# 摘 要

随着互联网的迅速发展，XML技术的不断成熟，XML已经成为Web上数据表示和交换的统一标准。研究XML数据管理技术对基于Web的应用领域有着十分重要的意义。为了有效地加工、分析和处理XML数据，国内外学者已经提出了各种XML的查询语言和存储管理技术。由于关系数据库是目前最成熟的一种数据管理技术，在存储和管理XML数据的各种方式中，基于关系数据库的XML数据存储和处理技术显然是一种可行而有效的方式，并在学术界受到了广泛的关注。

本文研究了XML数据管理的相关技术，介绍了XML语法的主要特点，对XML数据的关系存储以及XML数据的查询技术作了系统论述。本文的研究工作包括以下几方面：

首先，对XML数据的存储方式进行了归纳和总结。然后，研究了基于关系数据库的XML数据存储技术，采用结构映射的方法，实现了XML数据到关系数据库的存储。接着，研究了基于关系数据库的XML数据查询技术，采用XPath和XQuery方法，实现了对XML文档的查询。最后，对XML索引查询的一些关键技术进行了初步探讨，采用了编码的方法，实现了对XML文档进行前序-后序节点编码。

**关键词：** XML，数据管理，关系数据库，存储，查询，索引

# Abstract

With the development of Internet and XML, XML has gradually become a standard of representing and exchanging data on the Web. Researching the technology of XML data management plays an important role in applications based on the Web. To organize and manage XML data efficiently, several XML data management approaches have been proposed. Relational Database technology has been almost perfect and dominant in business data management area.Using relational database to manage XML data is a viable and effective manner.

This thesis researches the relevant technology of XML data management, analyzes the characteristics of XML, and does a thorough research and discussion on the relational storage of XML data and the querying process. The primary tasks of the paper are as follows:

First, we summarize and classify XML storage approaches. Second, we study on XML storage technologies based on relational database, and store the XML data into a relational database using Structure-mapping method. Third, we study on XML query technologies based on relational database, and do the query of XML data using XPath and Xquery. Finally, we study on some key technologies of XML index, finish the coding of XML data using Dietz-coding method.

**Key Words:** XML, data management, relational database, storage, query, index

## 声　明

本学位论文是我在导师的指导下取得的研究成果，尽我所知，在本学位论文中，除了加以标注和致谢的部分外，不包含其他人已经发表或公布过的研究成果，也不包含我为获得任何教育机构的学位或学历而使用过的材料。与我一同工作的同事对本学位论文做出的贡献均已在论文中作了明确的说明。

研究生签名：荆旦建　　2009年1月9日

## 学位论文使用授权声明

南京理工大学有权保存本学位论文的电子和纸质文档，可以借阅或上网公布本学位论文的部分或全部内容，可以向有关部门或机构送交并授权其保存、借阅或上网公布本学位论文的部分或全部内容。对于保密论文，按保密的有关规定和程序处理。

研究生签名：荆旦建　　2009年1月9日

# 1 绪论

## 1.1 论文研究的背景和意义

互联网的迅速发展，改变并冲击着整个世界，带来了经济的发展和生活的丰富，网上飞速流传的信息也越来越庞大和复杂。作为互联网最主要应用的Web已经成为最大的信息资源库。HTML（Hypertext Markup Language，超文本标记语言）在Web的发展过程中起到了很重要的作用。HTML以其简单易学、灵活通用的特性，使人们发布、检索、交流信息都变得非常简单。它继承了SGML（Standard Generalized Markup Language，标准通用标记语言）的结构化、实现独立和可描述等特点，同时对SGML作了极大简化。它还引入了超链接概念，使用尖括号作为标记以便任何文本编辑器都可创建。伴随着互联网的发展HTML成为了主要的信息发布形式。

HTML在互联网的发展过程中起到了非常重要的作用，然而，电子商务、电子出版、远程教育等基于Web新兴领域的全面兴起使得传统的Web资源更加复杂化、多样化，数据量的日趋庞大对网络的传输能力也提出更高的要求。同时，人们对Web服务功能的需求也达到更高的标准，比如：用户需要对Web进行智能化的语义搜索和对数据按照不同的需求进行多样化显示等个性化服务；公司和企业要为客户创建和分发大量有价值的文档信息，以降低生产成本，以及对不同平台、不同格式的数据源进行数据集成和数据转化等等，这些需求越来越广泛和迫切。传统的HTML由于自身特点的限制，不能有效地解决上述问题。尽管HTML推出了一个又一个新版本，已经有了脚本、表格和帧等功能，但始终满足不了不断增长的需求。于是，可扩展标记语言XML(eXtensible Markup Language)应运而生。1998年2月，全球互联网联盟W3C(World Wide Web)推出了XML1.0的正式版本，并正式推荐XML作为下一代互联网数据的表示规范。

XML是SGML的一个简化子集，它将SGML的丰富功能与HTML的易用性结合到Web应用中，以一种开放的自我描述方式定义了数据结构，在描述数据内容的同时能突出对结构的描述，从而体现出数据之间的关系。XML同HTML兼容，具有平台无关性，同时又是一种真正的扩展语言，能以可读的格式提供数据而又不受到表现形式的限制，因此，XML的灵活、开放和基于标准的格式使它很快变成商业世界中用来交换商业数据的最广泛使用的语言，通过XML数据发布和交流信息已被企业广泛采用。目前，以XML为主导的一系列规范已成为新一代的Web标准。

XML凭借自身的优越性，已经逐渐成为网络上数据表示和交换的标准。XML数据在Web上广泛流行，从企业报表、商业广告到技术文档，几乎无所不在。如何有效地管理XML数据成了Web应用领域中一个亟待解决的重要问题。XML数据管理技术已经成为了XML技术研究内容的关键组成部分，一直是XML技术研究中的热点和焦点。

国外的许多大学、研究机构和各种基金都已经或正在开展 XML 数据处理技术的研究。

## 1.2 国内外研究现状

在国外，XML 与关系数据库的相关技术已经引起了高校科研院所及商业数据库公司的关注，其中有代表性的有美国 STANFORD 大学的 Lore[31]项目，Lore 主要的研发重点是为 XML 定义数据模型与查询语言，支持 XML 数据的存储与处理；还有加拿大多伦多大学的 XML 搜索引擎 TOX[31]；法国国家信息与自动化研究院(INRIA)的 Xyleme 计划的 Dynamic Data Warehouse for the XML Data of the web(web 上 XML 数据的动态数据仓库)项目[31]，该项目的设计目标是将互联网上的所有 XML 数据都整理装入 XML 数据仓库。XML 已经成为近年来的 SIGMOD、VLDB、ICDE 和 PODS 等数据库顶级会议的研究热点。

目前，许多主流的数据库厂商如微软、oracle、IBM 和 Sybase 等都在把 XML 支持结合到其产品中，或者提供可在其数据库中使用 XML 的工具，微软的.NET 技术就是基于 XML 的架构上的。这些计算机业的巨头都非常看好 XML 的前景，大有得 XML 者得天下之势。

在我国，对 XML 也展开了初步的研究及应用，于 2000 年 12 月 27 至 28 日由国家高技术发展计划智能计算机主题专家组主持，在北京召开了“中国 XML 技术 2000 研讨会”，研讨会较为全面地反映了目前我国 XML 的研究水平。国内相关研究则起始于新世纪之初，一些大学及有实力的软件公司、科研所都进行了相关的研究。如因科公司的因科 XML 网关(IXG)成功实现了 XML 和传统关系数据库的实时转化和交互，以及 XML 文件和 VoiceXML、WML 等之间的实时自动翻译等；中科院软件研究所的 XML 与关系数据库转换系统 Xtrans 也实现了从 XML 文档到关系数据库数据，以及从关系数据库数据到 XML 文档的双向转换。目前我国对 XML 的研究取得了一定成果，但是仍然有很多领域需要去开拓。

## 1.3 论文的主要内容

本文在深入了解 XML 数据管理技术研究现状的基础上，详细描述了基于关系数据库的 XML 数据管理系统的主要技术，重点研究了 XML 的存储、查询以及索引技术。存储方面主要研究了基于 DTD 的关系映射方法，查询方面研究了 XPath 和 XQuery 两种技术，索引方面重点研究了前序-后序节点标号的方法。

本文的内容组织如下：

第一章是绪论，主要介绍了 XML 的特点和发展现状，以及 XML 数据管理技术的研究意义。

第二章介绍了 XML 的基本概念和相关标准，以及 XML 技术的相关应用，这是实

现 XML 数据管理的基础。

第三章详细描述了几种主要的 XML 数据管理技术。

第四章详细描述了一个基于关系数据库的 XML 数据管理系统的实现过程，实现了对 XML 数据的存储、恢复、更新和查询等功能。

第五章对 XML 的索引技术进行了研究和探索，重点研究了前序-后序节点标号法的索引技术，实现了对 XML 文档进行编码。

最后一章是对本文工作的总结和展望。

# 2 XML 预备知识

## 2.1 XML 基本概念

### 2.1.1 XML 的由来

1969年，IBM公司由Goldfarb领导的研究项目组创建了通用置标语言GML(General Markup Language)，用来编辑和格式化文本以及在子系统之间实现信息检索。GML可用于标记任何数据集合的结构，是一种元语言（meta-language），能够描述其他语言及语法和词汇。1986年10月，GML被国际标准化组织采纳为国际性的数据存储和交换标准，并被更名为“标准通用置标语言”（SGML）。

1989年7月起，W3C组织60多位精心挑选的SGML信息结构专家专门成立了一个SGML小组，提出了“网络上的SGML”计划。该小组删除了SGML中非核心的、未被使用的和含义模糊的部分，剩下了短小精干的标记工具，这就是XML。XML保留了SGML80%的功能，而将其复杂程度降低到原来的20%。W3C于1998年批准了XML的1.0版本。

### 2.1.2 XML 语法

一个文档称为XML文档，它必须是格式良好的，也就是说按照W3C制定的XML规格标准语法或语义是正确的，格式不是良好的文档不能被接收处理，浏览器无法正常显示。格式良好的XML文档简化了解析器的内部代码，加快了文档解析速度。一个格式良好的XML文档由三个部分组成：可选的序言（prolog），包括注释、处理指令和文档类型定义等；文档的主体（body），由一个或多个元素组成，其形式为一个可能包含字符数据的层次树；可选的尾声（epilog），其中内容包括注释、处理指令和紧跟后面的空白。图2.1.2.1是一个格式良好的XML示例文档。

文档的第一行是一个XML声明，XML声明是XML处理指令的一种，XML文档中的处理指令以“<?”和“?>”标识，一个XML文档以一个XML声明作为开始。XML声明给出了XML文档采用XML版本号，是否和外部DTD配合使用，以及数据所采用的编码方式等信息。

文档的第二行起是XML文档的实质内容，即XML文档所表示的数据。XML通过元素来组织XML数据，元素是XML文档内容的基本单元，XML元素包括标记和字符数据。一个元素包含一个起始标记、一个结束标记以及标记之间的数据内容。XML文档中有元素、属性、文本等几种基本的数据类型。此外，还有一些其他的结构用以实现一些特殊的功能，如XML指令等。一个元素的形式是：<标签 属性名=属性值……>数据内容</标签>。其中数据内容可以是元素的值，或者是其他元素，也可以是两者的混

合。每个 XML 文档有且仅有一个根元素，其他任何元素都是根元素的子孙，即都出现在根元素的内部。

```
<?xml version="1.0" encoding="gb2312"?>
<books>
    <book>
        <name>xml 应用</name>
        <author>张三</author>
        <price>55.00</price>
    </book>
    <book>
        <name>asp 应用</name>
        <author>李四</author>
        <price>65.00</price>
    </book>
    <book>
        <name>java 应用</name>
        <author>王五</author>
        <price>34.00</price>
    </book>
</books>
```

图 2.1.2.1 book.xml 文档示意图

### 2.1.3 XML 文档的结构

完整的 XML 文档包含物理结构和逻辑结构，文档的物理结构和逻辑结构均可适当嵌套。

1）XML 文档的物理结构[1]

在物理结构方面，整个 XML 文档由一系列称为实体（Entities）的存储单元组成，每个实体均包含一定的内容并通过名字命名。实体按照树型结构组织，其根节点是文档实体（Document Entity），包含整个 XML 文档的内容。文档实体是个特定的实体，是 XML 处理程序的起始工作点，但却不指定名字且不指定处理程序如何定位。实体根据所包含的内容可分为解析实体（Parsed Entities）和非解析实体（Unparsed Entities）两类，其中解析实体的内容可通过其名字被其他实体引用（Reference），其内容是作为整个文档的一个完整性组成部分；而非解析实体则包含了非 XML 内容的资源，XML 对非解析实体的内容并没有约束。

2）XML 文档的逻辑结构[1]

在逻辑结构上，XML 文档由若干元素（Elements）组成，每个元素均通过起始标记和结束标记界定（只有空元素例外，没有结束标记）并由一个通用的标识符命名，还可包含一些属性的说明。元素的定义包括对元素类型、属性列表及相应的约束条件的指定，并可进一步说明下级子元素允许的类型。所有的属性定义均采用名字－数值对的形式。

XML 文档包含混合的数据和标记，标记包括起始标记、结束标记、空元素标记、实体引用、字符引用、注释、数据分节、文档类型说明及处理指令。文档的非标记部分构成整个文档的字符数据。XML 处理程序将属性值传送到上层应用进行有效性检查之前必须首先对属性值进行规格化处理，即对实体的引用进行递归处理并对字符串参数中的空格进行处理，然后将数值赋给属性。

XML标记的主要作用是描述XML文档的存储和逻辑结构并说明其属性，通过XML文档的类型说明可定义逻辑结构和存储单元之间的约束。XML 文档的 DTD 通过一定的语法格式进行，可包含一个指向内部或外部的标记子集，并可以采用独立文档说明的方式指定在涉及到外部标记子集时文档实体的缺省属性以帮助 XML 处理程序工作。另外，可选部分是文档类型说明的外部子集中未包含在逻辑结构中的部分，和外部文档类型说明子集一样可包含若干说明、注释、处理指令或内嵌的部分。

## 2.2 XML 的相关标准

### 2.2.1 DTD 和 Schema

DTD（Document Type Definition，文档类型定义）是一套关于标记符的语法规则。它是XML1.0版规格的一部分，是XML文件的验证机制，属于XML文件组成的一部分。DTD 是一种保证 XML 文档格式正确的有效方法，可以通过比较 XML 文档和 DTD 文件来看文档是否符合规范，元素和标签使用是否正确。一个 DTD 文档包含：元素的定义规则，元素间关系的定义规则，元素可使用的属性，可使用的实体或符号规则。XML文件提供应用程序一个数据交换的格式，DTD 正是让 XML 文件能够成为数据交换的标准，因为不同的公司只需定义好标准的 DTD，各公司都能够依照 DTD 建立 XML 文件，并且进行验证，如此就可以轻易的建立标准和交换数据，这样满足了网络共享和数据交互。

XML Schema 是一种描述信息结构的模型，用来定义 XML 文档的文本结构、数据类型等 XML 文档描述规则。它为一类文档建立一个模式，规范了文档中的标记和文本的可能组合形式。它不仅包括了 DTD 所能实现的所有功能，而且它还提供了一系列的新特色，大大弥补了 DTD 的不足。XML Schema 本身就是 XML 文档，使用标准 XML 语法，因此可直接用一般 XML 解析器对其进行语法分析，并且有强大、易用的扩展功能；XML Schema 支持丰富的数据类型包括数字型、布尔型、整型、日期时间、URI(统一资源标识符)、十进制数等等，它允许对数据进行更严格的合法性检查，而且还支持由这些简单的类型生成更复杂的类型；XML Schema 支持名字空间，保证了标记的唯一性，利用名字空间将文档中特殊的节点与 Schema 说明相联系，一个 XML 文档可以有多个对应的 Schema。

### 2.2.2 XPath 和 XQuery

XPath 即为 XML 路径语言(XML Path Language)主要用于对文档元素进行寻址，与此同时，它也为字符串、数字和布尔操作提供了基本手段。之所以要引入 XPath 的概念，目的就是为了在匹配 XML 文档结构树时能够准确地找到某一个节点元素。可以把 XPath 比作文件管理路径：通过文件管理路径，可以按照一定的规则查找到所需要的文件；同样，依据 XPath 所制定的规则，也可以很方便地找到 XML 文档结构树中的任何一个节点。XPath 将一个 XML 文档视为一棵树进行操作，为此 XPath 定义了树状模型（Tree Model）。该模型仅仅是概念上的，并且不要求任何特定实现。该树包含了七种节点类型，分别为：根节点、元素节点、正文节点、属性节点、命名空间节点、处理指令节点和注释节点。每种类型的节点都能被映射成一个字符串值。在默认情况下，XPath 以文档次序（Document Order）遍历文档树。

XQuery 是一种功能强大的数据查询语言，它能够从 XML 文档中选择并抽取出复杂的模式，进而把查询结果重构成用户需要的新的 XML 文件结构。它极有可能成为 XML 中的 SQL, 因为 SQL 语言是对建立在关系代数基础上的数据表进行操作的语言, 而 XQuery 则是对建立在 XQuery 数据模型上的 XML 数据进行查询的语言，并且它还是一种书写格式自由的语言。

