q1=a//b$，(a_3,c_1,d_2)满足$Q2=a//c//d$，但他们合并起来并不满足Q，这就产生不必要的路径归并，降低了连接效率。

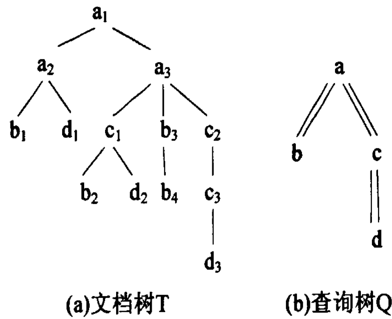


图 3.2 XML文档树和查询树

本课题设计的小枝查询方案不需要归并分支路径解，而是直接输出最终生成的链表中元素，即为满足查询请求的结果。不仅在时间方面节省中间结果归并的开销，在空间方面也只是用到了部分栈和链表的结构，并不占用太多的内存空间。

3.2 设计目标

本文提出的方案可以支持XML数据到关系数据库的映射、生成数据的发布处理、文档的动态更新、小枝模式查询等功能。通过这些功能的实现就可以较好解决前面提到的问题。主要设计目标的详细描述如下：

(1)XML数据到关系数据库的映射

此部分的功能是根据一定规则将用户给定的XML文档数据映射到关系中，此处映射要求对所有的XML文档都能进行，即映射与XML模式无关。同时要求在映射过程中能保持文档中结点的顺序，且映射中的优化策略对用户是成熟和透明的。

(2)生成数据的发布

此部分的主要功能是提供一种方法，当用户请求被响应时，能在由映射生成的数据基础上进行处理，将XML数据进一步处理为用户所要求的数据形式发布。

(3)文档的动态更新

此功能是映射方案的优势所在，由于许多映射策略并没有考虑更新的功能，当用户修改XML文档中数据时（如简单地添加一个元素结点），不得不对XML文档进

行重新映射。本文提出的映射策略事先考虑了更新的需求，当更新时，本功能只需对改动的结点进行回写即可。

(4)小枝模式查询

这一部分提供给用户快速的查询响应，当用户提出复杂的路径查询时，此查询设计不仅能够实现功能上的正确响应，而且能以较快速度回应。

3.3 模型方案

具体来说，将XML文档进行映射有如下的方法和策略：

(1)将XML文档看成是一个有序有向边标记图，设计一个（或若干个）关系存储XML图中边信息和结点值，该策略属于边模型映射，称为边模型映射方法。

(2)设计若干个关系来存储XML文档树中结点信息，结点值和结构信息（通过区间编码或路径串编码），该策略属于结点模型映射，称为结点模型映射方法。

上述方法各有优缺点，边模型映射方法仅仅维护XML文档树的边信息，因此，为了处理用户的查询，需要连接边表形成一个路径。单一边表方法（Edge边表方法）非常简单，仅仅维护边标记，但当计算路径表达式时需要大量的连接操作。Monet边表方法需要建立大量的边表，但是对于简单路径表达式的计算却是十分有效的。边模型映射有一个共同的缺点：对于支持需要改变XML文档结构（增加或删除路径）的应用，是非常困难的。结点模型映射的优点是：通过非等值连接，能够容易地判断两个结点之间的包含关系。但由于关系数据库中没有特殊的索引机制来支持非等值连接，其代价也不可忽视。

通过上述的分析，自然想到分别利用两种模型映射的优点来解决实际问题，比如，为了使映射过程速度加快，可以借鉴结点模型中的思想来快速判断结点之间的关系，本课题提出的映射方案就是采用路径串的编码方法来快速定位结点之间的祖先/后裔关系。由于新的映射方案设计重点考虑更新功能，如何使重新编码的结点范围最小是追求的目标，由于边模型映射是分别存储路径边信息的，所以自然想到加入边模型的思想。

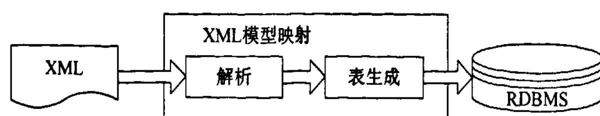


图 3.3 映射方案设计图

第四章介绍的存储方案XSC就是结合上述两种模型思想来设计的，在借鉴结点模型映射XRel方法的基础上，加入边模型映射思想。图3.3是映射方案的概况图。

3.4 小枝查询方案

Twig是XML处理中一个重要的查询模式，其主旨就是搜索XML树得到满足树状结构的查询模式的结果，当前该研究方向的参照主要是查询匹配问题，给定具有n个结点的Twig查询模式Q和一个XML数据库G，Q在G中的一个匹配是指Q的结点与G的结点之间存在如下的映射关系：

- 对应于查询结点的目标结点所含的数据必然满足查询结点上的谓词条件；
- 目标结点间的结构关系与查询结点组合间的结构关系必须一致，包括父亲-孩子关系和祖先-子孙关系。

Twig整体匹配算法在求解时是将查询作为一个整体来处理，在求解方式中影响计算性能的因素主要有两个：一个是结构连接算法的效率；另一个就是流式处理Twig。显然，如果需要结构连接计算的数目少，整体的计算则必然会有较好的性能表现。

(1)提高结构连接操作性能

有两种方式来实现：一是设计新的连接算法，减少传统关系数据库中连接算法中构建全部连接记录之后过滤出所需结果的过程，采取按照所需进行连接的方式；第二种方式是在结构连接中引用XML索引结构信息，达到减少连接记录数目的目的。常用的XML索引形式为结点记录类中序号对索引。

(2)流式处理

流式处理方式的思想：首先依照Twig查询树设置相应的堆栈，每个堆栈对应Twig查询树中相应的结点，堆栈的次序与前序遍历Twig查询树次序相同；然后，顺序扫描XML数据，将扫描中遇到的对应于Twig查询树标签的元素顺序地压入相应的堆栈，当遇到对应于Twig查询树叶结点标签的元素时，回溯经过的相应堆栈，这样，符合Twig查询模式结构关系的元素序列即为满足查询请求的结果。

本课题提出的小枝模式查询方案TwigWM吸收了上述两种思想：TwigWM中引入了结点记录类的索引（本文采取的是路径串编码），当实行算法匹配时，能够减少重复的结点扫描，并且在算法运行过程中，可以通过索引定位到合适的结点位置；另

外TwigWM通过设置部分栈和链表结构来实现流式处理，将XML文档中的元素组织成标签流，直接对标签流进行扫描，减少顺序遍历XML文档树的时间。

第四章 可支持更新的XML存储方案XSC

基于目前存储方案在XML文档更新方面支持得不够友好，本章提出了XSC(XML Storage sChema)存储方案。XSC方案基于一个新的编码：先序编号和路径串编码的结合。之所以这样设计在于：更新结点时，只需对兄弟结点进行少量编码，就可正确的实现XML查询和发布。XSC方案在具体存储时将元素表分割设计，目的是加快XML数据装载的效率。

4.1 引言

如何对XML数据进行有效地存储已成为一个研究的热点，已提出的模型映射方法具有一定的灵活性，因为当使用模型映射方法对XML文档进行存储时，不但存储了XML文档本身的数据信息，同时还存放了其相应的模式信息，因而可以实现异构系统间数据的无缝集成，根本不需要考虑文档的模式信息，即使源文档的结构发生了一定的变化也不会影响系统间数据的交换。结点模型映射又是较常用的方法，如XRel方法 [9] 和XParent方法 [10]。

数据被存储一段时间后往往需要修改以反映现实世界的真实情况，一个来自用户需求的修改、初始信息的错误或者增量存储的要求都需要对原始数据进行修改，因此，XML要想成为通用数据表示方法和共享格式，高效的更新功能是不可或缺的。但是，已提出的大多数存储方法只能提高对XML数据的有效存储，当XML数据需要频繁地更新时，现有的编码方案需要将XML树中的大量结点重新编码，以维持XML文档的顺序，这样便产生了很高的更新代价，形成了系统性能的瓶颈。

目前在编码的更新方面，区间编码采用预留编码空间的方法，针对不同特征的XML 数据和应用环境提出了一整套预留算法，但是，这种方法当涉及频繁的更新时，很容易将预留的空间用完，导致必须对整个XML文档重新编码。前缀编码只需要将插入结点的右兄弟结点及其子孙结点重新编码，所以，在对更新操作的支持上，前缀编码方案重新编码的结点数量小于区间编码的方案。

本章提出的存储方案XSC是在借鉴结点模型映射XRel方法的基础上，加入边模型映射思想。XSC在编码方面采用前缀编码，同时又考虑将前序遍历的序号作为标识结点的一部分，这样可以避免当插入新结点时对其余结点进行过多的二次编码。

4.2 XSC映射实现

由于本章提出的XSC模型映射是在借鉴XRel映射思想基础上考虑支持更新功能，并且在实验分析一节会将两种映射进行比较，因此，下面先简单介绍XRel映射。

4.2.1 XRel映射思想

XRel方法将XML文档映射到四个表中，分别是Path、Element、Text和Attribute，其中，Path表描述路径信息，Element、Text和Attribute表分别用于描述文档中的元素、属性和文本信息。XRel模式组成如图4.1所示：

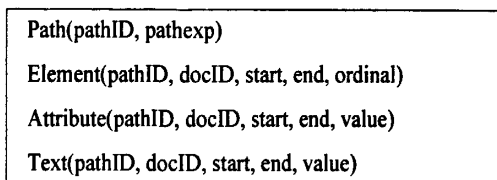


图 4.1 XRel关系模式

其中，*pathID*为标记路径的标识，*pathexp*属性存储标记路径，为了实现路径表达式的字符串匹配操作，将标记路径中的“/”替换为“#/”进行存储。*start*和*end*界定结点的位置，*ordinal*反映该结点在兄弟结点中的位置序号。对于Element、Attribute和Text表，键是(*docID, start*)，*pathID*是外键。

XRel的最大优点在于它与XPath标准的紧密结合，从而能够对基于XPath的查询给予相当好的性能支持，但同时它也存在很大的不足之处，一个简单的修改结点名字的操作，都需要相当复杂的操作。

4.2.2 XSC映射策略

XSC是一种基于不同结构的XML文档映射方法，无论文档是否符合特定的模式，都可以映射到XSC存储模式中。其基本思想：对XML文档树中的所有对象进行先序遍历（即深度优先遍历），产生这些对象的先序遍历序号，并将这些对象的先序遍历序号按升序进行列表。XML文档树中的对象包括元素结点、属性结点以及文本结点。具体的处理方法如下：

(1)对于所有具有相同元素标记名Tag的元素结点建立一个先序列表Tag_Elem，该列表中的每一个记录是标识该结点的一个六元组(*DocID, Order, ParentOrder,*

LeftOrder, RightOrder, Path)。其中，*DocID*是该结点所在文档的文档标识；*Order*是该结点的先序遍历序号，它是一个文档树中所包含的所有元素结点按先序遍历所产生的先序遍历序号；*ParentOrder*表示该结点的父亲元素结点的*Order*，以反映结点之间父亲/孩子关系；*LeftOrder*(*RightOrder*)表示该结点的左（右）兄弟结点的*Order*，以反映结点之间的兄弟关系，同时也用于实现更新操作；*Path*表示从根到当前结点的路径，可以用来检测结点之间的祖先/后裔关系，同时也用于实现路径表达式的字符串匹配和更新操作。

(2)对于所有属性结点建立统一的先序列表Attribute，该列表中的每一个记录是标识该结点的一个五元组(*DocID, Order, AttrID, AttrName, AttrValue*)。其中，*DocID*是该结点所在文档的文档标识；*Order*表示属性结点所属元素结点的先序遍历序号；*AttrID*是属性结点的序号；*AttrName*是属性结点的名称，*AttrValue*是属性值。

(3)对于所有文本结点建立统一的先序列表Text，该列表中的每一个记录是标识该结点的一个四元组(*DocID, Order, TextID, TextValue*)。其中，*DocID*是该结点所在文档的文档标识；*Order*表示文本结点所属元素结点的先序遍历序号；*TextID*是文本结点的序号；*TextValue*是文本值。

