TopX 查询子系统的分析与改进

学　　院　计算机科学与技术

专　　业　计算机科学与技术

年　　级　2004 级

姓　　名　吴宗远

指导教师　张坤龙

2008 年 6 月 15 日

摘　　要

随着互联网的迅猛发展，XML 数据的数据量日益增长。为了更好的查询和利用这些信息，XML 信息检索技术应运而生。TopX 是一个比较先进的 XML 信息检索系统，对其进行分析和改进有着理论和实用上的重要价值。

论文分析了 TopX 查询子系统的查询处理过程。TopX 建立在支持内容和结构条件联合的概率信息检索模型之上，支持所有 XPath 轴的路径条件作为精确或模糊的结构性限制条件，支持基于本体的语词项和标记名的扩展作为内容条件，是一个对 XML 和普通文本数据进行排序检索的搜索引擎。为了了解和掌握 XML 信息检索技术，熟悉查询处理的工作流程，本文主要通过结合源代码的形式，剖析其查询处理中的关键技术，了解其实现机制，并对不足的对方加以改进和完善。其中分析的关键技术包括索引访问调度、高代价谓词的引入和处理、提前候选对象删除等。

论文对 TopX 的查询子系统作出了改进，改进工作主要包括对其架构、字符编码的改进以及中文查询的实现。

关键词：XML；信息检索；查询处理；TopX；top-k

ABSTRACT

As the Internet develops, XML data grows quickly, to query and make better use of these information ,it is necessary to develop the technology of XML information retrieval. TopX is a relative advanced XML information retrieval system based on this technology, to analyze and improve it is make sense in theory and practice.

This thesis analyze the work flow of query processing in TopX's subsystem of query process. TopX supports a probabilistic-IR scoring model for full-text content conditions and tag-term combinations, path conditions for allXPath axes as exact or relaxable constraints, and ontology-based relaxation of terms and tag names as similarity conditions for ranked retrieval. TopX is a search engine for ranked retrieval of XML and full-text data. To understand and master the technology of XML information retrieval, know the work flow of query processing, We combined with the source code to analyze the key technologies in query processing, understand their realization mechanism and improve and perfect its shortages. The main analyzed technologies include these: index access scheduling ,the introduce and process of expensive predicates, early candidate pruning and so on.

Based on the analysis we give some improvements on the original TopX system. Including the improvements on its architecture, character encoding and the realization of Chinese retrieval.

Key words：XML; information retrieval; query processing; TopX; top-k

目　录

第一章　绪论

1.1　文本数据的三种类型

计算机技术的飞速发展导致一个全新的信息时代的到来。在这个时代里，互联网上存储有海量不同领域不同格式的数据信息。尽管这些数据信息表现出不同的形式，而且多媒体数据(如语音、图象、视频等)的数量在快速增长，但是文本数据依然是一种非常基础的、被广泛使用的信息存储形式。文本信息从结构上大致可以分为三种类型，分别是无结构数据、结构化数据和半结构化数据[1]。

无结构数据指的是用标点符号和语法标记分隔的原始文本。在互联网上可以获得的传统 HTML 文件可以看作是一种无结构数据。HTML 文档中的标记只具有语法性质，几乎不能表现出任何与内容有关的语义信息。而且，HTML 没有严格的结构，它允许文件包含没有正确嵌套和与之对应的结束标记的开始标记。这些特性给开发者的使用带来了很多方便，但同时也失去了从中提取结构信息或元数据的能力。

结构化数据指的是具有预定义的严格格式的数据，例如数据库记录，其中的元数据给出了其内容的类型、长度和其它属性的清晰定义。用户可以通过查询获得针对其需要的具体、明确的答案，但只局限于简单、固定和参数化的查询模式。如果不了解数据库的元数据就无法获得所需的信息。

半结构化数据则介于以上二者之间，它具有一个比无结构数据更明确的结构限制却又不像结构化数据那样严格。半结构数据通常采取信息和描述信息的模式一并出现的形式，它具有自描述特征，因此它的信息展示比较容易，适合在互联网上广泛传播。半结构化数据在信息展示和数据交换上有着自身固有的优势，它是无结构数据和结构化数据的一种中间状态，既能够表达内容信息，又能够传递出结构信息，因此半结构化数据是一种非常有价值的数据形式。