XQuery 是基于 Quilt[9]语言提出的，综合了 SQL、OQL、XML-QL、XQL 及 Lorel 等诸多语言的特点。其语法采用范式表达。一个 XQuery 查询由查询序言和查询体组成。查询序言是一系列声明和定义，建立了查询处理的环境。主要包括命名空间声明、模式导入、xmlspace 声明、缺省的 collation 以及函数定义。查询体是待求值的一段查询表达式，是 XQuery 的主体部分。从 XQuery 语言的语法构成可以看出 XQuery 具有以下特点：（1）XQuery 是一种典型的函数语言。XQuery 建立在被成为查询的表达式之上，而不是建立在语句之上。在该语言中，除了查询序言之外的每个构造都是一个表达式。表达式能够被任意组合，并允许嵌套。只要类型相符，一个表达式的结果可以作为另一个表达式的输入。（2）XQuery 是一种强类型语言。XQuery 的类型系统是基于 XML Schema 的，可以从一个或几个 XML Schema 中导入类型，然后 XQuery 能基于这些类型进行查询运算。此外，XQuery 还支持静态类型分析，这意味着系统能够针对查询的类型执行某些推断，这种推断是建立在查询输入的类型基础上的。静态类型分析能够较早地检测错误，可以成为某些形式化的基础。

### 2.2.3 SAX 和 DOM

SAX 的全称是 Simple APIs for XML，也即 XML 简单应用程序接口。SAX 并不是由 W3C 官方所提出的标准，可以说是“民间”的事实标准。实际上，它是一种社区性质的讨论产物，是 XML_DEV 邮件列表中的成员根据应用的需求自发定义的一套对

XML文档进行操作的接口规范。SAX成功的一个主要原因是初始的规范，并为许多流行的XML解析器提供了前端驱动。其它解析器提供商如IBM、Sun和Oracle等很快在它们的解析器中集成了最早的SAX接口，这样它们的产品就可以和现有的应用程序一起运行。最终的SAX规范是根据Java接口书写的。

SAX提供了一种对XML文档进行顺序访问的模式，这是一种快速读写XML数据的方式。当使用SAX分析器对XML文档进行分析时，会触发一系列事件，并激活相应的事件处理函数，从而完成对XML文档的访问，所以SAX接口也被称作事件驱动接口。用解析器读取文档时，当解析器发现标签时就会告知程序它发现的标签，包括会告知它何时发现了一个开始标签，何时发现了一些特征数据，以及何时发现了一个结束标签，这就叫做事件驱动接口，因为解析器告知应用程序它遇到有含义的事件。事件驱动是用来编写应用程序以响应发生的如鼠标点击等事件，事件驱动解析器就是一旦事情要开始发生，你不需要调用解析器，而是解析器调用程序。SAX接口之所以叫做简单应用程序接口，是因为这个接口确实非常简单，绝大多数事情分析器都没有做，而是需要应用程序自己去实现。它的基本原理是由接口的使用者提供符合定义的处理器，XML分析时遇到特定的事件，就去调用处理器中特定事件的处理函数。SAX接口规范是XML分析器和XML处理器提供给XML更底层的接口，它能为应用提供较大的灵活性。

DOM文档对象模型(Document Object Model，文档对象模型)是基于树型结构的API，是由W3C制定的标准接口规范。DOM依据XML文档的结构将其转换为树型结构的文档对象模型，用户通过对该对象模型的访问，可以动态地创建文档，遍历文档结构，对XML文档中的数据进行修改、移动、删除和插入等操作。之所以要定义这样一个接口，是因为如果在使用XML数据时没有一个统一的接口，每个程序员就要去处理XML语法细节，那些XML的语法分析器也必然有一个接口让程序员通过它访问XML数据。如果所有语法分析器的接口都不相同，开发就必须针对某一个分析器，当换了另一个分析器时，就不得不重写程序。正如数据库有标准的ODBC/JDBC这样的接口规范一样，要做XML应用开发，就需要有一个统一的XML数据接口—DOM。XML DOM的设计概念为，将待操作的XML文件放入XML DOM树对应的各种节点之中，也就是当用DOM来操作XML文件时，XML文件中所有的内容都变成了XML DOM树上的一个节点。

DOM分析器是把整个XML文档转化成DOM树放在了内存中，因此应用程序可以随时对DOM树中的任何一部分进行访问与操作，没有访问与操作次数以及文档大小的限制，并且可以通过修改文档树，进而修改XML文档。但当XML文档比较大或者文档结构比较复杂时，对内存的需求就比较高且耗时。所以DOM分析器对机器性能的要求比较高，实现效率不十分理想。不过，由于DOM分析器的树结构的思想与XML文档的结构相吻合，而且通过DOM树机制很容易实现随机访问，因此DOM分析器有很

广泛的使用价值。SAX 分析器与 DOM 不一样，它在对 XML 文档进行分析时，触发一系列的事件，应用程序通过事件处理函数实现对 XML 文档的访问，但不能处理任意大小的 XML 文档。由于事件触发本身是有时序的，因此 SAX 分析器提供的是一种 XML 文档的顺序访问机制，不支持对文件的随意存取，并且只能读取 XML 文档内容，而不能修改，开发上比较复杂，需要自己来实现事件处理器。对程序设计人员而言，可以用 SAX 建立自己的 XML 文档模型。因此，SAX 和 DOM 相比显得更灵活，有自己独特的优点。

基于对 SAX 和 DOM 的分析与比较，它们有不同应用领域和发展前景：(1)SAX 适合处理下面的问题：对大型文件进行处理；只需要文件的部分内容，或者只需要从文件中得到特定信息。(2)DOM 适合处理下面的问题：需要对文件进行修改；需要随机对文件进行存取，例如 XSLT 解析器。随着网络编程技术的发展和 XML 的广泛应用，SAX 技术与 DOM 技术凭借其各自的优势将被业内人事广泛地应用到今后的应用程序编写中，它们将会伴随 XML 技术的发展而不断地发展和完善。

## 2.3 本章小结

本章主要介绍了 XML 的基本概念、语法和一些相关技术标准，如 DTD 和 Schema、XPath 和 XQuery、SAX 和 JDOM 等。概括地说，DTD 和 XML Sehema 是用来表述 XML 的内容和结构的，规定了 XML 文档中能出现什么、不能出现什么和以哪种方式出现；XPath 用来定位文档中的节点，是其它很多标准的基础；XQuery 是 W3C 推荐的基于 XML 的查询语言；SAX 和 DOM 都是用来解析 XML 文档的规范。以上这些内容是进行 XML 数据管理的基础。

# 3 XML 数据管理技术

目前 XML 数据的存储格式可以分为四种：文本文档格式、对象类模式、关系表模式和为 XML 数据专门设计的存储结构[31]。对应不用的存储策略，有着不同的 XML 数据管理技术，它们分别是基于文件系统的 XML 管理技术、基于面向对象数据库系统的 XML 管理技术、基于关系数据库系统的 XML 管理技术以及所谓的 Native XML 数据管理技术[23]。

## 3.1 基于文件系统的管理技术

### 3.1.1 基于文件系统的管理技术的概念

XML 数据的一般表现形式是文本文档，存储于文件系统中。因此管理 XML 数据的最简单而又直接的方法就是直接将它们存储于文件系统中，基于文件系统进行 XML 数据的管理与查询操作。在这种方式中，XML 数据的访问入口是文件系统提供的句柄。目前有三种查询方法是基于这种存储方式的：基于信息检索技术的关键词查询、采用 XSLT 语言的文档格式转换形式的查询以及用户定制的基于 DOM 或 SAX 接口的查询。

### 3.1.2 基于文件系统的管理技术的原理

基于文件系统的 XML 管理系统的管理方式如图 3.1.2.1 所示。XML 文档被直接存储到文件系统中，借助文件系统的管理技术来查询 XML 数据。

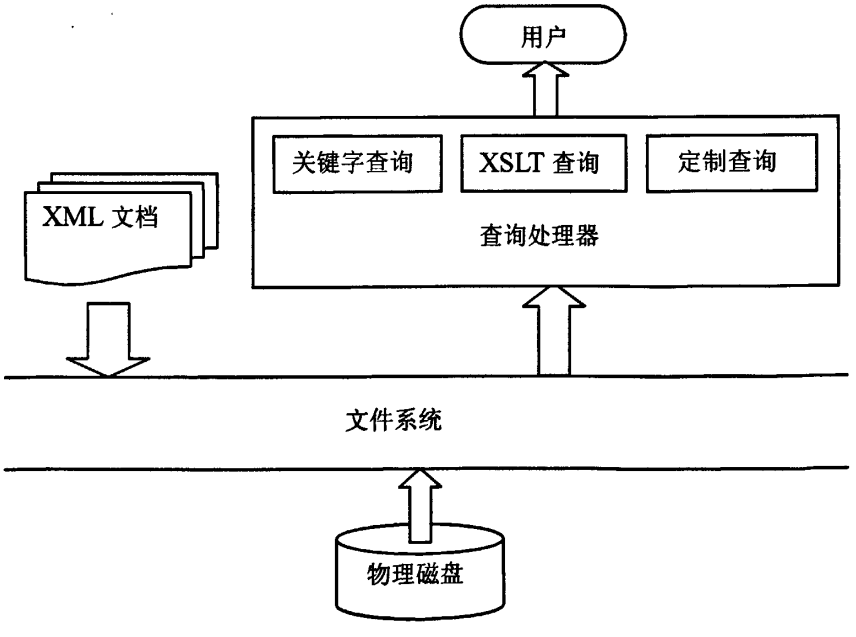



图 3.1.2.1 基于文件系统的 XML 数据管理系统结构图

基于文件系统的 XML 管理方式的优点是：存储方便，不需要任务额外操作；信息检索技术可以直接应用；可以使用基于 DOM 或 SAX 的工具。缺点是：没有存储控制；不能为结构化查询提供支持；IR（Information Retrieval，信息检索）方法查询功能简单；DOM 方式无法支持大文档查询；每查询一次都要重新解析。因此，对于大规模的 XML 数据存储与管理，基于文件系统的方法是不适用的。

## 3.2 基于对象数据库的管理技术

基于对象数据库的管理技术是使用对象数据库来存储和管理 XML 数据。ODMG 标准为面向对象数据库定义了对象模型（Object Model）、对象定义语言（Object Definition Language,ODL）、对象操作语言（Objece Manipulation Language,OML）和对象查询语言（Object Query Language,OQL）。OQL 语言提供了对于复杂查询的支持，包括简单路径表达式查询以及嵌套查询等。因此，一些系统采用对象数据库系统管理 XML 数据。

文献[7]提出了两种基于对象数据库的 XML 数据存储映射技术。第一种是用不变的 ODMG ODL 模式存储 XML 数据，模式类似于 XML 文档对象模型（DOM）。在该模型中，element 类型被分为两类：TAG 与 PCDATA，分别对应于普通元素节点和文本节点；attribute 类型有四种，分别为 CDATA、IDREF、ID 和 IDREF。这种方法适合于无模式的 XML 数据。第二种方法则把每一种 XML 元素类型映射为一个对象模型，XML 元素之间的关系使用 ODMG 提供的类之间的联系来实现，适用于处理有模式的 XML 数据。虽然对象数据库模型以其复杂类型支持和丰富的查询语义，使得它在表示与查询 XML 数据上有着天然的优势，然而商用对象数据库的技术还没有得到广泛的认可和应用，技术发展得并不是很成熟，特别是对大规模数据的复杂查询能力还有欠缺。

## 3.3 Native XML 数据管理技术

### 3.3.1 Native XML 数据管理技术的概念

作为一种新的数据格式，XML 数据有着与关系型数据和对象数据都不同的性质，因此专门为 XML 数据设计存储模型与索引技术，研究特定的查询处理技术以及查询优化策略成为目前的一个技术热点。这样的 XML 数据管理系统被称为 Native XML 数据库系统。Lore 以及 Tamino 等都是这样的 XML 数据库系统。

### 3.3.2 Native XML 数据管理技术的原理

Native XML 数据管理技术的结构如图 3.2.2.1 所示。XML 数据文档以某种模式存储于 XML 数据库中，XML 数据查询由 XML 查询处理器直接处理，不需要任何形式的翻译与转换操作，查询结果本身就是 XML 数据，可以直接发布。

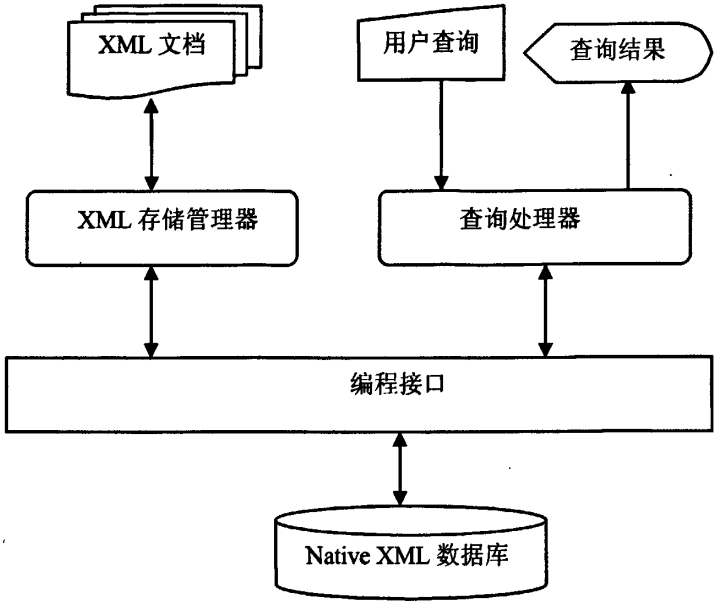



图 3.3.2.1 Native XML 数据管理系统结构图

## 3.4 基于关系数据库的管理技术

### 3.4.1 基于关系数据库的管理技术的概念

基于关系数据库的 XML 数据管理系统就是要将 XML 数据存储在关系数据库系统中，然后借助关系数据库系统的成熟、完备的技术实现对 XML 数据的管理。这种 XML 数据管理系统的基础平台是关系数据库系统。系统的主要设计思想是:首先，利用 XML 文档模式信息设计关系表结构。然后，解析 XML 文档，结合模式信息，将 XML 文档的内容分别存储到对应的关系表中。这使得查询操作不需要再访问 XML 文档。完全依赖于关系数据库系统。最后，利用 XML 文档的模式信息将得到的查询结果构建成 XML 文档格式。

根据生成关系模式的过程不同，基于关系数据库的 XML 管理方式可以分为两大类。第一种方式中，与 XML 数据相关的关系模式是从 XML 数据相对应的 DTD 信息中提取的，具有不同 DTD 定义的文档所对应的关系模式也是不同的，这样的方法被称为依赖于文档（Document-dependent）的方法。另一种方式中，由于 XML 数据可以被看作是一棵文档树，所以可以从如何使用关系模式描述树结构入手，开发新的 XML 存储技术，这种方法被称为独立于文档（Document-independent）的方法。在独立于文档的方法中，XML 文档的 DTD 信息是不需要的，任何格式的 XML 文档对应的关系模式也都是一样的。由于依赖于文档的方法是基于 XML 文档的逻辑结构来构建关系模式，而独立于文

档的方法是基于 XML 数据的树型结构建关系模式，因此这两种方法也被分别成为结构映射（Structure-mapping）方法和模型映射（Model-mapping）方法[23]。

### 3.4.2 基于关系数据库的管理技术的原理

基于关系数据库的 XML 管理系统的结构如图 3.4.2.1 所示。

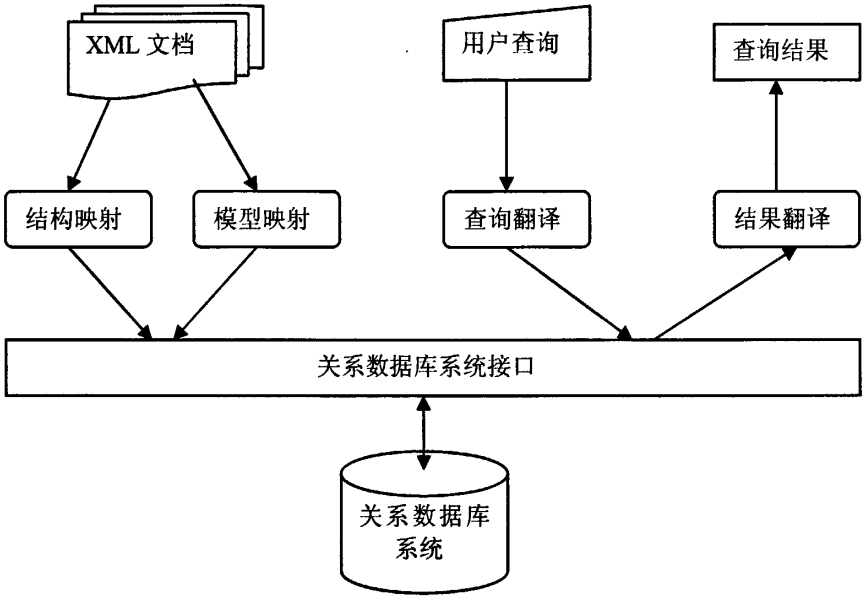



图 3.4.2.1 基于关系数据库的 XML 管理系统结构图

基于关系数据库管理 XML 的基本过程描述如下：

1）将 XML 数据转化为关系模式存储到关系表中。

2）利用模式信息和统计信息将采用 XML 模型的查询（XQuery 或 XPath）转换成 SQL 语句。

3）将转换后的 SQL 语句进行优化，然后执行。

4）将执行结果转换并重构为 XML 格式提供给用户。

### 3.4.3 结构映射方法

结构映射方法就是使用 XML 数据的 DTD 信息产生存储 XML 文档的关系模式。这是一种依赖于文档的方法，是由 Shanmugasundaram 等人提出。结构映射方法分为三种：Basic-inlining、Shared-inlining 和 Hybrid-inlining[31]。无论是哪种方法，在进行 XML 数据存储映射之前要做的工作都是简化 DTD 模式。由于 DTD 内的一些语法是与 XML 存储无关的，例如在 DTD 中定义了<!ELEMENT a(b*,(c | d | e)*,c,e)?>这样一个复杂的元素，对于 XML 的存储而言只需要知道 a 的孩子节点有可能是类型 b、c、d 和 e 就足够了。因此在映射之前，需要将 DTD 内的层次嵌套关系简化，可参见文献[29]。

