摘要

在网络技术飞速发展的同时，web 上涌现出的数据也呈指数级增长．XML 逐渐成为互联网上描述和交换信息的标准格式，广泛应用于电子商务、数字图书馆等应用实例和产品．XML 文档集检索的有效性成为研究一个重要方向．

XML 关键字查询近年来成为 XML 数据检索的一个研究热点．相较于 XML 查询语言，如 XQuery 等，XML 关键字查询有其独特的优势．用户不需要额外学习复杂的查询语言，也不需要深入了解查询信息的内部底层结构，只需要提供相关内容的关键字就可以实现数据的检索．同时 XML 关键字检索以元素为粒度进行，结果只返回包含用户提供的全部关键字的 XML 文档片段，提高了检索的速度．

论文的主要研究内容如下：将 XML 树中实体节点和属性节点类比于数据库中 E-R 模型的实体和属性，提出实体子树的概念，并将实体子树做为语义相关单元的基本元素．定义了标识节点语义信息的语义三元组，从而将关键字和特定的环境语义关联在一起．用户提前确定关键字的语义，可以提高查询结果与用户查询意图相匹配的几率．将关键字分为主关键字和从关键字，在查询过程中过滤从关键字，仅保留主关键字可以减少查询的范围，提高查询的效率．在 XML 关键字查询过程中定义了关键字匹配节点间的语义相关性，通过寻找关键字的相关语义单元提高了查询结果的语义相关性．在上述内容的基础上描述并实现了基于语义相关性的关键字查询算法．由于关系型存储具有严密的理论及成熟的实现技术，XML 信息的后台存储利用关系数据库来实现．最后将基于语义相关性的关键字查询方法和 MLCA 方法的查询结果进行比较．实验表明，基于语义相关性的关键字查询方法更好地表达了用户的查询意图，在查询的有效性和查询效率上都有较大改进．

关键词：XML；语义相关性；关键字查询；实体子树；关系数据库

分类号：TP311

ABSTRACT

With the development of the network technology, large amount of data come forth exponentially in the web. XML is becoming the standard to describe and exchange information on the Internet. XML is widely used in e-commerce, information systems, and digital libraries and so on. Naturally, efficient information retrieval from these great amounts of XML documents is becoming extremely important.

XML keyword search becomes a research hotspot in XML data searching field in recent years. Compared with XQuery and other XML query language, XML keyword search has its unique advantages. The customer doesn't need to study complicated query language, nor need to have thorough understanding of the structure of the XML document. In fact, the customer only needs to supply the keywords related to the contents he is interested in, then the result can be returned. Furthermore, because the granularity of XML keyword search is based on elements, it can only return the parts of the document including a keyword so that the search is efficient.

The main researchful contents of the thesis are as follows: Comparing entity-node, attribute-node of the E-R model with XML's, we defined Entity Sub Tree as an element of Semantically Relevant Unit. We defined the semantic relevance of keyword matching nodes which makes each keyword be related to its corresponding context semantics. This allows users to choose particular semantics which makes the result match users' real intentions for querying. We divided keywords into primary keywords and lesser keywords to improve query efficiency. Finding semantically relevant units improves the semantic relevance of the result. Based on the above, we described and implemented XML keyword search arithmetic based on semantic relevance. Because relational database has strict theory and mature technology, we use it to store the XML information. The result of our experiments demonstrates the effectiveness and the efficency of the new query method, and it does better in expressing the user's querying intention.

KEYWORDS: XML; semantic relevance; keyword search; entity subtree; Relational Database

CLASSNO: TP311

独创性声明

本人声明所呈交的学位论文是本人在导师指导下进行的研究工作和取得的研究成果，除了文中特别加以标注和致谢之处外，论文中不包含其他人已经发表或撰写过的研究成果，也不包含为获得北京交通大学或其他教育机构的学位或证书而使用过的材料．与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确的说明并表示了谢意．

学位论文作者签名：李 签字日期：2009年6月17日

学位论文版权使用授权书

本学位论文作者完全了解北京交通大学有关保留、使用学位论文的规定．特授权北京交通大学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，并采用影印、缩印或扫描等复制手段保存、汇编以供查阅和借阅．同意学校向国家有关部门或机构送交论文的复印件和磁盘．

（保密的学位论文在解密后适用本授权说明）

学位论文作者签名：[signature]　　导师签名：[signature]

签字日期：2009 年 6月17日　　签字日期：2009年 6月17日

致谢

在撰写整个论文的过程中，我的导师王宁副教授给了我细心的指导和无尽的关怀．无论从论文的选题、撰写，直至最终定稿，王老师都在耐心地指导我，帮助我逐步理清思路，最终完成本论文．

我在攻读硕士研究生期间，深深受益于王老师的关心和指导．在学习上，王老师一方面对我严格要求，另一方面又给予充分的信任，当我遇到困惑时常常能够点拨迷津、提出关键问题，使我茅塞顿开．王老师渊博的学识、严谨的治学态度和勤奋务实的工作态度让我受益匪浅．在生活和为人上，王老师更是给予我无微不至的关怀，不仅勉励我自励上进，更时刻教育我要正品端德．在此，我对恩师表示最崇高的敬意和最诚挚的感谢！

此外，还要感谢师兄蔺旭东博士．在蔺师兄的指导下，我逐渐融入了实验室这个大家庭，能够较好地适应在实验室的研究生活．在完成课题的过程中，他给了我很多帮助．在与蔺师兄的交流和探讨中，我才逐渐发现问题并理清思路，最终完成论文的撰写．

在实验室工作及撰写论文期间，袁玲、冯浩、程友忠、王东伟等同学对我论文中的研究工作给予了热情帮助，在此向他们表达我的感激之情．

感谢我的父母，他们的理解和支持使我能够在学校专心完成学业．

最后，衷心地感谢在百忙之中审阅论文的各位老师和专家，恳请各位老师多多批评指正，并提出宝贵的意见．

1 引言

当网络技术飞速发展的同时，web 上涌现出了大量的数据．而随着 XML 逐渐成为互联网上信息描述和信息交换的标准格式，互联网上符合 XML 规范的数据也越来越多地存在于当前的信息社会中，大量的 Web 应用，如电子商务、数字图书馆、网格、咨询系统等采用 XML 作为数据的表示形式．因此，用户如何有效地查询 XML 文档成为一个重要的研究方向．

1.1 研究背景

XML 查询方式可以分为两类：XML Query 查询模式和关键字查询．

XML Query 查询模式主要利用结构化查询语言实现 XML 文档的查询．这类结构化查询语言，如 XPath[1]、XQuery[2]和 XML-QL[3]等，通过定义格式良好的，复杂精确的描述语言，采用正则表达式[4-6]，从结构上来寻找 XML 数据单元之间的关系和内容，进而搜索 XML 数据．因此，它可以获得准确的预期数据．但是这类查询方式有明显的缺陷：首先，大多数的普通用户并不了解或熟悉查询语言的相关语法机制，即不会使用结构化查询语言．其次，即便用户掌握了查询语言，对于结构查询而言，用户仍然需要了解所查询 XML 文档的数据组织情况，才能够构造查询表达式．但是大多数的 XML 文档并没有提供其结构信息．即使存在文档结构说明，对相同的查询也要为每个异构文档编写不同的查询表达式．

与结构化查询相比，关键字查询最大的优势在于它的简单易用性．对用户来说，他们只需要提供简单的关键字信息，而不需要掌握复杂的查询语言，也不需要了解所查询的目标 XML 文档的数据结构就可以实现数据的检索，这大大方便了普通用户的使用．

虽然 XML 关键词查询和传统的信息检索技术的查询表达式均为若干关键字，但两者有着显著的区别．在传统的信息检索中，查询的目标是整个文档，查询返回的结果也是整个文档，即：只要某个文件中包含查询表达式中所有的关键字，就将整个文档其作为一个查询结果返回．与传统的信息检索技术不同，在 XML 文档关键字查询中，查询对象有可能是单个的、包含大量内容信息的 XML 文档，而查询返回的结果是该 XML 文档的一些片段．

1.2 研究现状

国外对 XML 数据的信息检索研究开始于 2001 年[7]，为了对研究者的 XML 检索方法进行统一评估，同时也为研究机构比较其成果提供一个论坛，在 2002 年，欧洲 DELOS Network of Excellence for Digital Libraries 与 IEEE Computer Society 共同启动 Initiative of Evaluation for XML Retrieval（INDEX）创新活动．每年都有众多研究机构与学者在 INDEX 的会议上参与 XML 信息检索问题的讨论．

目前，XML 关键字查询算法大都是在树型存储模型上展开，以 LCA（Lowest Common Ancestor）来判断 XML 文档中的任意两个节点是否语义相关，利用 Dewey 编码等编码方式记录节点间的层次关系．已有的方法包括 Equix[8]，Meet[9]，MLCA[10]，XSEarch[11]，XRank[12]，SLCA[13]等．由 Sara Cohen 等人提出的 Equix 是搜索 XML 的语言，它提供了图形化的抽象语法和有形的具体语法来支持 EquiX 查询，并结合了模式匹配，用逐步增加匹配节点的方法对匹配节点集所包含的信息进行约束．此后他们又提出了 XSEarch 算法，XSEarch 是一个基于关键词语义的 XML 搜索引擎，它为那些普通用户提供了简单的查询语言，返回语义相关的文档片段．XSEarch 允许用户指定标签-关键字对，并为非完全结构 NFS（Non-full structure）查询结果的判断提供了 interconnection relationship，即互联判断标准和方法．具体而言，对于 XML 文档中的两个节点，若在连接这两个节点的路径上没有出现两个相同标签的节点，则认为它们语义相关；否则认为两者不相关．Interconnection 这种判别方法具有其合理性，能够起到较好的语义区分作用，但 XSEarch 的不足之处在于，若它预先计算所有节点对之间是否相关，就需要占用大量的存储空间，而在查询时即时判断节点对之间的关系需要涉及递归计算，时间效率较低，特别是在 XML 文档深度较大时更加耗费时间．Meet 运算返回节点的最小公共祖先（LCA）．它提出了一种递归计算 LCA 的算法，但是没有考虑节点的语义相关性问题，所以查询的准确率往往较低．密歇根大学的 Yunyao Li 等人在 LCA 的基础上提出 MLCA（Meaningful LCA）的概念．对于 MLCA 而言，XML 文档中的节点只与某个节点集合中与其最近的节点语义相关．MLCA 的判断标准在大多数情况下都是适用的，能够得到用户满意的查询结果．MLCA 通过向 XQuery 添加一些新的功能来自动计算有意义的匹配节点集 MLCAS（Meaningful Lowest Common Ancestor Structure），并扩展不明确的标签名．MLCA 的查询语言使用扩展的 XQuery 语言，因此要求用户在编写查询表达式时预先知道文档的结构，并且对于不同的 XML 文档，需要编写不同的查询表达式．XRank 是最早考虑到 XML 文档的分层和超链接结构，以及关键词二维接近概念的 XML 检索系统．它提出了一种基于栈的 LCA 算法，给出了关键字在 XML 文档上的相似查询，通过

ElemRank 来衡量 XML 元素的客观重要性，采用类似于 PageRank 的算法，并考虑到 XML 的嵌套结构来计算不同 XML 元素的排序，但 XRank 不区分关键字和标签，同样没有考虑 LCA 的语义．加利福尼亚大学的 Yannis Papakonstantinou 等人提出 Smallest Lowest Common Ancestor（SLCA）的概念，使得基于结构的 XML 关键字查询的研究达到了一个新的水平，并且提出 Indexed Lookup Eager（ILE）、Scan Eager（SE）和 Stack 算法，SLCA 利用特殊的 B+树索引提高检索效率．尽管上述各种方法在实现细节上有所不同，但它们的查询都分为两个阶段进行：首先根据关键字找到匹配节点集，然后从匹配节点集生成结果集．从这一特点，我们不难看出使用原始算法进行关键字查询的缺陷．对于第一阶段，根据关键字找到的匹配节点集往往比较大，难免对后续计算的效率带来不利影响；在第二阶段，由于节点间节点关系只通过编码加以保持，因此，在关键字匹配节点存在缺失时，会产生节点组合的有效性问题，即结果中关键字匹配节点的组合是否是用户真正关心的信息，这需要进行有效性判断运算，会对算法效率产生不利影响．

也有人对关键字查询的结果集进行研究[14-15]．Yannis Papakonstantinou 等人对关键字在结果集中的连接方式进行了重点研究，并提出 SA 算法[14]，有效的返回包含关键字结点间的连接．2007 年，Ziyang Liu 和 Yi Chen 提出 XSeek 算法[15]，通过参考结构化查询中将输入关键字进行分类的思想，将关键字分成两类：指定查询条件的关键字和表明返回结果的关键字，以此体现更多的查询语义，从而达到减小结果集规模的目的．但是，由于 XSeek 算法仍是在典型关键字查询算法得到的结果集的基础上对结果集做进一步的运算，并且它的两种类型的关键字没有充分发挥各自不同的作用，用户的查询意图在多数情况下仍无法准确捕捉．

国内对 XML 查询的研究，一方面集中在 XML 数据的索引结构[16-17]上，包括基于结构连接的索引和基于路径的索引[18]，现在的工作大多放在以数据为中心的 XML 数据的数据库存储和检索技术上．另一方面主要是研究基于关键词的 XML 检索模型[19-20]．国内的研究大多停留在数据库领域，对信息检索和数据库的融合的研究较少．

由此我们可以看到，目前的研究工作没有有效地解决 XML 文档关键字查询所存在的问题．有些没有考虑查询结果的语义问题，有些需要复杂的有效性验证，有些实现技术的时间、空间复杂度较高．

1.3 本文完成的工作

随着互联网技术的飞速发展和网络上数据的快速增加，XML 逐渐成为了一种描述信息和交换信息的标准，对 XML 数据的查询也逐渐增多．所以，如何能够简

单并有效地查询 XML 文档成为一个研究的热点.

对用户来说，关键字查询方法简单，具有易操作性. 但典型的关键字查询过分地强调了节点之间的结构信息，而没有充分利用节点之间的语义相关性，导致查询结果包含了许多没有意义的节点.

在上述背景之下，本文完成的工作是：

- 首先，介绍了 XML 查询的研究背景，总结了目前 XML 关键字查询的研究内容和方向.
- 其次，叙述了 XML 的基础理论，XML Query 查询模式和 XML 关键字查询的相关技术. 基础理论部分主要介绍了 XML 的基本概念、XML 相关技术规范和标准和 XML 的应用. XML Query 查询模式介绍了 XML 查询语言 XPath 和 XQuery. XML 关键字查询的相关技术部分重点叙述了关键字查询的理论基础，包括 XML 树型结构，Dewey 编码，关键字匹配的 SLCA 方法以及怎样用关系数据库存储 XML 文件.
- 第三，在分析过典型的关键字查询方法的不足之后，给出了关键字环境语义的概念，使用户能够选择关键字的语义，以符合其查询意图；进一步将关键字区分为主关键字和次关键字，用来提高查询效率. 类比关系数据库中的 E-R 模型，提出了实体子树的概念，定义了关键字匹配节点之间的语义相关性，通过语义相关单元的概念来提高查询结果的语义相关性. 随后给出了完整的查询算法及具体的实现过程.
- 最后，通过实验比较了新的基于语义相关性的 XML 关键字查询方法和传统关键字查询 MLCA 方法的有效性和效率. 有效性实验主要对比了两种查询方法的查全率和查准率. 效率测试实验主要从不同查询用例的返回时间、关键字的数目对返回时间的影响以及数据集的大小对返回时间的影响 3 个方面进行两种查询方法的比较. 实验证明，基于语义相关性的 XML 关键字查询方法可以返回更匹配的查询结果，具有较高的查询效率和准确度.

1.4 论文的组织结构

第 1 章给出了课题的出发点以及研究的问题及范围，简要介绍了 XML 查询领域的研究背景、XML 关键字查询的研究现状以及本文所完成的工作.

第 2 章介绍了与课题相关的一些理论背景. 包括 XML 的基本概念，XML Query 查询模式和 XML 关键字查询技术. 重点叙述了 XML 的关键字查询技术.

第 3 章描述了算法的基础知识，算法的概要思想和算法的具体实现.

第 4 章给出了本文的实验过程和结果，证明了算法的优势.

第 5 章总结全文，对本文研究做了分析和总结，并给出了未来的研究内容和方向.

图 1.1 是本文研究内容的组织结构图.

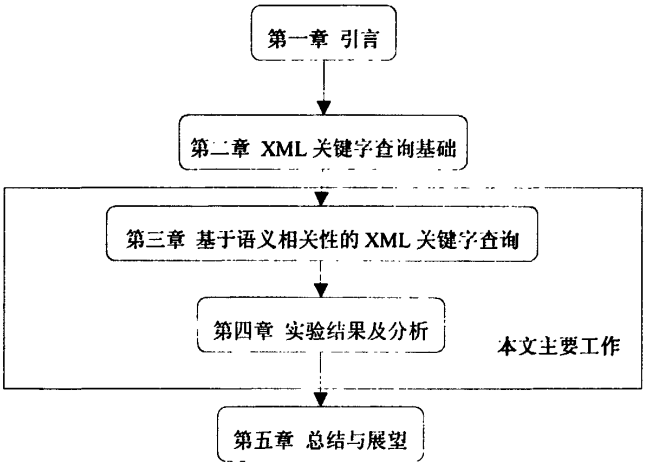


图 1.1 论文的组织结构

Figure1.1 Structure of the thesis

2 XML关键字查询基础

随着互联网的飞速发展，人们的工作、学习和生活方式发生了巨大的改变，人们获取知识和信息的途径越来越多，网站和Web页面等信息也急剧膨胀，HTML（hypertext markup language，超文本标记语言）开始不能满足网络设计的需求，逐渐暴露出其局限性. HTML缺乏扩展性，定义也较为模糊，会使错误蔓延扩展. 人们需要一种标准化的、可扩展的和具有严谨结构的新语言，XML也因此孕育而生. 最初XML的提出是为了增强应用程序从Web上获取文档的解释和操作能力，之后人们对查询这些文档内容的可能性产生极大的兴趣. 随着大量XML数据的出现，如何有效地存储、管理和查询这些XML数据成为研究的一个重要课题.