根据先序列表，XML文档的XSC关系存储模式被设计为如图4.2所示，有以下注意点：

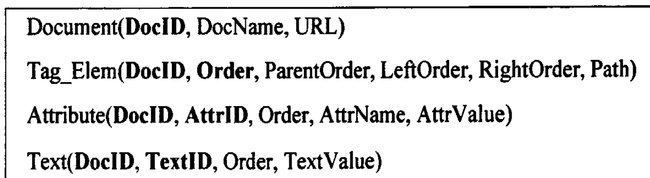

Document(**DocID**, DocName, URL)
Tag_Elem(**DocID**, **Order**, ParentOrder, LeftOrder, RightOrder, Path)
Attribute(**DocID**, **AttrID**, Order, AttrName, AttrValue)
Text(**DocID**, **TextID**, Order, TextValue)

图 4.2 XSC关系模式

(1)关系的码属性被用粗体表示。将所有的关系都按码属性建立聚集索引，以加快查找的执行速度。

(2)对于Tag_Elem表，为了有效地检测结点之间的祖先/后裔关系，对*Path*属性存储的路径串进行变换，将路径串中的路径步替换为路径步与相应元素结点的先序遍历序号的连接，例如（图4.3）：Title_Elem 表中有一结点的标签路径为/pub/book/title，存储时*Path*属性值为#/pub-1#/book-2#/title-3。

图4.3是XML文档应用XSC进行映射的文档树模型。为了简化，图中未标出文本结点，且图中的结点编码只显示*Order*和部分*Path*的值。

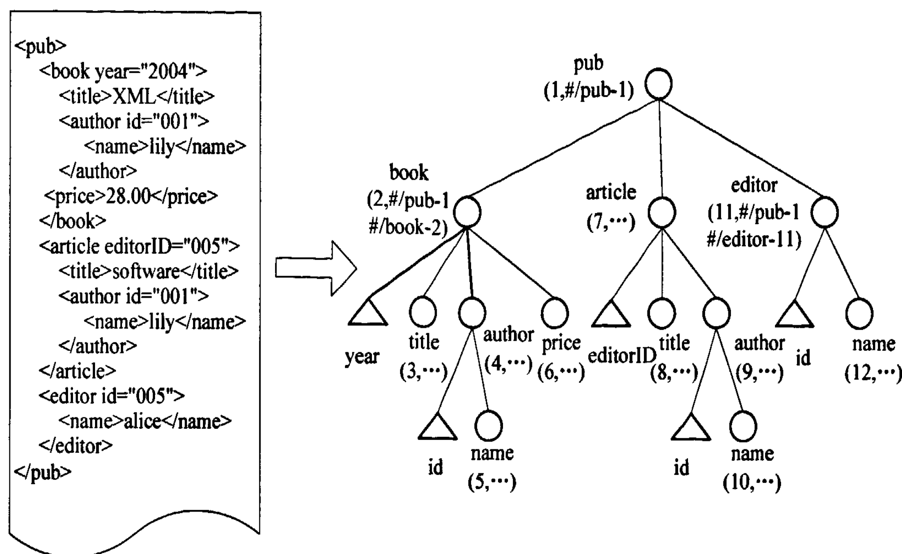


图 4.3 XSC映射数据模型示例

4.2.3 XSC映射算法

XSC的映射算法如下：

输入：XML文档信息和文档结构树*T*

输出：关系表Document、Tag_Elemnt、Attribute、Text中的数据

步骤：

Step1：将XML文档的名称和URL存入关系表Document中的*DocName*和*URL*的相应属性列中，并分配唯一的文档序号存入*DocID*属性列中；

Step2：深度优先遍历此文档结构树*T*中的每个结点V_e，对每个结点进行如下数据存储：

if（V_e是元素结点）{

假定元素名称为Tag，判断数据库中是否已建立Tag相应的元素表Tag_Elem，若未建立，先建该表；若已建立，将元素名称、先序遍历的序号，路径信息分别存入Tag_Elem表中相应的属性列中，若此元素是根结点元素，则令*ParentOrder*属性列值为-1（表示此元素结点无父结点），否则将父结点元素的先序序号存入*ParentOrder*属性列中，同时将元素结点的左兄弟结点的先序序号存入*LeftOrder*属性列中，将该元素结点的先序序号存入左兄弟结点的*RightOrder*属性列中。

} else if（V_e是属性结点）{

将此属性结点的名称、属性结点的值、所属元素结点的先序序号存入Attribute表中相应的属性列中，由于一个元素结点可以拥有多个属性结点，所以分配一个序号*AttrID*用于唯一标识同一个元素结点下的属性结点，*DocID*属性列的值从Document表中导入。

} else if（V_e是文本结点）{

将此文本结点的值、所属元素结点的先序序号存入Text表中相应的属性列中，并分配一个序号*TextID*用于唯一标识同一个元素结点下的文本结点，*DocID*属性列的值从Document表中导入。

}

Step3：$V_e = V_e \rightarrow next$，若存在下一个结点，则转向Step2，否则结束。

Text

docID	TextID	Order	TextValue
1	1	3	XML
1	2	5	lily
1	3	6	28.00
1	4	8	software
1	5	10	lily
1	6	12	alice

图 4.4 XSC映射中文本表

Attribute

docID	AttrID	Order	AttrName	AttrValue
1	1	2	year	2004
1	2	4	id	001
1	3	7	editorID	005
1	4	9	id	001
1	5	11	id	005

图 4.5 XSC映射中属性表

图4.4 是图4.3中XML文档实际存储的文本关系数据；图4.5 是图4.3中XML文档实际存储的属性关系数据；图4.6是图4.3 中XML文档实际存储的元素关系数据，若某个元素结点无左（右）兄弟结点，则其*LeftOrder*(*RightOrder*)值为-1。

Title_Elem

docID	Order	ParentOrder	LeftOrder	RightOrder	Path
1	3	2	-1	4	#/pub-1#/book-2#/title-3
1	8	7	-1	9	#/pub-1#/article-7#/title-8

Author_Elem

docID	Order	ParentOrder	LeftOrder	RightOrder	Path
1	4	2	3	6	#/pub-1#/book-2#/author-4
1	9	7	8	-1	#/pub-1#/article-7#/author-9

Name_Elem

docID	Order	ParentOrder	LeftOrder	RightOrder	Path
1	5	4	-1	-1	#/pub-1#/book-2#/author-4#/name-5
1	10	9	-1	-1	#/pub-1#/article-7#/author-9#/name-10
1	12	11	-1	-1	#/pub-1#/editor-11#/name-12

图 4.6 XSC映射中部分元素表

4.3 XML文档重组

在XML重组背景下，以XML形式重组数据的过程如图4.7所示，其中，重组过程需要一个XML转换层。在XML转换层中，有一个视图构造器，它根据视图定义，构造XML视图，XML视图中的元素和关系表中的数据对应关系保留在XML到关系的映射中。由于在这一背景下，数据实际存储在关系数据库中，因此不存在构造关系模式和载入数据的问题。当用户针对XML视图提出查询的时候，先经过查询翻译器进行翻译。

XSC按照结点的类型对文档进行分解，分别存储到对应的关系模式中，存储的关系表中包含了所有与结点相关的信息。按照图4.7思想，为了加快查询与重组文档效率，基于元素结点对应的关系表创建视图View(*DocID, Order, Name, ParentOrder, LeftOrder, RightOrder, Path*)，由视图中结点的编码信息可得如下关系性质：

性质一：视图中任意两个元组n_1和n_2，如果$n_1.DocID = n_2.DocID \wedge n_2.path$ is prefix of $n_1.path$，则n_2在XML树中对应的结点是n_1对应的结点的祖先结点。

性质二：视图中任意两个元组n_1和n_2，如果$n_1.DocID = n_2.DocID \wedge n_2.Order = n_1.ParentOrder$，则$n_2$在XML树中对应的结点是$n_1$对应结点的父亲结点。

性质三：视图中任意两个元组n_1和n_2，如果$n_1.DocID = n_2.DocID \wedge n_1.RightOrder = n_2.Order(n_1.LeftOrder = n_2.Order) \wedge n_1.ParentOrder = n_2.ParentOrder$，则$n_2$对应的结点是$n_1$对应的结点的右（左）兄弟结点。

根据上面的性质以及存储关系中的*DocID*、结点的编码信息能有效地对

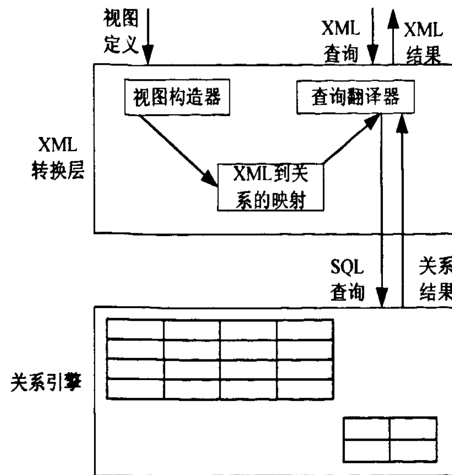


图 4.7 以XML形式重组数据过程

源XML文档进行恢复。

XSC文档重组算法如下：

输入：视图View、关系Attribute、关系Text中的各个元组，且所有元组具有相同的*DocID*

输出：重组的XML文档

步骤：

Step1：在View视图中找到*Order*值是“1”的元组，将它构造为XML文档的根元素，并在Text和Attribute表中找到*Order*值也是“1”的元组（若存在的话），构造根元素的文本和属性；

Step2：在View视图中找到*ParentOrder*值是“1”的元组，将其构造为根元素的子元素，并保存此元组中属性*Order*的值，以此*Order*值通过Text和Attribute表构造出元素的属性和文本；

Step3：找出View视图中*ParentOrder*取值等于上一个步骤中保存的*Order*值的元组，将这些元组构造为上一个步骤构造出元素的子元素，并保存元组中*Order*属性的值，以此*Order*值通过Text和Attribute表构造出元素的属性和文本；

Step4：递归的运行Step3，直到构造完所有元素的子元素为止；

Step5：输出结果XML文档。

4.4 XML查询处理

根据图4.7所示，XSC对用户提交的路径表达式的查询要转化成SQL语句，再提交给关系引擎进行查询。下面结合XSC存储模式介绍将XPath的一个子集翻译到SQL的方法。

对于查询*Q*1：//regions//description，由于View视图中Path属性记录了从根结点元素到每个结点元素的路径信息，因此，像*Q*1这样的路径表达式可以通过字符串的匹配操作来实现，如可以翻译如下（假定视图名为*View*）：

Select *View.Order*, *View.Name* From *View*,
(Select *Order*, *Name* From *View* Where *Name*= 'regions') *View*2
Where *View.Path* LIKE '%#/ *View*2.*Name* – *View*2.*Order* #%'
and *View.Name*= 'description'

对于稍微复杂的查询，如*Q*2：//regions[africa/quantity='1']//item/description，就不能直接这样翻译了。*Q*2中包含有三层意思：一是要存在路径//regions/africa/quantity，二是要存在路径//regions//item/description，三是路径//regions/africa/quantity对应的元素结点的文本值为1。在翻译的时候要把这三层意思考虑进去。由于要翻译的两条路径在查询里面被谓词分割成几个碎片，因而，对于这样的查询，需要将查询分割，然后将若干片段里的路径组合，形成一个完整的路径，再进行翻译。文献 [9]提供了两个步骤来完成这个查询，第一步是将此查询转换为XPathCore 图，例如*Q*2对应的XPathCore图如图4.8所示，其中每个结点代表一个表达式，或者一个谓词，或者文字、数字、布尔值。例如结点n_1代表了表达式//regions，n_5代表了表达

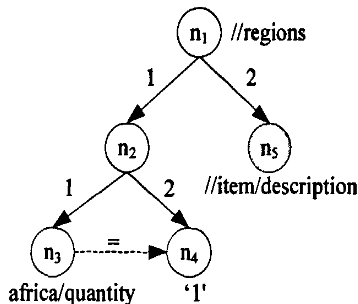


图 4.8 *Q*2的XPathCore图

式//item/description，n_2代表了一个谓词[africa/quantity='1']，阴影结点是输出结点。

图中有两类边：一类是用实线表示，边上标出的数字表示孩子结点的序号；另一类用虚线表示，这种边代表了运算，边上标出的是运算符。第二步就是生成SQL查询。具体算法见文献 [9]。