Basic-inlining 方法的原则是在存储一个元素节点的时候，尽可能多地将这个元素的后代节点存储在一个数据表中，例如形如直线而无分支的数据路径上的节点。这些后代节点作为祖先表中的属性域而存储，而节点之间的嵌套关系则采用关系表之间的外键来解决。

在 Shared-inlining 映射方式下，为以下三种 DTD 节点生成独立的关系：入读为 0 的节点，即根节点和入度大于 1 的元素节点；节点的孩子节点（将值集元素作为单独的关系保存）；出现在环上的入度为 1 的节点。生成的关系以节点的元素类型命名，DTD 中其他节点被内联为其最近祖先节点对应的表中，作为某个属性而存在。为了保存边信息，每个关系表中存储被映射节点的标识，并增加其父节点的标识。

Hybrid-inlining 与 Shared-inlining 的不同之处在于仅将边指向的节点和环路上的节点存储为单独的关系，而其他节点被内嵌入祖先节点中。入度为 1 的节点根据其父节点而被分割到不同的关系表中。

### 3.4.4 模型映射方法

1）模型映射的主要方法

模型映射方法是目前在数据库中分解存储 XML 的一类重要方法。目前，XML 模型主要有 XPath、 Edge DOM、Infoset 等几种[37]。在基于模型的映射方法中，XRel、Xparent 是基于 XPath 的模型映射方法，Edge 是基于 Edge 的模型映射方法。下面将对 XRel、Xparent、Edge 等方法做一个简单的介绍。

① XRel 方法

根据 XML 语法，所有的 XML 文档都是一棵树。在 XRel 中，则进一步将这些节点分为七类，每一个节点都具有一个路径值，路径值记录了从根节点到该节点的路径信息。为了能够恢复文档，节点还必须保留前序信息和祖先后代信息，XRel 用一对整数来记录这些信息，对于有值的节点（属性节点和文本节点）还要记录这些值。这样 XRel 就能将一棵树分解成多个节点的集合，同时，也有足够的信息能将这个节点集恢复成一棵树。XRel 定义了四个关系：

Element(docID, pathID, start, end, index, reindex)

Attribute(docID, pathID, start, end, value)

Text(docID, pathID, start, end, value)

Path(pathID, pathexp)

来存储模型分解后得到的节点集，从而实现模型向关系数据库的映射。

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，尤其是它的 PATH 信息具有很大的冗余信息，一个简单的修改节点名字的操作，都会需要相当复杂的操作。

② Xparent 方法

Xparent 方法也是基于 XPath 模型的。它定义了四个关系来实现 XPath 模型向关系数据库模式的转换：

LabelPath(ID, Len, Path) //每一个路径都有一个唯一的 ID，path 中存储了具体路径，len 记录了路径中包含的边的条数。

DataPath(Pid, Cid)PID //是父亲节点的 ID，CID 是儿子节点的 ID。这也是该方法称为 Xparent 的原因。

Element(PathID, Did, Ordinal) //根据 PathID 可获得该元素的路径，DID 是节点被赋予的唯一 ID，Ordinal 记录了该节点在其兄弟节点中的序号。

Data(PathID, Did, Ordinal, Value) //PathID, Did, Ordinal 含义同上。Value 记录了该数据节点的文本值。

Xparent 方法的最大优点也在于它与 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，但同时它也存在路径冗余的缺点。

③ Edge 方法

Edge 方法中，将 XML 数据存储在一个数据表 Edge 中，该表的模式如下：

Edge (Source, Ordinal, Target, Label, Flag, Value)

Source 记录了一条边的源节点的 ID，Target 则记录了该边的目的节点的 ID，Ordinal 记录了目的节点在其兄弟节点中的序号。Label 则记录了源点的 Label 值。Flag 指示该目的节点是一个元素节点，还是一个文本节点，如果是文本节点的话，则其文本值被存储在 Value 中。

Edge 方法的最大优点在于它存储 XML 文档仅需要一个表，表的设计相当的简洁，但又具有足够的信息。降低了管理多个表所带来的复杂性，避免了在多个表之间进行连接查询。在一个具体的 Edge 表中，空值比较少，节省了存储空间。它的缺点在于，以边形式存储 XML 文档，其恢复算法比较复杂，效率比较低。其次，对查询的支持力度比较小，如果执行一条基于 PATH 的查询，通过边表获得一条路径，操作相当复杂，远不如 XRel 和 Xparent。

2） 模型映射执行过程

模型映射方式的执行过程大致如下：

① 定义一个类似于 DOM 的通用 XML 文档模型。一般而言，这个模型是一个树状结构。

② 设计一个(或者多个) 能够存储这个模型的关系模式。

③ 提供一个算法能够将具体的 XML 文档转换为该模型的实例，并将该实例分解存储到数据库相应的表中去。

在存储具体的 XML 文档的时候，首先，将文档数据转换为该模型的实例；其次，

运用映射将 XML 数据映射成符合关系数据库的数据，并存储到数据库中。从数据库存储数据合成（恢复）原始 XML 文档的过程则是上述过程的逆过程。

3） 模型映射方案的特点：

① 模型映射后，原有数据表的数目和字段都是确定的。

② 模型映射可以保证数据的完整性，不会造成数据的丢失。

③ 模型映射支持基于内容和结构的查询。

④ 模型映射也适用于无模式的文档。

## 3.5 本章小结

本章主要介绍了几种不同的 XML 数据管理方法：基于文件系统的方法、基于对象的方法、基于关系的方法和 Native XML 数据管理方法。其中基于关系的方法是目前应用最广泛的，STORED、Agora、Monet 和 VXMLR 等都是使用关系数据库来存储和查询 XML 数据。在这些系统中，XML 数据按照某种规则映射成关系表进行存储，针对 XML 的查询则被翻译成 SQL 语句，而真正的查询操作是由底层关系数据库系统完成的。

# 4 基于关系数据库的XML数据管理系统

## 4.1 系统概述

本章我们将详细描述一个XML数据管理系统的实现过程，该系统是基于关系数据库而实现的。将XML存储到关系数据库中就是利用XML文档模式信息设计关系表结构，然后解析XML文档，结合模式信息，将XML的内容分别存储到对应的关系表中。利用关系数据库系统来处理XML数据的方式具有如下的优点：一方面，当前的关系数据库的技术十分成熟，商用关系数据库系统都具有高性能的查询引擎，良好的可扩展性、安全性和稳定性，因此，利用关系数据库系统管理XML数据可以重用数据库的查询优化器和事务处理机制，能够保证XML数据的一致性和完整性；另一方面，目前大量的WEB数据主要存放在关系数据库中，采用这种方法便于在关系数据库上建立适于二者的应用，使关系数据库在WEB领域发挥更大的作用。

## 4.2 系统分析

我们要实现的是一个XML数据管理系统，它需要具备数据管理中所需要的基本功能：数据的存储、数据的删除、数据的更新和数据的查询等。我们用数据流程图和系统功能图对这个系统进行分析。

### 4.2.1 系统数据流程图

数据流程图(DFD)是结构化的系统模型设计方法。DFD能够把复杂的业务逻辑结构用数据流程的方法表示出来，它采用了四种简单的记号来描述流程图，分别是：(1)数据流(2)数据存储集(3)处理功能(4)外部实体。我们采用自上而下，逐层展开的方法绘制这个数据流程图。

1）顶层数据流程图

系统可以实现存储XML文档，并对XML文档进行管理，可以将系统中保存的数据表逆转换为XML文档到指定位置。如图4.2.1.1所示：

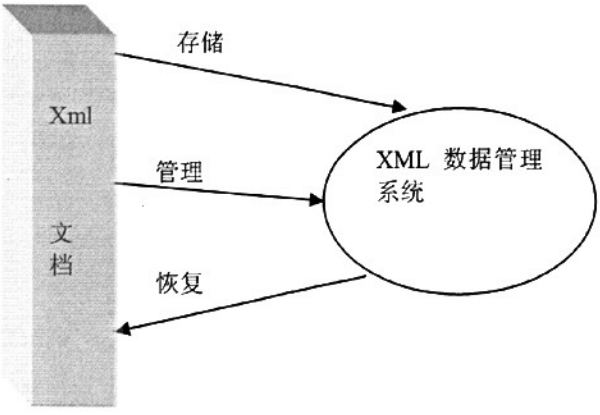



图 4.2.1.1 顶层数据流程图

2）一层数据流程图

一层数据流程图是对顶层数据流程图的细化，依据自顶向下，逐步求精的准则细化。本系统的一层数据流程图如图 4.2.1.2 所示。

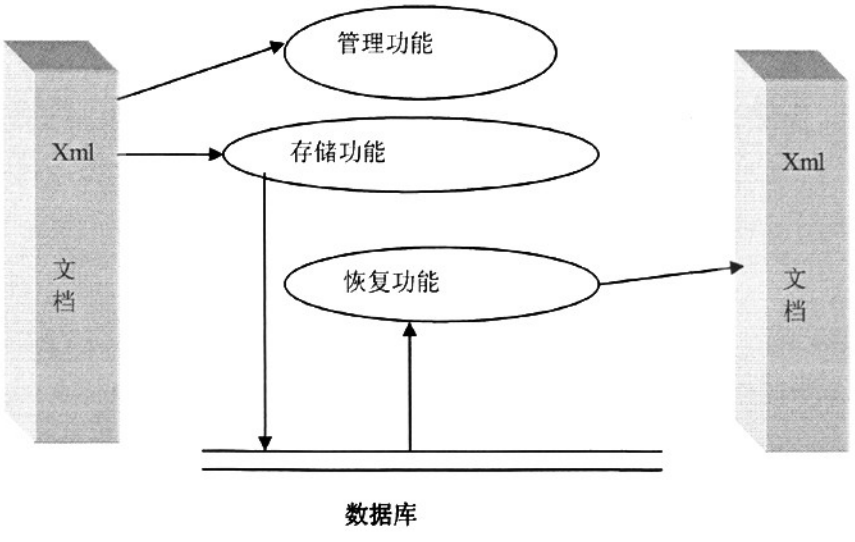



图 4.2.1.2 一层数据流程图

从数据流程图中我们可以分析出,这个系统主要由 3 大部分构成：XML 数据的存储、XML 数据的管理以及 XML 数据的恢复。数据库在系统中起到了存储数据的中间

介质作用。

### 4.2.2 系统功能结构图

如图 4.2.2.1 所示，系统总的分为三个模块，XML 文档的存储模块、XML 文档的恢复模块、XML 文档的管理模块。存储模块实现从 XML 数据到关系数据表的转换；恢复模块实现从数据表到 XML 文档的逆转换；管理模块实现对 XML 文档中的数据进行更新、查询等操作管理。

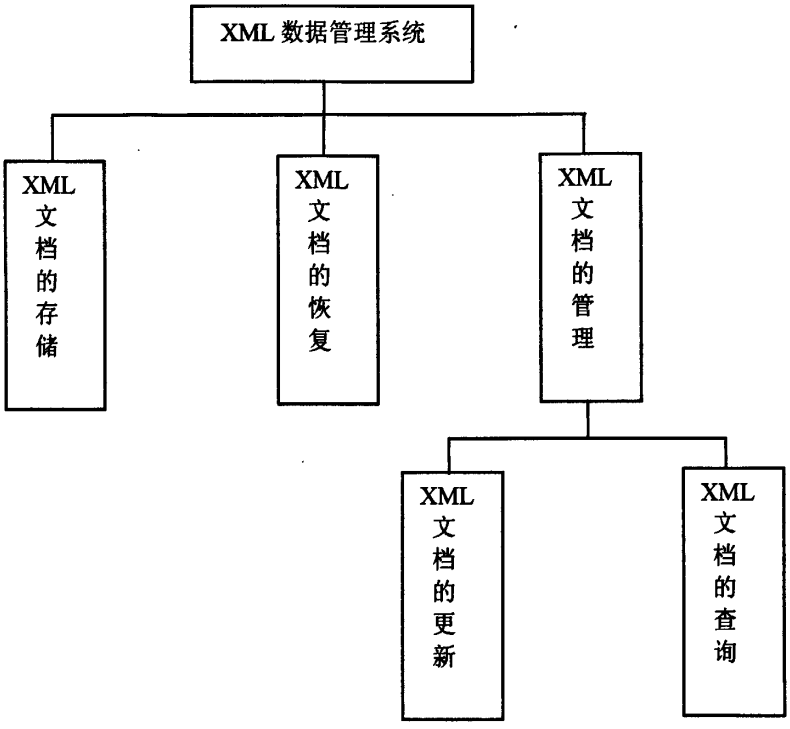



图 4.2.2.1 系统功能结构图

### 4.2.3 系统用例图

下面我们使用 UML 建模，模拟用户使用这个系统来分析系统的体系结构。

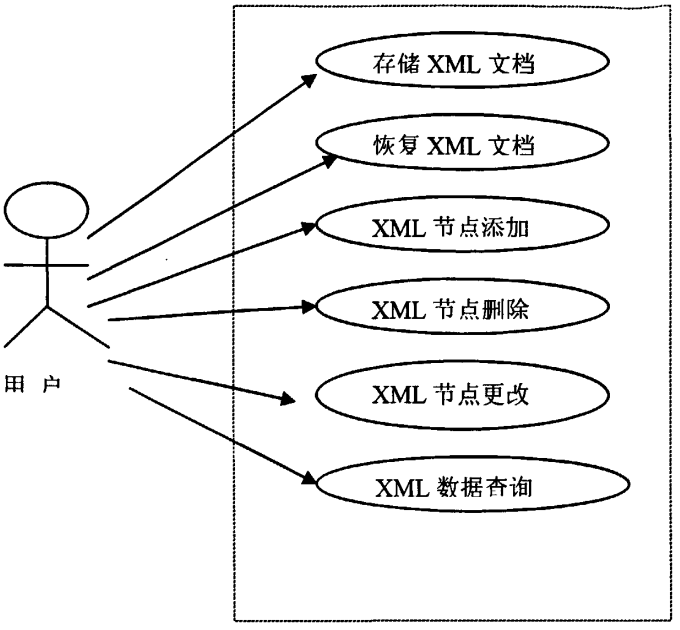



图 4.2.3.1 系统用例示图

从图 4.2.3.1 可以帮助分析系统功能。用户导入指定的外部 XML 文档到系统数据库中，也可以将系统数据库中的表结构恢复为 XML 文档。此外，用户还可对 XML 文档进行各种操作。

以上是对 XML 数据管理系统的具体分析，下面我们要详细描述这个系统的各个部分的实现过程。

## 4.3 系统的实现

### 4.3.1 XML 文档的关系存储

要想有效的管理 XML 数据，首先要考虑的是如何将它存储起来。一种可行的方法是把 XML 模式转换为关系模式，利用成熟的关系数据库技术存储 XML 数据。前面我们已经介绍过，基于关系数据库的 XML 数据管理技术主要有两种方法：结构映射方法和模型映射方法。这里我们主要来详细描述结构映射方法的实现过程。这种方法的核心就是要从 XML 数据对应的 DTD 信息中提取与 XML 数据相关的关系模式，将文档结构映射到关系数据库表中。

关系数据库由一系列表组成，每个表包含一系列元组，每个元组由一系列字段组成，而每个字段由字段/字段值对组成。由于XML是半结构化的数据，关系数据库以表为组织的，是结构化数据，为了能够在XML文档中和数据库中之间传送数据，有必要将XML文档的结构映射成数据库表的结构。在这里我们利用XML的DTD 模式映射成数据库表的