用户为了在数据量巨大的Web上查询符合自己特定需求的信息，搜索引擎成为一个十分重要的工具，而关键字查询技术又是搜索引擎的核心支撑技术. 目前通用的搜索引擎主要是针对静态的HTML设计的. 但HTML只是一种简单的表示语言，用来显示内容，并不能描述内容，因此无法做到对检索信息的精确定位，从而导致查询的结果极不精确，Web信息传输量巨大. 与HTML不同，XML是一种开放式的、自描述性的语言，它不仅可以定义数据的结构，也能描述数据的内容. 所以XML既可以看作是一种数据描述语言，也可以看作是一种标识语言，它使数据的结构和表示相分离，也就让基于知识、基于语义和基于内容的搜索成为可能.

XML关键字搜索与HTML关键字搜索的最大区别在于两者搜索结果的粒度不同. 因为HTML文档中的标签仅显示指令而没有任何语义信息，所以文档很难分割，HTML搜索结果的粒度是整个文档. 而一般的HTML文档的规模都不大，因此可以直接将整个文档返回给用户. 相较之下，XML搜索结果的粒度是文档的特定片段. XML文档中的标签是包含一定语义信息的，它可以指出所包含的数据内容的含义. 正因如此，XML关键字搜索只需返回与用户提交的关键字相关的XML文档片段，而不需要返回用户整个文档. 一般来说，XML文档的规模要比HTML文档大很多，返回结果粒度的精细化能够帮用户过滤掉很多无用或不相关的信息，提高了搜索的性能.

XML关键字查询技术的进一步研究将有利于提高XML搜索的查询效率及准确度，而广泛使用XML技术的众多应用，如电子商务，咨询系统整合等也因此有更进一步的发展前景.

2.1 XML基本概念

XML是一种可扩展的标记语言，具有自描述性、可扩展性以及数据的结构和表示相分离等特性．下面我们具体介绍XML的基本概念，XML相关的技术规范和标准以及XML的功能和应用．

2.1.1 XML概述

20世纪末60年代初，IBM公司为了解决不同专用格式创建的法律文件无法相互移植的问题，提出了通用标记语言，后来建立了标准通用标记语言SGML（standard generalized markup language）．1986年，国际标准化组织ISO采用了SGML．1996年7月，SGML专家和W3C（World Wide Web Consortium）组织了一个SGML工作小组，但是SGML过于复杂，不适于在WWW上使用．而Web上的通用标识语言HTML（SGML的一个子集），随着互联网的迅速发展也逐渐显示其不足之处．HTML是用于描述数据的显示形式而非描述数据本身，因此，在Web上对HTML文档内含数据的处理显得非常困难．而且HTML仅支持固定及有限的标签集，用户无法根据自己的需要设置和添加有意义的、可以让他人使用的标记．于是1996年11月该工作小组提出了XML的初稿，并于1998年2月正式发布了XML1.0版．

此后W3C的XML工作组定义了XML（extensible markup language，可扩展的标记语言）．对XML语言的描述[21]如下："XML是SGML的子集，其目标是允许普通的SGML在Web上以目前HTML的方式被服务、接收和处理．XML被设计成易于实现，且可在SGML和HTML之间相互操作．"XML组合了其前语言SGML强大的功能和可扩展性，并达到了Web团体要求的简洁性．W3C同样阐述了XML的10个设计目标[21]：

- XML应该可以直接用于互联网；
- XML应该支持各种应用程序；
- XML应该与SGML兼容；
- 编写处理XML文档的应用程序应该很简单；
- XML中可选特性的数目应该尽可能地少，理想情况是零；
- XML文档应该便于阅读而且相当清晰；
- XML的设计应该很快准备好；
- XML的设计应该正式而且简洁；
- XML文档应该易于创建；

- XML标记的简洁性是最不重要的.

XML是一种元标记语言（Meta-Markup Language），是一种把数据表示为一个文本字符串的语言，这个文本字符串还包括用于描述数据的散布的“标记(tag)”.使用标记允许把文本和与它的内容或形式相关的信息散布在一起.所以XML可以提供描述结构化资料的格式，是用于描述数据结构的一种标准，可以用于不同组织间的数据交换.HTML中数据的元素标记用来表示数据的显示，而XML中数据的元素标记用来表示数据的意义.标记提供了一种给文档添加元内容和结构信息的机制.为了注释，XML创建了标记的层次.元素是XML的基本组成，元素之间可以包含或者嵌套子元素，从而表示数据之间的联系.一个XML文档由嵌套的元素层次结构构成，每个文档有一个唯一的根节点，一个元素有一个标记，描述该元素的含义，一个元素由从起始标记到终止标记的区域构成，该区域可以是嵌套的子元素，也可以是属性或文本串值.XML定义了语义标记的规则，这些规则用于创建标记语言，并且可以使解析器处理所有新创建的标记语言.XML是能够进行自描述（Self-Describing）的语言，它的标记并没有预先定义，需要使用者自己定义所需的标记.XML使用文档类型定义DTD（document type definition）显示数据，使用XSL定义描述文档显示的机制，通过SAX和DOM技术来解析XML文档、使用XPath和XQuery等查询语言进行文档查询.

与HTML相比，XML具有很多优点和自己的特性：

（1）自描述性.它允许实现者定义自己的一套标记，也允许根据不同的规则制定不同的标记，从而实现了用定义自己的标记集来说明文档内容的功能，能够比较好地表现多种复杂的数据关系.基于XML的应用程序可以在XML文件中忽略不相关的部分，准确高效地搜索相关的数据.

（2）可扩展性.XML是一种元语言，基于它提供的规则，可以任意定制标语言.XML为注释和标记字符数据提供了一种可扩展的机制，避免了HTML的重载[22].标记的可扩展性，能够使各种不同格式的数据比较容易地转化为XML数据.

（3）简单易用性.XML的规则简单明了，这些规则可用于创建标记语言，也可以使解析器处理所有新创建的标记语言.XML不是专有的，它易于阅读和编写，人和计算机均能够很容易的理解XML文档.而且XML允许最佳的输出格式，如HTML等格式，并格式化应用程序，这使它成为在不同的应用之间交换数据的理想格式[22].

（4）开放的国际化标准.XML是W3C制定的开放标准，完全可用于Web和工具的开发，从而使基于XML的应用具有广泛性.XML不仅能在不同的计算机系统之间交换信息，而且能跨国界和超越不同文化疆界交换信息.

（5）XML支持对文档内容的验证.XML文档的结构和内容由其语法定义.如

文档类型定义 DTD 和 XML 模式.

（6）支持高级搜索. 由于文档内容的结构和含义很容易了解，在搜索中还能加入与数据相关的上下文信息，使得 XML 文档的搜索更为容易和精确.

（7）XML 使数据的结构和表示相分离. 这简化了应用的开发与维护.

（8）具有集成数据和文档的能力.