图4.8实际上将*Q*2分割成了若干片段，分割时的一个原则是，每个片段中的表达式应尽量是最长的简单路径表达式（即不含谓词）。得到XPathCore图后，进行路径串联，即将从根结点到某一结点的路径上所有表达式的结点的值串联起来，得到完整的路径。

4.5 动态更新XML文档

XML文档的更新一般可以分为3种操作：插入结点，删除结点和修改结点内容。删除和修改很容易实现，以下对在XML文档中插入新结点的几种不同情况分别讨论（令新插入的结点为*n*，且其*Order*的取值在视图中唯一）：

(1) 左右兄弟同时存在。设左兄弟结点为n_1，右兄弟结点为n_2 。在此情况下令

$n.ParentOrder = n_1.ParentOrder$，$n_1.RightOrder = n.Order$，

$n.LeftOrder = n_1.Order$，$n.RightOrder = n_2.Order$，$n_2.LeftOrder = n.Order$。

(2)右兄弟存在，左兄弟不存在。设右兄弟结点为n_1。在此情况下令

$n_1.LeftOrder = n.Order$，$n.RigthOrder = n_1.Order$，$n.LeftOrder = -1$，

$n.ParentOrder = n_1.ParentOrder$。

(3)左兄弟存在，右兄弟不存在。设左兄弟结点为n_1,在此情况下有

$n_1.RightOrder = n.Order$，$n.LeftOrder = n_1.Order$，$n.RightOrder = -1$，

$n.ParentOrder = n_1.ParentOrder$。

(4)在无孩子的结点下插入。设无孩子的结点为p，在此情况下有

$n.LeftOrder = n.RightOrder = -1$，$n.ParentOrder = p.Order$。

关于结点*n*的*Path*属性赋值略。从以上分析可见，新结点插入操作引起改动很少，并不需要其余结点过多的重新编码。

图4.9显示了上面几种不同情况，其中结点编码显示了(*Order, Path, LeftOrder, RightOrder*)的值，黑色结点表示新插入的结点，更新引起改动结点的编码属性用下划线表示，Path赋值略。

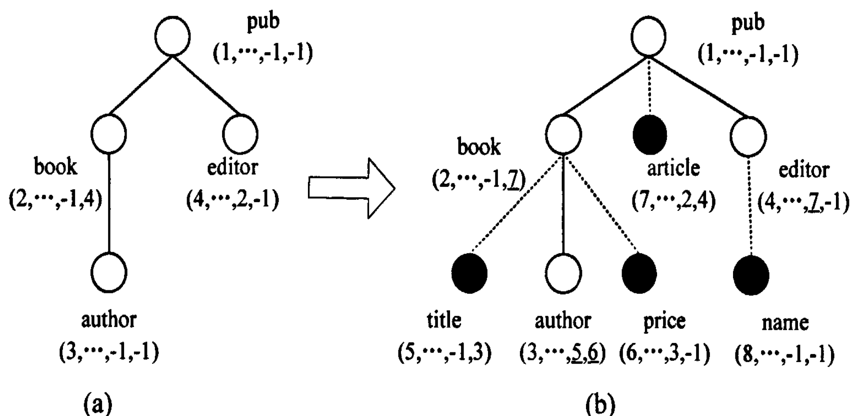


图 4.9 插入结点前后的对比

4.6 实验分析

为了准确分析XSC的性能，从实验上将XSC方法与XRel方法进行对比。下面主要考虑文档存储与路径查询效率方面，采用XMark [59]作为实验数据集。实验的硬件环境：CPU为Pentium 4，主频为2.8GHz，内存为512MB；软件环境：操作系统为Windows XP，开发语言为Java，关系数据库管理系统为Oracle 10g。对每组数据分别测试5次，取平均值。

4.6.1 XML存储性能分析

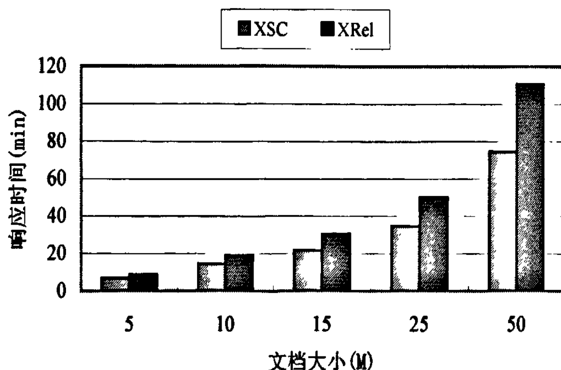


图 4.10 两种方法存储时间的比较

图4.10显示了两种方法在存储时间方面的比较，测试文档对应的标签对分

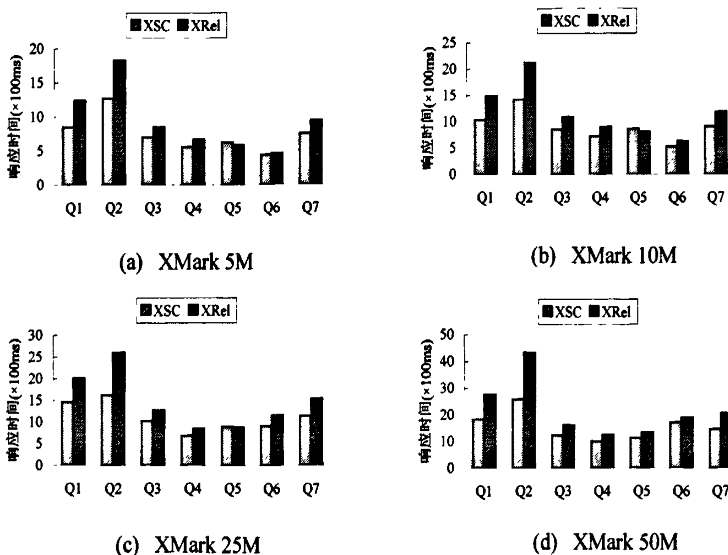
别是（单位：个）：74369，151289，223108，368176，752126。从图中可以看出随着XML文档的增大，XSC效果比较显著。究其原因：XRel方法将元素结点信息存储在一张Element表中，随着表中记录数增加，插入操作要花费一部分时间对主键的判断，而XSC将元素结点信息分散存储到多张表中可以减轻这种负担。

4.6.2 XML查询性能分析

表 4.1 实验中所用到的XPath查询用例

Name	XPath
Q_1	/site/open_auctions/open_auction/annotation/description/text/bold
Q_2	/site/closed_auctions/closed_auction/annotation/description/parlist /listitem/parlist/listitem/text/keyword/emph
Q_3	/site/regions/australia/item/description
Q_4	/site/closed_auctions/closed_auction
Q_5	/site//australia//description
Q_6	//item
Q_7	/site/people/person[1]/profile

为了充分考虑XML查询中单表查询、多表连接等不同的方面，从XMark中包含



(a) XMark 5M (b) XMark 10M

(c) XMark 25M (d) XMark 50M

图 4.11 两种方法查询时间的比较

的20个查询语句中选取长路径，中等路径，短路径进行测试。由于对查询得到的记录集重构成XML结果片段的步骤是一样的，且所花费的时间占很大一部分时间，容易掩盖两种方法对路径表达式查询的时间差异，因此测试时略去XML文档重构的时间。实验所选用的查询用例见表4.1。

从图4.11可以看出，对于长路径*Q*1，*Q*2，随着测试文档大小的增加，XSC的查询性能有较显著地改进，这主要因为XRel方法中对应的表中记录数多得多，而XSC查询时：一方面由于对应的表中记录数要少，所以执行连接操作时可以节省时间；另一方面XSC方法中存储的边信息（Path属性）对于简单路径表达式的计算是非常有效的。分析*Q*5，XSC查询时间比XRel多，这是因为对于包含祖先分隔符的XPath表达式，XSC的SQL查询语句中的where条件子句使用Like运算符，而XRel方法中处理的where条件子句使用算术比较运算符。对于其它路径，XSC的查询响应时间都有一定程度的提高。

第五章　基于非归并的小枝模式查询方案TwigWM

小枝模式查询是XML查询中重要的操作，已经有许多种算法提出，如 TwigStack和TJFast算法等，但是他们都是基于归并思想，不能避免大量的不必要的路径归并。本章提出的TwigWM算法使用部分栈与链表的结构来实现非归并查询，由于从路径串编码能够直接得到祖先元素结点的编码，所以TwigWM算法采用路径串编码。通过与归并算法TwigStack、TJFast，非归并算法Twig2Stack等进行比较，TwigWM算法具有良好的性能。

5.1　引言

目前基于XML查询处理的方法主要有基于序列的方法、结构连接方法和小枝模式(Twig)方法等，其中连接操作是最重要的查询操作。对Twig查询模式通行的求解算法大多采取如下步骤 [60]:

(1)将Twig查询模式分解为二元结构关系（父亲-孩子或祖先-后代）；

(2)利用结构连接算法（即判断两元素满足结构关系的连接算法）从XML数据库中寻找所有满足上述结构关系的结点集合；

(3)将得到的中间结果合并为满足整体结构关系的最后结果。

直观地看，上述Twig查询模式求解方式中影响计算性能的因素主要有两个：一个是结构连接算法的效率；另一个就是Twig模式分解的规模。显然，如果需要结构连接计算的数目少，整体的计算则必然会有较好的性能表现。整体匹配算法在求解时将查询作为一个整体来处理，因而速度得到了较大的提升。文献 [18]最早提出了小枝模式查询算法TwigStack。文献 [51]提出的TwigStackList算法能够较好地处理小枝模式中的父子关系。文献 [19]针对扩展的Dewey编码提出了TJFast算法，它只需访问叶子结点。但是上述算法对Twig形式的查询而言，需要执行路径的合并操作，不能避免大量的不必要的路径归并。Twig2Stack [20]与TwigList [21] 分别采用层次栈和队列的结构而不需要归并，因此性能优于前述算法，但是他们需要遍历文档树中的每个结点。针对上述现象，本章提出了TwigWM(Twig Without Merge)算法，TwigWM是用非归并思想输出结果解，同时TwigWM不需要遍历XML文档中的所有结点。

5.2 相关概念及说明

5.2.1 结点编码

XML文档树的常用编码主要有两大类：基于区间的编码和基于路径的编码。

基于区间编码是给每个结点赋予一个区间编码$(begin, end)$，让其表示结点在XML文档中的顺序位置，文档树T中的结点u是结点v的祖先，则有

$$begin(u) < begin(v) \wedge end(v) < end(u) \tag{5.1}$$

区间编码典型代表是Zhang编码 [17]，它对XML文档树中的每一个结点赋予一个二元组$< begin, end >$，对树T 中的所有结点进行先序遍历，每一个结点在遍历时分别被访问两次并产生两个序号：一次是在遍历该结点的所有后裔结点之前访问该结点，并产生该结点的序号$begin$；另一次是在遍历完该结点的所有后裔结点后再访问该结点，并产生该结点的另一个序号end。因此，树T 中的任意两个结点u 和v 是祖先/后裔关系，当且仅当满足式5.1，即祖先结点u 的区间编码$[begin(u), end(u)]$包含后裔结点v 的区间编码$[begin(v), end(v)]$。显然，这种祖先/后裔关系的判断条件可以进一步改写为式5.2所示

$$begin(u) < begin(v) \wedge begin(v) < end(u) \tag{5.2}$$

另外，树T 中的每一个结点也被赋予一个$level$值，该值用于标识该结点在树中所处的层数。对于该编码方案，$begin$ 作为结点的唯一标识。一个Zhang 编码的示例如图2.3所示。

基于路径编码方案则是利用XML文档的树形结构，给从文档根结点开始所能到达每个路径上的元素结点赋予一个编码，典型代表是前缀编码（Dewey编码），它直接将一个结点的父亲结点编码作为该结点编码的前缀，具体见2.3.2部分。

为了支持文档更新，第四章提出了一种混合编码方案进行模型映射，在查询方面，本章将混合编码方案进行变换，只选取其中的路径串编码，同时参考扩展Dewey编码 [19]。扩展Dewey编码在x的计算方面不同于Dewey编码。下面是扩展Dewey编码的详细定义：