1.2　XML 的产生及意义

XML(EXtensible Markup Language)数据是一种最典型的半结构化数据。XML 的前身是 SGML(The Standard Generalized Markup Language)。SGML 有两个重要的思想：一是文件中能够明确地将标示与内容区隔，二是所有文件的标签使用方法均一致。由于 SGML 是一种非常严谨的文件描述法，导致它过于庞大复杂（标准手册就有 500 多页），难以理解和学习，进而影响其推广与应用。人们对 SGML 进行了简化衍生出了 HTML，HTML 仅用来在浏览器里显示网页文件并被使用了数十年之久。但是，HTML 慢慢也显露出它的不足：缺乏对复杂结构的支持，难以实现自动的数据交换，不利信息重用等。

为了解决以上问题，专家们对 SGML 进行了精简，并依照 HTML 的发展经验，产生出一套使用上规则严谨，但是简单的描述数据语言：XML。XML 目的

在于提供一个对信息能够做精准描述的机制，藉以弥补 HTML 太过于表现导向的特质[20]。

XML 如今被广泛用于信息展示、程序设计、数据通信、服务配置等众多领域，并已经发展成为互联网上数据展示和信息交换的新标准。与 HTML 不同，XML 允许用户自定义标记来识别标记之间包含的数据信息，每一个数据块都需要封装在一对开始和结束标记之中，不同数据元素的关系通过嵌套和引用表示出来。XML 数据可以利用 XSL、XSLT 等语言来定义其展现格式，因此使用 XML 可以将信息内容与信息展示这两个问题分离开来，方便为同一数据提供多样的视图，而且不会混淆文档的核心信息和形式信息。那种从 HTML 网页中提取有用信息的费力、易错且无法维护的“擦屏”方法的使用将大大减少，因为 XML 是为数据展示而设计的——它简单、易解析、是自我描述的，而且 XML 还可以在一些共同的语法或 DTD 的基础上作为大范围的组织间数据整合的平台。

```
<?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>
</bookstore>
```

图 1.1　XML 文档示例 books.xml

图 1.1 是一个 XML 文档的实例。通过其中的标记名称和层次关系，我们可以清晰的了解这个书店的图书信息。

XML 在结构化数据(如数据库记录)和无结构数据(如 HTML)之间建立起了一座连接的桥梁，由于其先天的优越性，它在各个领域的应用也不断深入和扩大，导致 XML 数据量成指数级快速增长。当大量的 XML 数据信息存在的时候，如

何有效的利用这些信息就成了一个必须去面对和解决的问题。人们也一直致力于 XML 数据的存储、索引和查询方面的研究。首先产生的就是简单的 XML 查询语言。

1.3 XML 查询语言及其局限

XML 查询语言产生的目的，主要是为了使用户能够从 XML 文档中抽取信息，能够在不同 DTD 之间翻译 XML 数据，能够从多个 XML 文档中结合数据，能够传送 XML 数据等。以下是几种典型的 XML 查询语言。

一、XPath(XML 路径语言)

XPath 是一个表达式语言，它作用在 XML 文档的抽象逻辑结构而非文档的表层语法上。这个逻辑结构，称为数据模型，在 XQuery/XPath Data Model (XDM) 中定义，它为 XML 文档、原子值(例如整数字符串布尔量)以及可能包含 XML 文档结点的引用和原子值的序列提供了一个树型表示。XPath 的目标是在 XML 文档树中寻址结点。XPath 的名字源于它使用一个路径表达式来导航 XML 文档的分层结构，XPath 使用一个紧凑的、非 XML 句法来促进 XPath 在 URIs 和 XML 属性值上的使用。

XPath 被设计成嵌入到一个宿主语言如 XSLT2.0 或 XQuery 中来使用，XPath 有一个子集可以用来进行匹配操作，即测试一个结点是否匹配一个模式[2]。