通过一个简单的 XML 文档示例图 2.1，我们可以看到 XML 与 HTML 在结构上的一些区别：XML 中的标签必须成对出现，HTML 没有此限制；XML 中所有的属性必须有值，并用双引号标示，HTML 的某些属性值可为空；XML 的嵌套顺序必须适合，HTML 没有要求，因此标签结构混乱；XML 的标签是可扩展的，HTML 的标签具有统一的格式. XML 的机构更为规范，更容易处理，也能够表达更多的信息.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<dblp>
    <incollection mdate="2004-03-08" key1=" BreitbartGS95">
        <author>Yuri Breitbart</author>
        <author>Hector Garcia-Molina</author>
        <author>Abraham Silberschatz</author>
        <pages>573-591</pages>
        <booktitle>Modern Database Systems</booktitle>
        <year>1995</year>
    </incollection>
</dblp>
```

图 2.1 XML 文档

Figure2.1 XML document

XML 面向内容，具有多样化的结构及丰富的语义，易掌握性，适于各种数据的交换. XML 具有自描述性、可扩展性、结构和内容两重特性[23]. 可以预言，XML 将成为数据组织和交换的事实标准[23]. 随着 XML 成为信息表示和交换的标准，XML 的应用也越来越广，将进一步促进 XML 在高级数据库搜索、网上银行、电子商务、医药、法律等领域的使用.

XML 文档必须要符合一定的语法规则才能被 XML 解析器解析，进而才能利用其中的数据. XML 文档有三种类型：

（1）无效的 XML 文档

没有遵守 XML 规范定义的语法规则. 如果已经在 DTD 或模式中定义了文档

能够包含什么，而相应的文档没有遵守对应的规则，则此文档也属于无效文档.

（2）规范的XML文档（Well-Formed XML）

符合W3C制定的基本语法规则的XML文档，但是可以没有DTD或模式定义．即使文档自身带有DTD或模式，也可以不遵守DTD或模式的定义.

（3）有效的XML文件（Validated XML）

文档首先要满足规范的XML文档的要求，在此基础之上还要在符合额为的一些约束．即文档的结构必须要符合某个和文档关联的DTD，必须在内部包含DTD或者显式地指明它所引用的外部DTD文件.

一般认为无效的XML文档是没有利用价值的文件，甚至不能称为一个XML文档．所以通常认为XML文档分为规范的和有效的两类XML文档.

2.1.2 XML相关技术规范和标准

XML是一种元标记语言，它能够开发各种不同应用的特定领域的标记语言.对不同的应用领域就需要制定对应的应用标准，包括标记表示的含义、附加的语法约束等．应用开发者可以使用XML各种标准的应用编程接口来获得和设置XML文档中的元素、属性和数据内容．而为了使XML能够进一步实用化，针对XML应用中的公用特征、方法或规则，W3C制定了一些XML的基础技术规范．例如，文档模式技术、文档样式技术、文档定位技术、命名空间、文档解析技术等.

XML文档的模式包括了DTD（Document Type Definition，文档类型定义）和XML Schema（XML模式）．XML文档的模式定义了XML文档的逻辑结构，规定了XML文档中的元素、属性、元素之间及元素与属性之间的关系.

DTD：DTD是一套关于标记符的语法规则．它是XML规范的一部分，是XML文的件验证机制．虽然DTD不是必须的，但是它为文档的编制带来了方便．通过比较XML文档和DTD文件能够检验元素和标签的使用是否正确，文档是否符合规范，因此DTD是一种保证XML文档格式正确的有效方法．结构满足DTD要求的XML文档称为有效的文档，DTD可以保证用户不会创建一个无效的XML结构.

XML Schema：XML Schema是另一种确定XML文档有效性的方法，它包含DTD所提供的全部功能并有所扩展．以XML语言为基础的XML Schema，针对将来的额外内容是可扩展的，支持命名空间（Namespaces），支持数据类型，内容比DTD丰富，有更强的表达能力，灵活性远远超出了现有DTD规范的方式.

XML的数据与显示是分离的，XML的文档样式技术CSS（Cascading Style Sheets，层叠样式表）和XSL（Extensible Stylesheet Language，可扩展样式表语言）定义了XML文档的显示样式.

CSS：为HTML开发的CSS可用于XML文档应用样式．CSS主要控制XML文档的显示，但是不会改变源文档的结构，也不具备基于XML发布通常所需的转换和生成结构的能力．

XSL：XSL是专门为XML设计的语法规范，不仅可以用来显示XML文档，还能够把一个XML文档转化为HTML、另一个格式的XML文档或其他格式的文档．XML用于承载数据，而XSL则用于设置数据的格式．一个XSL样式表集合了一系列设计规则，用于从XML文件中抽取信息，并将其转换成HTML等其他格式．XSL不是内容管理工具，不能用于更改XML文档的内容或编辑信息．XSL包括两个部分：XSLT（XSL Transformation）是用于转换XML文档的语言，XPath（XML Path Language，XML路径语言）是用于XML文档导航的语言．

XML文档的定位技术是XSLT和XPointer中用于对XML文档各部分导航定位的Xpath语言．

XPath：XPath提供了定位XML文档中指定部分的一种高效方式．它不是结构化语言，而是一种基于字符串的表达式语言，描述了如何识别、选择、匹配XML文件中的各个构成元件，包括元素、属性、文字内容等．XPath定义了一些可对XML文档中的数据进行“寻址”的路径表达式，通过路径表达式来选取XML文档树状结构中的节点或节点集．

Namespaces（命名空间）：Namespace限定了与之关联的所有元素的作用范围，避免了标记名重名的情况，用于保证XML与DTD中名字的一致性，以便不同的DTD中的名字在需要时可以合并到一个文档中．

文档解析是指对XML文档的内容和结构进行访问和分析．文档解析技术包括DOM（Document Object Model，文档对象模型）和SAX（Simple API for XML）．

DOM：DOM是W3C为在内存中建立XML文档的树型结构提供的一种标准规范，是为XML文档的已解析版本定义的一组独立于平台和语言的接口，一种标准API．其不仅提供了对存储在内存中的XML文档的一个完全的表示，也提供了随机访问整个文档的方法．因此，可将DOM看作一个标准的连接文档和应用程序或脚本语言的结构体系，提供给用户一个接口以装载、定位、操作和序列化XML文档．

SAX：SAX是用于XML的简单API，是一个基于事件的XML文档解析技术，通过事件驱动来识别XML文档的内容．它不需要像DOM那样建立一个完整的文档树，不会创建任何对象，而是在读取文档时激活一系列事件，这些事件被推给事件处理器，然后事件处理器提供对文档内容的访问．基于事件创造的对象需要程序开发人员自己完成．SAX的这一特性使应用程序开发人员可以在相应的事件中写入特定的处理代码．

DOM需要构建整个文档驻留内存的树，当文档很大时就要就有足够大的内存空间．而且DOM要创建表示原始文档中每个东西的对象，包括元素、文本、属性及空格，若人们只是关注原始文档的极小的一部分，创建的那些永远不被使用的对象就造成很大的浪费．DOM解析器必须在代码取得控制权之前读取整个文档，这对大型文档来说会引起显著的延迟．与DOM相比，SAX不需要将整个文档一次加载到内存，也无需为所有节点创建对象，内存消耗小，可解析大于系统内存的文档，不必等待所有要处理的数据就可以立即分析数据．但正因为没有把整个文档存放到内存中，因此SAX不能随机的定位到文档的特定部分，不能实现复杂的搜索，也不可能对数据进行更改．

DOM和SAX两种技术的根本区别在于应用程序得到的XML文档数据的方式不同，DOM模型基于对象，而SAX是事件驱动．选择哪种技术来解析XML文档主要考虑到以下几个因素：

- 应用程序的目的：当需要对数据本身进行修改，并作为XML输出时，应该使用DOM．
- 数据量的大小：当文件比较大时,SAX是比较好的选择．
- 数据的使用：如果仅需要使用文档中的一小部分数据，可以选择SAX方法．但需要引用已经处理过的信息时，DOM方法优于SAX方法．
- 速度、内存的需要：SAX的实现比DOM要快，其内存的需求也大大小于DOM方法．

2.1.3 XML的功能与应用

作为互联网上数据表示和交换标的准，XML已经广泛应用于电子商务、金融、电子政务等诸多领域．XML的功能有很多种，例如XML可以从HTML中分离数据，可以使数据在不兼容的系统中进行交换；普通文本文件可以通过XML实现数据共享、数据存储．总的说来XML的应用模式可分为四类[24]：

（1）数据的交换．XML能够在不同数据源之间充当交换数据的媒介．当用户所需的数据来自不同的数据源，具有各自不同的复杂的格式时，用户与这些数据之间只能通过标准语言XML进行交互．XML的自描述性和可扩展性让它有足够的能力表达各种类型的数据．

（2）客户端文件处理．将大量运算负荷分布在客户端，减轻了服务器的压力．客户可根据自己的需求选择和制作不同的应用程序以处理数据，而服务器只须发出同一个XML文件．应用XML将处理数据的主动权交给了客户，服务器所做的只是尽可能完善、准确地将数据封装进XML文件中．XML的自解释性使客

户端在收到数据的同时也理解数据的逻辑结构与含义，从而使广泛、通用的分布式计算成为可能.

（3）同一数据内容，多种形式呈现．将同一数据以不同的面貌展现给不同的用户．面对不同用户，不同的需求，将相同的数据封装为不同的格式，能够让不同的用户理解或使用相同的数据.

（4）信息的过滤和再利用．网络代理可以对所取得的信息进行编辑、增减从而适应不同用户的需要.

XML 在不断地发展，XML 的应用模式也会出现新的变化．在实际生活中，XML 有着大量的应用实例和产品，例如，Microsoft 的根据 XML 所定义的电子商务通用语言 BizTalk，它有助于企业交换并共享资料；通用汽车公司使用 DataChannel 公司的 XML 伺服器将咨询系统的资料转换成 XML 格式的网页，供使用者浏览；美国财务服务科技委员会发表的银行网上付款系统，其指令讯息和系统反应都采用 XML 格式.

2.2 XML Query 查询模式

XML Query 查询模式是 XML 查询方式中非常重要的一类查询方式，它定义了精致的查询语言，能够让用户借助这些查询语言来描述自己感兴趣的模式，然后将用户的模式交给实际的 XML 数据处理系统处理，最后返回与模式相匹配的结果.

2.2.1 XML 数据查询方法简介

XML Query 查询模式主要是通过使用结构化的查询语言来实现 XML 文档的检索．面向 XML 数据的查询语言有很多种，包括 Lorel[25]，XML-QL[1]，XML-GL[26]，Quilt[27]，XQL，XPath[2]，XQuery[3]等．斯坦福大学 S.Abiteboul 等人设计实现的 Lorel 可以看作是所有后续出现的半结构化查询语言的祖先，之后增加了对 XML 的支持．XML-QL 是由 AT&T 实验室的 T.Deutsch 等人在完成 Strudel 项目时提出的，是第一个嵌入 XML 的查询语言．XML-QL 扩展了 SQL，增加了 CONSTRUCT 语法，用于从查询的返回结果构造 XML 文件，同时它采用元素模式来匹配 XML 数据，而这种元素模式本身也符合 XML 语法，XML-QL 也为 Quilt 的出现提供了许多可借鉴的经验．XML-GL 是由 Politecnico dimilano 的 S.ceri 等人提出的，是一种面向 XML 的图形界面可视化查询语言，它用有向标记图表示图形的 XML 数据，是一种用户界面友好的查询语言．XQL 是一种用于查询和过滤 XML 数据的语言，

它继承了XSLT的许多特征，在XSL基础上增加了布尔操作等，为查询、定位提供了单一的语法形式，XQL非常适用于XML文档段的查找. Quilt由D.Chamberlin（IBM Almaden），J.Robie（Software AG）等人开发，是一种面向XML的查询语言，它借鉴了以前许多查询语言的特点，并试图统一它们. 它的设计目的是实现最强的查询表达能力，能最大限度地结合XML的特点，以及完成不同的数据源之间的集成，Quilt集成融合了前面许多查询语言的特征. XPath和XQuery是XML查询语言的代表. 由W3C创建的XPath，是实现XML数据周游的基本语言，也是XQuery的基础. XQuery也是由W3C组织提出的一种XML查询语言标准，它吸收了多种已有的XML查询语言的优点，已成为现在公认的XML查询语言标准.

2.2.2 XML查询语言XPath

XML查询语言虽然种类繁多，但都是使用基于XPath数据模型下的规则路径表达式来查询XML数据. XPath是XML数据查询语言的核心，用来对XML文档的内容进行定位、检索. 在W3C的规范里描述了XPath1.0[28]. XPath提供了XML文档的树型表示—节点树. 节点分为根节点、元素节点、属性节点、文本节点、命名空间节点、处理指令节点、注释节点七类. 而XPath最主要的构件路径表达式就是通过节点树来跟踪路径，识别出所有被路径表达式检索的节点.

表达式是XPath的主要构件，最重要的表达式是定位路径（location path）表达式，也就是通常所说的路径表达式. 我们给出一个简单的定位表达式示例：

/dblp/book/title

定位路径分为相对定位路径和绝对定位路径两种方式. 每个定位路径表达式由一个或多个定位步（location step）组成，定位步之间用正斜杠（/）分开. 绝对路径以正斜杠（/）开始，从文档树的根节点开始定位；相对路径则直接从某个定位步开始定位路径. XPath中描述定位路径的求值过程的是上下文节点集，就是表达式中给定点确定的当前节点集. 上下文定义为正在处理的当前节点. 一个定位步包括三个部分：一个指定了定位步选择节点与上下文节点之间的树状关系的轴；一个节点测试，它指定定位步选择节点的节点类型或节点名；零个或多个谓词，它使用专有的谓词表达式来进一步筛选定位步选择的节点集合.

定位路径表达式的计算过程是从左到右一次计算每一个定位步. 每个定位步的计算在其上下文节点集的基础上进行，需要对基于它的上下文节点集中的每一个节点进行求值. 绝对定位路径的初始上下文节点集仅包含文档节点，相对定位路径的初始上下文节点集包含当前上下文节点（集），并依赖与表达式使用的位置. 首先要在初始上下文节点集上计算第一个定位步，计算的结果作为下一个定

位步计算的上下文节点集；然后依次对定位路径中的每一个定位步进行计算，最后产生的结果节点集就是该定位路径表达式的结果.

2.2.3 XML查询语言标准XQuery

XQuery规范启动于1998年W3C发起的查询语言波士顿专题讨论会. 但直到2001年2月，查询语言工作组才开始了发布工作，并在随后进行了重要更新，2004年7月又加入了XML Query和XPath完全文本（full-text）的正式工作草案.

XQuery是一个从XML格式的数据源中获取数据的查询语言，由Quilt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. XQuery定义为对XML数据集进行查询的语言，XML数据不仅指XML文档，还指一切看起来像XML的数据，包括关系数据库中的数据. XQuery对于XML数据就像SQL对于关系数据一样. 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.

每个XQuery查询包括一个或多个查询表达式，常用的XQuery语法有：路径表达式、FLWR表达式[29]、序列表达式、算术表达式与布尔表达式、条件表达式、构造器、定量表达式以及函数调用等. 常用的XQuery查询表达式有路径表达式和FLWOR表达式.

路径表达式：路径表达式是一种对XML文档的内容进行定位、检索的表达式，适合用来对树型结构数据进行查询.XQuery中的路径表达式沿用了XPath的语法，使用标签路径从XML文档中选取需要的节点.

FLWOR表达式：FLWR表达式是一种典型的能够完成具有某种实际意义的查询表达式，包含了模式匹配、过滤选择和结果构造3种操作. FLWR表达式是由FOR-LET-WHERE-ORDER BY-TURN 4个关键字定义的字句构成. 它支持迭代，可以把变量绑定到中间结果，当需要对两个或多个文档进行连接和重构数据时常用到这种表达式. 每个FLWOR表达式可以有一个或多个For子句和Let子句、一个可选的Where子句和Order子句、以及一个Return子句. For子句通过将节点绑定到变量来继续循环遍历序列中的每一个节点；Let子句为一个变量赋一个值或一个序列；Return子句定义每个元组要返回的内容；Where子句指定条件对绑定的变量进行过滤，如果其有效布尔值为真，则该元组被保留，并被绑定用于Return子句中，值为假时就废弃该元组；Order子句指定返回结果的排列顺序.