(1) 如果v是文本结点，则$x = -1$ 。

(2) 如果v 的标记名是$CT(t_u)$ 中的第k个标记，其中t_u 是v的父亲u 的标记，则

- 如果v是u的第一个孩子，则$x = k$；
- 否则，假设v的左兄弟编码的最后一部分为y，则

$$x = \begin{cases} \lfloor y/n \rfloor \times n + k & (y \bmod n) < k; \\ \lceil y/n \rceil \times n + k & \text{其它}. \end{cases} \tag{5.3}$$

其中$CT(t_u)$表示结点标记t_u的孩子标记集合，并且集合中的标记按某种方式有序（如按字母顺序有序），n表示集合$CT(t_u)$的大小。

经过仔细分析，式5.3有部分不合理之处，现改进扩展Dewey编码如下：

如果v的标记名是$CT(t_u)$中的第k个标记，其中t_u是v的父亲结点u的标记，则

- 如果v是u的第一个孩子，则$x = k$；
- 否则，假设v的左兄弟编码的最后一部分为y，则

$$x = \begin{cases} \lfloor y/n \rfloor \times n + k & (y \bmod n) < k; \\ \lfloor y/n \rfloor \times n + n + k & (y \bmod n) = k; \\ \lceil y/n \rceil \times n + k & (y \bmod n) > k. \end{cases} \tag{5.4}$$

通过这种编码方法，结点的扩展Dewey编码与标记之间存在对应关系，如果$d(v)$定义为$d(u).x$，则：

(1)如果$x = -1$，则v是文本结点；

(2)否则，v的标记是$CT(t_u)$中序号为$x \bmod n$的标记，n表示集合$CT(t_u)$的大小。

图5.1(a)是采用扩展Dewey编码的XML文档示例。

5.2.2 相关定义

一个XPath路径表达式查询可以通过小枝模式(Twig pattern)进行建模，称为小枝模式查询。小枝模式查询也称为树模式查询。下面给出了相关的形式化定义：

定义 5.1 (XML文档树模型). 一个XML文档树T用四元组(r_T, V_T, E_T, F_T)来表示，其中V_T是结点的集合；E_T是边的集合；r_T是唯一的根结点；$F_T : V_T \rightarrow \sum$是一个映射函数，表示每个结点的类型，$\sum$是$T$中的类型集。

定义 5.2 (小枝模式查询). 一个小枝模式查询Q用五元组$(r_Q, V_Q, E_Q, F_Q, O_Q)$来表示，其中$V_Q$是结点的集合；$E_Q = C_Q \vee D_Q$是边的集合，$C_Q$和$D_Q$分别表示孩子边和后代边；$r_Q$是唯一的根结点；$F_Q : V_Q \rightarrow \sum$是一个映射函数，表示每个结点的类型，$\sum$是$Q$中的类型集；$O_Q$是输出结点。

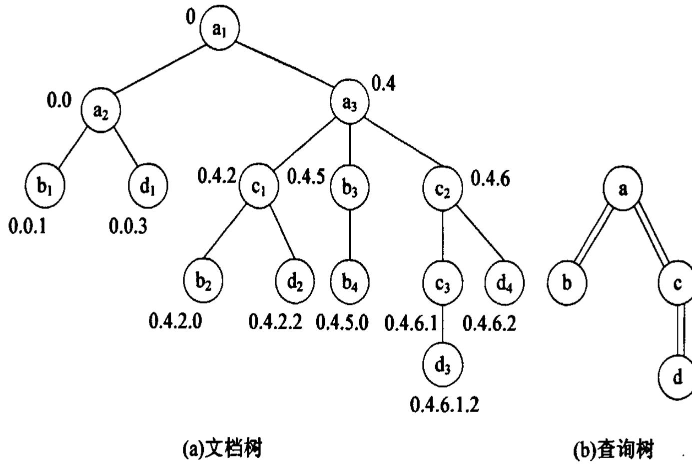


图 5.1 XML文档树及Twig查询树

定义 5.3 (小枝模式匹配). 给定小枝模式查询Q和文档树T，$lab(u)$表示结点u的类型标签，则Q在T上的匹配是一个映射$f: V_Q \to V_T$，且满足：

- 保持结点类型，即$\forall u \in V_Q : lab(u) = lab(f(u))$；
- 保持边关系，即$\forall u, v \in V_Q$:若$< u, v > \in C_Q$，则在T中$f(v)$是$f(u)$的孩子结点；若$< u, v > \in D_Q$，则在T中$f(v)$是$f(u)$的后代结点。

图5.2给出了小枝模式匹配的示意图，其中的一个匹配为：(a_1, b_1, c_1, d_2)。

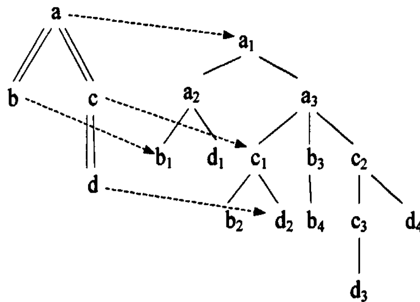


图 5.2 小枝模式匹配

进而可定义Twig模式匹配问题为：给定Twig查询模式Q和一个具有某种索引结构的XML数据库，所谓Twig模式匹配问题就是搜索XML数据库得到所有满足Q模式的XML数据片段，表示为n元组的形式。

定义 5.4 (标签流). 查询树Q中的结点a对应的标签流是指在文档树T中所有具有相同标签名的元素结点的集合，同时集合T_a中元素按前缀顺序有序（如“0.4”先于“0.4.2.0”）。

图5.3所示是图5.1中结点a，b，c，d对应的标签流T_a，T_b，T_c，T_d，指针指向的元素是叶子结点标签流中被初次定位的元素（见算法1）。

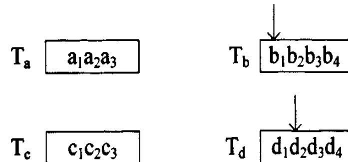


图 5.3 标签流

定义 5.5 (分支结点). 在一个小枝查询模式中，如果一个结点有多个子结点（大于一个），那么这个结点就叫分支结点。

为了叙述方便，本文中提及的结点是指查询树中的结点，而元素结点（简称元素）是指文档树中的结点。

5.3 查询匹配算法TwigWM

5.3.1 数据结构及说明

为了实现非归并的思想，本文使用了链表结构，将查询树中每个结点V_i关联一个链表L_i，链表中存放最终输出的元素结点和指向孩子元素结点的指针。

将非终端结点V_b关联一个栈S_b，算法运行时，不断的用当前元素结点去触发栈中的元素，若栈中的元素被弹出，还要决定其是否能够进入链表L_b中。图5.4所示是TwigWM算法用到的数据结构。

关于标签流T_e有如下操作：current(T_e)函数取T_e当前指针指向的数据元素，advance(T_e)函数使T_e的指针指向下一个数据元素，eof(T_e)函数判断T_e中的当前指针是否指向末尾。

descendant(V_i)函数获取查询树Q中结点V_i的后代结点，twigLeafNodes(V_q)函数获取查询树Q中以V_q为根结点的子树中叶子结点集合。

P_n表示结点n的路径模式。如图5.1(b)中P_d为$a//c//d$。

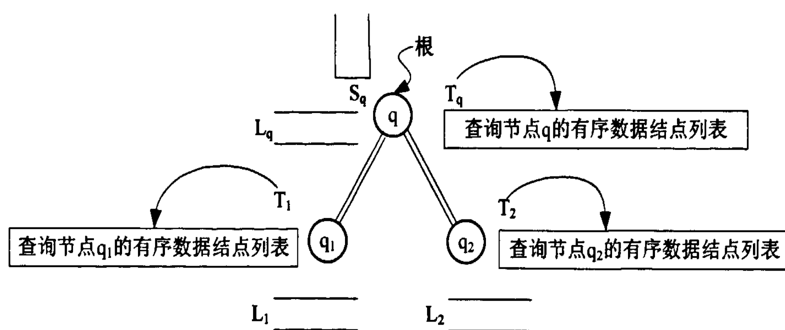


图 5.4 TwigWM算法用到的数据结构

5.3.2 算法思想

本章在实验结果一节要与TwigStack算法 [18]、TJFast算法 [19]、Twig2Stack算法 [20] 进行比较，由于Twig2Stack算法已在第2.6.2节介绍过，所以下面先简单介绍TwigStack、TJFast的执行过程，然后再介绍TwigWM的思想。

- TwigStack算法

TwigStack算法用栈保存线性路径匹配结果：首先找到可能满足小枝模式的元素并将它们保存在对应的栈中，若遇到叶子结点对应的元素时，则输出线性路径匹配结果；然后归并所有线性路径结果得到最终满足小枝模式匹配结果。在处理只有祖先后裔关系的查询时，TwigStack算法得到的可能满足查询Q的元素实际上一定满足Q，对于有父子关系的则不一定满足，需要在最后的归并过程中去掉不满足Q的中间结果。

图5.5所示是TwigStack执行过程，TwigStack算法为a，b，c，d四个结点建立四个栈S_a，S_b，S_c，S_d来保存小枝模式树中结点对应的元素。同时栈S_v中每个元素还有一个指针指向$S_{parent(v)}$的栈顶，用来指明所对应的祖先元素。TwigStack算法运行过程：首先得到元素a_1和b_1（a_2由于没有c类型的后裔元素被丢弃），分别将它们压入S_a、S_b栈中，同时b_1的指针指向S_a的栈顶，如图5.5(c)中(1)所示，因为b_1是小枝模式树中叶子结点对应的元素，所以输出线性路径匹配结果(a_1,b_1)，并将b_1弹出S_b；图5.5(c)中(2)是将元素a_3、b_2入栈后的状态，并且输出线性路径匹配结果(a_1,b_2)、(a_3,b_2)；图5.5(c)中(3)是将元素c_1、c_2、d_2入栈后的状态，并且输出线性路径匹配结果(a_1,c_1,d_2)、(a_3,c_1,d_2)、(a_1,c_2,d_2)、(a_3,c_2,d_2)；当遇到元素d_3时，由于c_2不是d_3的祖先被弹出，d_3进入S_d，如图5.5(c)中(4)所示，输出线性路径匹配结果(a_1,c_1,d_3)、(a_3,c_1,d_3)。最后归并这些线性匹配结果得到最终满足小枝查询树的结果。

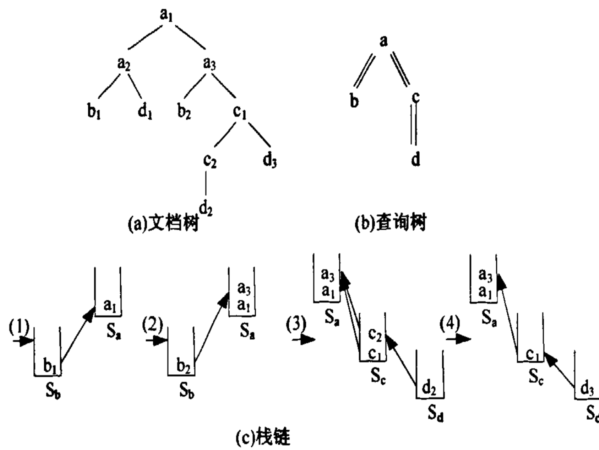


图 5.5 TwigStack算法执行示例图

- TJFast算法

TJFast算法分为两个阶段：第一阶段计算单个路径的解；第二个阶段将这些解合并，形成小枝查询的解。根据扩展Dewey编码的性质可知，给定一个元素的扩展Dewey编码，可以很容易知道其路径以及其路径是否与某一路径模式匹配。因此，算法的关键是判断一条路径解是否参与整个小枝的解，也即是否能与其它路径解进行合并。而两条路径可以合并，其必要条件是两条路径具有相同的元素以匹配模式中的分支结点，因此，算法中将每个分支结点b的匹配元素缓存在S_b栈中。