形式。

DTD是Document Type Definition(文档类型定义)。在DTD中，提供包含文档的元素、标记、属性和实体的清单以及其相互关系。我们在进行文档和数据库转换时，可以充分利用DTD文档，从而建立起较为符合原文档的数据库结构,并将文档信息保存在数据库中，且可将数据库信息导出成XML文档。XML的标准并不要求使用DTD，但绝大多数使用DTD，当XML文档含有DTD时，对XML文档的存储，查询和交换都有重要作用，符合DTD 的任何文档可以存储在关系模式中。通过DTD 生成关系数据库表模式，称为DTD 到关系模式的映射。

DTD 文档具体说明了与之相关的 XML 文档中元素和属性的名称，元素和属性的隶属关系，元素与子元素之间的嵌套关系，子元素的出现次数等信息。通过 DTD 文档再加之某种映射策略可以设计出存储 XML 文档数据需要的若干个关系模式。

本文采用的映射策略为基于 DTD 图的综合内联技术。具体算法如下：

①遍历 DTD 图,读取根节点及根节点所有的性质为叶节点的孩子节点和性质为正则运算符的孩子节点的叶子孩子节点,然后在不改变读取的所有节点父子关系的前提下,用这些节点构造出一棵被称为元素子图的树。

②对于其它的所有代表XML文档标记的非根节点处理方法同上。

③修改元素子图,无 ID 子节点的元素子图增加一个 ID 子节点。

④元素子图的根节点为关系名称,各个子节点为关系属性。

下面以图4.3.1.1所示的XML文档为例，详细描述这一过程。

```
<?xml version="1.0"?>
<!DOCTYPE book SYSTEM"book.dtd">
<books>
<book>
<ISBN >10439</ISBN >
<TITLE> XML数据库设计</TITLE>
<AUTHOR>尹志军</AUTHOR>
<PUBLISHER>机械工业出版社</PUBLISHER>
<PRICE>49</PRICE>
</book>
<book>
<ISBN>22056</ISBN>
<TITLE>Java XML应用程序设计</TITLE>
<AUTHOR>侯要红</AUTHOR>
<PUBLISHER>机械工业出版社</PUBLISHER >
<PRICE >38</PRICE>
</book>
</books>
```

图4.3.1.1 book2.xml文档示意图

这个XML文档可以映射为如表4.3.1.1所示的数据表的形式：

表4.3.1.1 转换后的数据表

| ISBN | TITLE | AUTHOR | PUBLISHER | PRICE |
|---|---|---|---|---|
| 10439 | XML数据库设计 | 尹志军 | 机械工业出版社 | 49 |
| 22056 | Java XML应用程序设计 | 侯要红 | 机械工业出版社 | 38 |

完成这一步之后的结果就是形成关系数据库表的形式，下一步就是映射成基于模型驱动的XML文档。这种映射关系分为基于表格的映射和基于对象的映射。许多转换XML到数据库的中间件都采用基于表格的映射，把XML看作一个(或一组) 表格，对于每个表格新建元素，对于表格中每个字段新建一个属性,或者只包含数据的子元素。对于每个表格字段中提供的主键的主键/外键都新建子元素。也就是说形成的XML文档是基于以下模型的:

```
<?xml version="1.0">
<?xml- stylesheet href="book.xsl" type="text/xsl"?>
<ExportFile>
<Schema>
<TableName> Book </TableName>
<Datatype name= 'ISBN' type='number' primary = 'yes'/>
<Datatype name= 'TITLE' type='char' maxLength='100'/>
<Datatype name='AUTHOR' type= 'char' maxLength= '20'/>
<Datatypename='PUBLISHER'type='char'maxLength= '200'/>
<Datatype name ='PRICE' type=' float'/>
</Schema>
<Data>
<Row>
<ISBN >10439</ISBN >
<TITLE> XML数据库设计</TITLE>
<AUTHOR>尹志军</AUTHOR>
<PUBLISHER>机械工业出版社</PUBLISHER>
<PRICE>49</PRICE>
</Row>
<Row>
<ISBN>22056</ISBN>
<TITLE>Java XML应用程序设计</TITLE>
<AUTHOR>侯要红</AUTHOR>
<PUBLISHER>机械工业出版社</PUBLISHER >
<PRICE >38</PRICE></Row>
</Data>
</ExportFile>
```

图4.3.1.2 book2.xml转换后的示意图

通过前面的部分已经实现了DTD到数据库表的映射，现在我们要做的就是通过编

程把XML数据写入到数据库表中。由于XML文档本身是一个文本文件，需要能够识别XML信息的解析器来解析并提取其中内容。由于我们只对部分XML文档感兴趣，无需形成整个XML文档的DOM树，因此我们可以选用SAX编程接口。根据需要我们只需重写以下几个事件，在文档开始和结束时，在一个元素开始和结束时，或者它在一个元素中找到字符时,以及其它若干点。文档开始时发送startDocument消息，结束时发送endDocument消息，当遇到元素的开始会发送startElement消息，同时把元素名以及长度，所包含的属性作为消息的参数传回来；遇到字符数据时发送character消息，并把内容作为参数送回；结束时发送endElement消息。

```
class saxbook extends DefaultHandler{
private String elem;
private StringBuffer buffer=new StringBuffer();
private boolean state;
public void parseURI(String uri)
{
try
{
SAXParserFactory spf= SAXParserFactory.newInstance();
SAXParser sp = spf.newSAXParser();
sp.parse(uri,this);
}catch (Exception e){e.printStackTrace();}
}
/**Start document*/
public void startDocument()
{buffer.append("Create Table");}
public void startElement( String namespaceURI,String localName String rawname,
Attributes attrs )
throws SAXException
{ elem=rawname;
if (elem.equals("Datatype")) {
if(! state) state=true;
else buffer.append(",");
    for (int i = 0; i < attrs.getLength(); i++) {
    String aname = attrs.getQName(i);
    String value = attrs.getValue(i);
```

```
    if (aname.equals("name"))        buffer.append(value+" ");
    if (aname.equals("type"))        buffer.append(value+" ");
    if(aname.equals("maxLength")) buffer.append("(" + value + ")");
    if (aname.equals("primary"))     buffer.append("" + "primary");
    }
    }
}
public void endElement (String namespaceURI,String localName,String rawname)
throws SAXException
{
if(rawname.equals("Schema")){
                              buffer.append(")\n");
                               }
}
public void characters(char[] cbuf,int start,int len)
{
if (elem.equals("TableName")) {String s = new String(cbuf, start, len);
                                 buffer.append(s);
                                }
}
}
}
```

这段源程序的执行结果是生成一个建表生成语句：

Create Table Book(ISBN number primary,TITLE char(100),AUTHOR char(20),PUBLISHER char(200),PRICE(float))

这就建立了与XML文档的DTD相关的数据表，从而实现了XML数据到数据表的转换，即完成了XML数据到数据库的存储。book2.xml所示的XML文档存储到数据库中后，形成了如图4.3.1.3所示的关系数据表。

| | ISBN | TITLE | AUTHOR | PUBLISHER | PRICE |
|---|---|---|---|---|---|
| | 10439 | XML数据库设计 | 尹志军 | 机械工业出版社 | 49 |
| | 22056 | Java XML应用程序 | 侯要红 | 机械工业出版社 | 38 |

图 4.3.1.3 数据库中的数据表

### 4.3.2 XML文档的恢复

前面已经实现了从XML文档到关系数据表的转换，有时我们需要将数据表恢复为XML文档，即实现前面的逆过程。这种过程主要是使用表名和字段名来驱动元素名(或属性名)，即给每一个表创建一个元素，为表中的每一个字段创建一个属性或者子元素；主键和外键信息可以用于创建文档的层次，即对每一个主键和外键关联，将主键所在的表创建的元素作为子元素插入外键所在的表所对应的元素中。先从选定的表开始并递归地使用主键和外键关系来找到所有的相关表，对于层次结构中的每一个表，将产生一个元素，它包含一系列只带有文本内容的元素(一个元素对应一个字段)和一系列带有子元素的元素(一个元素对应一个相关的表)。

把关系表格式化为XML文档的过程如下:

1）写出XML文档的声明。

2）写出根元素的开始标记。

3）反复检索数据表中的每条记录，以列名作为元素名，以数据表中的列值作为XML文档中元素的属性值。

4)写出根元素的结束标记。

主要的代码如下所示：

```
public void create(String root,String rootchild)
{
    DocumentBuilderFactory dbf=DocumentBuilderFactory.newInstance();
    Document doc=null;
    try
    {
        DocumentBuilder db = dbf.newDocumentBuilder();
        doc = db.newDocument();
    }catch (ParserConfigurationException e){e.printStackTrace();}
    Element rootelement=doc.createElement(root);
    doc.appendChild(rootelement);
    Iterator attri=attrilist.iterator();
    Iterator ele=elelist.iterator();
    try
    {
        while(rs.next())
        {
            Element rootchildelement=doc.createElement(rootchild);
            while(attri.hasNext()) /* 循环到属性集合中取元素名称，之后从查询结果集
```

```
中取得数据，创建元素属性 */
                {
                  SaveAttrName temp=(SaveAttrName)attri.next();
rootchildelement.setAttribute(temp.getAttributeName(),rs.getString(temp.getIndex()).trim());
                }
                rootelement.appendChild(rootchildelement);
                while(ele.hasNext())
                {
                   /* 循环到元素集合中取元素名称，之后从查询结果集中取得数据，创
建节点*/
                   SaveEleName temp=(SaveEleName)ele.next();
                   Element tempelement=doc.createElement(temp.getElementName());
                   Text temptextelement=
                   doc.createTextNode(rs.getString(temp.getIndex()).trim());
                   tempelement.appendChild(temptextelement);
                   rootchildelement.appendChild(tempelement);
                }
                attri=attrilist.iterator(); //重复循环到集合中取值
                ele=elelist.iterator();
            }
        } catch (Exception e){e.printStackTrace();}
        writeXml(doc);
    }
    private void writeXml(Document doc)
    {
     try
        {
            FileOutputStream outStream = new FileOutputStream(url);
            OutputStreamWriter outWriter = new OutputStreamWriter(outStream);
            ((XmlDocument)doc).write(outWriter, "GB2312");
            outWriter.close();
            outStream.close();
            System.out.print("\n文件完成!\n");
        } catch (Exception e){e.printStackTrace();}
```

```
    }
}
```

以上程序用ResultSet对象转换成XML文件，只需把查询结果集（ResultSet对象）和要生成的XML文件的路径传入，然后自己指派属性名称、元素名称并与对应的查询结果集中的字段相对应，最后调用designOver()函数，就可以生成所期望的XML文件了。

数据库中的数据表经过重构后，恢复成了如图4.3.2.1所示的XML文档，恢复后的XML文档与原来的XML文档数据基本保持一致。

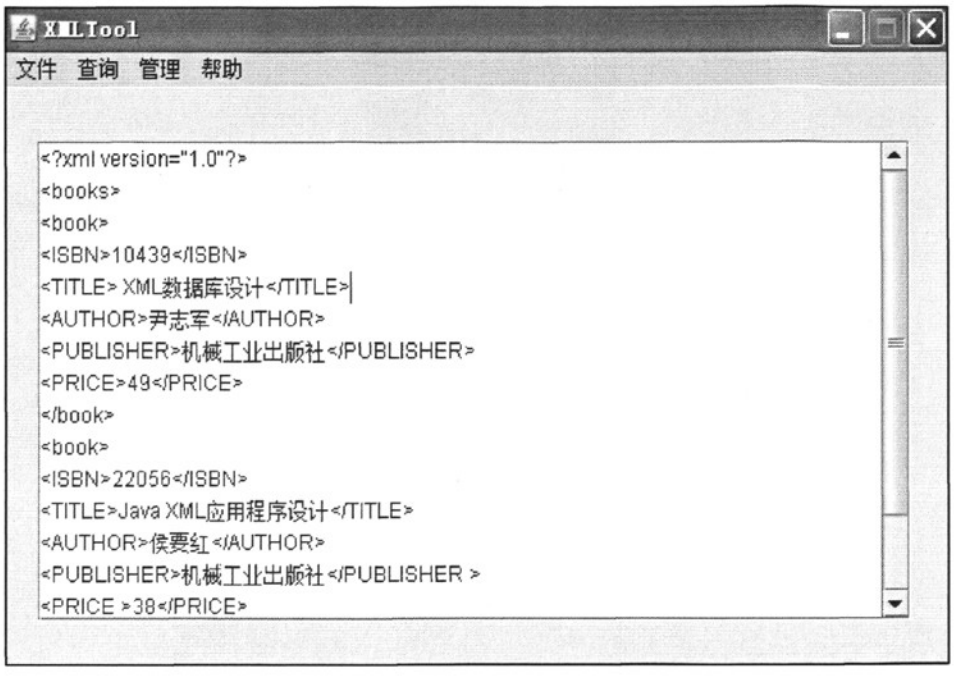



图4.3.2.1 恢复后的XML文档示意图

### 4.3.3 XML文档的更新

XML文档的更新主要用来实现文档中节点的添加、删除和修改，要实现这些操作，首先是要找到这些操作的位置，即找到这些操作的对象元素。我们可以使用路径查询法查找要找的元素。路径查询法的主要过程是：从路径的起始节点开始，检索起始结点的子元素，使其元素名与路径中的下一个元素名相同；通过检索可能得到多个子元素结点符合条件，对其中每个子元素，都要沿着路径的其余部分重复上面的步骤，直到路径结束；最后，返回满足条件的元素。路径查询法的流程如图4.3.3.1所示。XPATH就是路径查询法中的典型方法，它可以得到我们想要查找的元素以及它的子元素和属性等。我们

将在后面详细介绍XPATH的查询方法。

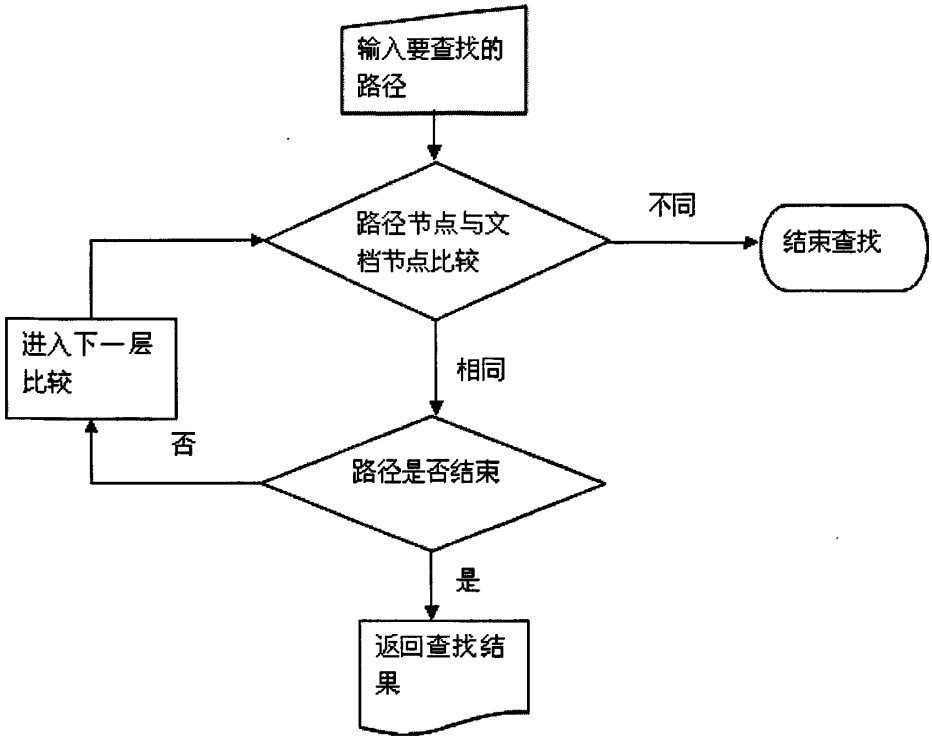



图4.3.3.1 路径查询法流程图

1) 节点的添加

XML文档中有着类型不同的节点，添加不同的节点要用不同的方法。

①标记型节点的添加

创建一个标记的过程可以分为两步：首先生成一个标记，然后再将该标记添加到某一个XML树的指定节点上。其语法为：

```
Element root = doc.createElement("root");
elem.appendChild(doc.createTextNode("my data"));
doc.appendChild(root);
```

②标记属性的添加

信息的存储可以是标记的形式，也可以是属性的形式。与生成标记的方式不同，这里不需要生成一个属性类型的节点，再将该节点添加到某一个标记当中；而是直接将属性添加到某个节点上。其语法是：

```
sex.setAttribute("stytle","red");
```

③注释型节点的添加

在XML数据类型中，注释是一种很重要的说明性信息，其对应的语法是：

```
Comment comment = doc.createComment("sex define");
root.appendChild(comment);
```

④CDATA类型节点的添加

CDATA代表XML文件中可以被解析的类型数据，这种类型的数据对应的是一种特殊的字符串类型。其中，可以包含一些XML文件的保留字，如标记等。其中的内容不会被当成XML程序处理，而只会被当成数据被读或写。其语法是：

```
CATASection cdata = doc.createCDATASection("<data>");
root.appendChild(cdata);
```