XQuery的查询处理过程[30]：首先，XML Schema导入工作，将查询所需的模式映射到XQuery类型系统.其次，XML数据文档加载，将XML文档映射为XQuery

数据模型的实例．接着对XML查询表达式进行处理，如表达式的解析、分析、优化、求值．最后将查询结果构造成XML文档形式串行化输出．

基于XPath的查询模式的主要内容分为3个层次[31]：线形路径表达式、分支路径表达式和路径树．由此可以看出实现XML Query查询模式的关键在于如何快速地确定任意两个元素之间的结构关系（containment relationship）．大多数的研究集中在索引方法的改进[32]，而一般的索引概括为两类：节点记录类索引方法和结构摘要类索引方法．

2.3 XML关键字查询的相关技术

当互联网上的XML文档数量越来越多，数据量越来越庞大，使用者逐渐扩展到更多的普通用户，XML Query查询语言也就显示出其缺点：大多数的普通用户难以掌握查询语言复杂的语法．而传统的面向HTML的检索是以文档为粒度的查询，检索单元是固定的、完整的文档，并且检索只对信息的内容进行索引，忽略了对被搜寻的概念语义的掌握，因此造成检索结果的查全率及查准率不高．XML关键字查询只需要用户提供与查询相关的关键字，而不需要用户掌握查询语言，查询业XML元素为粒度，仅返回原XML文档的片段，提供内容和结构上的双重检索，使得检索结果更加准确，传输的数据量也就相应减少．因此XML关键字查询也就成为一个非常重要的研究方向．XML关键字查询算法大都是在树型存储模型上展开，以LCA来判断XML文档中的任意两个节点是否语义相关，并利用Dewey编码等编码方式记录节点间的层次关系．

2.3.1 树型结构

一般而言，描述XML的数据模型有图模型和树模型两种．为了便于XML数据的处理，通常在研究中将XML数据映射为树状结构，称作XML树[33]．XML数据模型采用树状结构作为数据的组织形式，每一份XML文档都是一棵有向树，有向树开始于唯一的一个根节点，并向更低级别的树状分支结构扩展．XML文档由若干元素组成，每个元素都是一个文档逻辑组件，而一个元素也可以包含若干的子元素．这种层次结构对应到XML树结构中，节点表示文档中的元素、属性和值，边代表了元素之间的嵌套关系．XML文档用树来表示可以让XML文件看起来层次更加清楚直观，而且对象级的操作比直接对文本的操作要更加容易．这里给出一个简单实例，图2.2(a)是一个XML文档，图2.2(b)是其对应的树状结构．

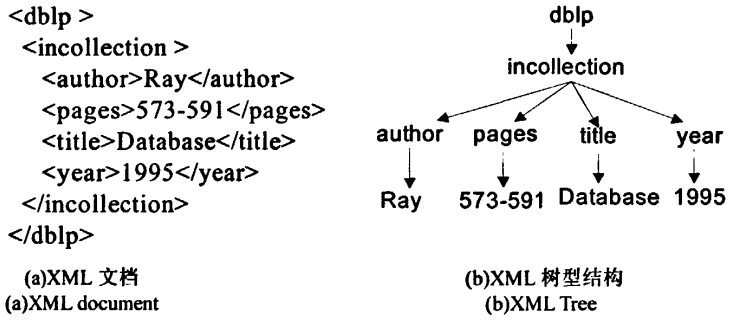


图 2.2 XML 文档及对应 XML 树型结构

Figure 2.2 XML document and XML Tree

2.3.2 Dewey 编码

由于 XML 数据具有半结构化的特点，对 XML 文档进行查询的一个关键就是对元素间结构关系的判断．判断的方法有建立 XML 文档树的路径索引和对 XML 文档树的节点进行编码．最常用的编码方式有区间编码和前缀编码[34]（即 Dewey[35] 编码）．Dewey 编码的简单、易于理解使它成为 XML 关键字查询中的基本技术．Dewey 编码包含了一个节点所在路径上所有节点的 Dewey 码信息，这些信息有利于建立与路径有关的关联关系，而在保持结构关系信息的同时也保存了整个文档的位置编码．图 2.3 是图 2.2(b)所示的 XML 文档树的 Dewey 编码．

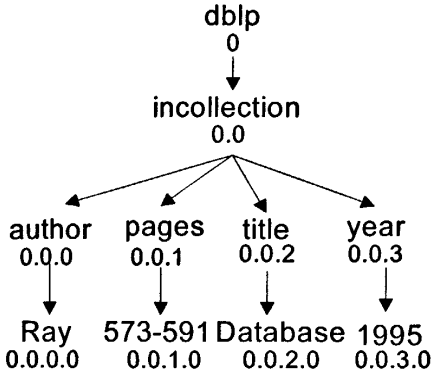


图 2.3 XML 树型结构的 Dewey 编码

Figure 2.3 Dewey of XML Tree

给定 XML 数据的标签有向树 $G=(V_G, E_G, r, A)$，其中：V_G 表示树中节点的集合，E_G 表示树中所有边的集合，r 为标签有向树的根，A 表示所有节点标签的集合，G 中任意节点的 Dewey 编码由下列规则确定：

（1）根节点 r 的 Dewey 编码为‘0’；

（2）在宽度优先遍历 G 的过程中，如果节点 v 是节点 u 的第 i 个孩子节点，那么，节点 v 的 Dewey 码为‘$D(u).i$-1’. 其中 $D(u)$表示节点 u 的 Dewey 码. 例如，某节点的编码为 u，该节点的第一个子节点的编码为 $u.0$.

Dewey 码 $D(u)$中被‘.’分割的整数的个数表示该节点的 Dewey 码长度，也就是该节点在 XML 树中的深度，以 l_u 表示. 取 XML 树中根节点 r 所在的层为 1，那么称 Dewey 码 $D(u)$中与第 i 层节点对应的整数为该 Dewey 码的第 i 层整数，表示为 $u(i)$. 由 1 到 i 层整数组成的 Dewey 码称为该节点 Dewey 码的第 i 层前缀，表示为 $P_u(i)$. Dewey 码的基本运算列举如下：

设 u 和 v 表示两个 Dewey 码，长度分别为 l_u 和 l_v

性质 1（相等关系）如果 $l_u=l_v$，并且从左至右逐层比较各层的整数都相等，则称 v 等于 u，表示为 $u=v$.

性质 2（包含关系）如果 $l_u<l_v$，并且 $P_v(l_u)=u$，则称 v 包含 u.

性质 3（大小判断）

（1）如果 v 包含 u，则称 v 大于 u，表示为 $v>u$；

（2）从左至右顺序比较 u，v 各层的整数，仅当第 i 层时二者的整数不同，如果 $u(i)<v(i)$，则称 v 大于 u.

性质 4（后代 Dewey 码），表示为 *descendent*(u,v).

（1）如果参数 u，v 其中之一为空，则返回另一个非空的 Dewey 码；

（2）如果 v 包含 u，则返回 v.

性质 5（公共前缀），表示为 *lca*(u,v)，计算如下：

（1）如果二者具有包含关系，那么返回被包含的 Dewey 码；

（2）从左至右顺序比较 u，v 各层的整数，仅当第 i 层时二者的整数不同，那么 $P_u(i$-1)即为二者的公共前缀.

根据 Dewey 码的定义，$P_u(i)$对应 u 节点路径上第 i 层节点的 Dewey 码. 可见，给定一个 Dewey 码也就自然地得到了该节点路径上所有节点的 Dewey 码. 各运算的时间复杂度可统一归结为 $O(l_{lca(u,v)})$.

2.3.3 SLCA 问题

XML IR（Information Retrieval）查询的一个重要方法是将传统信息检索中的关键字查询方式应用到XML数据中. 由于XML数据独有的特点，针对XML数据的关键字处理也有了新的特征：查询结果通常不是整个文档，而是包含给定关键字的更小粒度的XML片段；针对所获XML片段的相似性度量必须包含结构相似性的

内容，前者是后者的基础．如何快速获得所有满足关键字组合语义的最紧致XML片段成为一个关键的问题．目前的研究是将此问题转化为SLCA[13]问题．已有的树模型都是以LCA（Lowest Common Ancestor）为基础形成的，认为包含所有关键字的最小数据片断就是一个好的结果．SLCA在LCA 的基础上要求返回的LCA 结点之间没有祖先后代关系，因此，只有部分LCA 才是满足条件的SLCA．

SLCA 问题：在给定初始条件的情况下，即给定了 XML 文档数据的相应的标签有向树 $G=(V_G,E_G,r,A)$ 和一组关键字序列 $W=\{w_1,w_2,...,w_k\}$，SLCA 问题就是确定满足给定关键字的最紧致 XML 片段（XML 子树）的根节点的问题．最紧致 XML 片段（XML 子树）S 应该满足一下条件：

（1）子树必须包含关键字序列 W，也就是说全部的关键字都分布于该子树 S 的叶节点中；

（2）S 中的任意子树都不可能包含全部关键字序列 W.

子树的根节点称为 SLCA 节点．图 2.4 中带下划线的节点表示关键字，虚线所框出的子树是满足给定关键字的最紧致 XML 片段，SLCA 节点就是子树的根节点 article.

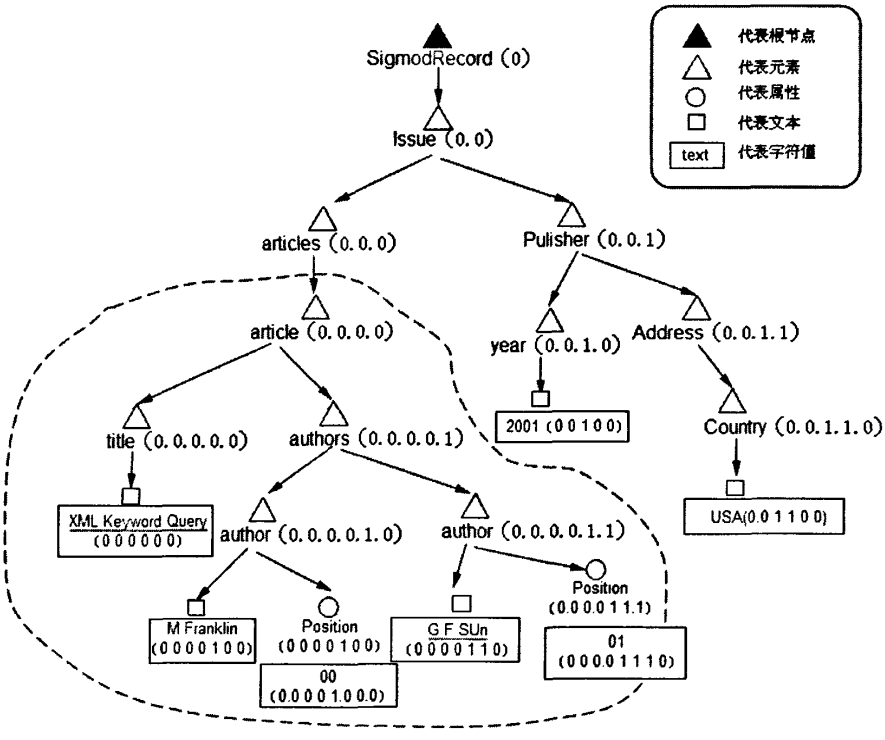


图 2.4 最紧致 XML 片段示例

Figure 2.4 the example of most-compact XML fragment

求解 SLCA 问题的方法概括如下：首先获取分别包含对应关键字的叶节点的集合，然后根据节点的相应 Dewey 编码信息求解 SLCA 节点．可以说 Dewey 编码是 SLCA 的基础，一般来说，SLCA 节点的 Dewey 码是关键字所在节点 Dewey 码的最长公共前缀．例如节点 article 的 Dewey 码 0.0.0.0 是 XML Keyword Query 和 G. F Sun Dewey 编码（0.0.0.0.0.0 和 0.0.0.0.1.1.0）的最长前缀 0.0.0.0．但是当一个 XML 文档中包含有多个对应同一关键字的节点时，计算最合适的 SLCA 节点并不能通过求解包含关键字节点的 Dewey 码的最长公共前缀简单地解决．

SLCA[13]中给出了基于 Dewey 编码计算 SLCA 的 3 种算法，分别是 Indexed Lookup Eager （ILE）算法、Scan Eager（SE）算法和 Stack 算法．由于 ILE 算法被证明具有较优的性能，特别是在各关键字出现频率相差很大的情况下，ILE 算法的搜索速度明显快于前人的算法．下面简要介绍一下 ILE 算法．

ILE 算法的基本思想是：将 XML 数据保存到 B+树结构；之后，借助于 B+树结构以及 Dewey 码的基本运算（大于、小于、子孙码、最长公共前缀等）计算 SLCA 节点．ILE 的基本思路可归结为如下具体步骤：

（1）修改 B+索引树，使它能够支持（keyword，Dewey 码）数据格式，keyword 表示关键字字符串，修改后的 B+树结构记为 DBPT（Dewey B-plus tree），并且支持 Dewey 码的相关运算．新的 B+树结构还应实现如下两个操作：lm（*keyword*，*dewey*），表示获取关键字为“*keyword*”的 Dewey 码集合中小于给定 Dewey 码“*dewey*”的最大 Dewey 码；*rm*（*keyword*，*dewey*），表示获取关键字为“*keyword*”的 Dewey 码集合中大于给定 Dewey 码“*dewey*”的最小 Dewey 码．

预先将 XML 数据分解保存到该数据结构中．插入 B+树的数据形式为（keyword，dewey），这相当于将 XML 中的数据按照关键字（keyword）和关键字在 XML 树中的节点 Dewey 码进行排序．

（2）获取对应给定关键字序列的 Dewey 码集合．每个关键字对应一个包含该关键字的节点的 Dewey 码集合，并将全部集合按照集合内元素的多少从小到大排序，用 S_1，S_2，…，S_k 表示，其中 S_1 对应元素数目最小的 Dewey 码集合．

（3）根据文献[36]，从 S_1，S_2，…，S_k 中求解 SLCA 的过程可通过公式 2.1 说明：

$$slca(S_1,...,S_k) = removeAncestor(\bigcup_{v1 \in S1} slca(\{v_1\},S_2,...,S_k)) \quad ...(2.1)$$

其中 $slca(\{v\},S_2,...,S_k)$ 的计算公式如公式 2.2 所示：

$$slca(\{v\},S_2,...,S_k) = slca(slca(\{v\},S_2,...,S_{k-1}),S_k) \quad ...(2.2)$$

而从某集合中求取对应单个 Dewey 码的 SLCA 节点的过程如公式 2.3 所示：

$$slca(\{v\},S) = \{descendant(lca(v,lm(v,S)),lca(v,(rm(v,S)))\} \quad ...(2.3)$$

从这3个计算公式可以看出，对S_1中的任意个Dewey码都要在DBPT上执行（k-1）次*lm*和*rm*运算；在得到对应*lm*和*rm*运算的两个Dewey码后，ILE还需要运行两次*lca*运算以及一次*descendant*运算才能从中计算得到一个准SLCA节点Dewey码.

进一步考虑实际实现：由于无法确定用户会输入哪些关键字，所以，ILE必须依据全部的XML节点构建DBPT结构. 如果XML数据有N个节点，并取B+树的分叉数为m，根据数据结构的理论，DBPT的高度不会小于$\lfloor \log_m^N \rfloor$，那么，ILE算法总共要进行$(k-1)\times|S_1|\times\lfloor \log_m^N \rfloor$次的*rm*和*lm*运算. 如果考虑到对应关键字的节点数目往往远远小于XML数据中的节点数目N，则已判断出由于ILE必须依赖全部XML节点求解SLCA，因此其计算效率并不高. 所以，提高SLCA节点的求解效率仍然是需要妥善解决的问题.

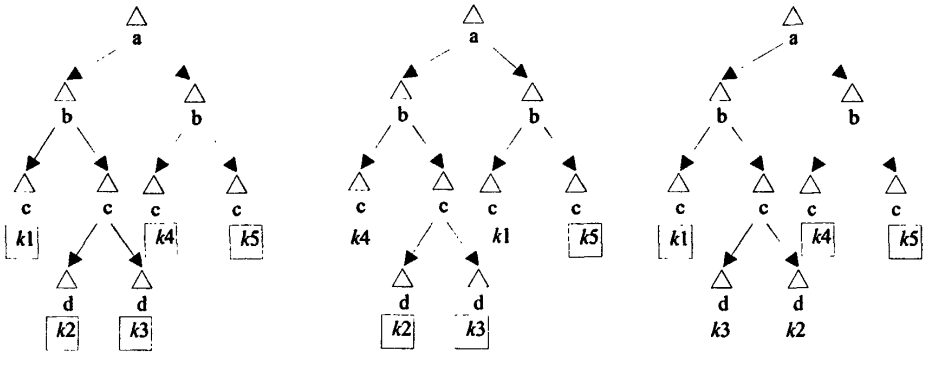
此外，ILE算法还存在如下不足：一是必须修改B+树结构支持必要的Dewey码的操作，*lm*，*rm*操作其实现较为复杂；虽然B+树索引结构在现有的数据库管理系统中普遍使用，但是都不适用于Dewey编码数据. 因此，如果试图考虑在实际应用中利用已有成熟的关系数据库管理系统，付出的代价会较大. 二是ILE算法计算SLCA的过程是“粗暴”的，首先将全部XML数据保存到改造过的B+树. 之后，根据获取对应不同关键字的Dewey码集合，反复调用Dewey码的基本操作和B+树上的匹配操作来求解SLCA节点，操作并不复杂，但效率并不高.

2.3.4 XML片段的相似性度量

XML片段的相似性度量是传统信息检索中十分重要的概念. 在获取了满足给定关键字的最紧致XML片段后，XML关键字查询处理的另一个重要步骤就是如何计算XML片段间相似程度的问题. XML数据的一个重要特点是其结构性，因此针对所获XML片段的相似性度量必须包含结构相似性的内容. 这一问题的研究方法分为两类：一类基于树结构编辑距离的方法；另一类是信息检索中TF*IDF技术的近似方法. 虽然标签有向树结构的编辑距离有着清晰的表述形式，但是其实际计算往往过于复杂，从而研究者更多地集中在传统信息检索技术TF*IDF的近似方法上[32].

TF*IDF的近似方法大致分为4种类型：基于TF*IDF的回归方法[12, 37, 38]、结构化的TF*IDF方法[39-40]、MLP（叶节点路径最大值：maximum leafpath）[41]方法和路径包模型[42]（path bag model）. 基于TF*IDF的回归方法，本质上讲可以看作是一种基于叶节点文本统计特性的多元回归问题，即首先直接利用TF*IDF方法计算XML片段中叶节点的统计值，之后利用XML片段中叶节点到SLCA节点间的

相对位置信息为参数求解对应于 SLCA 节点的影响因子，所以，这种方法并不能有效地捕捉 XML 片段的结构特征，即这类方法通常不能区分图 2.5(a)和图 2.5(b)之间的结构差异，而是会将它们作为同一结构；结构化的 TF*IDF 方法同样吸收了 TF*IDF 技术中“单元（term）”的概念，只是此时的“单元”变成了具有结构信息的“Twig 单元（twig unit）”. 所以，这类方法的核心就是如何分解 XML 数据，以便得到能够体现 XML 数据结构特征的“Twig 单元”向量. 实践证明，这种分化的计算是复杂的[37]；MLP 方法与其他近似方法都不相同，它将 XML 片段的相似性度量的计算分成了两个部分. 其中一部分为引入 MLP 概念，进而构建 MLP 向量以捕捉 XML 片段的结构特征. 所谓 MLP 就是指某节点到以之为根的所有叶节点的最长距离. 另一部分是利用节点标签集合的覆盖程度来捕捉 XML 片段的语义相似性. 但是，基于 MLP 向量的方法在捕捉 XML 片段的结构特征上并不理想，由于图 2.5(a)和图 2.5(b)的 MLP 向量是相同的，所以，它们之间的结构差异也就不能分辨. 路径包模型的核心思想是，采用 XML 片段中标签路径的分布信息作为 XML 片段结构特征的近似. 由于没有考虑到节点间的关联结构，对节点标签比较敏感的，这类方法不能区分图 2.5 中 3 个 XML 片段的结构差异，因为它们具有相同的标签路径. 这些近似方法存在如下不足：既不能很好地分辨 XML 片段结构上的细微差异，部分算法也过多地依赖于 XML 数据的标签或节点相对位置信息，从而不能满足 XML 片段来自不同 XML 数据源的情况.



(a) XML 源片段
(a) Source XMLfragment

(b) 交换 $k1$ 和 $k4$ 后的片段
(b) Exchanging $k1$ and $k4$

(b) 交换 $k2$ 和 $k3$ 后的片段
(b) Exchanging $k2$ and $k3$

图 2.5 标签相同的 3 个 XML 片段

Figure 2.5 Three XML fragments with same labels

由上述内容可知，目前XML片段相似性度量的方法都不太理想，针对XML片段相似性度量的研究还有待进一步深入.

2.3.5 关系数据库存储XML文件

为了充分利用XML的优势，完成XML检索，首先要解决的就是XML的有效存储问题．XML的存储方法可以分为3类，分别利用文件系统、关系数据库和面向对象存储来实现．