图 1.2 是一个 XPath 查询示例，返回图 1.1 所示的 books.xml 中 bookstore 元素下第一个 book 结点。

```
set xmlDoc=CreateObject("Microsoft.XMLDOM")
xmlDoc.load("books.xml")
xmlDoc.selectNodes("/bookstore/book[0]")
```

图 1.2 XPath 查询示例

二、XQuery

XQuery 是用于 XML 数据查询的语言，它对 XML 的作用类似 SQL 对数据库的作用。XQuery 被构建在 XPath 表达式之上，被所有主要的数据库引擎支持 (IBM、Oracle、Microsoft 等等)且是 W3C 标准。XQuery 是用来从 XML 文档查找和提取元素及属性的语言，XQuery 1.0 和 XPath 2.0 共享相同的数据模型，并支持相同的函数和运算符。

XQuery 可以被用来提取信息以便在网络服务器中使用，生成摘要报告，把 XML 数据转化为 XHTML，为获得相关信息而搜索网络文档等[3]。

三、NEXI(Narrowed Extended XPath I)

NEXI 是一个由 XPath 衍生出的语言，其中增加了 about 函数。与 XPath 不

同，NEXI 中的语义并未定义，而是由检索引擎推断出来的，这个显著的特点与面向数据库的查询语言不同。对于内容查询，NEXI 不允许对 XML 元素的限制，同样也不需要对目标元素进行指定。但是对于既包含内容又包含结构限制的查询，就需要明确或隐含的结构性限制条件[4]。

XML 查询语言在一定程度上能够返回用户需要的结果，但是它也存在一些不足。如某些 XML 查询语言只能够进行模式匹配式的工作，主要侧重于结构查询，不能充分表达出内容上的信息。即使像 NEXI 这样的查询语言，可以兼顾结构和内容信息，但是它缺少一个相似度评价系统，不能很好地评估查询目标，更无法返回一个排序的结果序列。

由于单纯 XML 查询语言的不足，我们需要借助另外一种技术来更好地处理查询语言，以在 XML 文档集上做出更明确更有效的查询，这便是信息检索技术。

1.4 XML 信息检索技术

信息检索(information retrieval)是指将信息接一定的方式组织和贮存起来，并根据信息用户的需要找出有关信息的过程。信息检索的发展已经成为包括信息的存储、组织、查询、提取等多个工作过程的一整套方法。信息检索主要采用模糊匹配方式和多样分类方法，它的查询语言往往采用自然语言，它致力于信息条目的相关性，而并不苛求完全匹配。

信息检索发展到今天，已经产生出四大核心技术，分别是：全文检索技术、自动分类技术、异构检索技术和智能检索技术。这四大核心技术在此不作深入讨论。需要指出的是，目前所有主流的搜索引擎都是基于关键字搜索，即利用关键字来表示查询语句和文档的核心信息。在信息检索领域，常用的模型有数据模型、检索模型等，它们都是一组抽象和形式化具体信息的方法[21]。

下面重点要介绍的是 XML 信息检索技术。

XML 作为一种半结构化数据，能够包含更多用户需要的信息，而且便于用户在其上做出更具表达能力的查询。XML 数据来源于许多不同的数据源，经常展现出异构结构，而且它的注解(即 XML 标记)不能够使用类数据库查询语言的 XPath 或 XQuery 进行充分搜索，通常这样的查询返回结果数量不是太多就是太少。XML 信息查询需要的是排序检索的范式、可松弛的搜索条件以及量化的相关性评价标准，它并不是简单的在 XQuery 上增加布尔文本搜索谓词。实际上，相似度评价和排序与数据类型是正交的，我们需要在查询中加入一些结构性属性，例如时间(如大约在 2008 年左右)、地理坐标(如天津大学附近)以及其它数字的和分类的数据类型。

将信息检索技术应用到 XML 数据上的研究已经开始了五六年，这个领域获得了相当的关注，取得了很大的发展[5]。本文将要介绍的 TopX 系统就是一个基于 XML 信息检索技术的 XML 信息检索系统，它着力于研究一种有效的模糊搜