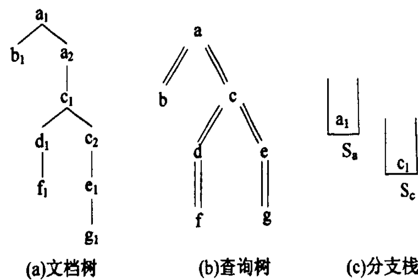


图 5.6 TJFast算法示例图

在图5.6中，最开始，$getNext(a)$递归的调用$getNext(b)$和$getNext(c)$（因为b、$c \in dbl(a)$），由于b为叶子结点，所以$getNext(b) = b$。在计算$getNext(c)$的过程中，需要计算$MB(f,c) = \{c_1\}$、$MB(g,c) = \{c_1, c_2\}$，所以$e_{max} = g$、$e_{min} = f$。将c_1进入S_c栈，然后返回，有$getNext(c) = f$。后面a_1进入S_a栈，有$getNext(a) = b$。最后输出路径解(a_1, b_1)、

(a_1, c_1, d_1, f_1)和(a_1, c_1, e_1, g_1)，并进行合并。注意，虽然(a_1, c_2, e_1, g_1)是$a//c//e//g$的一个路径解，但是它并不输出，因为c_2不在S_c栈中。

- TwigWM算法

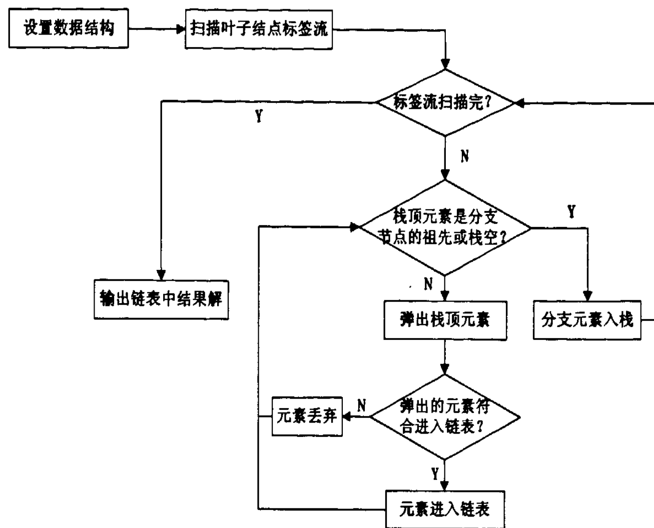


图 5.7 TwigWM算法思想流程图

首先依照Twig查询树结构为结点设置相应的数据结构，包括标签流、链表、分支栈等，标签流中的元素按一定顺序排列，Twig查询树中每个结点对应一个链表，链表的排列次序与前序遍历Twig查询模式中的结点次序相同；然后，顺序地扫描Twig查询树中叶子结点对应的标签流中元素数据，在扫描过程中，将元素数据对应的路径信息中包含的分支结点元素压入相应的分支栈，在压入之前先判断栈顶元素是否是压入元素的祖先，若不是，则根据条件判断弹出的栈顶元素是否能够进入相应的链表中；当扫描完叶子结点对应的标签流中元素数据后，符合Twig查询模式结构关系的元素序列会进入相应的链表中，并且链表通过指针互相连接起来，这样，不再需要归并分支路径解，直接输出链表中的元素即为满足查询请求的结果。图5.7是TwigWM算法中心思想的流程图表示。

5.3.3 具体算法描述

TwigWM算法（见算法1）的输入：q是具有n个结点$\{V_q, V_2, \cdots, V_n\}$的Twig查询树，其中$V_q$是根结点。$T$是$q$中结点对应的文档标签流集合$\{T_q, T_2, \cdots, T_n\}$。TwigWM算法第1-3行使标签流的指针定位到合适的位置，该位置所在的元素匹配结点路径模

算法 1 TwigWM(q, T)

1: **for** $f \in$ twigLeafNodes(V_q) **do**
2: findMatchedLabel(f);
3: **end for**
4: 对每个$V_i \in V(q)$,初始化其链表L_{V_i};
5: **while** ($\neg$end(V_q)) **do**
6: //end(V_q): Return $\forall leaf \in$ twigLeafNodes(V_q) $\rightarrow$ eof(T_{leaf});
7: 令e为叶子结点标签流T_i中前缀顺序最小的当前元素,且其结点类型为V_e;
8: **for** $R \in$mapAN(e) **do**
9: 令S是R元素的结点类型所关联的栈;
10: trigger_Stack(S, R);
11: **for** $g \in R$ **do**
12: **if** ($\neg$(S contain g)) **then**
13: each child of V_g in q, V_c do
14: $g.start_{V_c} \leftarrow$ length(L_{V_c})+1;
15: push(S, g);
16: **end if**
17: **end for**
18: **end for**
19: append e into list L_{V_c};
20: advance(T_e);
21: findMatchedLabel(V_e);
22: **end while**
23: trigger_Stack(S, R_{EOF});
24: outputList(q, L);

算法 2 Procedure findMatchedLabel($leaf$)

1: **while** ($\neg$ eof (T_{leaf}) $\wedge \neg$ (current(T_{leaf}) matches P_{leaf})) **do**
2: advance(T_{leaf});
3: **end while**

算法 3 Procedure trigger_Stack(S, R)

```
1: for r ∈ R do
2:    flag ←true;
3:    while (S ≠ ∅ ∧ ¬(r starts with top(S))) do
4:       flag ←false;
5:       e ←pop(S),其结点类型为V_e;
6:       each child of V_e in q, V_c do
7:          e.end_{V_c} ← length(L_{V_c});
8:       if ∀V_e's child V_c : e.start_{V_c} ≤ e.end_{V_c} then
9:          append e into list L_{V_e};
10:      end if
11:   end while
12:   if flag=true then
13:      return ;
14:   end if
15: end for
```

式，例如图5.3中b、d 首次定位到b_1、d_2的位置；第5-22行循环处理当前元素是否可以进入链表L中，其中mapAN函数返回满足祖先结点路径模式的元素集合，例如mapAN(b_2)={(a_1, a_3), (c_1)}；trigger_Stack(S, R)（见算法3）：若S的栈顶元素与R集合中某个元素不满足祖先关系，则弹出栈顶元素并判断弹出的元素是否可以进入链表L中（trigger_Stack函数第8-9行进行判断），其中$flag$标记作用：若栈顶元素不弹出则栈中其余元素不用判断就可知是R中某个元素的祖先元素。例如在图5.1中，当e是d_2时，栈S_a中元素为(a_1, a_3)，则调用trigger_Stack(S_a, R)后，S_a栈不用弹出元素。TwigWM算法第11-14行将不在栈中的元素赋上合理的$start$属性值入栈，第23行用R_{EOF}集合强制将栈中所有的元素（若满足条件）进入链表L中。第24行非归并输出链表中元素outputList(q, L)，outputList输出过程参考了文献 [21]中的思想。

在TJFast算法中，每一个进行比较的元素都要先与它所对应的查询路径进行比较，判断其是否满足查询路径，只有匹配查询路径的结点元素才可以进一步判断其是否产生最终结果，这样对于每一个结点元素都需要进行一次查询路径匹配判断，这将耗费很大。而在TwigWM算法中，由于分支栈的存在，当确定叶子结点元素之间存在共同的分支结点元素时，就将分支结点元素进入分支栈，这样不需要扫描非

算法 4 Procedure outputList(q, L)

1: 令$pre[1..n]$,$post[1..n]$分别表示元素的$start$和end属性值;
2: 根据父亲链表中第一个元素的$start$,end属性值初始化其余结点V_i的$pre[i]$,$post[i]$值; //根结点V_1的$pre[1]$,$post[1]$值分别是0, $length(L_{V_1}) - 1$
3: $move[1..n] \leftarrow pre[1..n]$;
4: **while** (true) **do**
5: 　输出结果解:($move[1], \cdots, move[n]$);
6: 　**if** ($\forall i : move[i] = end[i]$) **then**
7: 　　**return** ;
8: 　**end if**
9: 　在查询树Q中选出满足$move[i] < end[i]$条件的结点V_i, 且V_i的所有后裔结点V_j满足$move[j] = end[j]$;
10: 　$move[i] = move[i] + 1$;
11: 　令e_i是V_i结点链表中$move[i]$指向的元素;
12: 　**for** ($\forall V_j \in$ descendant(V_i)) **do**
13: 　　根据父元素（以e_i为根元素开始）的$start$,end属性值重置$pre[j]$,$post[j]$和$move[j]$;
14: 　**end for**
15: **end while**

叶子标签流中元素数据。

以图5.1(a)所示的XML文档和图5.1(b)所示的Twig查询树为例，算法执行过程如下（如图5.8）：

(1)插入b_1后栈和链表的状态，a_1和a_2都是b_1的祖先，由于当前S_a栈为空且a_1与a_2满足祖先关系，所以a_1和a_2同时进S_a栈；(2)插入b_2后栈和链表的状态，a_2由于不满足b_2祖先a_3的前缀关系被弹出，同时a_2由于不符合进入L_a的条件而被丢弃；(3) 插入d_2后栈和链表的状态，由于当前S_c栈为空，所以d_2的祖先c_1进S_c栈；(4)插入d_3后栈和链表的状态，由于c_1不是c_2的祖先被弹出，同时判断c_1符合进入L_c的条件，所以c_1进入L_c且关联的区间是$< d_2 >$；(5)插入d_4后栈和链表的状态，由于c_3不满足d_4祖先c_2的前缀关系被弹出，同时判断c_3符合进入L_c的条件，所以c_3进入L_c且其关联的区间是$< d_3 >$；(6)用R_{EOF}触发后栈和链表的最终状态，这时候栈中所有结点都被弹出，并且都符合进入链表的条件，如a_3关联的区间$< b_2, b_3, b_4 >< c_1, c_3, c_2 >$。

outputList(q, L)算法输入有两个参数：查询树q和已经创建好的链表L，在

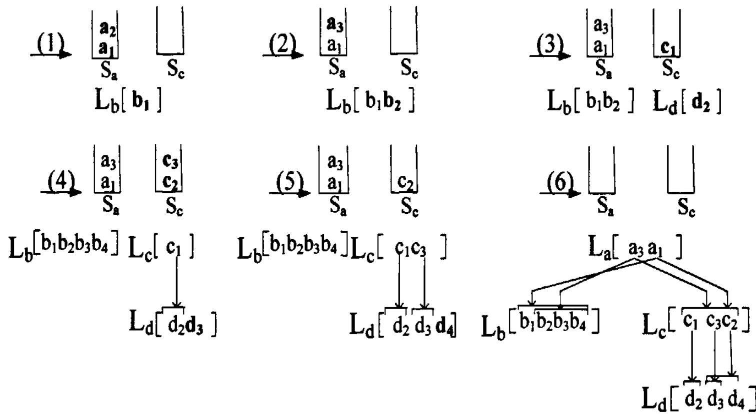


图 5.8 TwigWM算法运行示例图

图5.8(6)中，此时有四个链表L_a，L_b，L_c和L_d。开始时，$pre = [a_3, b_2, c_1, d_2]$，$post = [a_1, b_4, c_2, d_2]$，这里$(a_3, a_1)$是由创建后根结点链表$L_a$中元素个数指定的（即outputList算法第2行），(b_2, b_4)区间是由a_3的*start*和*end*属性值决定的。此时$move = [a_3, b_2, c_1, d_2]$记录当前要输出整体结果解的位置，当输出$(a_3, b_2, c_1, d_2)$后，运行第10行，$move = [a_3, b_3, c_1, d_2]$记录当前要输出整体结果解的位置，所以输出$(a_3, b_3, c_1, d_2)$，接着输出$(a_3, b_4, c_1, d_2)$，当运行算法第12-14行时，$pre = [a_3, b_2, c_3, d_3]$，$post = [a_1, b_4, c_2, d_3]$，于是输出结果$(a_3, b_2, c_3, d_3), \cdots$。

5.4 算法分析

给定Twig查询树Q和具有n个元素结点的XML文档树T，Q中的分支结点与其子结点之间只有祖先/后裔关系，$|b|$和$|f|$分别表示文档树T中非叶子和叶子元素结点的数目，d表示查询树Q中结点的最大度，$|Q|$表示查询树Q中结点的个数。