文档存储XML容易实现，不需要额外的管理系统和专门的存储机制，但每次访问XML文档时都需要重新调入内存解析，而且在整个查询期间被解析的XML文档都要常驻内存，这些都会严重影响查询XML文档的性能．虽然对XML文档建立外部索引可以在一定程度上解决这些问题，然而索引的维护也是非常困难的.采用面向对象的方法存储XML能够XML文档中的子对象按照其在初始XML文档中的顺序进行存储（比如书的“作者”和“出版社”能够在相邻位置存储），这会明显提高处理复杂查询时构造查询结果的速度；但是这种方法实现起来比较复杂，需要专门的面向对象存储系统，而且现在的面向对象数据库管理系统不便于一般用户使用．现阶段，关系数据库系统是一项十分成熟的技术，有着严密的理论体系和成熟的实现技术，应用广泛，在数据存储和管理方面占据优势．所以XML的存储业主要集中在关系数据库上．因此，充分研究XML与关系型数据库的关系，建立基于关系型数据存储和关键字查询的数据交换模型，利用关系型数据库的成熟技术使XML在数据交换中的优势得以充分发挥，进而促使XML成为系统间数据交换的标准有着十分重要的现实意义．

利用关系型数据库进行XML文档的存储的具体方式主要有三种：(1)将XML文档作为整体存入关系型数据库的LOB型字段中；(2)根据模式文件对XML文档进行解析，之后将解析过的数据存入多个关系表中，这种技术被称为内联法；(3)利用新型数据字段XMLType对XML文档进行存储．还有将其中两者联合使用的混合方法．

对于第一种方法，LOB字段中存储的XML文档一般只进行整体查询，文档内部的查询效率极其低下．第二种方法，又分为基本内联法，共享内联法和综合内联法[43-44]，后两者应用较多．它们都是利用XML模式和关系模式之间的映射关系，将XML文档解析后存入关系表，然后利用SQL语句或XQuery进行节点级或子树级的查询．第三种方法由数据库厂商在其新的商用数据库版本中予以支持．其主要存储形式是在关系型数据库中新建一种数据类型XMLType专门用于XML存储．XML文档在XMLType字段中以树型结构存在，是一种与纯XML数据库存储形式类似的存储方式．

IBM公司有两篇文献[45-46]总结了利用关系模型存储XML的弊端：处理递归和

重复元素困难；无法适应XML模式的经常性变化；关系表间的大量Join操作影响查询效率；利用XQuery查询时存在转换困难的问题. 同时，利用关系型存储XML也有一些优势：大量关系型数据库的成熟技术可以用来进行多用户、索引、事务完整性、安全性等方面的管理；不必进行初始节点集合的查询和比较；不存在有效性判断的问题等.

本文算法中XML利用关系数据库进行存储，选取的方法是内联法，利用模式文件对XML的语义约束解析XML文档，并按照一定的规则存储到相应的关系表中，然后利用SQL语句执行节点级的查询.

2.4本章小结

本章首先介绍了XML的基础理论知识，包括XML的产生发展过程和XML的特点、XML的相关技术，以及XML广泛的应用；紧接着介绍了传统的XML数据查询方法XML Query查询模式，包括XPath和XQuery查询语言；最后重点介绍了XML关键字查询技术，包括XML树形结构、Dewey编码、SLCA问题、XML片段的相似性度量、XML文档检索结果的排序问题，以及利用关系数据库存储XML文件的相关知识.

3 基于语义相关性的 XML 关键字查询

XML 关键字查询简单易用，用户只需要提供相关查询的关键字就能够实现查询. 而且关键字查询以 XML 元素为粒度进行，返回结果仅仅是用户感兴趣且符合关键字包含关系的 XML 文档的一些片段，检索结果更为准确，传输的数据量也因此大大减小. 但是 XML 关键字查询的表达能力有限，少数几个关键词很难表达完整的查询需求，往往导致查询结果中包含大量与用户期待结果不一致的地方，而用户很难从大量结果中识别出自己感兴趣的内容. 如何能够完整地表达出用户的查询意图，怎样快速寻找到结果集的根节点及如何限定结果集的范围成为进一步研究的问题.

3.1 问题描述

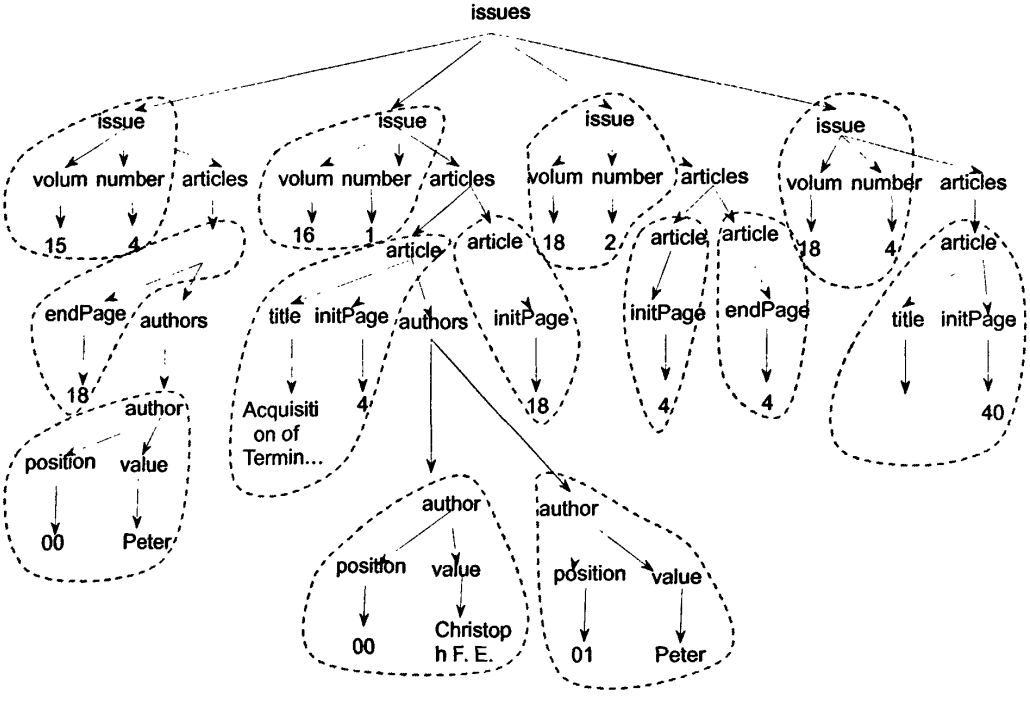


图 3.1 XML 片段

Figure 3.1 XML Fragments

关键字查询是一种从 XML 文档中获得信息的重要方法，也是一种对用户友好的查询 XML 文档的方法. 典型的关键字查询方法都分为两个阶段进行：首先根据关键字找到匹配节点集，然后以包含所有关键字匹配节点的最小子树作为查询结

果．XSearch 方法[11]，MLCA 方法[10]，SLCA 方法[13]和 XRANK 方法[12]等都属于此类方法．虽然有相关研究[47]对上述方法的查询结果进行了改进，但它在获得查询结果子树时仍然采用 SLCA 方法．在[47]中，查询结果子树的根节点被统一称为 VLCA 节点，因此，在本文中，我们将上述方法统一称为 VLCA 方法，将以 VLCA 节点为根节点的查询结果子树称为 VLCA 子树．

对于图 3.1 所示的 XML 文档片段我们给出关键字查询用例 Q1 和 Q2 来分析 VLCA 方法的查询结果．

$Q1$：article volume 18 number 4

$Q2$：article Peter 00

对于 $Q1$，最有可能的查询意图是“查询属于第 18 卷第 4 期的出版物中的所有文章．”对应此查询意图，$Q1$ 中的每个关键字都有其隐含的意义．例如，关键字“18”隐含的意义是指“第 18 卷”，关键字“4”则指“第 4 期”．我们将关键字隐含的意义称为关键字的环境语义，它的正式定义将在 3.3.1 节给出．一个关键字查询的用例所反映的查询意图可以从查询用例中关键字的环境语义加以推断．对于一篇 XML 文档，每个关键字都可以对应多个环境语义．因此，一个关键字查询用例可以反映出多个查询意图．而利用 VLCA 方法返回的 VLCA 子树可能对应于不同的查询意图，所以，并不是所有的 VLCA 子树都是用户需要的．

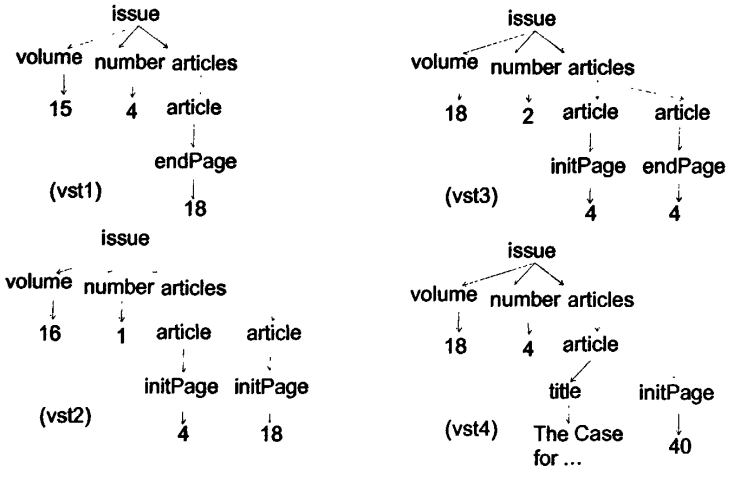


图 3.2 Q1 的查询结果

Figure 3.2 Query results of Q1

观察图 3.2 所示的 $Q1$ 的查询结果．它是由四个 VLCA 子树 *vst*1 到 *vst*4 构成的．但是，依照上文提到的 $Q1$ 的最有可能的查询意图，我们发现只有 *vst*4 与此意图相符．对于 *vst*1，关键字“18”的环境语义是“文章的结束页码为 18”．在 *vst*2

和 *vst3* 中，同样有关键字的环境语义与 *Q1* 最有可能的查询意图不相符．因此，对于 *Q1* 最有可能的查询意图而言，*vst1*，*vst2* 和 *vst3* 都是无关的信息

从这一例子我们得出结论如下：如果我们不能明确查询用例中各关键字的环境语义，查询结果中必将包含许多与用户实际查询意图不符的无关信息．为了解决这一问题，有两个方案可供选择．第一个方案是查询出所有的候选结果，然后根据关键字的环境语义将候选结果分级．第二个方案是在查询候选结果之前先明确关键字的环境语义．第一个方案的处理过程类似于 XSEarch[11]，XRANK[12]等方法，不同之处在于第一个方案用关键字的环境语义代替了 XSEarch[11]，XRANK[12] 等方法结构化的分级因子（如“不同关键字匹配节点间的距离”和“查询项出现频率”等）．但是，我们认为，查询出与用户意图不符的信息是完全没有必要的．因此,本文采用第二个方案，其细节将在 3.3 节中给出．

如果所有关键字的环境语义都与用户的查询意图相匹配，VLCA 方法是否能保证查询结果中不包含无关的信息？观察图 3.3 中 *Q2* 的查询结果．尽管在 *vst5* 和 *vst6* 中，关键字的环境语义都是相同的，但关键字匹配节点间的关系是不同的．在 *vst5* 中，关键字“00”和“Peter”的匹配节点有一个共同的“author”祖先节点，因此，我们认为 *vst5* 是有意义的．而 *vst6* 是无意义的因为其中的关键字匹配节点有不同的“author”祖先节点．

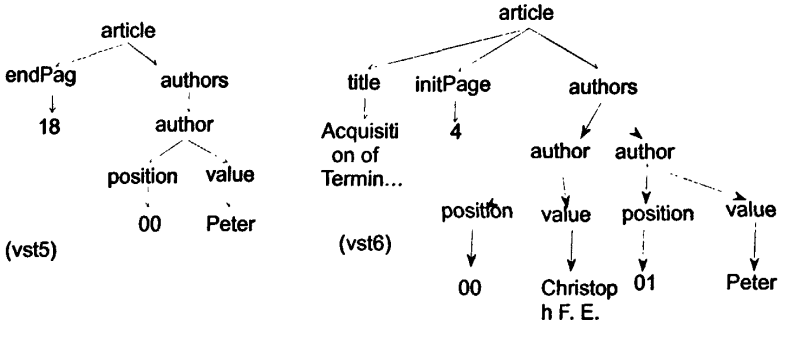


图 3.3 Q2 的查询结果

Figure 3.3 Query results of Q2

从这一例子中，我们可以知道，除了考虑关键字的环境语义，不同关键字匹配节点间的关系也会对查询结果的质量产生影响．在文献[11]和[48]中分别提出了对关键字匹配节点间的关系进行判断和筛选的方法．虽然它们都在一定程度上过滤了无意义的信息，但是，性能问题却成为这些方法的瓶颈，因为它们在对关键字匹配节点间的关系进行判断和筛选时，需要对包含关键字匹配节点的子树中的每个节点都进行运算．本文则提出了主关键字的概念和实体子树的概念来提高关键字匹配节点间有意义判断的有效性和效率．

3.2 语义和实体子树

XML一般以树型模型作为它的数据模型．从XML的语义相关性考虑，首先将介绍实体子树和节点语义三元组的概念．

3.2.1 XML数据模型

本文将XML文档建模为无序的多叉树，即XML树．XML树的的所有内部节点称为名称节点，而它所有的叶子节点称为值节点．XML树的内部节点又可以分成三类[49]：实体节点、连接节点和属性节点．根据XML文档的DTD定义三类节点如下：

（1）实体节点

实体节点在对应的DTD中是一个*型节点，即能够重复出现在父节点中，并且可以拥有实体、连接、属性等类型的子孙节点．

（2）属性节点

属性节点在对应的DTD中不是一个*型节点，在一个实体节点中只能一次，有一个值节点作为它的子孙节点．

（3）连接节点

既不属于实体节点，也不属于属性节点的节点称为连接节点．一个连接节点仅仅能够包含实体类型的子节点．

DTD中，具有一对一操作{‘?’, ‘,’}的节点为属性节点，而具有一对多操作{‘*’, ‘+’}的节点为实体节点或连接节点．连接节点只能拥有一种类型的实体节点．

在下文中，我们用T或$T(r)$表示XML树，其中，r表示T的根节点，T的节点集合用$N(T)$表示，它是由$NE(T)$（实体节点集合），$NA(T)$（属性节点集合），$NC(T)$（连接节点集合）和$NV(T)$（值节点集合）组成的．

每个$n \in N(T)$都可以用一个Dewey编码表示，$nodeValue(n)$表示n的标签名或值．当n是名称节点时，$nodeValue(n)$代表了n的标签名；而当n是值节点时$nodeValue(n)$代表了n的值．

3.2.2 实体子树

关系数据库中的E-R模型具有较强的语义表达能力，能够方便、直接地表达应用中的各种语义知识，也比较简单、易于用户理解，是抽象和描述现实世界的有力工具．观察XML树中实体节点和属性节点，它们的作用类似于E-R模型中的实体

和属性[50]．由此可知，实体节点及其子孙属性节点，值节点能够组成语义相关的信息单元—实体子树，其正式定义如下：

定义1：实体子树（Entity Sub Tree，EST）．如果一个XML子树以实体节点为根，并且除了根节点外，只包含属性节点和值节点，则称此XML子树为实体子树，可表示为est或est(r)．根节点r的标签名称为est(r)的类型．

在图3.1中所示的XML树，虚线框中的节点构成了不同的实体子树．

如果一棵实体子树没有子孙实体子树，此实体子树就被称为叶子实体子树，否则，这棵实体子树称为内部实体子树．

3.2.3 节点的语义三元组

由上可知，每一个XML节点都对应一个语义三元组，其定义如下：

定义2：节点语义三元组（Node Semantic Triples）．对于一棵XML树T(r)和其节点n，pe(*n*)、ces(*n*)和pa(*n*)分别代表n的父实体节点，子实体节点和父属性节点．节点n的语义三元组nsd(*n*)定义如公式3.1所示：

$$nsd(n)=\begin{cases}(nodeValue(n),null,nodeValue(n)) & n\in NE(T)\\ (nodeValue(pe(n)),nodeValue(n),nodeValue(n)) & n\in NA(T)\\ (nodeValue(pe(n)),nodeValue(pa(n)),nodeValue(n)) & n\in NV(T)\\ (nodeValue(ces(n)),"ancestor",nodeValue(n)) & n\in NC(T)\end{cases}\quad\cdots(3.1)$$

nsd(*n*)中的第一个和第二个元素称为n的本位实体和本位属性，它们标识了与*n*有最近语义关系的实体节点和属性节点，而最后一个元素标识了*n*的标签名或值．

在语义三元组的定义中，对连接节点进行了特别的处理．为了简化问题，我们将XML树中的连接节点视为它们的子实体节点的名为“ancestor”的属性的值．图3.4(a)和3.4(b)分别表示出了原始的XML片段和将连接节点转化为“ancestor”属性值后的XML片段．

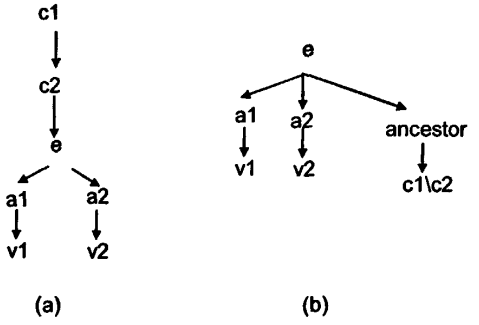


图 3.4 Ancestor 属性

Figure 3.4 Ancestor attribute

图3.4中，nsd(c1)=nsd(c2)=(e，“ancestor”)．因此，XML树中的所有节点都可以看作是实体子树的内部节点．

3.3 语义相关性

每个节点都具有语义，而实体树是一个语义相关的信息单元，下面主要介绍了关键字的环境语义和怎样判断关键字匹配节点之间是语义相关的．

3.3.1 关键字的环境语义

一般情况下，用户给出的一个关键字查询用例，总是表达了自己的一个明确且唯一的查询意图．然而在XML文档中，由于关键字的环境语义的不同，同一个查询用例在大多数情况下可能会被理解为多个不同查询意图，因此查询的效率降低，查询的结果也与用户期望的有所差距．我们先给出关键字环境语义的定义，然后对查询意图与关键字环境语义的关系进行分析．

定义3：关键字环境语义（Keyword Context Semantic）．对于一个关键字k，n是k在XML树T上的一个匹配节点．n的语义三元组nsd(n)即为k的环境语义，用kcs(k)表示．n的本位实体和本位属性也被称为k的本位实体和本位属性．

很显然，对于一棵XML树，一个查询关键字可以对应多个环境语义．对于一个关键字k，其对应的所有环境语义可以组成一个环境语义集合，用KCS（k）表示．对于图3.1所示的XML片段，KCS(18)={(article，endPage，18)，(article，initPage，18)，(issue，volume，18)}．

不同的关键字环境语义，可以反映出不同的用户查询意图，也导致了查询结果的多样性．例如，关键字“4”和“18”的环境语义如果分别是kcs(4)=(issue，

number，4)，kcs(18)=(article，endPage，18)，那么，$Q1$的查询意图就可以解释为“查询所有属于第4期出版物并且结束页为18的文章信息及其所在的出版物的卷号”，VLCA方法查询所得的图3.2所示的vst1与此查询意图相对应．如果关键字环境语义变为kcs(4)=(article，initPage，4)，kcs(18)=(article，initPage，18)，$Q1$的查询意图也相应变成“查询所有起始页为4或18的文章信息及其所在的出版物的卷号和册号”，VLCA方法查询所得的图3.2所示的vst2与此查询意图相对应．我们已经说过，通常情况下，当用户给出一个关键字查询用例时，他只想表明一种明确的查询意图．这就意味着，查询用例中的每个关键字都应当明确地与一个环境语义相对应，并且，不是所有的VLCA子树都应作为查询结果被返回．为了达到这一目的，我们将常规的关键字查询用例改良成为扩展语义的关键字查询用例，在查询关键字匹配节点之前，首先明确用户的查询意图．

3.3.2 扩展语义的关键字查询用例

在完成XML文档解析的过程后，用关系数据库的二维关系表R来存储所有XML节点的语义三元组．表的元素就是XML节点的语义三元组．对于一个关键字查询用例，所有可能的关键字环境语义都可以在查询实际的XML数据之前从关系表中得到．常规的关键字查询用例也将演化成为相应的语义扩展的关键字查询用例：

定义4：语义扩展的关键字查询用例（Semantic Extended Keyword Query Case，SEKQC）．对于一棵XML树T和一个关键字查询用例Q，如果Q在T中对应多个环境语义（语义三元组）的所有关键字都与R中的一个对应的环境语义相关联，则Q就演化成为语义扩展的关键字查询用例．

例如，对于图3.1所示的XML文档片段，$Q1$对应的部分SEKQC在表3.1中列出．

表3.1 关键字查询用例和对应的SEKQCs

Table3.1 Keyword query case and the corresponding SEKQCs

Keyword query case	$Q1$：article volume 18 number 4
SEKQCs	$Q11$：[(article)(volume)(issue, volume, 18)(number)(issue, number, 4)] $Q12$：[(article)(volume)(issue, volume, 18)(number)(article, initPage, 4)] $Q13$：[(article)(volume)(issue, volume, 18)(number)(article, endPage, 4)] ……

对应于同一关键字查询用例的所有SEKQC是按照它们的语义跨度进行排列的．

定义5：语义跨度（Semantic Span）．对于一个SEKQC，其所包含的不同关键

字的个数称为Q的语义跨度，表示为ssp(Q).

举例来说，在表1中，ssp(Q12)=6，ssp(Q13)=7.