XML文件中的一个CDATA类型节点对应一个CDATASection类，第一句中createCDATASection的参数是具体的CDATA数据内容。第二句是将该CDATA类型的节点添加到根节点的最后。

⑤处理指令性节点的添加

XML文件中处理指令不用于表示特定数据，往往是对特定XML数据处理程序的提示性信息，提示XML处理程序应该执行的算法或执行的动作。处理指令的生成采用ProcessingInstruction类来表示。其语法是：

```
ProcessingInstruction proc =
doc.createProcessingInstruction("print","length=2");
root.appendChild(proc);
```

构造方法中第一个参数print，是处理指令的名字；第二个参数是处理指令中的一个具体属性。第二句将处理指令添加到根节点上。处理指令在XML文件中的位置是任意的，包括根标记的外面。

以下是一个添加节点操作的完整例子：

```
/*
* 此方法为得到Document对象实例
*/
public static Document getInstance(String xmlPath) {
Document doc = null;
try {
BufferedReader bf = new BufferedReader(new FileReader(xmlPath));
DOMParser ps = new DOMParser();// xml解析器
ps.parse(new InputSource(bf));// 解析xml
doc = ps.getDocument();// 获得Document对象
} catch (Exception e) {
```

```
e.printStackTrace();
}
return doc;
}
public static void addElement(Document dc) {
Element ebook = dc.createElement("book");
Element eisbn = dc.createElement("ISBN");
Element etitle = dc.createElement("TITLE");
Element eauthor = dc.createElement("AUTHOR");
Element epublisher = dc.createElement("PUBLISHER");
Element eprice = dc.createElement("PIRCE");
Text tisbn = dc.createTextNode("1234");
Text ttitle = dc.createTextNode("XML 编程实践");
Text tauthor = dc.createTextNode("邵敏");
Text tpublisher = dc.createTextNode("清华大学出版社");
Text tprice = dc.createTextNode("36");
estu.appendChild(eisbn).appendChild(tisbn);
estu.appendChild(etitle).appendChild(ttitel);
estu.appendChild(eauthor).appendChild(tauthor);
estu.appendChild(epublisher).appendChild(tpublisher);
dc.getDocumentElement().appendChild(ebook);
}
```

该程序在 book2.xml 文档中添加了一个新的元素节点：

```
<book>
<ISBN>1234</ISBN>
<TITLE>XML编程实践</TITLE>
<AUTHOR>邵敏</AUTHOR>
<PUBLISHER>清华大学出版社</PUBLISHER >
<PRICE >36</PRICE>
</book>
```

添加后的 XML 文档如图4.3.3.2所示，这个 XML 文档比原来的增加了一个新的 book 节点。

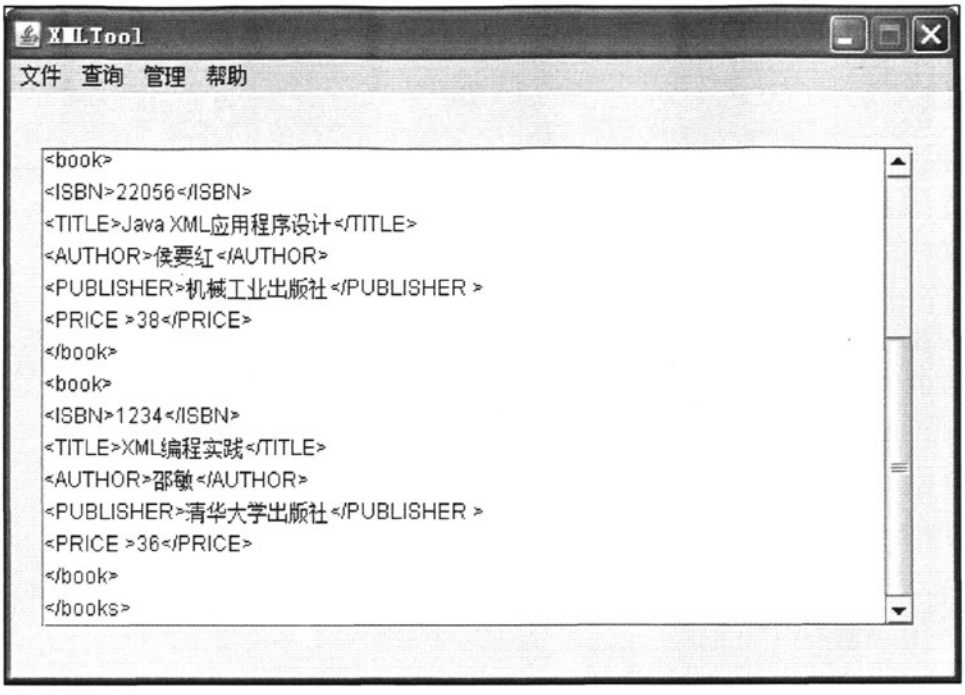



图4.3.3.2 添加节点后的 XML 文档

2) 节点的删除

XML文档中节点的删除比较简单，主要任务就是找到并确定我们需要删除的对象，不仅要元素名称一样，而且其属性值也必须一致。其主要的代码如下：

```
public static void delElement(Document dc, String idValue) {
NodeList nl=dc.getElementsByTagName("book");
Element e;
NamedNodeMap nnm;          //属性集合
Attr att;
for(int i=0;i<nl.getLength();i++){    //遍历节点，查找要删除的对象
e=(Element)nl.item(i);
nnm=e.getAttributes();
if(nnm!=null){
for(int j=0;j<nnm.getLength();j++){
att=(Attr)nnm.item(j);
if((att.getName().equals("id"))&&(att.getValue().equals(idValue))){
(e.getParentNode()).removeChild(e);    //删除该节点
}
```

```
}
}
}
}
```

以上代码中用到了 removeChild()的方法，在使用这个方法时需要注意亮点：（1）一个节点只能被它的父节点移除；（2）方法中的参数是要移除的节点对象。

3) 节点的修改

XML节点的修改可以归纳为添加和删除的结合。删除掉不需要的信息，再添加我们需要的信息，这就打到了修改的目的。其主要的代码如下所示：

```
public static void updateElement(Document dc, String sttitle, String stauthor) {
NodeList nl = dc.getElementsByTagName("TITLE");
Element e;
for (int i = 0; i < nl.getLength(); i++) {      //遍历文档中的节点
e = (Element) nl.item(i);
if (e.getFirstChild().getNodeValue().equals(sttitle)) {
Node n = e.getParentNode();
NodeList nd = n.getChildNodes();
for (int j = 0; j < nd.getLength(); j++) {
Node node = nd.item(j);
if (node.getNodeName().equals("author")) {
node.getFirstChild().setNodeValue(stauthor);    //修改新的值
}
}
}
}
}
```

### 4.3.4 XML 文档的查询处理

#### 4.3.4.1 XML 查询分类

现在互联网上已经存在大量的 XML 数据，这些数据基本上仍是以文件的形式存放。这些 XML 文档一般都不大，但包含了丰富的数据，因此需要一个可以从文档中抽取信息的 XML 文档查询工具。当前，对 XML 查询的研究主要有两类：（1）基于内存的 XML 文档查询处理及优化，其关键工作包括索引设计、查询处理的逻辑及物理优化等；（2）基于关系或面向对象数据库来研究 XML 文档的查询，目前主要研究基于关系数据库的

查询问题[1]。

1）基于内存的XML文档的查询处理

XML数据通常以文档的形式存在，因此，有必要从文档的角度来研究XML查询和优化。与基于关系数据库的XML查询处理不同的是，XML文档的查询处理必须实现逻辑优化、物理优化、索引设计和查询执行的各个环节。在以上几个环节中，最难的还是XML文档的索引设计。因为XML的数据模型比较复杂，采用任何一种单一的索引结构都不能很好地解决问题，需要多种索引机制的联合使用才可以收到良好的效果。

Lore系统的研究人员提出了四种索引：值索引、父索引、父子索引和路径索引。路径索引是最常用也是最复杂的索引，利用这种索引可以直接获得一个路径表达式所能达到的所有元素的集合，而不用对文档进行扫描。路径索引的设计中比较成熟的是Lore系统的DataGuide，它是源文档结构的一种概括，本身也是一个文档，可看成一棵树，其中包含了源文档中所有可能存在的路径，并且源文档中相同的路径在DataGuide中只出现一次，它在处理常规路径查询时可以收到良好的效果。但是它不支持XQuery中经常出现的“//”操作，在搜索该类路径时就必须遍历整个DataGuide树。另外，Data Guide只记录从文档根节点出发的路径，从而丧失了许多优化的可能性。基于内存的查询处理和优化在物理实现上必须考虑数据的物理组织方式。因为CPU速度和内存访问速度之间的差距越来越大，现代计算机通常都提供了一级或二级的高级缓存，以缓冲CPU和主存之间的冲突。但由于缓存通常比较小，因此必须采用各种方法以降低运行时的缓存失配率。

2）基于关系数据库的XML查询处理

XML查询处理建立在关系数据库系统之上的优点是可以直接利用关系数据库成熟的查询优化和索引技术，也不用为并发控制、安全性等问题做额外的工作。但是，基于关系数据库的XML查询结果需要以XML的格式对外发布。目前采用的方法有：

① 使用视图的方法：首先在关系数据库上建立XML视图（视图是虚拟的），然后基于视图使用XML查询语言定制XML数据。

② 不使用视图的方法：不使用视图这一中间环节，使用专门的查询语言（如扩展SQL语言或者自定义的语言）直接在关系数据库上构造XML文档。

4.3.4.2 基于XPath的XML查询

1） XPath查询原理

XPath是一种对XML文档的内容进行定位和检索的语言，是后续更强大的数据检索语言如XQuery的基础。XPath因使用路径标记在XML文档的层次结构中进行导航而的名，其工作方式与语法有些类似操作系统中用于文件定位的路径以及互联网中用于资源定位的URL。当然，XPath比后二者复杂的多。XPath不独立使用，主要嵌入在XSLT

、XPointer、DOM 等宿主语言中应用。比如，在 XSLT 的应用中，XPath 用在模板中来检索数据以及定义匹配模式。

数据模型即 XPath 看待它操作的 XML 文档方式，了解数据模型有助于更好地了解 XPath 的语法以及工作方式。XPath 将 XML 文档视为一个节点树，如图 4.3.4.2.1 所示，节点有七类，包括根节点、元素节点、属性节点、文本节点、命名空间节点、处理指令节点和注释节点。XPath 对每种节点都定义了计算其字符值的方法，不同的节点计算方法可能不同。

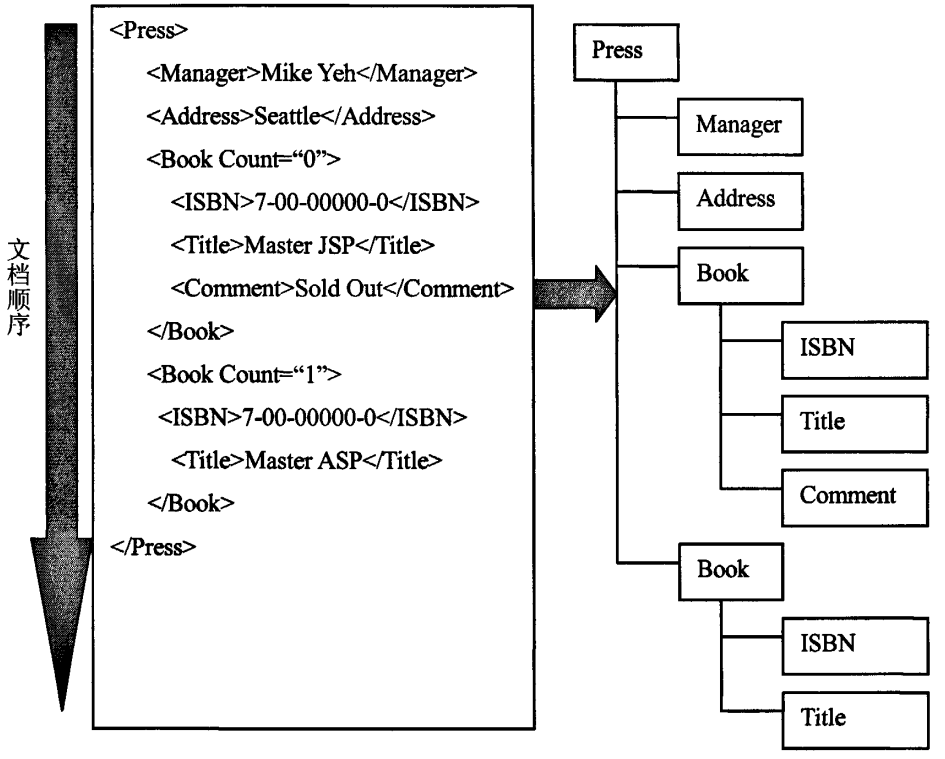



图 4.3.4.2.1 XPath 查询示意图

文档顺序定义了每个节点在 XML 文档中出现的顺序。此顺序与 XML 文档中节点的物理位置一致。如图 4.3.4.2.1，根节点为文档的第一个节点，每个元素节点出现在其子节点之前。根节点和元素节点可以有一个有序列表的子节点（即子节点之间有一定顺序关系，不能任意排列）。节点不共享子节点，即除了根节点，每个节点都有唯一的，明确的父节点。父节点可能是根节点或元素节点。一个节点的后代包括它的子节点以及子节点的后代。值得注意的是，属性节点肯定有其父节点，但属性节点不是其父节点的子节点，即属性节点不被视为子节点。

定位路径是 XPath 最重要的一种表达式，它选择与上下文节点相关的一个节点集作

为计算结果。XPath 有两种定位路径，相对路径和绝对路径。在相对路径中，每一个定位步骤都选择与上下文节点相关的一套节点，然后再这一套节点中的每一个节点都作为后续定位步骤的上下文节点进行选择。如此从左到右进行选择，最后选择的所有节点组合在一个，得到定位路径结果。

XPath 定位步骤有三个部分：①一个轴，指明定位步骤选择的节点与上下文节点的节点树上的关系；②一个节点测试，指明定位步骤选择的节点的类型和扩展名称；③零个或多个断言，使用任意的表达式进一步精简选择的节点集。一个定位步骤选择节点的过程为：初始的节点集由那些与上下文节点有相应关系的节点构成，其中关系由轴指出。同时，这些节点还必须有着由节点测试部分指定的节点类型和扩张名称。最后，初始的节点集将由断言逐个过滤，以得到此定位步骤选择的最终结果。