- 空间方面：

因为每个非叶子元素结点的扩展Dewey编码被孩子元素结点重复存储，而叶子元素编码只被存储一次，所以最多情况下的存储为

$$S = d^2 \times |b| + d \times |f| \tag{5.5}$$

又因为$|b| < n$、$|f| < n$，所以其空间复杂度为

$$S(n) = O(d^2 \times n) \tag{5.6}$$

- 时间方面：

由于TwigWM算法只需访问$|f|$个叶子结点，但每次访问叶子结点时，要判断分支结点是否进入分支栈，又因为$|f| < n$，所以在构造链表L时的时间复杂度为

$$T_1(n) = O(d \times |f|) = O(d \times n) \quad (5.7)$$

当链表L建好以后，假定最终结果解的个数为m，因为输出时，Q中每个结点在结果输出时都要输出，且一般情况下$m < n$，所以outputList输出算法的复杂度为

$$T_2(n) = O(|Q| \times m) = O(|Q| \times n) \quad (5.8)$$

所以TwigWM算法时间复杂度为

$$T(n) = T_1(n) + T_2(n) = O(d \times n + |Q| \times n) \quad (5.9)$$

因为$d \ll n$、$|Q| \ll n$，所以时间复杂度为$O(n)$。

5.5 实验分析

5.5.1 实验环境

表 5.1 实验中所用到的小枝模式查询用例

Name	DataSet	Twig
XQ_1	XMark	//site//closed_auctions//closed_auction//price
XQ_2	XMark	//site[//regions//parlist//text//keyword]//closed_auction//date
XQ_3	XMark	//item[location]//description//keyword
XQ_4	XMark	//item[//description//text//bold]//mailbox//mail//date
XQ_5	XMark	//open_auctions[//reserve]//bidder[//time]//personref
DQ_1	DBLP	//dblp//inproceedings[title]//year
DQ_2	DBLP	//title[//i]//sup
DQ_3	DBLP	//dblp//article[author][//title][//url][//ee]//year
DQ_4	DBLP	//dblp//article[author][//title]//year
DQ_5	DBLP	//article[//sup]//title//sub

为了准确分析TwigWM算法的性能，本文实现了四种算法：TwigStack、TJFast、Twig2Stack、TwigWM。实验的硬件环境：CPU为Pentium 4，主频为2.8GHz，内存

为1 GB。软件环境：操作系统为Windows XP，开发语言为Java。实验用到两个著名的数据集：生成的数据集XMark [59] 和真实的数据集DBLP [61], 其中XMark数据集大小为50MB，DBLP数据集的大小为60MB。在两个数据集上分别进行不同的5个小枝模式查询用例测试。实验用到的查询用例见表5.1。

5.5.2 实验结果

图5.9和图5.10 给出了4种算法在两个数据集下 小枝模式查询的运行时间，4种

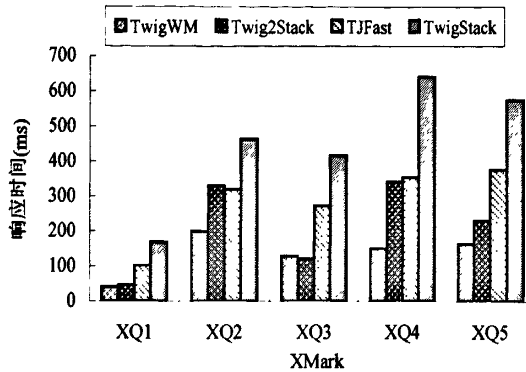


图 5.9 四种算法在XMark上执行时间的比较

算法在标签流上的执行时间相同，所以没有包含输入标签流的时间。从图中可以看出TwigWM，Twig2Stack，TJFast算法都好于TwigStack算法， 原因在于前三个算法都

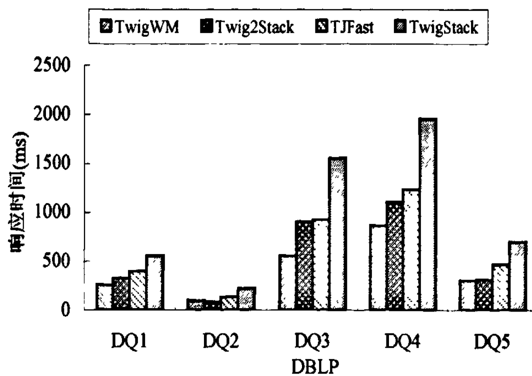


图 5.10 四种算法在DBLP上执行时间的比较

是从整体小枝模式树(Twig)上考虑结构连接的。由于TJFast算法在输出结果时是基于

归并的，并不能避免大量的不必要的路径归并，所以TwigWM与Twig2Stack算法好于TJFast算法。TwigWM算法总体上好于Twig2Stack算法，原因归结于TwigWM算法只需遍历叶子结点对应的标签流，而Twig2Stack算法则遍历整个文档标签流，同时当小枝模式树复杂时Twig2Stack算法用比较多的时间构造层次栈。

第六章 XML数据管理在Office数据源实例中的应用

随着XML技术不断地发展，越来越多的信息集成系统采用XML作为信息存储、交换及发布的载体。在XML数据管理的实际应用中，用户希望按照自己的查询需求对系统内的文档进行信息检索。

以上各章具体介绍了XML数据管理的技术，给出了XML映射方案及更新的方法，还针对小枝模式查询提出了一种非归并的查询算法。本章将上述知识组成一个整体，设计一个XML 数据管理在Office数据源实例中的应用。

6.1 实例背景

新的Office 系统（如Office 2007）采用基于XML 的文件格式（Office XML 格式），新的格式改善了文件和数据管理、数据恢复以及与行业系统的互操作性，任何支持XML 的应用程序都可以访问和处理新文件格式的数据。在Office系统中引入Office XML 格式有很多优点：

• 易于将业务信息与文档集成。利用Office XML 格式能够快速地从分散的数据源创建文档，从而促进文档组合和内容重用，简化了Office 应用程序和企业业务系统之间的数据交换。可以通过选择的应用程序（不必使用Office 应用程序）读取和写入XML，更快、更精确地发布、搜索和重用信息。

• 可互操作性。可以通过使用能够处理XML 的标准工具和技术来改写Office 文档中的信息或创建文档，而无需访问Office 应用程序。例如，需要编辑word 文档的页眉中的文本，对于一份文档来说，该任务的执行并不合情理。但是，若在成百上千的不同文档的页眉中改变文本，可以利用XPath 查询找到旧的文本，用新的文本替换页眉部件，并重复此过程直到更新完每一个文档。

• 内容共享和重用。Office XML 的模块性使得在一次生成内容后在多个其它文档中加以重用成为可能。例如，可以将在一个word 文档中创建的表用于其他word 文档中。

基于上述介绍，本文以Office系统中的文档作为实例的数据源，进行XML文档的存储、小枝模式查询等应用。

6.2 实例设计

6.2.1 概要设计

本实例开发的目的是为了提高XML数据管理的效率。通过友好方便的用户界面，可以实现任何以数据为中心的XML文档的存储、更新及查询。本实例提供的主要功能有：

- 数据装载

数据装载模块可以实现将XML文档中的数据自动导入，当接收用户输入的XML文档时，先对文档进行解析，然后将解析后的数据按XSC映射规则转换成多条记录，再存储在关系表中。数据装载模块还包括关系模式的自动创建。

- 数据检索

数据检索模块可以响应用户提交的查询请求，将表达式如XPath转换为关系数据库可以接受的SQL语句，将检索数据以XML文档片段的结果格式返回。

- 数据更新

数据更新模块可以将XML文档中结点数据的变化自动反映到关系表中，包括增加文档片段（或结点），修改结点内容，删除文档片段（或结点）。

- 数据发布

XML发布模块是将关系表中的数据以XML文档的形式呈现给用户。

- 小枝模式查询

小枝模式查询模块是直接针对文档文件的，可以对复杂的查询请求给出快速地响应，例如在此实例中，在不打开文档文件的情况下，可以实现对数据格式及内容的同时检索。

图6.1给出了实例系统的整体框架图。

本实例的数据处理主要包含三大功能：数据存储、数据更新及数据查询。所有的功能都是在人机交互的基础上完成的，因此，良好的用户界面是必须的。当用户通过界面提交某一请求时，实例系统调用相应的功能模块，这些模块与后台数据库或文档交互，将处理结果通过界面形式显示给用户。所以实例系统分成三层：表示层，逻辑业务层和数据层。其中：表示层提供用户界面；逻辑层实现XML数据存储、更新及查询功能；数据层提供持久数据信息，在本实例中可以是关系数据库，也可以是XML文档。

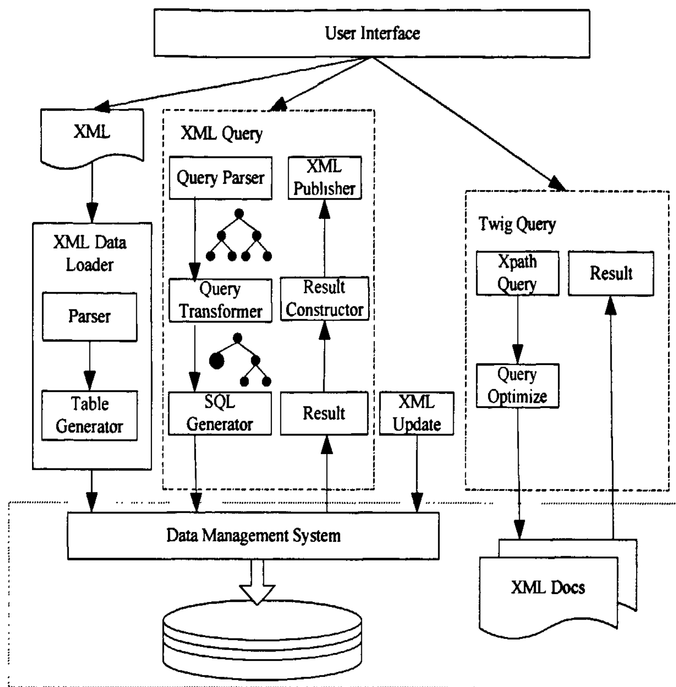


图 6.1 实例整体框架

6.2.2 详细设计

1.表示层

表示层主要由界面组成，可以选择表示层的不同界面来执行相应的功能。该实例共有以下几个主要窗口：

• mainFrame主窗口，该窗口是一个主界面窗口，提供功能菜单供用户选择不同的操作。

• storeFrame存储窗口，根据界面提示选择XML文档，调用业务逻辑层中的函数按照XSC映射规则进行存储。

• publishFrame发布窗口，根据存储在关系中的数据，结合创建的视图以XML形式重组数据。在本实例中，发布即相当于将数据重组为word文档的XML形式。

• searchFrame 检索窗口，将用户给出的路径表达式转化成SQL语句，再提交给关系引擎进行检索。

• insertFrame 插入文档（或结点）窗口，在word中可能需要向某个地方插入文本数据，而用XML表示时就是向XML文档中插入片段文档。

- modifyFrame 修改文档（或结点）窗口，例如对word文档中某个值进行替换。

- twigQueryFrame 小枝模式查询窗口，此窗口根据给出的XPath表达式（比如，查询文档中是否存在蓝色背景、粗体、红色的宋体文字）用TwigWM算法在XML文档中进行非归并查询。

2. 逻辑业务层

(1)存储模块层。主要有以下几个方法：

- parseDoc.java。用JDOM解析XML文档。

- encodeDoc.java。对解析后的XML进行编码，此处按照第四章提出的方案进行编码。

- relationCreate.java。在存储之前，先根据XML文档自动创建关系结构。

- dataLoader.java。将XML文档中的元素结点、属性结点及文本结点数据装入到创建的关系中。

- viewCreate.java。视图创建，当用路径表达式对关系中的数据进行检索时，可以利用视图来提高效率。

(2)检索模块层。主要有以下几个方法：

- xpathAnay.java。对提交的路径表达式进行分析，辨别其中包含的分隔符是祖先分隔符还是后裔分隔符。

- sqlGenerate.java。将XPath表达式换为XPathCore图，再生成SQL语句。

- xmlDisplay.java。根据关系引擎返回的数据重构成XML文档格式。

- publishXML.java。根据视图中的数据进行word文档的XML形式重组。

(3)更新模块层。主要有以下几个方法：

- insertXML.java。在任意指定的位置插入XML片段，且必须保持XML原有标签之间的关系。

- modifyXML.java。对XML文档中的结点名称进行修改，例如将word文本中所有“XPath”改成“XQuery”。

- delXML.java。将XML文档中的结点（或片段）删除，若删除父标签结点，则子标签结点也要删除。

(4)小枝查询模块层。主要有以下几个方法：

- labelStream.java。此函数将XML文档中的元素数据归类成不同的标签流。

- twigTreeCreate.java。将提交的XPath表达式组织成模式树。
- twigWM.java。运行TwigWM算法中的主函数，不断地将分支解进入链表中。
- outputResult.java。输出查询的结果，且以XML片段的形式返回。