语义跨度越小，更多的关键字就能被组合在一起成为短语．在XML树上，短语对应于连续的节点片段．因此，较小的语义跨度意味着更紧凑的结果单元，这更有可能符合用户的查询意图.

每个SEKQC都对应唯一的查询意图．一旦用户选择了一个SEKQC，关键字查询就将在唯一的用户查询意图下展开.

3.3.3 SEKQC 中的主关键字

对于一个常规的关键字查询用例，所有的关键字都只能独立地去匹配XML节点，因为不同关键字间的关系在查得关键字匹配节点之前无法推断．但是，在SEKQC中，关键字与环境语义相关联，这有助于将关键字分组并进一步找到主关键字．具体地说，SEKQC中具有相同的本位实体和本位属性的关键字可以被分为一组.

定义6：主关键字．在SEKQC的一个关键字组中，匹配最底层XML节点的关键字，被称为关键字组的主关键字，组中的其它关键字称为从关键字.

例如，Q11中的关键字可被分为三组：(article)，(volume 18)和(number 4)．三个关键字组的主关键字分别是“article”，“18”和“4”.

在许多情况下，主关键字对应于XML树中的值节点．因此，较之同一关键字组中的其它关键字，主关键字通常能匹配较少的XML节点．根据这一特点，查询算法可以只利用SEKQC关键字组中的主关键字而忽略从关键字．由于每一关键字都与明确的环境语义相关联，因此，这样做既能保证查询语义不发生改变，又能减少I/O操作，提高查询效率.

3.3.4 关键字匹配节点的语义相关性

基于上述讨论，我们知道如果一个关键字查询利用了 SEKQC，那么与关键字的环境语义不相符的关键字匹配节点将不会被选择. 但是，我们在 3.1 节中讨论过，对于与环境语义相符的关键字匹配节点，如果利用 VLCA 方法来进行后续的处理，我们仍然不能保证产生的 VLCA 子树都是有意义的.

因此，我们提出语义相关单元的概念作为查询结果的基本信息单元，而不是采用 VLCA 子树.

定义 7：语义相关单元（Semantically Relevant Unit，SRU）．对于一棵 XML

树 T 和一个关键字查询用例 Q，$ST(LESTS, IESTS)$ 是 T 的一棵子树，其中，*LESTS* 和 *IESTS* 分别是 *ST* 的叶子实体子树集合和内部实体子树集合．当 *ST* 满足下列四个条件时，我们称 *ST* 是一个语义相关单元：

（1）Q 中每个关键字至少有一个匹配节点在 ST 中；

（2）*IESTS* 中的任意两个实体子树都是不同类型的；

（3）*LESTS* 中的任意一个实体子树都至少要包含 Q 中一个关键字的匹配节点；

（4）在 *LESTS* 中，如果两个实体子树的类型相同，则对应的关键字匹配节点的类型及语义三元组也都是相同的．

关键字匹配节点间的语义相关性决定了每个语义相关单元中的关键字匹配节点的组合是有意义的．语义相关单元中关键字匹配节点的语义相关性可分为四类：内部语义相关，层次语义相关，同宗语义相关和同类语义相关．下面给出它们的定义：

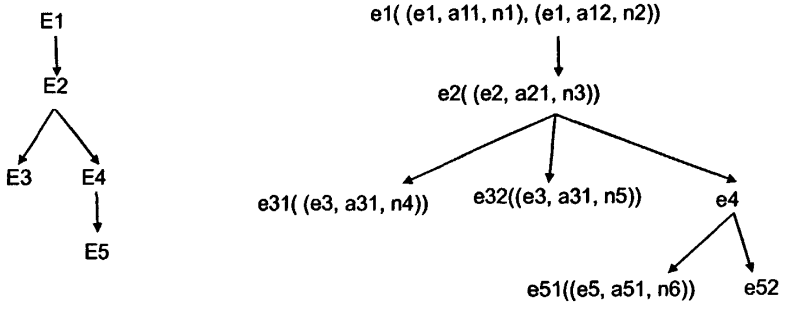
在定义 8 到定义 11 中，我们用 *n*1 和 *n*2 表示被比较的两个关键字匹配节点．*n*1 和 *n*2 所在的实体子树用 *est*1 和 *est*2 表示．

定义 8：内部语义相关．如果 *est*1 与 *est*2 相同，我们称 *n*1 和 *n*2 是内部语义相关的．即 *n*1 和 *n*2 位于同一实体树内．

定义 9：层次语义相关．如果 *est*1 与 *est*2 之间有祖先与后代的关系，我们称 *n*1 和 *n*2 是层次语义相关的．

定义 10：同宗语义相关．虽然 *est*1 与 *est*2 之间没有祖先与后代的关系，但它们的最低公共祖先实体子树是 *aest*．从 *est*1，*est*2 到 *aest* 的路径上，如果没有相同类型的实体子树，我们称 *n*1 和 *n*2 是同宗语义相关的．

定义 11：同类语义相关．如果 *est*1 与 *est*2 是两棵相同类型的实体子树，同时，*n*1 和 *n*2 具有相同的类型和语义三元组，我们称 *n*1 和 *n*2 是同类语义相关的．



(a) XML模式
(a) XML schema

(b) XML实例
(b) XML instance

图 3.5 XML 模式及实例

Figure3.5 XML schema and instance

观察图 3.4 中的例子，当我们把连接节点看作子实体节点的“ancestor”属性的值，XML 文档的模式树的形式就类似于图 3.5(a)．图中的每个节点都代表一棵实体子树．图 3.5(b)为符合图 3.5(a)中的模式树的 XML 文档片段．根据定义 7，此文档片段是一个 SRU，表 3.2 列出了所有关键字匹配节点的语义相关关系．其中，*i*，*h*，*co* 和 *cl* 分别代表内部语义相关性，层次语义相关性，同宗语义相关性和同类语义相关性．

表 3.2 图 3.4(b)中关键字匹配节点的语义相关性

Table3.2 Semantic relevance of keyword matching nodes in Figure 3.4(b)

	n1	n2	n3	n4	n5	n6
n1		i	h	h	h	h
n2	i		h	h	h	h
n3	h	h		h	h	h
n4	h	h	h		co	cl
n5	h	h	h	co		cl
n6	h	h	h	cl	cl	

3.4 查询算法描述

本文的核心算法是语义相关单元查找算法(Finding Semantically Relevant Unit，FSRU)，基于上述预备知识，我们给出 FSRU 的基本描述．

步骤 1：根据用户提供的关键字查询用例，寻找每个关键字的环境语义，并组合出全部可能的 SEKQC．

步骤 2：由用户选择符合其查询意图确定的一个 SEKQC，并把这个确定的 SEKQC 返回给算法．

步骤 3：将确定的 SEKQC 中的关键字按照语义三元组中的本位实体和本位属性进行分组．也就是语义三元组中的本位实体和本位属性同时相同的关键字分为一组，记为 eaGroup 级分组．

步骤 4：对每一个 eaGroup 级分组，去除从关键字，保留主关键字．

步骤 5：在步骤 4 的基础上，对同一个 eaGroup 组中的关键字，按照本位属性的不同再一次进行分组，记为 aGroup 级组．

步骤 6：查找关键字的匹配节点．

步骤 7：对同一 aGroup 级分组中的关键字对应的匹配节点进行并运算，也就是将这些节点合并为一个匹配节点集合 aSet．

步骤 8：对同一 eGroup 级分组中，各个 aSet 匹配节点集合进行交运算，保留各个 aSet 集合都有的匹配节点，将其作为一个集合 eSet．

步骤 7 和步骤 8 两步首先对相同类型实体子树中的关键字匹配节点进行比较

和运算，以便发现具有内部语义相关性或同类语义相关性的实体子树.

步骤 9：对于不同类型的实体子树，将关键字按照父子关系分组. 具有父子关系的 eSet 匹配节点集合将作为 fcSet 集合.

步骤 10：同一 fcSet 集合内的各个 eSet 集合做父子运算，仅保留父节点. 这些父节点更新了 fcSet 集合.

步骤 11：对上述不同 fcSet 集合的各个匹配节点进行组间运算，保留具有同宗语义相关的关键字匹配节点.

步骤 10 和步骤 11 对不同类型的实体子树中的关键字匹配节点进行比较以发现具有层次语义相关和同宗语义相关的关键字匹配节点.

最后产生的查询结果就是按照 FSRU 算法得到的最终查询结果.

3.5 基于语义相关性的 XML 关键字查询的实现

基于语义相关性的 XML 关键字查询的实现包括了 XML 文档的解析和存储，数据结构的设计和具体算法的流程示意图. 本节最后给出一个小的示例用来演示具体的实现过程.

3.5.1 查询示例系统结构

查询示例系统由输入输出界面、查询表达式处理模块、XML 文档处理模块、关键字查询模块、数据库存储模块五个部分组成.

（1）查询表达式处理模块主要负责将查询表达式中的关键字进行提取，并对连接节点进行判断和移除，同时对表达式中的关键字的有效性进行判断，如是否是空关键字，是否有重复的关键字等一些基础的，不涉及算法的情况.

（2）XML 文档处理模块，主要是对 XML 文件的解析过程.

（3）查询模块是整个示例系统的主要部分，通过基于语义相关性的关键字查询算法实现根据用户提供的查询关键字，搜索 XML 文档，将符合用户需求的 XML 片段返回给用户. 这里简化为片段 XML 子树的根节点.

（4）数据库模块用来存储 XML 数据，采用内联法来将 XML 映射到关系数据库中. 数模库模块主要是关系表的设计.

（5）简单的用户界面可以方便用户的使用.

整体结构如图 3.6 所示.

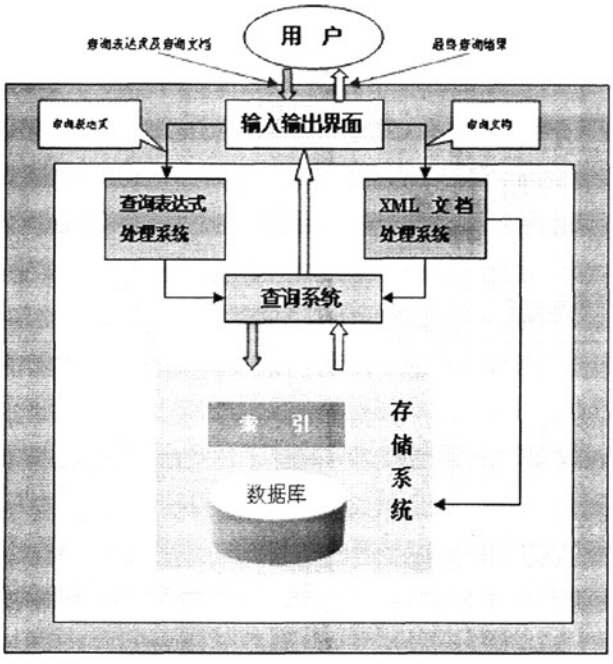


图 3.6　整体结构

Figure3.6 whole structure

以下主要介绍 XML 文档处理模块、关键字查询模块、数据库存储模块三个主要模块.

3.5.2　XML 文件的解析

这一模块的主要功能是对 XML 文档进行解析，获得所需结构的数据. 由于数据量比较大，而 SAX 技术需要消耗的内存较小，因此这里采用 SAX 方法作为 XML 文档的解析方法.

对实体、属性、连接节点、值节点的判断则是根据 XML 文件的 DTD 文档来实现. 没有出现在 DTD 文件中的均视为值节点；而拥有多对一操作{'*', '+'}的节点是实体节点或连接节点，当一个元素只包含相同的元素时，这个节点被人为是连接节点，否则认为它是实体节点，例如 DTD 中有<!ELEMENT articles (article)*>，则节点 articles 是连接节点，而<!ELEMENT article (title,authors)>的对应节点 article 是实体节点；在一个实体节点的子节点中仅能出现一次，同时又且仅有一个值节点作为子节点的节点是属性节点，如<!ELEMENT title (#PCDATA)>，title 只能在 article 中出现一次，而且有字符串作为子节点，因此 title 是一个属性节点，DTD 元素的属性同时也做为属性节点. 为了简化算法的实现，解析时过滤了连接节点，

同时也忽略了自嵌套的节点.

解析后的每个节点对应的结构内容为：节点的 Dewey 编码、节点的标签、节点所属本位属性的标签、节点所属本位实体的标签和 Dewey 编码. 既有 XML 节点的 Dewey 编码，也有其语义三元组. 解析后的内容暂存在中间文本文件中，为数据库载入数据做准备. 为了节省存储空间，可以在解析文件的同时将数据存入数据库中，在以后的工作中将进一步改进.

3.5.3 关系存储表的设计

主要的关系表有四个：eanode、entity_tree、eaTable 以及 valueTable. eanode 和 entity_tree 表的基本结构如表 3.3(a)和表 3.3(b)所示，而表 eaTable 和 valueTable 的结构相同，如表 3.3(c)所示.

表 3.3 关系表的结构

Table 3.3 The structure of relational tables

node	type
article	e
author	e
mdate	a
...	...

(a)

label	dewey
article	0.0
author	0.0.0
editor	0.0.1
...	...

(b)

id	node_dewey	node_label	attribute_label	entity_label	entity_dewey
...	...	...	...	...	...
14665	0.600	article	null	article	0.600
14690	0.601	article	null	article	0.601
...	...	...	...	...	...
14711	0.601.10	volume	null	volume	0.601.10
14712	0.601.10.0	1	null	volume	0.601.10

(c)

eanode 表主要用来存储节点的类型，但只对实体节点和属性节点进行存储. 由于实体节点和属性节点的个数比较少，XML 文档的 DTD 一般简单明了，因此根据 DTD 文件把实体节点和属性节点单独抽取出来比较容易. 而在 XML 文档中重复出现的几率很大，根据 eanode 表很容易判断一个关键字的类型，不需要查询整个 XML 节点的表. eanode 表是查询的辅助表，能够提高查询的效率.

表 entity_tree 同样是一个辅助表. 同 eanode 表一样，entity_tree 表的数据也是根据 XML 文档相应的 DTD 文件得到的. 这个表主要用来存储实体节点之间的关系. 此处的 dewey 并不是 XML 文档的 dewey 编码，而是专门针对实体节点进行的编码. 编码能够反映任意两个实体之间是否有父子关系，或共同的祖先是哪个，

以及两个实体之间的层次数等信息.

表 eaTable 用来存储 XML 文档的属性和实体节点，而表 valueTable 则专门存储值节点. 最初的设计是将所有的 XML 节点信息均放在一张表中，但通过 eanode 表，我们很容易判断出一个节点的类型，那么在查询时，表中的很多信息相对来说是没有用的. 将一张表拆为 eaTable 和 valueTable 两张表就大大减少了查询的数据规模，也就减少了查询的时间，提高了查询的效率. 表中的属性 id 主要是为了节点的排序设计的. 顺序解析 XML 文档时，标号在前的节点的 Dewey 编码一定其后节点的 Dewey 编码小，算法中有些地方需要对记录进行排序，此时 id 属性将是一个很好的设置.

3.5.4 查询算法的具体实现

我们在 3.4 节对基于语义相关性的 XML 关键字查询算法进行了简单描述，下面来具体说明算法的实现过程.

（1）主要数据结构的设计

在查询算法实现中最多用到的就是链表，以下主要介绍这些链表的作用以及一些链表之间的关系.

首先，链表 enode 和 anode 分别用来存储关系表 eanode 中的实体节点标签和属性节点标签，而链表 entitytreenode 和 entitytreedewey 则分别存储表 entity_tree 中的标签和编码. 将两个关系表暂存在内存中，这是因为两个表的数据量都很小，装入内存后可以很方便地进行多次存取，从而减少访问数据的次数.

其次，链表 smtSet 用来存储每一个关键字的全部的 SEKQC，一个关键字的一组 SEKQC 作为 smtSet 的一个单元来存储. 而 keysSmtG 是用来存储关键字的确定的 SEKQC. elGroup 可以用来存放不同本位实体和属性实体的关键字组. nodeSet_U 的存储单元存放的是与 elGroup 存储单元对应的关键字组的匹配节点集合. eGroup 则是存放不同本位实体的关键字组. sameEntityGroup 存放 eGroup 的每个单元中相应的关键字匹配节点的 Dewey 编码. el 仅仅存放 eGroup 的每个单元对应的本位实体标签. 链表 keysSmtG、elGroup、nodeSet_U、eGroup、sameEntityGroup、el 之间的关系如图 3.7 所示.

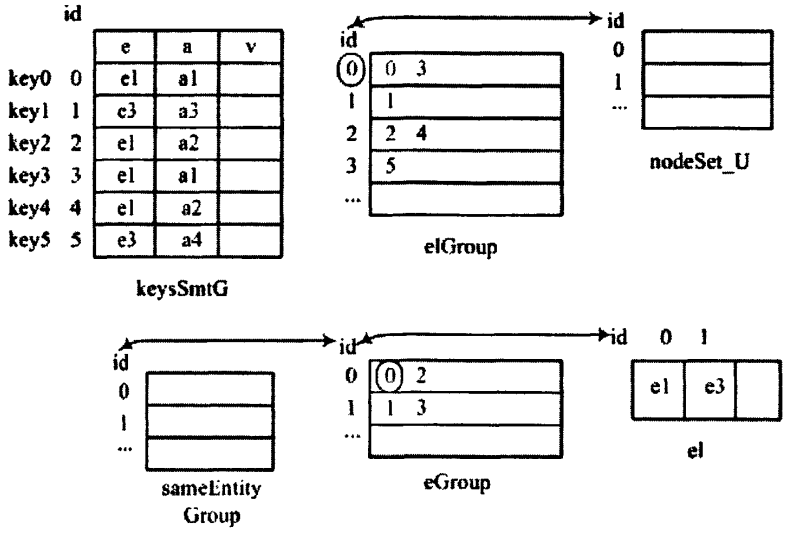


图 3.7 链表之间的关系

Figure3.7 The relationship of Lists

在图 3.7 中，每一个表的一行代表了一个存储单元，id 是存储单元的编号，从图中我们可以看出：keysSmtG 的编号按本位实体和本位属性相同分组并存储在 elGroup 中，如红色的数字是对应的．elGroup 的编号按相同实体进行分类并存储在 eGroup 中，可以从用圆圈起的数字相对应看出来．而 elGroup 与 nodeSet_U 的编号是一一对应的，同时 sameEntity、eGroup、el 之间，编号是相对应的．

除了这些链表，还有一些二维或一维的字符串数组．如 InStr 是一个二维字符串数组，InStr[0]用来存放输入的关键字，而 InStr[1]则存储关键字对应的节点类型（通过查询 enode 和 anode 来实现）．

（2）查询过程的流程图

根据 3.4 节的算法描述，我们可以得到整体的算法实现流程图 3.8.

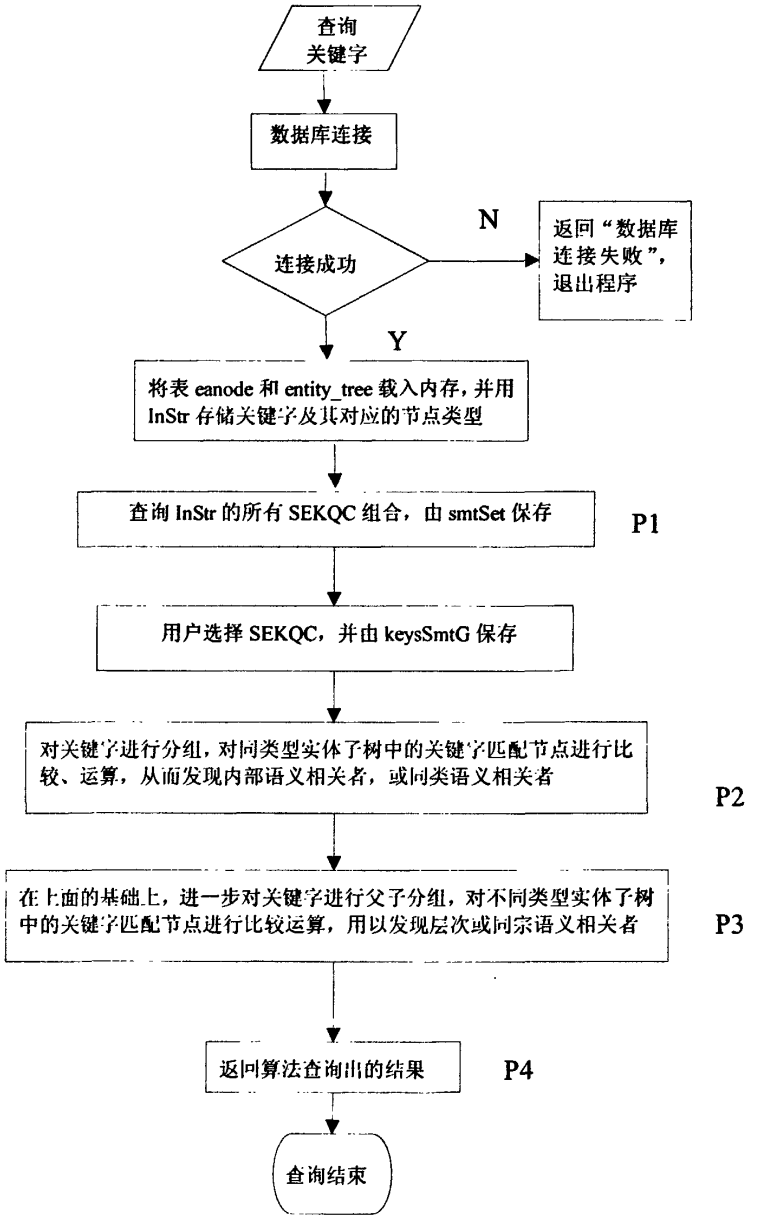


图 3.8 算法的整体流程图

Figure 3.8 general flow chart of the algorithm

对于图 3.8 中的模块 P1、P2、P3 我们分别在图 3.9，图 3.10，图 3.11 中给出其具体过程. 这里用(e，a，v)来简记关键字匹配节点的语义三元组. e 代表实体子树的根节点，a 代表节点的属性，v 代表节点的值.

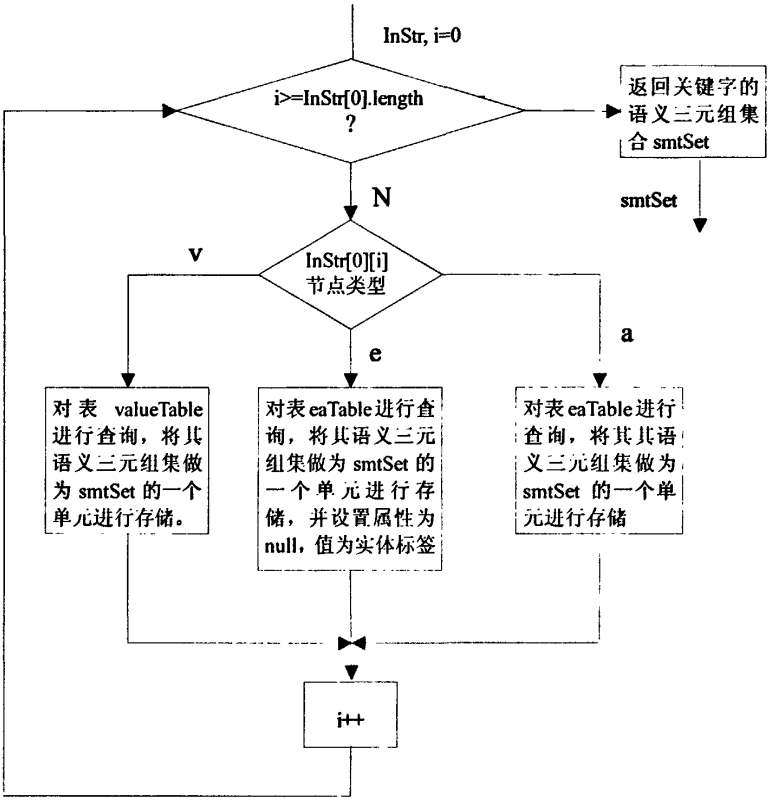


图 3.9 P1 的流程图

Figure 3.9 flow chart of module P1

图 3.10 描述了 P2 模块的具体工作内容，P21 将 keysSmtG 中语义三元组(e，a，v)中 e、a 相同的元素的下标序号抽取为一组，然后作为 elGroup 的一个元素，相当于将提供的关键字按照同类实体子树进行分组．主要实现如下：