2） XPath 查询运用

XML 文档实例如图 4.3.4.2.2 所示：

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<bookstore>
<book category="COOKING">
  <title lang="en">Everyday Italian</title>
  <author>Giada De Laurentiis</author>
  <year>2005</year>
  <price>30.00</price>
</book>
<book category="CHILDREN">
  <title lang="en">Harry Potter</title>
  <author>J K. Rowling</author>
  <year>2005</year>
  <price>29.99</price>
</book>
<book category="WEB">
  <title lang="en">XQuery Kick Start</title>
  <author>James McGovern</author>
  <author>Per Bothner</author>
  <author>Kurt Cagle</author>
  <author>James Linn</author>
  <author>Vaidyanathan Nagarajan</author>
  <year>2003</year>
  <price>49.99</price>
</book>
<book category="WEB">
  <title lang="en">Learning XML</title>
  <author>Erik T. Ray</author>
  <year>2003</year>
  <price>39.95</price>
</book>
</bookstore>
```

图 4.3.4.2.2 bookstore.xml 文档

这里主要介绍一下如何使用 XPath 来查询 XML 文档中的节点，通过使用 selectNodes()函数从 XML 文档选取节点，来了解 XPath 的基本查询过程：

① 选取所有的 book 节点：xmlDoc.selectNodes(“/bookstore/book”)

查询结果如图 4.3.4.2.3 所示：

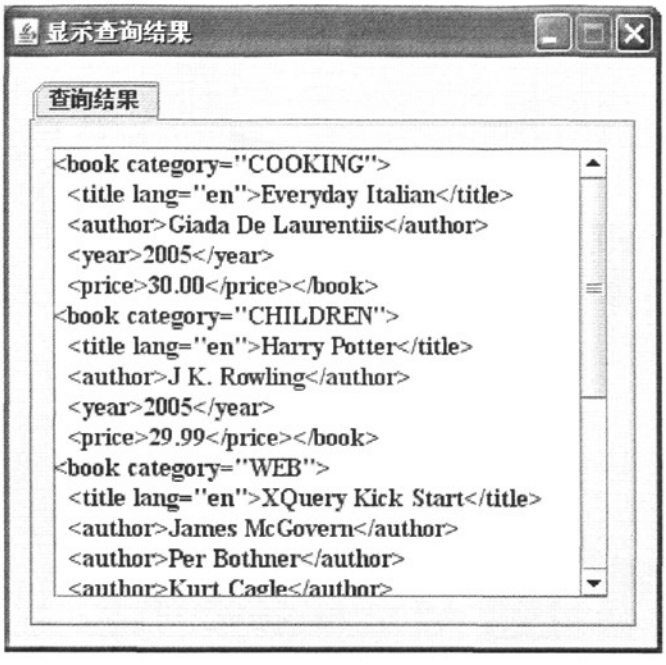



图 4.3.4.2.3 查询结果 1

② 选取第一个 book 节点：xmlDoc.selectNodes(“/bookstore/book[0]”)

查询结果如图 4.3.4.2.4 所示：

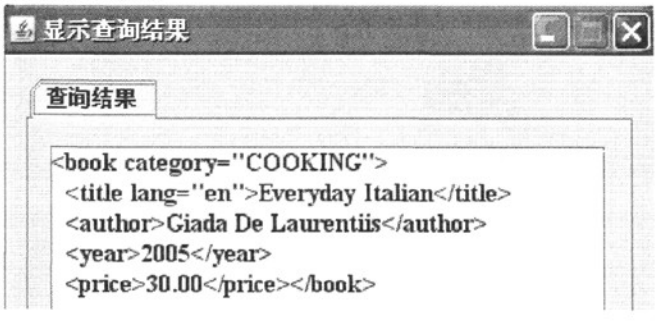



图 4.3.4.2.4 查询结果 2

③ 选取 price 节点：xmlDoc.selectNodes(″/bookstore/book/price/text()″)

查询结果如图 4.3.4.2.5 所示：

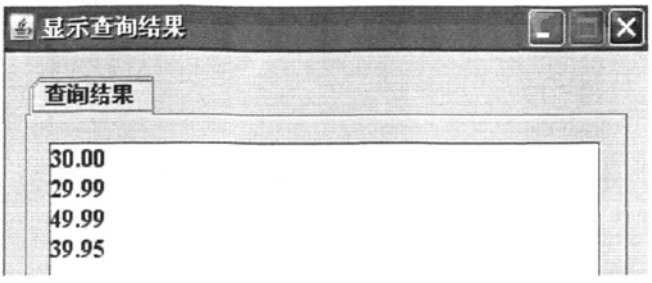



图 4.3.4.2.5 查询结果 3

④ 选取价格高于 35 的 price 价格：

xmlDoc. selectNodes(“/bookstore/book[price>35]/price”)

查询结果如图 4.3.4.2.6 所示

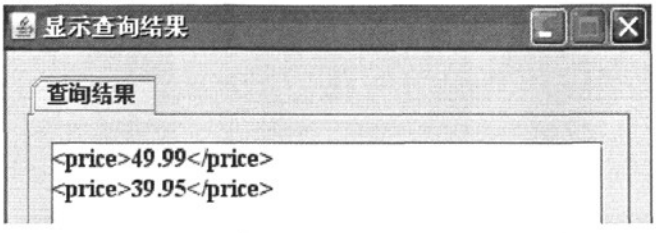



图 4.3.4.2.6 查询结果 4

⑤ 选取价格高于 35 的 title 节点：

xmlDoc. selectNodes(“/bookstore/book[price>35]/title”)

查询结果如图 4.3.4.2.7 所示：

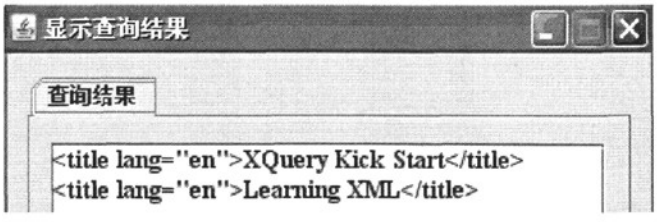



图 4.3.4.2.7 查询结果 5

以上是使用 XPath 对 XML 文档进行查询的基本操作，通过这些操作我们了解了 XPath 的基本查询原理：XPath 把整个 XML 文档看成一棵节点树，它通过特定的路径规则来查找需要的节点。XPath 是其他 XML 查询语言的基础，了解了 XPath 的查询原理和过程，能够更好的掌握和使用其他 XML 查询语言。

### 4.3.4.3 基于 XQuery 的 XML 查询

XQuery 由 Quilt 所衍生而来，同时又从 XPath 和 XQL 中吸收了路径表示语法以适应层次结构文档的需要，融入了 SQL 中基于关键字系列子句的思想，为数据重建提供了类似 SQL 的模式，并吸取了 OQL 中由几种不同表达式全嵌套组成的功能语言概念.

XQuery 作为一种将查询表示成表达式的功能语言，可以完全嵌套，故而沿用了子查询的功能与用法。

1） XQuery 到 SQL 的转换

① 转换实现的要求

XML 文档在关系数据库中进行查询处理时，查询处理器将 XQuery 转换成 SQL 需要满足一定的要求.

a 查询转换的正确性

对于一个 XQuery 查询和它转换后生成的 SQL 查询，应该实现相同的功能，SQL 所得到的查询结果正是 XQuery 查询所期望的。

b 查询转换完全性

要求所有的 XQuery 功能都有对应的 SQL 语句来实现。

c 查询效率

有两方面因素要考虑：一是 XQuery 到 SQL 的转换效率，与转换方法本身的复杂度直接相关；二是转换生成的 SQL 语句的查询执行效率，与转换方法和映射方法都相关，这是因为关系表的物理存储结构直接影响到 SQL 查询操作的执行过程，其中所涉及的 SQL 连接操作数量是一个尤其重要的因素。

d 对关系查询引擎的利用程度

XQuery 转换为 SQL 的目的之一就是要将 XQuery 查询处理的功能分解，充分利用关系数据库成熟的查询引擎功能，达到降低问题难度及提高处理效率的效果。

② 转换的主要过程

XQuery 语言转换成 SQL 语言有多种方法，但各种方法大都包括主要的三个部分：XQuery 语言的解析与内部查询代数形式表达；进一步生成 SQL 语句的处理过程；按 XML 格式输出结果。一般的，各种转换方法所使用的内部查询代数以及后继处理过程差别较大。主要处理步骤如图 4.3.4.3.1 所示：

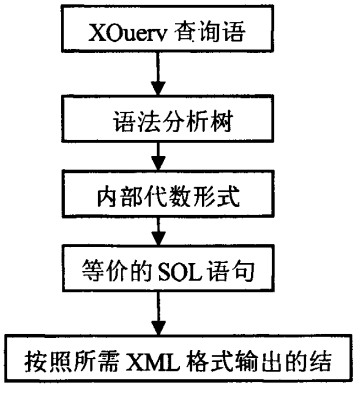



图4.3.4.3.1 XQuery查询的主要步骤

a 解析

接收 XQuery 语言编写的语句文本并将之转换生成语法分析树。XQuery 语句经过词法分析器 Lexer，得到单词序列作为词法分析的输出。对 XQuery 查询语句按照其语法定义，将各个语法部分分解开来并重新组织建立它们之间的语法联系。这与 SQL 查询的语法分析相似，形成一棵语法分析树。

b 内部代数表达

按照第一步的语法分析树生成内部代数表达形式。输入是表层语法树，输出是符合各转换方法内部代数系统的表达形式，一般是树状形式，类似于 SQL 查询的逻辑表达式树形式。不同的内部代数系统往往差异很大，因此在不同的转换方法中实现该步骤时有其具体的处理过程和原则。

c 对 SQL 的转换处理

有了内部代数表达形式就便于向 SQL 查询方向转换。一般来说每种 XQuery 查询的转换方法使用自己的一套代数系统来体现 XQuery 的查询功能，虽然各种代数系统存在差异，但其目的都是为了有利于与 SQL 查询操作的处理相对接。内部代数系统一般都包括丰富的操作符，大都与 SQL 的同类操作符作用相近，含义相似，另外内部代数系统对查询的表达形式也大都采用树形结构，与 SQL 查询代数表达式树相近，这都便于转换过程的顺利进行。每种转换方法在该步骤可能有着不同的处理过程，但为了更完全的转换到 SQL 查询，都要对内部代数的表达形式作变换，将能够由关系操作或一系列关系操作代替的内部代数操作都尽可能的提取出来转变为 SQL 语句。

d 查询结果输出

当 XQuery 查询语句经过处理得到其目标数据之后，下一步就要对外输出，这时主要的任务是对数据组织格式的安排。输出时所要使用的格式是由用户书写的 XQuery 语句具体指定的，XQuery 语言也提供了这样的功能。因此，有关结果输出格式的信息处理在整个查询处理过程的前面阶段就已经开始了，在查询获取目标数据之后就仅剩下结果处理的问题了。在查询处理的解析阶段，除了词法和语法分析外，在分析过程中要把输出格式单独列出，并且将格式中的待求值部分与解析后的各要素对应起来，在获得查询结果后，依据上面的对应关系使得正确的结果能够被放置在正确的位置上。

2） XQuery 查询运用

有一 XML 文档片段如图 4.3.4.3.2 所示，我们可以用编程的方法来查询图该文档，验证下 XQuery 的查询结果。

```
<order Id="10">
<custname>John Gates</custname>
<items> <item desc="generator"><cost>8000</cost></item>
        <item desc="derailleur"><cost>24000</cost></item>
</items>
<payments>
      <payment due="1/10/08"><amount>20000</amount></payment>
      <payment due="1/5/08"><amount>12000</amount></payment>
</payments>
</order>
<order Id="9">
……
</order>
```

图4.3.4.3.2 order.xml文档片段

主要代码如下：