3.数据层。

数据层实现数据的持久化存储，在本实例中用文档文件和关系数据库实现持久化存储，因此这个层次的设计主要是关系模式的设计和文件的组织设计。可以借鉴现有的技术，因此，不再详细阐述。

6.3 实例实现

6.3.1 实现环境及数据源

本实例开发的环境配置如下：CPU使用Pentium 4，主频为2.8GHz，内存为1GB，硬盘为80GB，操作系统使用Windows XP，关系数据库管理系统为Oracle 10g。

该实例开发采用java编程语言，开发环境为Eclipse3.2。解析XML采用JDOM。Java作为主流的程序设计语言，支持面向对象、面向组件、面向服务的软件开发，Java具有可移植性，重用性，简单易学等优点。XML本身也是一种语言，可以和Java很好的结合起来。Java具有跨平台的特性，实现一处编译，多处运行的良好机制，这正是本课题开发实例的要求。Eclipse是一个开放源代码、基于Java的可扩展开发平台，给系统开发带来方便。JDOM是基于Java 2的API，用于快速的解析XML，它不仅具有SAX的高效性，能够快速解析和输出，同时具有DOM的便利性。

本实例所使用的数据源是由word 2007生成的文档，图6.3是图6.2中黑色框内数据对应的XML数据源。

6.3.2 结果展示

本实例提供良好的界面，可以通过窗口中的对话框，按钮和文本框等组件与实例进行交互，实现相应的功能，图6.4、6.5、6.6、6.7 、6.8是实例运行的部分截图。其中图6.6中的查询语句相当于在word文档中检索“黄色背景，字体为Dotum格式的文本”，图6.8中黑色框的文本是图6.7查询结果对应的word文本。

Extensible Markup Language (XML) is a simple, very flexible text format derived from SGML (ISO 8879). Originally designed to meet the challenges of large-scale electronic publishing.

XML is also playing an increasingly important role in the exchange of a wide variety of data on the Web and elsewhere.

This page describes the work being done at W3C within the XML Activity, and how it is structured. Work at W3C takes place in Working Groups. The Working Groups within the XML Activity are listed below, together with links to their individual web pages.

You can find and download formal technical specifications here, because we publish them. *This is not a place to find tutorials, products, courses, books or other XML-related information. There are some links below that may help you find such resources.*

You will find links to W3C Recommendations, Proposed Recommendations, Working Drafts, conformance test suites and other documents on the pages for each Working Group.

图 6.2 word数据

```
    <wr wrsidRPr="007F5CAF">
        <wrPr>
            <wcolor wval="FF0000"></wcolor>
            <whighlight wval="yellow"></whighlight>
        </wrPr>
        <wt>XML is also</wt>
    </wr>
    <wr wrsidRPr="007D1FFE">
        <wrPr>
            <wrFonts wascii="Arial" whAnsi="Arial" wcs="Arial">
            </wrFonts>
            <wcolor wval="000000"></wcolor>
            <wlang></wlang>
        </wrPr>
        <wt xmlspace="preserve"></wt>
    </wr>
    <wr wrsidRPr="007F5CAF">
        <wrPr>
            <wb></wb>
            <wcolor wval="FF0000"></wcolor>
            <whighlight wval="yellow"></whighlight>
        </wrPr>
        <wt>
            playing an increasingly important role in
            the exchange of
            a wide variety of data on the Web and elsewhere.
        </wt>
    </wr>
</wp>
```

图 6.3 数据源部分示例

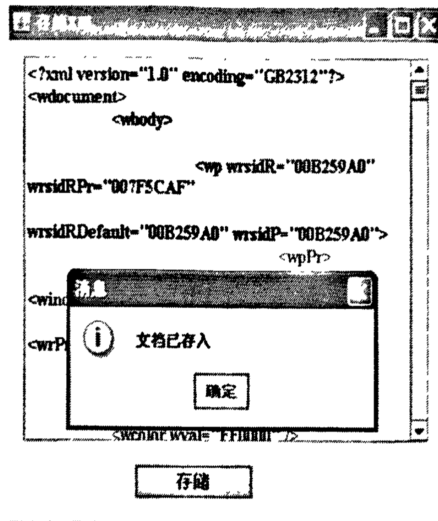


图 6.4 存储文档示例

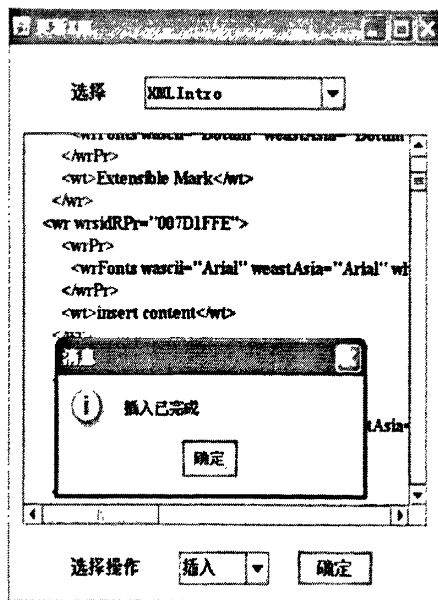


图 6.5 更新文档示例

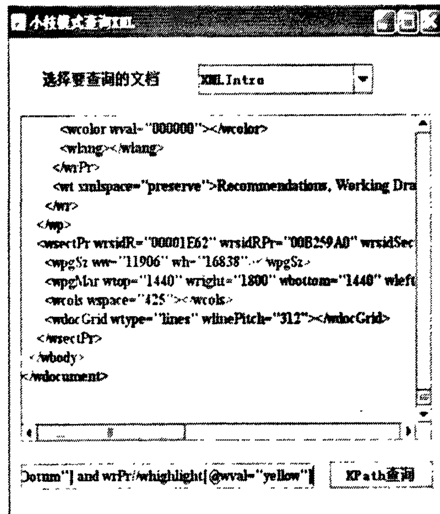


图 6.6 查询文档示例

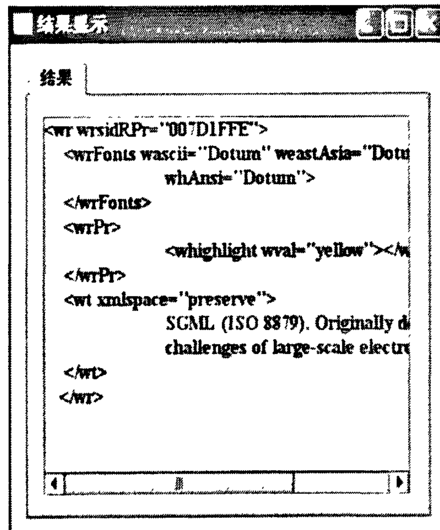


图 6.7 查询结果显示示例

Extensible Markup Language (XML) is a simple, very flexible text format derived from SGML (ISO 8879). Originally designed to meet the challenges of large-scale electronic publishing. XML is also playing an increasingly important role in the exchange of a wide variety of data on the Web and elsewhere.

图 6.8 查询结果对应的word文本示例

第七章　总结与展望

7.1　总结

XML作为网络数据交换和信息集成的工具语言，以其自描述、跨平台交换性等特点，在网络中得到广泛应用。越来越多的半结构化数据采用XML格式进行存储和交换，为此对XML的数据管理显得日益重要。

本文结合XML数据自身的特点和当前实际应用的基础上，就XML映射方法、文档动态更新及小枝模式查询算法等关键技术进行了研究，目的是提高XML数据管理的效率，所做的主要工作如下：

(1)提出了一种编码方案，将先序编号与路径串相结合对文档树中的结点进行编码。该编码方案是后面存储与查询的基础。

(2)通过对现有映射策略的分析总结，提出了一种新的映射方案XSC，该映射方案是在借鉴结点模型映射XRel基础上加入边模型映射思想提出的，接着给出了XSC 的详细映射算法，XSC映射是无损映射（即可完全恢复的），最后还通过实验将XSC与XRel的效率进行了比较。

(3)XSC映射方案支持动态更新功能，本文给出了在几种不同位置插入新结点的情况分析，当向文档中插入新结点时，只需要对其余结点进行少量的二次编码就可以实现正确的XML发布与查询。

(4)目前小枝模式查询算法基本上是基于归并思想的，即当得到分支解后，还要再进行合并去除一些无用的分支解。本文提出了一种非归并的小枝模式查询算法TwigWM，TwigWM算法采用了路径串的编码，利用栈和链表等数据结构实现结果解的非归并输出，通过与别的查询算法如TJFast算法的比较，TwigWM 算法具有良好的性能。

(5)设计并实现了XML数据管理应用的实例。根据前面提出的XML存储映射，更新功能及查询算法的基本思想，分析和设计了此实例的体系结构，接着给出了各个模块中主要方法的实现。运行结果表明，该实例可以把以数据为中心的XML文档数据无损的映射到关系中，利用小枝查询方案可以实现更快的结点检索。

7.2 展望

本文对XML数据管理中的部分算法和技术做了详细的介绍，并提出了一些新的观点和方法，但是研究中不免存在一些不完善的地方，所提出的算法也不一定是最优的算法，今后对这方面还有待于进一步的完善。主要体现在以下几个方面：

(1)在XSC存储映射方案中，本文对结构信息考虑的还不是很充分。虽然XML文档有无模式信息，XSC都可以将XML文档进行映射，但这并没有充分利用到XML模式中信息，如：函数依赖、语义信息等。因此，以后将对结构信息进行更多的支持。

(2)在XML查询中，本文主要考虑了小枝模式查询的现状，给出了一个基于非归并的匹配算法TwigWM，虽然该算法优于许多归并的小枝模式算法，但该算法在与某些非归并的算法比较中并不占优势，因此，以后考虑加入更多的索引技术来快速地定位元素结点，更大程度上提高算法的性能。

(3)不能对复杂的谓词进行处理。在查询路径表达式中常含有复杂谓词，这些表达式能表达更复杂的查询。已有的解决方法是将复杂谓词表示为单独的抽象语法树，可以单遍地处理复杂Twig表达式。接下来可以采用类似的思想在TwigWM算法中实现对复杂谓词的支持。

(4)本文主要考虑了XML数据管理在Office数据源方面的应用，未来可以拓宽其它应用领域。

总之，在XML数据管理中，本文给出了一些自己的观点和方法，在一定程度上促进了XML数据管理相关技术的研究。此外，将会继续查找方法的不足，争取给出更加有效的理论与方法。

参考文献

[1] W3C. Extensible markup language(xml) 1.0 (fifth edtion)[EB/OL]. http://www.w3.org /TR/2008/PER-xml-20080205.

[2] 周傲英，胥正川，郭志懋，周水庚. VXMLR系统存储模式的自适应调整[J]. 计算机学报, 2004, 27(4): 433–441.

[3] 孟小峰，王宇，罗道峰，陆世潮，安靖，陈妍，蒋瑜，欧建波. OrientX: 一个Native XML数据库系统的实现策略[J]. 计算机科学, 2003, 30(10): 111–115 .

[4] 孟小峰. XML数据管理概念与技术[M]. 北京：清华大学出版社, 2009.

[5] 王静，孟小峰，王珊. 基于区域划分的XML结构连接[J]. 软件学报, 2004, 15(5): 720–729.

[6] 万常选，刘云生，徐升华，刘喜平，林大海. 基于区间编码的XML索引结构的有效结构连接[J]. 计算机学报, 2005, 28(1): 113–127.

[7] H. Schoning and J. Wasch. Tamino-an Internet Database System[C]. Proceedings Of the 7th International Conference on Extending Database Technology, 2000: 383–387.

[8] A. Chapman, H. Jagadish and S. Khalifa. Timber: A Native XML Database[J]. VLDB Journal, 2002, 11(4): 274–291.