索，其中的搜索条件是出现在元素名和元素内容中的语词项的内容条件和 XPath 风格的路径条件的组合。结构性相似度的要求是，一个文档无需满足所有的路径条件但依然可以成为查询结果。使用基于叙词表的相似度度量方法，标记名和内容语词可以放松限制，从而使查询得到合适的扩展。这些技术的实现，可以提高 XML 检索系统的性能，返回一些更符合用户需求的排序结果。

1.5 论文的组织结构

论文的主要工作是分析 TopX 系统的查询处理过程，并在此基础上对它做出改进。

论文的第一章介绍了 XML 数据的重要性、XML 信息检索技术的意义和 TopX 的由来。第二章将讲述 TopX 的系统结构及其数据模型、查询模型和索引结构。第三、四、五章是全文的核心所在，重点介绍 TopX 的查询处理过程，其中第三章简要介绍 TopX 的 top-k 算法及其查询处理流程，第四、五两章从内容条件和结构条件两方面来深入阐述 TopX 查询处理中使用到的一些关键技术，如高代价谓词处理、索引访问调度、提前候选对象删除等。第六章介绍对于 TopX 的改进工作，包括其架构的改进、字符编码的改进和中文查询的实现。第七章是结论部分，总结了 TopX 查询处理的工作流程及对它的改进，并提出了一些需要深入研究的问题。

第二章　　TopX的系统结构

2.1　TopX系统概述

TopX是一个对XML和普通文本数据进行排序检索的搜索引擎，它是由德国的马克思－普朗克信息科学研究所开发出的。TopX基于支持全文本条件和标记语词项合并的概率信息检索模型，它支持所有XPath轴的路径条件作为精确或可松弛的查询限制条件，以及基于本体的语词项和标记名的扩展作为排序检索的相似度条件。TopX使用了多种技术来加快top-k查询，包括：使用概率模型作为一个有效率的基于改进的阀值算法的相似度得分预测器，对为扫描索引列表而做的顺序访问和为计算全相似度得分而做的随机访问进行审慎的调度，随需应变地对索引列表进行增量式合并，运用自定位的查询扩展，以及一组为评估结构路径条件专门设计的预先计算好的索引。

TopX经过了压力测试和多种数据集上的性能评估，包括TREC的T级数据测试平台、INEX(INitiative for the Evaluation of XML Retrieval，是一个评价XML信息检索系统性能国际科研组织)的XML信息检索测试平台以及维基百科的一个XML版本。TopX还被作为INEX－2006的特别子任务基准测试一个参考引擎，它可以被用来在不同的数据集上进行交互式查询[5]。

TopX系统主要由两部分组成：1、TopX索引器和一组数据导入及建立索引的方法，2、TopX搜索servlet及其底层的类库。

TopX是一个XML信息检索的原型系统，它尝试使用了一个对top-k选择查询进行组合的顺序和随机访问的模型。TopX实现了有效查询评估和通用的排序结果输出评分模型二者之间的无缝整合，它位于数据库工程和信息检索研究之间的连接点上。从数据库的角度出发，TopX为对大量数据进行可扩展的、top-k类型的处理提供了一个有效的算法基础，它重点在于研究使用自适应的、面向磁盘的代价模型来存取大型的磁盘索引结构，以及高度成熟的存储和有效查询大量文档集合(可能在T字节的数据)的解决方案。通过观察可知，顺序磁盘I/O受益于异步预取及硬件和处理器高速缓存分层结构的高度局部性特征，其均摊代价比随机磁盘存取小得多，因为后者需要额外的索引结构和对单独对象标识符进行关键字查找。TopX的查询处理方法侧重于这些代价较低的顺序磁盘访问，但是一定数量的随机访问对于解析出特定候选对象的最终得分并明确最终的结果排序来说也是很关键的，因此TopX中也选择性地使用了随机访问。

从信息检索的角度来看，TopX提供了一个对全文本、半结构化数据和结构化数据进行索引和有效搜索的完整框架。TopX查询处理器支持强连接性和宽松的信息检索风格的检索选项，以及高代价文本谓词例如短语、强制语词或是否定语词或短语。它是一个自包含(不需要依赖其它程序)的查询引擎，支持高级的IR

技术或针对全部 IR 应用程序的各类评分模型，它为 Web IR、结构化属性以及包含 XML 全文本搜索的排序 XML 检索提供了一整套富于创新且高效的评分方法。