```
private static String XMLFile = "order.xml";
XQEngine m_engine = new XQEngine();//设置 XQEngine 相关属性
String query = "/order/custname";  //查询语句,查询 order 下的 custname 元素
try {
    SAXParser parser = spf.newSAXParser();
    XMLReader reader = parser.getXMLReader();
     m_engine.setXMLReader( reader );
    }
    catch( Exception e )
    {
      throw e;
    }
 m_engine.setDocument(XMLFile);  //加载 xml 文档
 ResultList results = m_engine.setQuery( query );  //执行查询
 results.getAST( ).dump("    " );
 System.out.println( results );
```

程序的执行结果如图 4.3.4.3.3 所示：

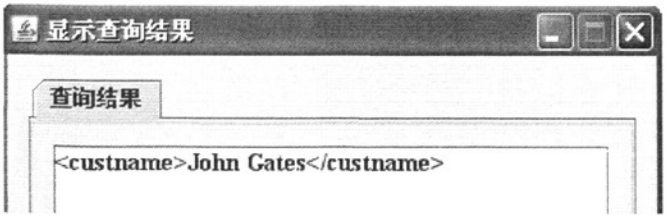



图 4.3.4.3.3 XQuery 查询结果

## 4.4 本章小结

本章详细描述了一个基于关系数据库的 XML 数据管理系统的实现过程，主要实现了存储、删除、恢复、更新和查询这几个功能模块。其中查询模块分为 XPath 查询和 XQuery 查询两个部分。XPath 是针对路径的查询语言，它能够定位文档中的所有节点，是其他 XML 查询语言的基础。XQuery 是在 XPath 基础上创建的，它的功能更为强大，它能够从 XML 文档中选择并抽取出复杂的模式，进而把查询结果重构成用户需要的新的 XML 文件结构。两者结合，能够很好地实现了查询功能。

# 5 XML 的索引查询技术

## 5.1 XML 索引技术介绍

随着互联网上 XML 文档的不断增多，对 XML 数据的使用越来越依赖于互联网搜索引擎强大的检索能力. 虽然传统的信息检索技术也可以应用到 XML 文档上，但是由于没有考虑 XML 文档中的标签、属性和结构等信息，因而就不能完全发挥 XML 所带来的好处。XML 语言提出以后，计算机界均认为它可以比 HTML 更好地解决搜索问题. 。这是因为基于 XML 的搜索引擎可以利用 XML 文档中的标记，可以确定在文档中的哪一部分查找，而不是像在 HTML 文档中那样，是在整个文档中查找；另外，它返回的结果只是文档的相关部分，这样不但可以提高搜索的精确度，而且还可以减少网络上的流量。

传统的信息检索技术只进行内容检索，信息查询式的提交是以提交关键词为主；XML 文档作为一种结构化文档，基于它的查询主要包括两个方面：基于关键字匹配的内容查询和对文档嵌套结构的查询. 对 XML 文档来说，文档内容是指元素的文本内容和属性值，文档嵌套结构是指元素之间的父子关系嵌套。文档查询侧重于内容查询，所以查询的执行一般以关键字作为入口，得到匹配该关键字的底层数据节点，再利用节点编码或其他方式，计算文档的嵌套结构。在这样的查询执行框架中，索引主要在两个方面发挥作用：关键字匹配和快速得到文档结构元素信息. 关键字索引的具体形式类似于传统的倒排文件等，但索引的粒度变小了，从文档级细化到了文档树中的节点级。对 XML 索引技术的研究，有利于提高 XML 数据的查询效率。XML 索引技术是当今 XML 数据管理技术中的热点。

## 5.2 XML 索引技术分类

XML 数据查询主要有以下两种[37](1) 值查询:通过限定在元素内容或属性上的取值而进行的选择查询。(2) 结构查询:通过路径表达式,对文件中标记的元素之间的结构关系进行查询。元素之间的结构关系包括:祖先/ 后裔(ancestor/ descendant)关系、双亲/孩子(parent/child) 关系、之前/之后(preceding/following) 关系、左兄弟/右兄弟(preceding-sibling/following-sibling) 关系等。根据上述两种查询方式,产生了两种不同的 XML 索引方法：基于路径的 XML 索引和基于编码的 XML 索引。第一种方法是建立 XML 文档树的路径索引,并通过路径索引加速 XML 查询的计算；第二种方法是对 XML 文档树中的节点(或边)进行编码,通过编码直接判断节点之间的结构关系或元素内容。

## 5.3 几种主要的 XML 索引

1） 基于路径索引的 XML 查询技术

路径索引的基本思路是:将 XML 文档转换成 XML 数据图,通过扫描 XML 数据图得到路径索引图,其中索引图是用较少的边来存储 XML 数据图中的边。目前主要的 XML 路径索引有: DataGuide、1-index、A(k)、D(k)、M(k)、APEX 和 Fabric 等。

① DataGuide 索引[37]

DataGuide 是斯坦福大学提出的 Lore 系统中所用的一种索引结构,它是对半结构化数据的一个简洁而精确的概括。它将 XML 数据图中由根节点开始经过相同标签值的所有路径存储在 DataGuide 中,在 DataGuide 中以一个节点集表示 XML 数据图中的多条边。一个 XML 数据图可能对应多个 DataGuide 索引图,为此 Lore 系统定义了一个“Strong Data Guide”模型,Strong Data Guide 的优点是如果在 XML 数据图中的简单路径到达的节点相同,那么这些简单路径存储在“Strong Data Guide”中的相应的节点集中。例如,图 5.3.1 中 XML 数据图它所对应的“Strong Data Guide ”索引如图 5.3.2 所示。Lore 系统支持文档的动态更新, 当文档更新时,“Strong Data Guide”也被更新。

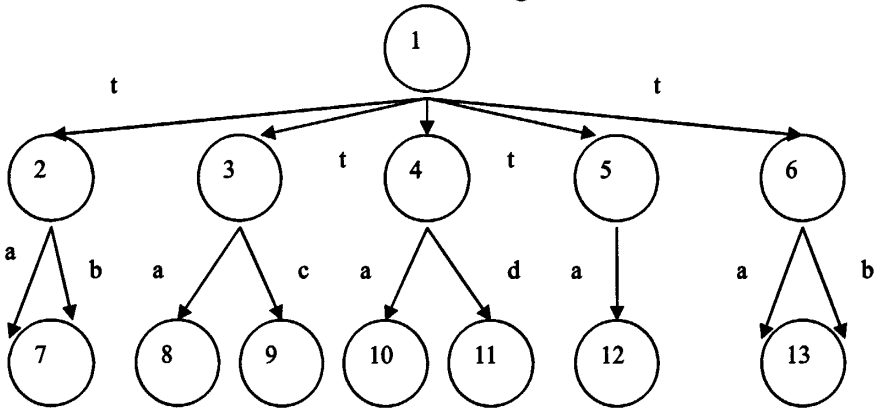



图5.3.1 XML数据图

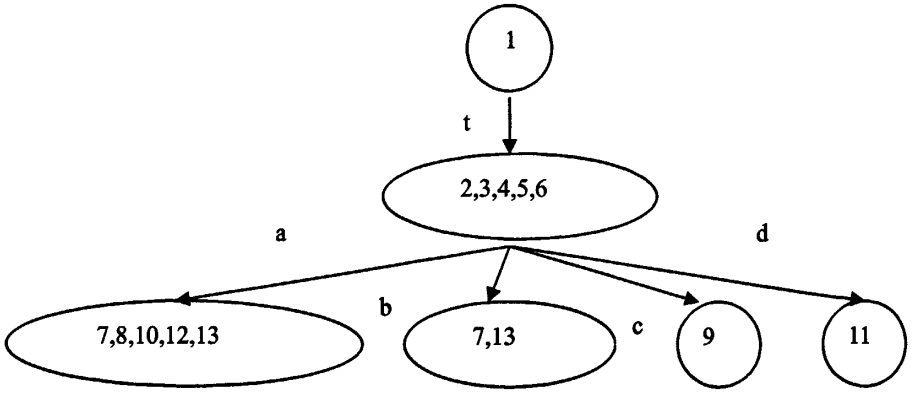



图5.3.2 Strong Data Guide索引图

② 1-index 索引[37]

DataGuide 存在下列不足：(1) Data Guide 是对 XML 数据图精确的概括，若 XML 数据图是图结构，那么建立 DataGuide 的时间和所需的空间可能是 XML 数据图大小的指数倍。(2) Data Guide 中各个节点的扩展集可能相交。为了解决 Data Guide 的上述两个问题，1-index 提出两节点“相似”概念，其含义是：若节点 u，v 具有相同的标签且节点 u′ 是 u 的父节点，节点 v′ 是 v 的父节点，则节点 u′ 与 v′ 相似。

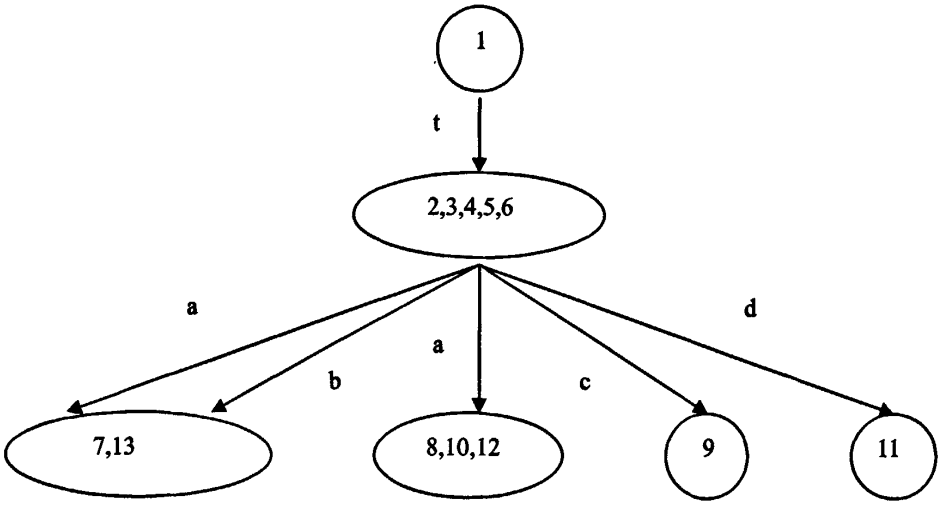



图 5.3.3 1-index 索引图

1-index 索引中将“相似”的节点存放在一个扩展集中，例如，图 5.3.1 中表示的是 XML 数据图，图 5.3.3 是 XML 数据图的 1-index 索引。图 5.3.3 所表示的 1-index 索引的扩展集为{7, 13}, {8, 10, 12}, {9}, {11}，各个扩展集之间并没有相交；图 5. 3. 2Data Guide 所表示的索引的扩展集为{7, 8, 10, 12, 13}, {7, 13}, {9}, {11}，很明显扩展集之间相交。这可能造成 Data Guide 中所有扩展集的节点总数是 XML 数据图中节点总数的指数倍。利用两节点“相似”概念使得 1-index 具有如下两个优点：(1) 索引大小和 XML 数据图大小成线性关系。(2) 索引的扩展集之间不相交，所有扩展集的节点总数和 XML 数据图中节点总数相等。

2) 基于编码的 XML 索引查询技术

基于路径索引的 XML 查询技术能够有效地解决单路径查询问题，但是不能很好地解决分支路径查询；编码索引查询技术不仅可以有效地查询单路径，也解决了分支路径查询问题。基于编码的 XML 索引查询技术主要有 XR+XRstack 算法，、Anc-Desc-B+算法等[16]。

① XML 数据的编码方案

目前常用的 XML 数据的编码方案主要有：位向量编码、区间编码等。

位向量编码：树 T 中的每个节点被编码为一个 n 位向量，n 是树 T 中的节点数量，在

某个位置 i 上的一个“1”惟一标识第 i 个节点；并且在一个自顶向下(或自底向上) 的编码方案中，每一个节点继承它祖先(或后裔) 节点的所有位上的“1”，如图 5.3.4 所示。

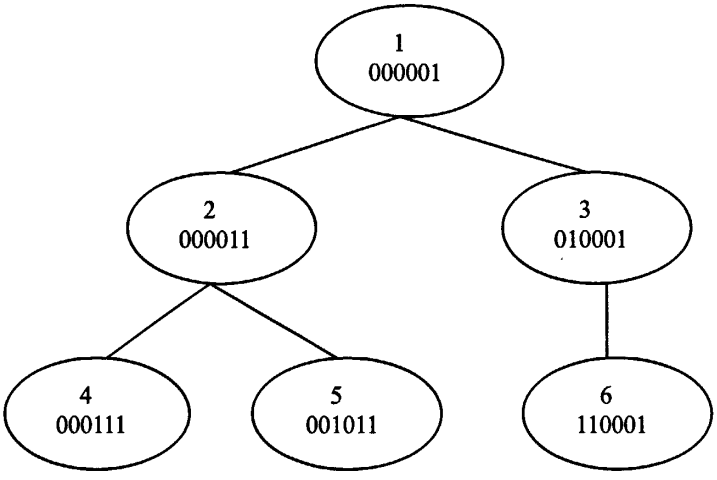



图5.3.4 位向量编码

区间编码：树 T 中的每一个节点被赋予一个区间编码[begin,end] ,并且满足：一个节点的区间编码包含它的后裔节点的区间编码,即树 T 中的节点 u 是节点 v 的祖先，当且仅当 begin(u)<begin(v)∧end(v)<end(u) 。基于区间编码的方案目前有三种: (1) Dietz 编码：树 T 中每个节点被赋予一个先序遍历序号和后序遍历序号的二元组〈pre,post〉。(2)Li-Moon 编码：树 T 中的每一个节点被赋予一个二元组〈order, size〉,order 位节点的扩展先序遍历序号,它的取值是非连续的,为节点的插入预留序号空间, size 为节点的后裔范围。(3)Zhang 编码：XML 文档树中的每一个节点被赋予一个二元组〈begin,end〉。对树 T 的所有节点进行先序遍历,每一个节点在遍历树时分别被访问两次并产生两个序号。一次是在遍历该节点的所有后裔节点之前访问该节点,并产生该节点的序号 begin；另一次是在遍历完该节点的所有后裔节点后再一次访问该节点,并产生该节点的另一个序号 end。

本节主要研究了 XML 查询技术中的两种主要方法，其中基于路径索引的 XML 查询方法只能够解决单路径查询，但是路径索引的创建不受 XML 文档结构的约束，即 XML 文档可以是树结构，也可以是图结构。基于编码方式下的 XML 索引查询方法能够有效地解决分支路径查询，但是这种方法都对相应的 XML 文档有要求，其所要查询的 XML 文档为树形结构。如果能很好地将上述两种 XML 查询方法中的优点结合起来，将会大大提高 XML 的查询效率。

## 5.4 前序-后序节点标号法的运用

前面简单介绍了基于路径的 XML 索引和基于编码的 XML 索引，接下来我们将详细研究一种基于编码的 XML 索引方法：前序-后序节点标号法[28]。

1） 前序-后序节点标号法

一般地，XML 文档分为两大类：数据型文档和叙述型文档。数据型文档的特点是数据只出现在物理树的叶节点上，是一个结构化的数据集，这很容易将其与一个支持子字段的关系型数据库相关联；叙述型文档，或称混合型文档，其特点是数据既可以出现在物理树的叶节点上，也可以出现在分支节点上，是一个结构化与非结构化相混合的数据集，这种情况就很难用一个关系型数据库结构来描述。对于大多数全文数据库而言，虽然都支持对结构化和非结构化数据的存储、管理和检索，但一般不支持子字段，也不像关系型数据库那样具有功能强大复杂的将多表进行连接操作的能力。结构化文本是指和表达的思想内容相对应，在物理形式上有明显的组织结构和层次关系的文本，比如书刊就是一种结构化文本。由于 XML 是一种半结构化的数据，因此可以采用前序-后序节点标号的方法来优化 XML 的查询速度。

采用一种特有的对节点进行标号的方法，就能记住 XML 文档中的祖先—后代的关系，这种方法在 1982 年由德国 Dietz 教授首次提出，并称之为前序-后序节点标号法，也被称为 Dietz 编码法。这种方法的主要思想是对一棵树进行前序遍历，每个节点获得一个前序遍历序号；再进行后序遍历，每个节点获得一个后序遍历序号；对每个节点采用其前序遍历编号和后序遍历编号来组成节点标号。在这种标号方法下，如果节点 X 是节点 Y 的祖先，当且仅当 X 的前序遍历序号小于 Y 的前序遍历序号，且 X 的后序遍历序号大于 Y 的后序遍历序号。可以用图 5.4.1 来说明前序—后序节点标号法。

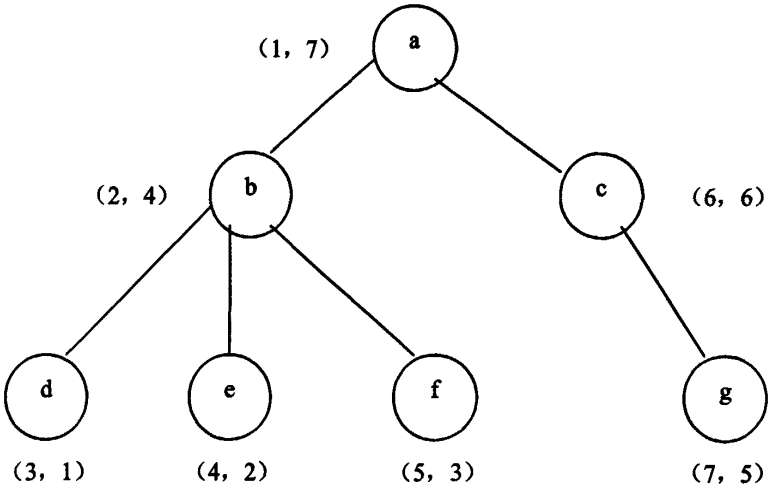



图5.4.1 前序-后序标号法示例

图中节点 a(2, 4)是节点 e(4, 2)的祖先，是因为前序序号 2<4，且后序序号 4>2；节点 b(2, 4)不是节点 g(7, 5)的祖先，是因为虽然前序序号 2<7，但后序序号 4<5。这种前序—后序编号法有一个不足是不能判定父—子关系，比如 a(1, 7)，b(2, 4) 节点是

e(4,2) 节点的祖先，根据规则，我们并不能判定 b(2,4) 节点是 e(4,2)节点的父亲。为此我们可以设计一个节点深度参数，节点所在的层次叫节点的深度(根节点的深度为0)。比如 b(2,4) 节点的深度为 1，e(4,2)节点的深度为 2,a(1,7)节点的深度为 0。在 Dietz 教授的规则上，再辅以节点深度参数，就能判断父—子关系，即在节点 x 已经是节点 Y 的祖先的基础上，如果节点 x 的深度比节点 Y 的深度小 1，则节点 x 是节点 Y 的父亲。

通过判断节点相关性,可以跳过很多与检索要求不相关的节点,因此这个算法将会提高检索的速度。在判断两个节点相关性的基础上,得到判断多个节点相关性的算法

前序—后序标号法不仅可以用来对 XML 文档的序进行描述，而且可以对其它树形、含有等级层次关系的文档进行描述，比如分类表、主题词表等。然而目前很少见到过用前序一后序标号法对诸如分类表、主题词表的描述，可能的原因是前序—后序标号法对文档更新比较敏感。当文档的层次结构需要更新时,会导致全部节点的标号都需要重新计算。因此，前序—后序标号法适合处理那些内容相对稳定的文档。

2） 前序-后序节点标号法的应用

一个 XML 文档树的先序遍历等价于它的文档顺序，即如果对文本形式的 XML 文档进行顺序存取，则一个元素被访问的顺序就是它们的先序遍历的序号；反之，XML 文档的文本能够以先序遍历它的文档树的形式进行重构。

前序-后序节点标号法可以产生如下四种关系模式：

Document（DocumentID，DocumentName）

Element（DocumentID，ElementPre，ElementPost，TagName，ParentPre，ParentPost）

Attribute（DocumentID，AttributeID，ElementPre，ElementPost，AttributeName，AttributeValue）

Text（DocumentID，TextID，ElementPre，ElementPost，TextContent）

表 Document 存储的是数据库中保存的 XML 文档的文件名，该表的主键为文档标号（DocumentID）。表 Element 存储的是 XML 文档中的元素，属性 ElementPre 和 ElementPost 分别表示元素的先序和后序编码,由于 XML 文档中元素最多有一个父元素，因此属性 ParentPre 和 ParentPost 分别表示父元素的先序和后序编码，属性 TagName 代表元素的标记名称，该表的主键为 DocumentID，ElementPre，ElementPost。表 Attribute 存储的是 XML 文档中属性节点，属性 AttributeID 表示每个 XML 文档中属性编号，根据 XPath 数据模型中对于属性节点的描述，表中还要保存该属性所从属的元素信息即需要保存元素的先序和后序编码。表 Text 存储的是 XML 文档中的文本节点，表中的属性分别表示文本编号，父元素的先序和后序编码，文本内容。

前序-后序标号法的主要程序代码如下：

```
private  DefaultMutableTreeNode MakeXMLTree(Element TempNode) { //将 XML 文档转换为树结构
```

```
        DefaultMutableTreeNode SecondLevel=new DefaultMutableTreeNode(TempNode);
        List ChildList = TempNode.getContent();
        ArrayList<Element> ElemList = new ArrayList<Element>();
        Iterator iterator = ChildList.iterator();
        while (iterator.hasNext()) {
            Object o = iterator.next();
            if(o instanceof Element) {
                ElemList.add((Element)o);
            }
        }
        for(int m=0;m<ElemList.size();m++) {
            SecondLevel.add(MakeXMLTree(ElemList.get(m)));
        }
        return SecondLevel;
    }
    while(Enum.hasMoreElements()) {          //遍历 XML 文档树，并对节点进行编号
                Object OE = Enum.nextElement();
                String ChildElement = OE.toString();
                DefaultMutableTreeNode TE = (DefaultMutableTreeNode)OE;
                Element ElementNode = ((Element)TE.getUserObject());
    if((TE.getParent())!=null){
        OMap.put(ElementNode.getAttributeValue("PreOrder"),String.valueOf(EPostOrder));
                    EPostOrder++;
                } else{