```
……
for（int i=0;i<keysSmtG.size()-1;i++）
   {
      for（j=i+1;j<keysSmtG.size();j++）
       if（haven=false）//haven 用来标记关键字 j 是否已被记录到 elGroup
         if（关键字 i，j 的 e、a 相同）
              将 j 记录到 i 对应的 elGroup 单元中，并标记 j 的 haven 为 true;
   }
……
```

在 P22 模块中消除从关键字时，只需要判断 elGroup 的每一个元素中中记载的编号对应的 keysSmtG 的元素是否是属性节点或实体节点，如果是则删除其在

elGroup 中的记录．仅保留主关键字有利于查询效率的提高．

在 P23 模块中，基于 P22 的 elGroup，寻找 elGroup 中具有相同实体 e 标签的 elGroup 存储单元，并将他们的编号作为 eGroup 的一个存储单元．对实体标签相等的判断也利用了 keysSmtG 链表．P22 把可能具有内部相关性的匹配节点对应的关键字分为一组，也就是说同一组的关键字的匹配节点可能同属于一个实体子树．

模块 P24、P25 的交、并运算的定义同集合运算中的概念相同．运算主要运用了节点 Dewey 编码的查找与比较．例如，函数 deweyComp()用来比较 Dewey 编码，函数 binSearch()可以在一个 Dewey 数组中查找一个 Dewey 编码．对于多个 Dewey 编码集合，每个集合的数据量可能很多，因此采用折半查找的方法，可以减少查找节点的时间．

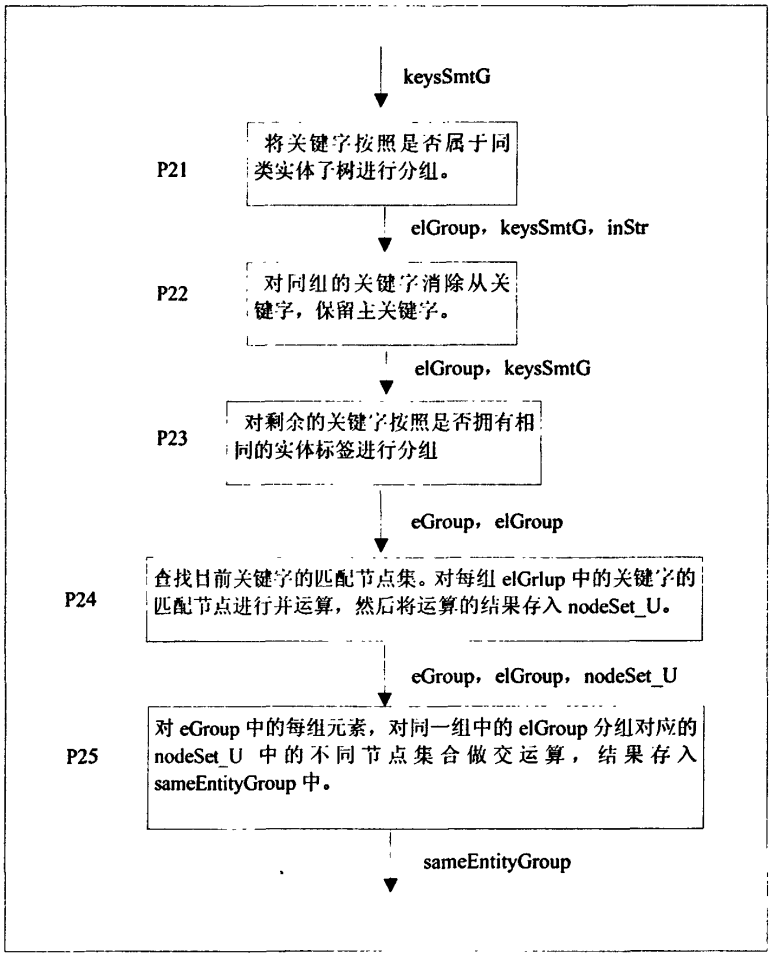


图 3.10 P2 的流程图

Figure 3.10 flow chart of module P2

在实现这几个模块时要把 keysSmtG、elGroup、eGrou、nodeSet_U 这四个链表的关系整理清楚，相关内容在前面对数据结构的描述中给出了具体介绍．同时也要注意 P21 模块中数据的初始状态．根据分组的规则，在 P2 中不同分组的关键字对应的匹配节点可能具有内部语义相关性，也可能具有同类语义相关性．

图 3.11 描述了模块 P3 的详细工作流程．

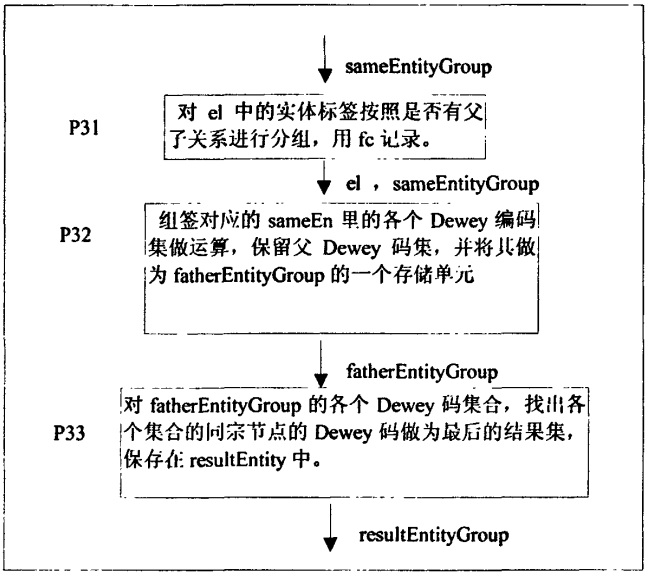


图 3.11 P3 的流程图

Figure 3.11 flow chart of module P3

在图 3.11 中，P31 中的 fc 定义为 int[][],fc[0]存放的是对应 el 的下标，fc[1]用来标记父子组，当 fc[1][i]=0 时，表明 fc[0][i]代表的子节点，否则，fc[1][i]代表了 fc[0][i]是第几组中的节点．我们给出以下例子：

表 3.4 fc 的内容

Table 3.4 contents of fc

下标	0	1	2	3	4	5	6	7
fc[0]	3	7	1	5	4	6	2	0
fc[1]	1	0	0	2	0	0	3	0

从表 3.4，我们可以清楚地看到有三组父子组合，组 1：el[3]、el[7]、el[1]，组 2：el[5]、el[4]、el[6]和组 3：el[2]、el[0]；P32 就是在组内的 el[k1]，el[k2]，el[kn]对应的 sameEntityGroup[k1],sameEntityGroup[k2]，sameGroup[kn]之间寻找祖先 Dewey 码，这些 Dewey 码就是 fatherEntityGroup[0]的内容，同理，我们可以得到 fatherEntityGroup[1]，fatherEntityGroup[2]．P33 从 P32 的结果集中寻找其共同祖先

的 Dewey 编码，将此做为查询结果返回．这里利用 entity_tree 来判断两个实体节点是否是有意义的同宗相关．

3.6 算法实例

下面我们针对图 3.1 所示的 XML 文档及 Q2 给出算法的一个小的实例．

首先对图 3.1 所示的 XML 文档进行解析并存入数据库中，解析后的数据格式大致如下：

('0.0' , 'issue' , 'null' , 'issue' , '0.0')

('0.0.0' , 'volum' , 'volum' , 'issue' , '0.0')

('0.0.0.0' , '15' , 'volum' , 'issue' , '0.0')

('0.0.1' , 'number' , 'number' , 'issue' , '0.0')

('0.0.1.0' , '4' , 'number' , 'issue' , '0.0')

('0.0.2' , 'article' , 'null' , 'article' , '0.0.2')

('0.0.2.0' , 'endPage' , 'endPage' , 'article' , '0.0.2')

('0.0.2.0.0' , '18' , 'endPage' , 'article' , '0.0.2')

('0.0.2.1' , 'author' , 'null' , 'author' , '0.0.2.1')

('0.0.2.1.0' , 'position' , 'position' , 'author' , '0.0.2.1')

('0.0.2.1.0.0' , '00' , 'position' , 'author' , '0.0.2.1')

('0.0.2.1.1' , 'value' , 'value' , 'author' , '0.0.2.1')

('0.0.2.1.1.0' , 'Peter' , 'value' , 'author' , '0.0.2.1')

……

接着我们查找 $Q2$ 中关键字的语义组合：article，Peter00

0: [(aritcle, null, aritcle)(author, value, Peter)(auhtor, position, 00)]

这里只有一组语义组合，可直接使用．

对关键字进行分组如下：

按照相同的实体和属性分为两组，去除从关键字后的分组内容为：

article

Peter, 00

查找关键字匹配节点的实体 Dewey 码如下：

article：0.0.2，0.1.2，0.1.3，0.2.2，0.2.3，0.3.2

Peter：0.0.2.1，0.1.2.3

00：0.0.2.1，0.1.2.2

对第二组的关键字 Peter 和 00 的匹配节点做交运算后我们得到两组实体

Dewey码：

0组：0.0.2, 0.1.2, 0.1.3, 0.2.2, 0.2.3, 0.3.2

1组：0.0.2.1

因为两组关键字语义环境中的实体节点间具有父子关系，我们将上述两组节点集做父子运算，保留父节点，得到结果：0.0.2．则以0.0.2为根节点的实体子树就是要返回的结果．我们直观地对图3.3进行观察，图3.3返回的结果有两个，FSRU算法返回结果只有一个，那就是图3.3(vst5)所示的实体子树，是一个有意义的结果单元．

3.7本章小结

本章针对已有关键字查询方法的不足之处，借鉴关系数据库E-R模型的思想，提出一种新的基于语义相关性的XML关键字查询方法，在定义关键字的环境语义的基础上，使用户的查询意图能够在查找XML文档的具体节点之前被确定，从而缩小了查询的范围．为了对关键字匹配节点的数量进行控制，论文提出主关键字的概念；为了确保查询结果是有意义的，而不是无关的、无用的信息，论文根据关键字匹配节点的语义相关性定义了语义相关单元，并给出基于语义相关性的XML关键字分类的查询算法及步骤；最后对算法的具体设计与实现给出了详细的说明．

4 实验结果及分析

实验所使用的硬件环境是：1.60GHz Intel Pentium Dual CPU，1GB 内存，120GB 硬盘，Windows XP sp2 操作系统.

实验程序使用 Java 语言开发，在 eclipse-SDK-3.4-win32 环境下运行. 后台数据库采用 Microsoft SQL Server 2005.

4.1 实验方法

本文在数据集 DBLP[51]和 XMARK[52]上，对基于语义相关性的 XML 关键字查询算法和 MLCA 算法的有效性和效率进行了对比.

4.1.1 数据集描述

实验过程中选取了实际数据集 DBLP 和自动生成的数据集 XMARK. DBLP 是一个非常流行的计算机科学参考文献的数据库，它被广泛地应用在测试有关 XML 查询技术等研究上，它的特点是结构简单，深度不高，但却包含大量的文本信息.

基于原始的 DBLP 数据集. 我们从中提取出 30000 个去除了引用子节点的“article”子树及 60000 个去除了引用子节点的“inproceedings”子树组成实验所用的 DBLP 数据集 dds1. 利用 XMARK 生成器. 我们生成了符合缺省 DTD 的大小为 100M 的实验所用的 XMARK 数据集 xds1. 为了比较数据集的大小对查询的影响，对上述 DBLP 数据集又抽取出同样规则的三组数据集 dds2、dds3、dds4，“article”子树和“inproceedings”子树的个数分别为 3000 和 6000，1000 和 2000，以及 300 和 600；而 XMARK 也同样再生成 3 组数据集 xds2、xds3、xds4，大小约为 20M、10M 和 1M.

4.1.2 实验方法描述

我们对查找语义相关单元算法（FSRU）和 MLCA 方法检索的结果进行实验的比较，以证明 FSRU 算法具有较好的有效性和较高的效率. 选取 MLCA 做为对比算法是因为 MLCA 方法的返回结果包含了有意义判断逻辑，这与本文所述的方法最为相似. 在实验过程中，MLCA 方法利用常规关键字查询用例作为查询条件，

而FSRU则利用常规关键字查询用例对应的所有SEKQC中的语义跨度最小的一个作为查询条件.

实验分为有效性测试和效率测试两个部分．有效性测试实验以XQuery的查询结果作为比对标准，利用查全率和查准率分析查询结果是否满足用户的查询意图；效率测试实验主要比较查询方法的返回时间，以及数据集的大小和关键字数目的多少对返回时间的影响度．表4.1是本实验所用的查询用例，$是关键字的分隔符.

表4.1 查询用例

Table4.1 Query cases

数据集	查询用例名	查询用例
DBLP数据集	*DQ1*	Article $ volume $ 1
	DQ2	Title$ number$ 3
	DQ3	Author$ pages$ 28
	DQ4	Volume$ 28$ year
	DQ5	Number$ 16$ Bioinformatics
	DQ6	Mdate$ 2004-07-30$ author$ Takeo Azuma$ 1999
	DQ7	Year$ 2002$ mdate$ 2002-08-13
	DQ8	Article$ James W. Fickett
XMARK数据集	*XQ1*	closed_auction$ price
	XQ2	closed_auction$ price$ date$ itemref$ quantity$ type$ seller$ buyer
	XQ3	closed_auction$ person76
	XQ4	person1$ address
	XQ5	closed_auction$ buyer$ person2064
	XQ6	person251$ person1869
	XQ7	seller$ person2417$ buyer$ price$ date
	XQ8	seller$ 07/24/1998

4.1.3 有效性测试实验

证明FSRU方法有效性的指标主要是查准率（precision）及查全率（recall）．整个实验以数据集dds1和xds1为基础，以XQuery的查询结果为标准，通过返回结果的数目及与XQuery结果的相符程度来说明算法的有效性．查准率和查全率的定义如公式4.1所示：

$$precision = (Rq \cap Rc)/Rq, recall = (Rq \cap Rc)/Rc \quad \ldots (4.1)$$

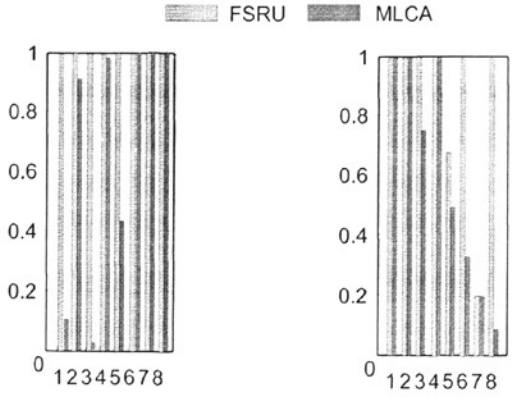
其中，*Rq*是某种具体查询方法的查询结果，而*Rc*是标准查询方法的查询结果．*Rq*⋂*Rc*表示具体方法与标准方法相重叠的结果个数．本实验的标准查询方法是XQuery．表4.2给出了不同的查询算法在两个数据集上针对相同的查询用例得到的结果的数目．

表 4.2 查询结果

Table 4.2 Query Result

数据集	查询用例	FSRU 返回记录单元的个数	MLCA 返回记录单元的个数	XQuery 返回记录单元的个数
DBLP	DQ1	485	4852	485
	DQ2	3855	4237	3855
	DQ3	13	278	13
	DQ4	266	272	266
	DQ5	135	312	135
	DQ6	1	1	1
	DQ7	245	245	245
	DQ8	3	3	3
XMARK	XQ1	9750	9750	9750
	XQ2	9750	9750	9750
	XQ3	2	2	2
	XQ4	1	1	1
	XQ5	3	3	3
	XQ6	1	2	1
	XQ7	2	2	1
	XQ8	8	65	8

算法的查询结果如表4.2所示．有效性实验的对比结果如图4.1和图4.2所示，其中，图4.1为算法FSRU和MLCA在数据集dds1和xds1上的查准率实验结果，图4.2为算法FSRU和MLCA在数据集dds1和xds1上的查全率实验结果．



(a)DQ1-DQ8　　*(b)XQ1-XQ8*

图 4.1 查准率

Figure 4.1 Precision

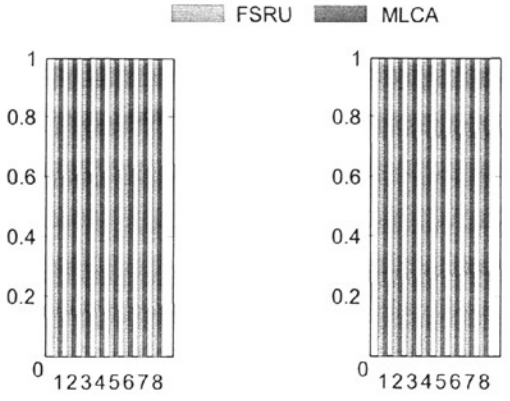


图 4.2 查全率

Figure 4.2 recall

如图4.1(a)和4.2(b)所示，对于DBLP数据集上的每个查询用例，FSRU都取得了最优的查全率与查准率，从而有力地证明了FSRU的查询结果符合用户的查询意图并且是语义相关的. FSRU与XQuery方法具有几乎相同的查询质量，而用户却无需学习复杂的XQuery语法，只需要对SEKQC进行选择，这是FSRU方法的优势之一. 由图4.2(a)可以看出MLCA方法的查全率同样是完美的. 但是，从图4.1(a)中*DQ*1到*DQ*5的查询结果我们可以看到，MLCA方法的查准率低于FSRU方法. 在*DQ*1到*DQ*5中，存在关键字在DBLP数据集中对应多个环境语义的情况. 因此，MLCA方法的查询结果都包含一些与用户查询意图不符的无关信息. 从图4.1(a)和图4.2(b)中我们可以看出，对于XMARK数据集，尽管FSRU方法的查全率和查准率都优于MLCA方法，但是它们都没有达到最优水平. 原因就在于，在XMARK数据集中，存在诸如“auction/buyer/person”和“auction/seller/person”的片段. 因此，如果我们仅将关键字与其本位实体和本位属性相关联，关键字仍然有可能对应多个环境语义. 我们将在未来的工作中解决这一问题.

4.1.4 效率测试实验

效率测试实验主要从不同查询用例的返回时间，关键字数目对返回时间的影响以及数据集大小对返回时间的影响 3 个方面进行 FSRU 查询方法与 MLCA 查询方法的比较.

首先，我们来分析在数据集 dds1 和 xds1 上，查询算法 FSRU 和 MLCA 对于查询用例 *DQ*1 到 *DQ*8 和 *XQ*1 到 *XQ*8 得到返回结果所使用的时间，结果如图 4.2 所示.

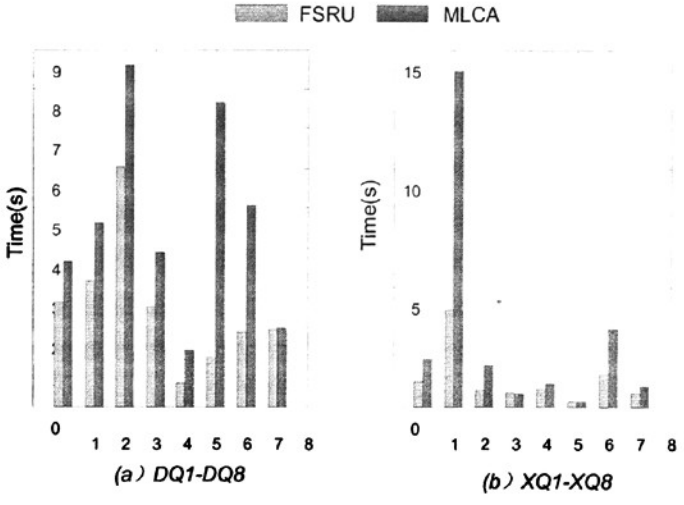


图 4.2 查询时间

Figure 4.2 Query times

图 4.2 可以帮助我们了解在不同的查询条件下 FSRU 查询方法的查询性能. 由于在FSRU算法中,用户对SEKQC进行选择的时间并不属于查询算法消耗的时间,因此，FSRU 的查询时间是查询关键字环境语义与查询语义相关单元的时间总和. 对于每一个查询用例，FSRU 的查询效率都优于 MLCA. FSRU 在查询效率方面具有优势的主要原因有两个：

（1）FSRU 只对主关键字的匹配节点进行查询运算. 通常，主关键字比从关键字匹配更少的 XML 节点. 因此，FSRU 查询运算过程中的 I/O 操作比 MLCA 显著地减少了，这是决定 FSRU 的查询效率性的关键因素.

（2）FSRU 通过比较关键字匹配节点的本位实体和本位属性来判断其语义相关性. 而 MLCA 方法则通过扫描包含关键字匹配节点的子树中的所有节点来判断关键字匹配节点的组合是否有意义.

其次，我们在同样的数据集 dds1 和 xds1 上，对查询用例 *DQ*6，*DQ*7 和 *XQ*2 分别进行修改，改变关键字的数目，观察关键字数目对查询性能的影响. 等价查询是指关键字数目不同，但查询的语义却不会发生变化，例如，*DQ*6 有 5 个关键字，其具有 3 个和 4 个关键字的等价查询分别是“2004-07-30 $ Takeo Azuma $ 1999”和“2004-07-30 $ author $ Takeo Azuma $ 1999”. 表 4.3 给出了查询用例的等价查询示例.

表 4.3 DQ6，DQ7 和 XQ2 对应的等价查询用例

Table 4.3 The equi-query cases of DQ6, DQ7 and XQ2

查询示例	关键字个数	等价查询用例
DQ6	3	2004-07-30$Takeo Azuma$1999
	4	2004-07-30$author$Takeo Azuma$1999
	5	mdate$2004-07-30$author$Takeo Azuma$1999
DQ7	2	2002$2002-08-13
	3	2002$mdate$2002-08-13
	4	year2002mdate$2002-08-13
XQ2	2	quantity$seller
	4	price$itemref$quantity$seller
	8	closed_auction$price$date$itemref$quantity$type$seller$buyer

图 4.3 给出了 FSRU 方法和 MLCA 方法在 *DQ6*，*DQ7* 和 *XQ2* 对应的不同数量的关键字的等价查询用例的查询时间曲线.

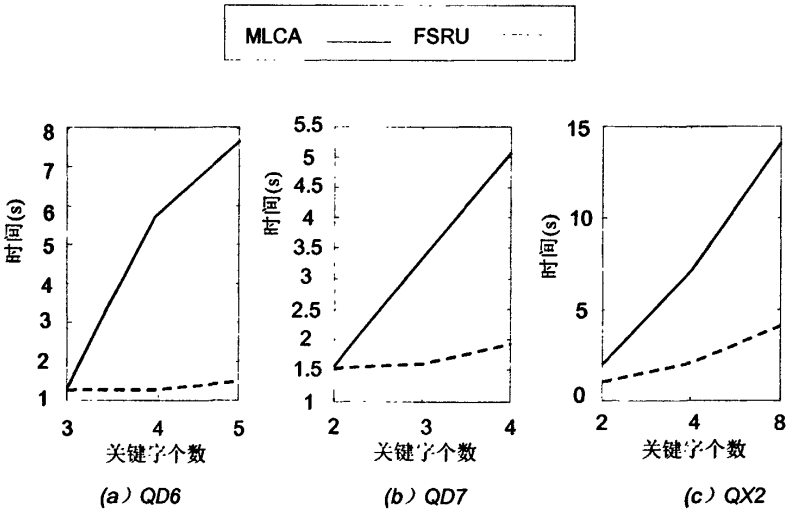


图 4.3 不同关键字数量对应的查询时间

Figure 4.3 Query times with different keyword .numbers

从图 4.3 中，我们可以明显地观察到一种趋势，即随着查询用例中关键字数量的增加，I/O 操作导致的查询效率的降低会越来越严重. 图 4.3(a)，4.3(b)和 4.3(c)分别给出了 *DQ6*，*DQ7* 和 *XQ2* 的不同关键字数量的等价查询的查询时间的曲线. 在图 4.3 中，每条 FSRU 曲线都比相应的 MLCA 曲线平缓. 因此我们可以说，FSRU 方法是比 MLCA 方法更稳定的一种关键字查询方法. 我们由此也可以知道，在能

够准确表达查询意图的前提下，用户应该尽可能的减少关键字的数目.

最后，我们考虑在不同大小的数据集 dds1 到 dds4 以及 xds1 到 xds4 上进行关键字查询，测试查询性能与数据集大小的关系. 实验结果如图 4.4 所示.

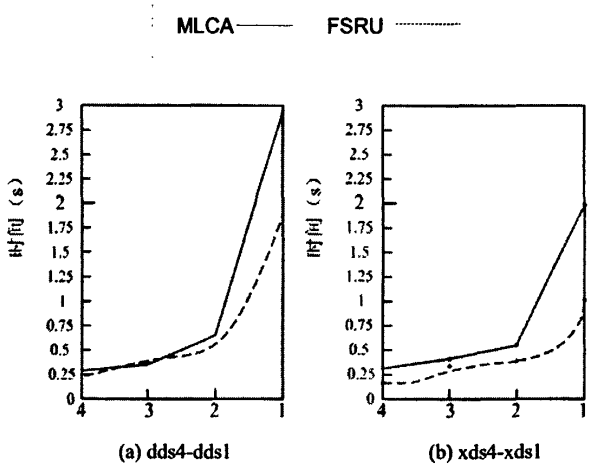


图 4.4 不同大小的数据集对应的查询时间

Figure 4.4 Query times with different dat set.sizes

图 4.4(a)给出了查询用例 DQ1 在数据集 dds1 到 dds4 上的查询时间，图 4.4(b)给出了查询用例 XQ1 在数据集 xds1 到 xds4 上的查询时间. 由图 4.4 我们可以看到，无论是 FSRU 方法还是 MLCA 方法，查询所需的返回时间都随着数据集的增大而增长，但是 FSRU 的曲线相对于 MLCA 的曲线来说比较平滑，波动没有 MLCA 大. FSRU 方法比 MLCA 方法更为稳定.

4.2 实验结论

本文提出的基于语义相关性的查询方法相比于传统关键字查询算法，更能够准确地将用户的查询意图传递给查询算法. 因此查询结果集大大减小，使得方法的有效性和效率等方面都取得了比较满意的效果.

5 总结与展望