[9] M. Yoshikawa, T. Amagasa, T. Shimura and S. Uemura. XREL:A Path-Based Approach to Storage and Retrival of XML Documents Using Relational Databases[J]. ACM Taransactions on Internet Technology, 2001, 1(1): 110–141.

[10] H. Jiang, H. Lu, W. Wang and J. Yu. XParent: An Efficient RDBMS-Based XML Database System[C]. Proceedings Of the 18th IEEE ICDE International Conference on Data Engineering, 2002: 335–336.

[11] J. Shanmugasundaram, K. Tufte, G. He, C. Zhang, D. DeWitt and J. Naughton. Relational Database for Querying XML Documents: Limitations and Opportunities[C].

Proceedings of the 25th International Conference on Very Large Data Bases, 1999: 302–314.

[12] A. Deutsch, M. Fernandez and D. Suciu. Storing Semistructured Data with STORED[C]. Proceedings of the ACM SIGMOD International Conference on Management of Data, 1999: 431–442.

[13] P. Bohannon, J. Freire, P. Roy and J. Simeon. From XML Schema to Relations: A Cost-Based Approach to XML Storage[C]. Proceedings of the 18th International Conference on Data Engineering, 2002: 64–75.

[14] 王庆，周俊梅，吴红伟，萧建昌，周傲英. XML文档及其函数依赖到关系的映射[J]. 软件学报, 2003, 14(7): 1275–1281.

[15] 何盈捷，王珊. 从DTD映射到关系模式:一种保持数据依赖的映射方法[J]. 计算机研究与发展, 2004, 15(5): 868–873.

[16] Q. Li and B. Moon. Indexing and Querying XML Data for Regular Path Expressions[C]. Proceedings of the 27th International Conference on Very Large Data Bases, 2001: 361–370.

[17] C. Zhang, J. Naughton, D. DeWitt, Q. Luo and G. Lohman. On Supporting Containment Queries in Relational Database Management Systems[C]. Proceedings of the 2001 ACM SIGMOD International Conference on Managementof Data, 2001: 425–436.

[18] N. Bruno, N. Koudas and D. Srivastava. Holistic Twig Joins: Optimal XML Pattern Matching[C]. Proceedings of the 2002 ACM SIGMOD International conference on Management of Data, 2002: 310–321.

[19] J. Lu, T. Ling, C. Chan and T. Chen. From Region Encoding To Extended Dewey: On Efficient Processing of XML Twig Pattern Matching[C]. Proceedings of the 31st International Conference on Very Large Data Bases, 2005: 193–204.

[20] S. Chen, H. Li, J. Tatemura, W. Hsiung, D. Agrawal and K. Candan. Twig2Stack: Bottom-Up Processing of Generalized-Tree-Pattern Queries over XML Documents[C].

Proceedings of the 32nd International Conference on Very Large Data Bases, 2006: 283–294.

[21] Q. Liu, X. Jeffrey and B. Ding. TwigList: Make Twig Pattern Matching Fast[C]. Proceedings of the 12th International Conference on Database Systems for Advances Applications, 2007: 850–862.

[22] 杨卫东，王清明，施伯乐. 针对XML流数据的复杂Twig Pattern 查询处理[J]. 软件学报, 2007, 18(4): 893–904.

[23] 郭红，沈煌. 一种复杂XML Twig 查询处理算法[J]. 小型微型计算机系统, 2008, 29(11): 2012–2015.

[24] W3C. XML Path Language (XPath) Version 1.0[EB/OL]. http://www.w3.org/ TR/xpath.

[25] W3C. XQuery: A Query Language for XML[EB/OL]. http://www.w3.org/TR/2001 /WD-xquery -2001215.

[26] F. Daniela, D. Alin and F. Mary. XML-QL:A Query Language for XML[EB/OL]. http:// www.w3.org/TR/NOTE-xml-ql.

[27] 万常选，刘喜平. XML数据库技术[M]. 北京：清华大学出版社, 2008.

[28] P. Dietz. Maintaining Order In A Linked List[C]. Proceedings of the 14th annual ACM symposium on Theory of Computing, 1982: 122–127.

[29] Dewey Decimal Classification[EB/OL]. http://www.oclc.org/dewey.

[30] N. Wirth. Type Extensions[J]. ACM Trans. Program. Lang. System, 1988, 10(2): 204–214.

[31] X. Wu, L. Lee and W. Hsu. A Prime Number Labeling Scheme for Dynamic Ordered XML Trees[C]. Proceedings of the 20th International Conference on Database Engineering, 2004: 66–78.

[32] 孔令波，唐世渭，杨冬青，王腾蛟，高军. XML数据索引技术[J]. 软件学报, 2005, 16(12): 2063–2079.

[33] R. Goldman and J. Widom. DataGuides: Enabling Query Formulation and Optimization in Semistructured Databases[C]. Proceedings of the 23rd International Conference on Very Large Data Bases, 1997: 436–445.

[34] T. Milo and D. Suciu. Index Structures for Path Expressions[C]. Proceedings of the 17th International Conference on Data Engineering, 1999: 277–295.

[35] T. Amagasa, M. Yoshikawa and S. Uemura. QRS: A Robust Numbering scheme for XML Documents[C]. Proceedings of the 19th International Conference on Data Engineering, 2003: 705–707.

[36] I. Tatarinov, D. Viglas, K. Beyer, J. Shanmugasundaram, E. Shekita and C. Zhang. Storing and Querying Ordered XML Using a Relational Database System[C]. Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data, 2002: 204–215.

[37] P. O'Neil, E. O'Neil, S. Pal, I. Cseri and G. Schaller. ORDPATHs: Insert-Friendly XML Node Labels[C]. Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data, 2004: 903–908.

[38] M. Kratky, J. Pokomy and V. Snasel. Indexing XML Data with UB-trees[C]. Proceedings of the 6th East European Conference on Databases and Information Systems, 2002: 155–164.

[39] P. Rao and B. Moon. PRIX: Indexing and Querying XML Using Prufer Sequences[C]. Proceedings Of the 20th International Conference on Database Engineering, 2004: 288–300.

[40] 周军锋，孟小峰，蒋瑜，谢敏. F-index:一种加速Twig查询处理的扁平结构索引[J]. 软件学报, 2007, 18(6): 1429–1442.

[41] R. Bayer. The Universal B-Tree for Multidimensional Indexing: General Concepts[C]. Proceedings of the International Conference on Worldwide Computing and Its Applications, 1997: 198–209.

[42] D. Floreseu and D. Kossmann. Storing and Querying XML Data Using an RDBMS[J]. IEEE Data Engineering Bulletin, 1999, 22(3): 27–34.

[43] A. Schmidt, M. Kersten, M. Windhouwer and F. Waas. Efficient Relational Storage and Retrieval of XML Documents[C]. Proceedings of the 3th WebDB International Workshop on the Web and Databases, 2000: 137–150.

[44] P. Harding, Q. Li and B. Moon. XISS/R: XML Indexing and Storage System Using RDBMS[C]. Proceedings of the 29th International Conference on Very Large Data Bases, 2003: 1073–1076.

[45] 刘云生，万常选，徐升华. 基于关系数据库有效实现RPE查询[J]. 小型微型计算机系统, 2003, 24(10): 1764–1771.

[46] S. Khalifa, H. Jagadish, N. Koudas, J. Patel, D. Srivastava and Y. Wu. Structural Joins: A Primitive for Efficient XML Query Pattern Matching[C]. Proceedings of the 18th International Conference on Data Engineering, 2002: 141–152.

[47] H. Jiang, H. Lu, W. Wang and C. Ooi. XR-Tree: Indexing XML Data for Efficient Structural Joins[C]. Proceedings of the 19th International Conference on Data Engineering, 2003: 253–264.

[48] F. Lam, W. Shui, D. Fisher and R. Wong. Skipping Strategies for Efficient Structural Joins[C]. Proceedings of the 9th International Conference on Database Systems for Advanced Applications, 2004: 196–207.

[49] H. Jiang, W. Wang, H. Lu and J. Yu. Holistic Twig Joins on Indexed XML Documents[C]. Proceedings of the 29th International Conference on Very Large Data Bases, 2003: 273–284.

[50] T. Chen, J. Lu and T. Ling. On Boosting Holism in XML Twig Pattern Matching Using Structural Indexing Techniques[C]. Proceedings of the ACM SIGMOD International Conference on Management of Data, 2005: 455–466.

[51] J. Lu, T. Chen and T. Ling. Efficient Processing of XML Twig Patterns with Parent Child Edges: A Look-ahead Approach[C]. Proceedings of the 13th ACM International Conference on Information and Knowledge Management, 2004: 455–466.

[52] 富丽贞. 基于小枝模式的XML数据查询处理技术研究[D]. 山西大学, 2008.

[53] A. Silberstein, H. He, K. Yi and J. Yang. BOXes: Efficient Maintenance of Order-Based Labeling for Dynamic XML Data[C]. Proceedings of the 21st International Conference on Data Engineering, 2005: 285–296.

[54] 郭启晶，洪晓光. XML文档结构索引的更新维护[J]. 计算机科学, 2004, 31(9): 98–101.

[55] 覃遵跃，徐洪智，卓月明. 基于分治策略的XML文档更新计算[J]. 计算机应用, 2009, 29(1): 331–333.

[56] 任家东，尹晓鹏. 一种新的基于区域的动态编码方案[J]. 计算机工程, 2006, 32(18): 79–81.

[57] 徐娟，李战怀，娄颖. 基于节点位置信息的降低更新代价前缀编码方案研究[J]. 计算机科学, 2009, 36(2): 167–171.

[58] D. Kha, M. Yoshikawa and S. Uemura. An XML Indexing Structure with Relative Region Coordinate[C]. Proceedings of the 17th International Conference on Data Engineering, 2001: 313–320.

[59] D. Florescu, R. Busse and M. Carey. XMark an XML benchmark project[EB/OL]. http://monetdb. cwi.nl/xml/index.html.

[60] 孔令波，唐世渭，杨冬青，王腾蛟，高军. XML数据的查询技术[J]. 软件学报, 2007, 18(6): 1400–1418.

[61] Dblp bibliography[EB/OL]. http://www.informatik.uni-trier.de/ ley/db.

攻读硕士学位期间发表的论文

已发表论文

[1] 赵圣猛，赵雷. 一种优化的XML文档模型映射方案[J]. 微电子学与计算机, 2009, 26(10): 174-177.

已投稿论文

[1] 赵圣猛，赵雷. 一种采用扩展Dewey编码非归并的小枝模式查询算法. 小型微型计算机系统.

致 谢

春去秋来，寒暑交替，三年的研究生生活不经意间在指尖滑过，在这里，我要向三年来给我指导、帮助、支持的所有老师和同学们表示衷心的感谢。感谢大家的支持和鼓励，帮助我顺利完成研究生阶段的学习和生活。

感谢父母这么多年的养育之恩。在我成长的道路上，你们给了我最无私的爱，包容了我的一切，让我无忧无虑的生活。你们对我的关心和鼓励是我在人生道路上前进的最大动力。

我要特别感谢我的指导老师赵雷副教授。正是因为这三年您在学习和科研上给我的莫大帮助，使我无论在理论学习，还是在项目开发上都取得了很大的进步。您总是不厌其烦地指导我解决各种各样的问题，传授我做人的道理和研究学习的态度方法。此外，您还扮演着朋友的角色，正因为您的平易近人，我们大家都愿意跟您分享自己的真实想法。

衷心的感谢杨季文教授对我的关心。杨老师治学严谨，带人谦和，总是在百忙之中抽出时间指导我们。您渊博的学识、忘我的工作热情、深刻的思想都深深地影响了我，令我受益匪浅，您将是我今后工作、学习的楷模。

感谢导师钱培德教授，您带我走进研究生生活，您的治学精神和个人魅力深深影响了我。

感谢寝室的好兄弟，感谢你们在生活上给我的帮助，在我情绪低落的时候给我以鼓励。同时还感谢实验室的伙伴们和师弟师妹，感谢你们在学习上给我的帮助，特别是在项目上给我指导，让我积累了很多经验。与你们的友谊将是我人生中宝贵的财富。

赵圣猛

二零一零年四月一十五日