OMap.put(ElementNode.getAttributeValue("PreOrder"),String.valueOf(EPostOrder));
                    ElementCount = EPostOrder+1;
                }
```

下面通过图 5.4.2 所示的 pub. xml 文档实例来验证前序-后序节点标号法的结果

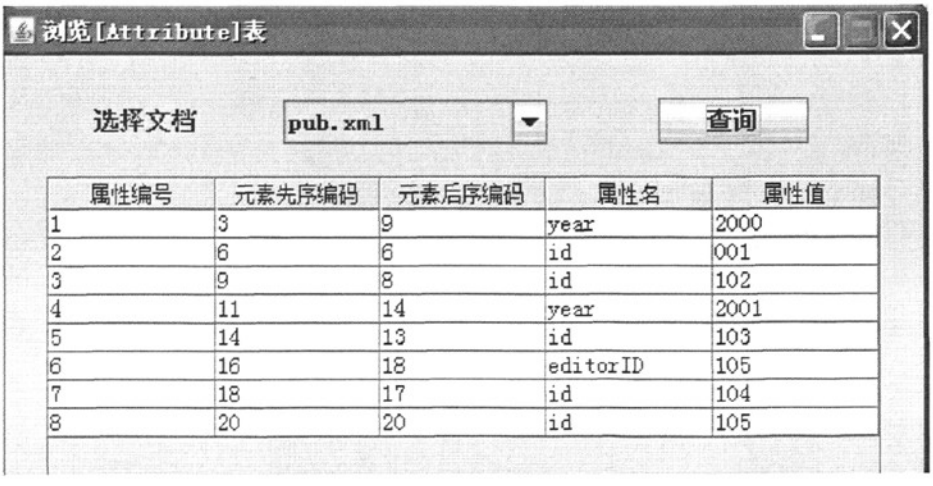
```
<pub>
    <library>Beijing Library</library>
    <book year="2000">
        <title>Database System Concept</title>
        <price>26.50</price>
        <author id="001">
            <name>Kaily Jone</name>
            <email>kjone@research.bell-labs.com</email>
        </author>
        <author id="102">
            <name>Silen Smith</name>
        </author>
    </book>
    <book year="2001">
        <title>Introduction to XML</title>
        <price>18.80</price>
        <author id="103">
            <name>Kaily Jone</name>
        </author>
    </book>
    <article editorID="105">
        <title>A Query Language for XML</title>
        <author id="104">
            <name>Kaily Jone</name>
        </author>
    </article>
<editor id="105">
        <name>A. Deutsch</name>
    </editor>
</pub>
```

图 5.4.2 pub.xml 文档

将图 5.4.2 所示的 XML 文档存储到数据库中后可以得到 3 张表，如图 5.4.3、图 5.4.4、图 5.4.5 所示。这三张表都是对 pub. xml 文档分别进行前序-后序遍历，并且对其中的节点进行编号后得到的结果。这 3 张表详细记录下了前序-后序节点标号后的 XML 中各个节点的编码，并且将元素节点、属性节点和文本节点分开编号。


| 元素先序编码 | 元素后序编码 | 元素标记 | 父元素先序... | 父元素后序... |
|---|---|---|---|---|
| 0 | 22 | #DOCUMENT | -131072 | 131072 |
| 1 | 21 | pub | 0 | 22 |
| 2 | 1 | library | 1 | 21 |
| 3 | 9 | book | 1 | 21 |
| 4 | 2 | title | 3 | 9 |
| 5 | 3 | price | 3 | 9 |
| 6 | 6 | author | 3 | 9 |
| 7 | 4 | name | 6 | 6 |
| 8 | 5 | email | 6 | 6 |
| 9 | 8 | author | 3 | 9 |
| 10 | 7 | name | 9 | 8 |
| 11 | 14 | book | 1 | 21 |
| 12 | 10 | title | 11 | 14 |
| 13 | 11 | price | 11 | 14 |
| 14 | 13 | author | 11 | 14 |
| 15 | 12 | name | 14 | 13 |
| 16 | 18 | article | 1 | 21 |
| 17 | 15 | title | 16 | 18 |
| 18 | 17 | author | 16 | 18 |

图 5.4.3 前序-后序标号后的元素节点编号




| 属性编号 | 元素先序编码 | 元素后序编码 | 属性名 | 属性值 |
|---|---|---|---|---|
| 1 | 3 | 9 | year | 2000 |
| 2 | 6 | 6 | id | 001 |
| 3 | 9 | 8 | id | 102 |
| 4 | 11 | 14 | year | 2001 |
| 5 | 14 | 13 | id | 103 |
| 6 | 16 | 18 | editorID | 105 |
| 7 | 18 | 17 | id | 104 |
| 8 | 20 | 20 | id | 105 |

**图 5.4.4 前序-后序标号后的属性节点编号**


| 文本编号 | 元素先序编码 | 元素后序编码 | 文本内容 |
|---|---|---|---|
| 1 | 2 | 1 | Beijing Library |
| 2 | 4 | 2 | Database Syst... |
| 3 | 5 | 3 | 26.50 |
| 4 | 7 | 4 | Kaily Jone |
| 5 | 8 | 5 | kjone@researc... |
| 6 | 10 | 7 | Silen Smith |
| 7 | 12 | 10 | Introduction ... |
| 8 | 13 | 11 | 18.80 |
| 9 | 15 | 12 | Kaily Jone |
| 10 | 17 | 15 | A Query Langu... |
| 11 | 19 | 16 | Kaily Jone |
| 12 | 21 | 19 | A. Deutsch |

图 5.4.5 前序-后序标号后的文本节点编号

3）前序-后序节点标号法的作用

用前序-后序节点标号法可以对 XPath 的查询方法进行优化。由于 XPath 路径表达式求值的计算过程是从 XML 文档树的根节点即文档节点开始定位的，在图 5.4.2 所示的 XML 文档中它的根元素是<pub>,<pub>的父节点就是文档节点，每一个路径表达式都要从文档节点开始定位的。为了便于将 XPath 转化为 SQL，需要将这个文档节点也要存储到表 Element 中。有如下定义：

定义：将 XML 文档节点（记为 D）存储到关系模式 Element（DocumentID，ElementPre，ElementPost，TagName，ParentPre，ParentPost）中，令 D 在属性 ElementPre 的取值为“0”设 D 所在的 XML 文档中的元素按照先序遍历的最大值为 M，则 D 在属性 ElementPost 的取值为 M+1，D 在属性 ParentPre 和 ParentPost 上取值分别为一对绝对值相等的相反数，其绝对值大于 XML 文档中元素的个数（在图 5.4.3 所示中这个绝对值取值为 131072）。

对于一个 XPath 查询来说，除了包含关系，如祖先/后裔关系、双亲/孩子关系，还有文档位置关系：如左兄弟/右兄弟关系（preceding-sibling/following-sibling）、之前/之后关系（preceding/following）关系。通过对 XML 文档进行前序-后序节点编码可以实现对 XPath 快速定位。XPath 查询一共有 13 个查询轴，其中有 4 个是最基本的，分别是祖先、后裔、之前、之后，对于 XML 文档树中任意给定的上下文节点 v，4 个基本轴

给出了该XML文档树的一种互不相交的剖分，即XML文档树的节点集合T被剖分成由4个互不相交的节点子集以及节点v自身构成：v/ancestor∪v/descendant∪v/preceeding∪v/following∪{v}

因此，对于每一个XPath定位步的计算实际上是计算一次集合集T的剖分，对于一个给定的上下文节点v，能够快速的确定4个基本轴的剖分。

如下所示，表示的是节点g的剖分情况：

g/ancestor={a, f}

g/following={i, j, k}

g/preceding={b, c, d, e}

g/descendant={h}

g/ancestor-or-self= g/ancestor∪{g}

g/descendant-or-self= g/descendant∪{g}

通过剖分，可以很快的确定节点g和其它节点的关系，同样的也可以确定其它节点之间的相互关系，从而在判断查询的过程中加快对节点的定位，提高查询效率。

通过归纳和总结得到前序-后序编码的XPath查询轴的计算判别条件如表5.4.1所示。

表5.4.1 XPath查询轴判别条件

| XPath查询轴 | 判别条件 |
|---|---|
| child(u)或attribute(u)<br>{ v\|v是u的孩子}或 { v\|v是u的属性 } | pre(u) = par(v) |
| descendant(u)<br>{v\|v是u的后裔} | pre(u) < pre(v)∧<br>post(v) < post(u) |
| descendant-or-self(u)<br>{v\|v是u的后裔或自身} | pre(u) ≤ pre(v)∧<br>post(v) ≤ post(u) |
| following(u)<br>{v\|v位于u之后} | pre(u) < pre(v)∧<br>post(u) < post(v) |
| following-or-sibing<br>{v\|v是u的右兄弟} | pre(u) < pre(v)∧post(u) < post(v)∧<br>par(u) = par(v) |
| parent(u)<br>{v\|v是u的双亲} | pre(v) = par(u) |
| ancestor(u)<br>{v\|v是u的祖先} | pre(v) < pre(u)∧<br>post(u) < post(v) |
| ancestor-or-self(u)<br>{v\|v是u的祖先或自身} | pre(v) ≤ pre(u)∧<br>post(u) ≤ post(v) |

| | |
|---|---|
| preceding(u)<br>{v\|v 是位于 u 之后} | pre(v) < pre(u) ∧<br>post(v) < post(u) |
| preceding-sibling<br>{v\|v 是 u 的左兄弟} | pre(v) < pre(u) ∧post(v) < post(u) ∧<br>par(v) = par(u) |

## 5.5 本章小结

本章主要介绍了基于关系数据库的 XML 数据管理技术中的重点内容：XML 数据的索引技术。本章在介绍了主要的索引技术后，重点研究了前序-后序节点标号的方法。详细描述了前序-后序节点标号法的原理，并且通过编程的方法实现了对 XML 文档中节点的编号，从而验证了这种方法的正确性。前序-后序节点标号法可以应用到 XPath 的快速定位查询中，提高 XML 文档的查询效率。

# 6 结束语

## 6.1 工作总结

随着Internet的发展,不断涌出的Internet应用迫切需要一种新的Web处理机制，使得在Web上传送具有统一标准并能够被机器所读取的数据格式。随着XML技术的不断成熟，XML已经成为Web上数据表示交换的统一标准。

本文分析了国内外对XML技术的研究现状，介绍了几种主要的XML数据管理技术，重点研究了基于关系数据库的XML数据管理技术。在各种XML处理方式中，关系数据库由于其成熟的技术和高性能，使得利用关系数据库来管理XML数据受到了广泛的关注，成为当前XML数据处理的主流方式，并在各种应用中得到了广泛使用。论文的主要工作有以下几方面：

1）归纳和总结了几种主要的XML管理技术，并简单描述了各种方法的原理以及各自的特点，重点介绍了基于关系数据库的XML管理技术的两种映射方法：模型映射和结构映射。

2）研究了基于关系数据库的XML数据存储技术，采用结构映射的方法，实现了XML数据到关系数据库的存储，同时实现了恢复和更新等功能。

3）研究了基于关系数据库的XML数据查询技术，采用XPath和XQuery方法，实现了对XML文档的查询。

4）归纳和总结了XML索引的分类：基于路径的索引和基于编码的索引；对XML索引查询的一些关键技术进行了初步探讨，采用了编码的方法，实现了对XML文档进行前序-后序节点编码。

## 6.2 前景展望

本文主要研究了基于关系数据的XML数据管理技术，虽然用关系数据库来存储和管理XML数据已经发展得非常成熟，应用的范围也十分广泛，能够对大量数据进行高效存取。但是这种方法依然有着明显的弱点。

1）XML的数据模型与关系模型有显著区别。XML数据是有序的、嵌套的树或图结构，而关系模型是无序的二维表结构。用简单模型来表达复杂数据会丢失语义信息，并且无论是存储还是查询都要经过格式转换才能完成。

2）XML数据不具有严格的规律性，可以没有预定的模式，而关系数据库系统需要明确的关系模式来进行数据存储和查询处理。

3）XML数据的结构和数据元素的类型可能发生变化，在关系数据库中处理模式变换是一个非常难的问题。

4）XML 查询语言通常包含许多复杂的语义，例如正规则路径表达式、嵌套查询以及查询结果重构等，而关系数据库查询语言 SQL 不能处理这些问题。

以上都是基于关系数据库的 XML 数据管理技术存在的问题，这些问题会影响 XML 数据存储的有效性和查询的效率，因此这种技术还需要改进和完善。希望在将来的研究工作中，这些问题可以得到有效的解决。

# 致 谢

衷心感谢我的导师侯晓霞教授，本课题是在侯老师的悉心教导下开展和完成的。在研究生的学习过程中，侯老师无论是在学习中，还是在生活中，都给了我们正确的引导和热诚的帮助。她深厚的学术功底，严谨的学术态度，让我们在各方面都受益匪浅。从尊敬的导师身上，我不仅学到了扎实、宽广的专业知识，也学到了做人的道理。在此，谨向侯老师表示我崇高的敬意和衷心的感谢。

同时还要感谢403教研室一起学习的同学们，在学习过程中他们给了我热心的帮助和耐心的指导。

最后还要感谢我的父母，是他们养育了我，让我能够取得今天的成绩。

# 参考文献

[1] MarkGrvaeS.尹志军等译. XML 数据库设计.北京:机械工业出版社.2002

[2] Elliotte Rusty Harold.刘文红，赵伟明等译.Java 语言与 XML 处理教程.北京:电子工业出版社.2003

[3] 邵敏,李力鸿郑震坤,何川.XML 编程实践.北京:清华大学出版社.2002

[4] Charles F.Goldfarb,Paul prescod.张晓晖,王艳斌,赵伟明等.XML 手册(第四版).北京:电子工业出版社.2003

[5] 黄莹,杨明福.XML 文档的存储方法研究.计算机工程.2002.28(5):281-283

[6] 李骥,陈福生.Native-XML 数据库综述.计算机工程与设计.2004.25(6):932-934

[7] 王红梅,朱洪秀,李欣.基于XML数据管理研究. 通化师范学院学报.2002. 23(2)22-25

[8] 孙一中等.XML理论和应用基础.北京:北京邮电大学出版社,2000

[9] 金玉玲,陈培久,裘江南.XQuery-一种全新的 XML 查询语言.情报学报.2002.21(4)

[10] Mark Johnson.Tamino XML Sever.Software AG.International Journal on Digital Libraries

[11] 翁亮,曾昭平,刘超俊,诸鸿文.XML 技术分析.计算机与网络.2000.1:24-25

[12] 候昌昌.XML 数据管理技术研究.南京师范大学硕士学位论文.2004,1

[13] 罗时辉.XML 数据存储管理系统.南京理工大学硕士学位论文.2003,1

[14] 邵蓉.XML 数据管理技术的研究.南京理工大学硕士学位论文.2005,7

[15] 周傲英等.基于关系的 XML 数据存储.计算机应用.2000,20(9)

[16] 徐德智，吴敏，赖同庆.XML 模式/查询和存储技术扫描.计算机工程与科学.2003,25(3)

[17] Albrecht Schmidt, Martin L.Kersten, Menzo Windhouwer, Florian Waas. Efficient Relational

Storage and Retrieval of XML Documents.InProc.ofWebDB.2000.47-52

[18] J.McHugh,S.Abiteboul,R.Goldman,D.Quass,J.Widom.Lore:A Database Management System for Semistructured Data.SIGMOD Record.1997.26(3)

[19] Catalina Fan,John Funderburk,Hou-in Lam,Jerry Kieman,Eugene Shekita,Jayvel Shanmugasundaram.XTABLES：bridging relational technology and XML.IBM Systems Journal 41.2002.11

[20] 袁升发.基于关系模式的 XML 数据存储技术研究.湖南科技学院学报.2006.5.27(5)

[21] 许卓明等.基于关系数据库的 XML 存储技术评述.计算机工程与应用.2003.21

[22] 美 阿吉夫著，马树奇等译.JavaXML 程序员参考手册.电子工业出版社.2002.5

[23] 王国仁,于戈,杨晓春,于亚新.XML 数据管理技术.电子工业出版社.2007.4

[24] D.Lee and W.W.Chu.CPI:Constraints-Preserving Inlining Algorithm for Mapping XML DTD to Relational Schema.J.Dada&Knowledge Engineering Vol,39,No.October 2001

[25] J.Mchugh,J.Widom.Query Optimaization for XML.Proceedings of the 25th VLDB Conference.1999

[26] Igor Tatarinov,Zachary G.Ives et al.Updating XML.In proceedings of the 2001 SIGMOD
Conference.2001

[27] Jonathan Robie,Don Chamberlin,Daniela Rlorescu.Quilt:an XML Query Language.

[28] Ioana Manolescu,Daniela Florescu,Donald Kossmann.Pushing XML queries inside relational databases.INRIA,2001

[29] M.Carey,et al..XPERANTO:publishing object relational data as XML.Workshop on the Web and Databases(WebDB),Dallas,Texas,May 2000

[30] J.Shanmugasundaram,et al.Querying XML views of relational data.VLDB Conference, Rome,Italy.September 2001

[31] 万常选.XML数据库技术.清华大学出版社.2005

[32] Feng Tian,David J.DeWitt,Jianjun Chen,Chun Zhang.The Design and performance Evaluation of Alternative XML Storage Strategies.SIGMOD Record special issue on"Data Management Issues in E-commerce".March 2002

[33] Serge Abiteboul,Sophie Cluet,Vassilis Christophides,Tova Milo,Guido Moerkotte,Jerome
Simeon.Querying Documnts in Object Databases.International Journal on Digital Libraries. 1997,1(1):5-19.

[34] R.Bourret.XML and Databases.IETF Data Engineering Bulletin.Vol.22,No.3,1999

[35] Ullas Nmabiar，Zoe Lacroix，Stephnae Bressna.Effieient XML Data Mnaagement: An Analysis.EC-Web2002，LNCS2455.2002:87-98

[36] George Feinberg.Anatomy of a Native XML Datbaase.XML 2004 Proeeedings by Sehemasotf.2004:1-12

[37] Dongwook Shin. XML Indexing and Retrieval with a Hybrid Storage Model. Knowledge and Information System.2001.3.255-161

[38] He Hao , Yang J un. Multiresolution Indexing of XML for Frequent Queries. In : Proceeding of the 20th International Conference on Data Engineering. Boston , USA , 2004,1:683～694

[39] Chien S Y, Vagena Z ,Zhang Donghui , et al . Efficient St ructural Joins on Indexed XML Document s. In : Papadias D , et al . eds.Proceedings of t he 28th VLDB International

Conference on Very Large Database , Hong Kong , China ,2002,1:263～274

[40] Li Quanzhong , Moon B. Indexing and Querying XML Data for Regular Pat h Expressions. In : Proceedings of t he 27th VLDB Conference ,Roma ,Italy ,2001. 361～370

[41] M. Yoshikawa, T. Amagasa, T. Shimura and S. Uemura,"XRel: A Path Based Approach to Storage and Retrieval of XML Documents using Relational Databases", ACM Transactions on Internet Technology, August 2001.110- 141.

[42] Florescu, D. and Kossmann, D. A performance evaluation of alternative mapping schemes for storing xml data in a relational database.Technical report.

[43] Pavle Mogin: XML Storage Techniques; COMP 442 Issues in Databases and Information Systems

[44] Haifeng Jiang, Hongjun Lu, Wei Wang, Jeffrey Xu Yu: Path Materialization Revisited: An Efficient Storage Model for XML Data. Australasian Database Conference 2002

[45] Elisa Bertino ,Barbara Carania. Integrating XML and databases.IEEE Internet Computing, 2001(4) .

[46] Richard Ho，LiBai，DavidElliman.A New Data Model for XML Databases. International Journal of Intelligent Systems in Accounting，Finance& Management.2003.