XML 关键字查询的简单易用性降低了用户的使用门槛，例如用户不必事先掌握复杂查询语言的语法，也不必了解 XML 文档的底层结构．因此关键字查询特别适用于大多数的普通用户，具有广泛的应用基础．如何使关键字查询在易用的基础上符合用户的查询意图并有较好的查询效率成为目前研究的一个重要方向．

本文在考虑 XML 节点间结构关系的基础上，研究基于语义相关性的 XML 关键字查询算法．算法利用用户对关键字环境语义的选择，在获取关键字匹配节点前就明确用户的查询目的，极大地缩小了查询的范围，也在一定程度上保证了查询结果是有意义的．论文将关键字进一步分为主关键字和次关键字，仅保留主关键字，从而进一步控制关键字匹配节点的数目，有效地提高了查询的效率和准确度．而在关键字匹配节点的语义相关性基础上形成的语义相关单元则确保了查询结果是有意义的．

本文的主要研究成果如下：

- 研究了 XML 关键字查询的相关技术，分析了现有关键字查询方法的现状与不足．
- 引入关键字的环境语义，将一般的查询用例改为扩展语义的查询用例模式．在寻找关键字的匹配节点前，将扩展语义的查询用例集提供给用户用来确定用户的查询意图，缩小查询的范围．
- 根据关键字的环境语义进行分组，同组中的关键字可分为主关键字和次关键字．当一组中同时包含主关键字和次关键字时，此关键字不起作用，仅需保留主关键字．从而可以进一步限制关键字匹配节点的数目．
- 为保证查询结果是有意义的片断，在定义关键字匹配节点的语义相关的基础上，进一步定义了语义相关单元的概念．
- 详细描述了算法的具体流程，对算法的实现给出了具体的说明，如 XML 文件的解析，关系存储表的设计，算法实现的数据结构，算法的流程等．
- 最后通过真实数据进行实验，对比基于语义相关性的 XML 关键字查询方法和 MLCA 方法，证明基于语义相关性的 XML 关键字查询方法可以准确地将用户的查询意图传递给查询算法，同时也具备了较好的有效性和较高的效率．

XML 关键字查询是从 XML 文档中获得信息的重要方法，也是目前数据库领域的一个研究方向．论文提出了基于语义相关性的 XML 关键查询的方法，并实现了该算法．接下来可以在此基础上做进一步的研究工作，主要包括：

（1）可以进一步将关键字分为查询关键字和结果关键字．结果关键字用来细化返回结果，可以满足用户的特定查询要求，如返回结果是实体子树的某个属性或值，而不是整棵实体子树．

（2）查询结果的排序问题也是本文未讨论到的地方．

（3）优化索引结构和查询处理过程，从而改善查询性能也是未来的研究重点．

（4）本文对 XML 文档进行了简化，忽略了 XML 树中的引用节点．未来的工作将把这些被忽略的内容考虑进来，使算法能够适用于更复杂的 XML 文档结构．

本课题的研究，令我对 XML 关键字查询领域有了深入的理解，同时进一步提高了自身的研究动手能力，对日后的工作有非常大的帮助．

参考文献

[1] 展胥嵘, 黄上腾. XML 查询语言 XML-QL 及其查询优化.计算机工程. 2002. 28(3). pp111-113.

[2] Y.Diao, M.Altinel, MJ.Franklin. Path sharing and predicate evaluation for high-performance XML filtering. ACM Transactions on Database Systems. New York. 2003. 28(4). pp296-336.

[3] D.Chamberlin, D.Florescu. XQuery: A query language for XML.Proceedings of the 2003 ACM SIGMOD international conference on Management of data. San Diego. 2003. pp682-682.

[4] D.Srivastava, D.Dar S .Jagadish H V, Levy. An Answering queries with aggregation using views. In: Proc. of 22th Intl. Conf. on Very Large Data Bases. Mumbai (Bombay), India: Morgan Kaufmann. 1996. pp318-329.

[5] Cohen S, Nutt W, Serebrenik A. Rewriting aggregate queries using views. In, Proc. of 18th Symposium on Principles of Database Systems. Philadelphia; ACM Press, May 1999.

[6] Arsany Sawires, Junichi Tatemura, Oliver Po, Divyakant Agrawal, Amr EI Abbadi, K.Selcuk Candan. Maintaining XPath Views in Loosely Coupled Systems. In Proceedings of VLDB. 2006. pp583-594.

[7] D.Carmel, Y.Maarek. XML and Information Retrieval: a SIGIR 2000 Workshop. ACM SIGMOD Record. 2001. 30. pp62-65.

[8] Sara Cohen, Yaron Kanza, Yakov Kogan, Werner Nutt. Equix-A Search and Query Language for XML. Journal of the American Society for Information Science and Technology. USA. 2002. 53(6). pp454 – 466.

[9] A.Schmidt, M.Kersten, M.Windhouwer. Querying XML document made easy: nearest concept queries. Proceedings of the International Conference on Data Engineering. 2001. pp321-329.

[10] Yunyao Li, Cong Yu, H.V.Jagadish. Schema-Free XQuery. In Proceedings of VLDB. 2004. pp72-83.

[11] S.Cohen, J.Namou, Y Kanza and YSagiv. XSEarch: a semantic search engine for XML. Proceedings of International Conference on Very Large Data Bases. 2003. pp45-56.

[12] L.Guo, F.Shao, C.Botev, J.Shanmugasundaram. XRank: ranked keyword search over xml documents. Proceedings of the ACM SIGMOD International Conference on Management of Data. 2003. pp16 -27.

[13] Y.Xu, Y.Papakonstantinou. Efficient keyword search for smallest LCAs in XML databases. In Proceedings of SIGMOD. Baltimore. 2005. pp527-538.

[14] V.Hristidis, N.Koudas, Y. Papakonstantinou, and D. Srivastava. Keyword Proximity Search in XML Trees. IEEE Transactions on Knowledge and Data Engineering. 2006. 18(4).

[15] Z.Y. Liu, Y.Chen. Identifying Meaningful Return information for XML Keyword Search. In Proceedings of SIGMOD. 2007. pp329-340.

[16] 万常选, 刘云生, 等. 基于区间编码的XML索引结构的有效连接. 计算机学报. 2005. 28(1). pp113-127.

[17] 王静, 孟小峰, 等. 基于区间划分的XML结构连接.软件学报. 2004. 15(15). pp720-729.

[18] 孔令波, 唐世渭, 等. XML 数据索引技术. 软件学报. 2005. 16(12). pp2063-2079.

[19] 文继军, 王珊. SEEKER: 基于关键词的关系数据库信息检索. 软件学报. 2005. 16(7). pp1270-1281.

[20] 王晓玲, 文继荣, 等.一种通过内容和结构查询文档数据库的方法. 软件学报, 2003, 14(5). pp976-983.

[21] Sihem Amer-Yahia, SungRan Cho, Laks V. S. Lakshmanan, Divesh Srivastava. Minimization of Tree Pattern Queries. Proceedings of the 2001 ACM SIGMOD international conference on Management of data. Santa Barbara, California, USA. pp497-508.

[22] Mark Graves. 尹志君等译. XML 数据库设计. 第一版. 机械工业出版社. 2002. pp23-47

[23] T.Bray, J.Paoli, McQueen CMS (Editors). Extensible Markup Language (XML) 1.0 (Third Edition). W3C Recommendation.1998.

[24] 刘光. 网页核心 XML 应用技巧与实例. 第一版. 北京. 清华大学出版. 2000. pp16-17.

[25] S.Abiteboul, D.Quass, J.McHugh, J.Widom and J.Wiener. The Lorel query language for semistructured data. International Journal on Digital Libr.aries (1). 1997. pp68-88.

[26] S.Ceri, S.Comai, E.Damiani and P.Fraternali. XML-GL: a graphical language for querying and restructuring xml documents. Int'l World Wide Web Conf. Toronto. 1999. pp1171-1187.

[27] D.Chamberlin, J.Robie, D. Florescu. Quilt: An XML Query Language for Heterogeneous Data Source. WebDB2000. Dallas, Texas. 2000. pp1-25.

[28] Miklau G and Suciu D, Containment and Equivalenc for a Fragment of XPath. Journal of the ACM. 2004. 51(1). pp2-45.

[29] www.w3.org/TR/xquery/#id-flwor-expressions,2007-01-23.

[30] 黄少荣. 新一代查询语言 Xquery 及查询优化. 埠南金融电脑. 应用技术. 2005. 21(4). pp445-449.

[31] Bruno N, Koudas N, Srivastava D. Holistic twig joins: Optimal XML pattern matching. In: Franklin MJ, Moon B, Ailamaki A, eds. Proc. of the SIGMOD Int'1 Conf. on Management of Data. Madison: ACM Press. 2002. pp310-321.

[32] 孔令波, 唐世渭, 杨冬青, 王腾蛟. XML 数据查询技术. 软件学报.2007. 18(6). pp1400-1418.

[33] Clark J, DeRose S. XML path language (XPath) version 1.0 w3c recommendation. World Wide Web Consortium. 1999.

[34] TATARINOV IVIGLAS S D, BEREY K, et a1. Storing and querying ordered XML using a relation database system. [C]//Procedings of the 21th ACM SIGMOD International Conference on Management of Data. NY, USA: ACM Press. 2002. pp204-215.

[35] I.Tatarinov, SD.Viglas, K.Beyer, J.Shanmugasundaram, E.Shekita, C.Zhang. Storing and querying ordered xml using a relational database system. Proc. of the 2002 ACM SIGMOD Int'l Conf. on Management of Data (SIGMOD). Madison: ACM Press. 2002. pp204-215.

[36] C.Botev, J.Shanmugasundaram. Context-Sensitive keyword search and ranking for XML. Proc.of the 8th Int'l Workshop on the Web&Databases(WebDB 2005). 2005. pp115-120.

[37] S.Amer-Yahia, N.Koudas, A.Marian, D.Srivastava, D.Toman. Structure and content scoring for XML. Proc.of the 31st Int'1 Conf.on Very Large Data Bases (VLDB). Trondheim:ACM Press. 2005. pp361-372.

[38] F.Weigel, H.Meuss, KU.Schulz, F.Bry. Content and structure in indexing and ranking XML. Proc.of the 7th Int'l Workshop on the web and Databases(WebDB). Malson de la Chimie: ACM

Press. 2004. pp67-72.

[39] T.Schlieder, H.Meuss. Querying and ranking XML documents. Journal of the American Society for Information Science and Technology. 2002. 53(6). pp489-503.

[40] R.Yang, P.Kalnis, AK.Tung. Similarity evaluation on tree-structured data. Proc.of the ACM SIGMOD Int'l Conf.on Management of Data (SIGMOD). Baltimore: ACM Press. 2005. pp754-765.

[41] K.Kailing, H.Kriegel, S.SchOnauer, T.Seidl. Efficient similarity search for hierarchical data in large databases. Proc.of the 9th Int'l Conf.on Extending Database Technology (EDBT). Greece: Springer-Verlag. 2004. pp676-693.

[42] E.Kotsakis. Structured information retrieval in XML documents. Proc.of the 2002 ACM Symp. on Applied Computing (SAC). Madrid: ACM Press. 2002. pp663-667.

[43] Jayavel Shanmugasundaram, Kristin Tufte, Gang He, Chun Zhang, David DeWitt, Jeffrey Naughton. Relational Databases for Querying XML Documents: Limitations and Opportunities. In Proceedings of VLDB. 1999.

[44] Mustafa Atay, Artem Chebotko, Dapeng Liu, Shiyong Lu, Farshad Fotouhi. Efficient schema-based XML-to-Relational data mapping. Information Systems. 2007.32(3). pp458–476.

[45] Kevin Beyer, Roberta J. Cochrane, Vanja Josifovski, et al. System RX: One Part Relational, One Part XML. In Proceedings of SIGMOD. Baltimore, Maryland, USA. 2005.

[46] Matthias Nicola, Bert van der Linden. Native XML Support in DB2 Universal Database. In Proceedings of VLDB. Trondheim, Norway. 2005.

[47] Liu ZY, Chen Y. Identifying Meaningful Return information for XML Keyword query. In: Chan CY, Ooi BC, Zhou AY, eds. Proc. of the 2007 ACM SIGMOD Int'l Conf. on Management of Data (SIGMOD). Beijing: ACM Press. 2007. pp329-340.

[48] Li XG, Yu G, Gong J, Wang DL, Bao YB. Towards Effective and Efficient NFS Querying on XML Document. Chinese Journal of computers. 2007. 30(1). pp57–67.

[49] Liu ZY, Chen Y. Identifying Meaningful Return information for XML Keyword query. In: Chan CY, Ooi BC, Zhou AY, eds. Proc. of the 2007 ACM SIGMOD Int'l Conf. on Management of Data (SIGMOD). Beijing: ACM Press. 2007. pp329-340.

[50] Chen PPS. The Entity-Relationship Model - Toward a Unified View of Data. ACM Trans. on Database Systems. 1976. 1(1). pp9-36.

[51] http://dblp.uni-trier.de/xml/.

[52] http://www.xml-benchmark.org/