TopX 假设所有单个属性的值都预先计算好并以恰当的形式存储在磁盘上，即存在一个关系型数据库系统中，或是使用一个更具有面向对象特性的倒排文档的方式，这些属性值包括可选的索引表元数据如得分直方图、相关性统计表和量化的属性相似度等。在查询处理阶段，针对多种属性的评分累加和结构排序在数据库系统之上进行，这是 TopX 引擎中专门的一块。

简单来说，TopX 的工作流程是，首先在一个静态资料库上建立索引，并将索引存入数据库管理系统中，然后 TopX 的搜索引擎在数据库管理系统上执行查询过程，并返回查询结果。

下面来看一下 TopX 的结构图。

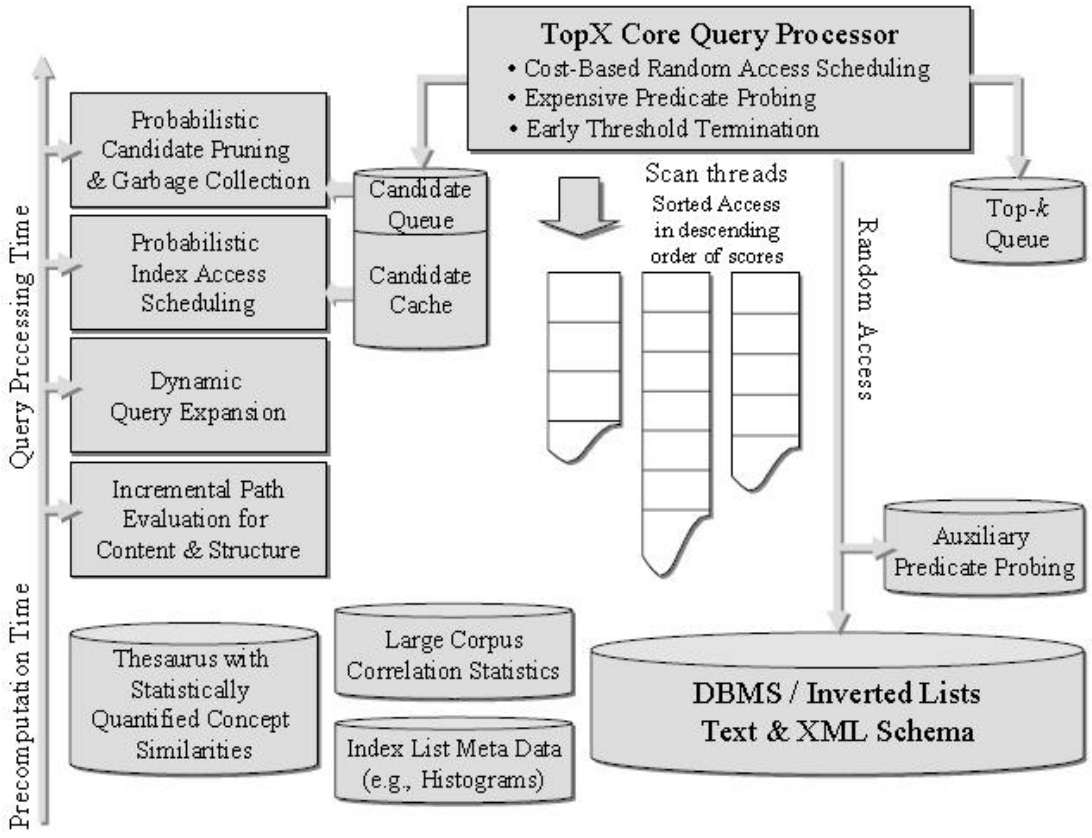


图 2.1　TopX 组件图[5]

大体来说，TopX 系统是一个组件式结构，只要处理好组件之间的接口，就可以实现其中一些组件的添加或删除。图 2.1 描述了 TopX 的主要组件结构，其中的 TopX 核心查询处理器负责记录中间结果并在一个多线程体系下管理对索引链表的顺序和随机访问，它为精确和有效的 top-k 查询处理建立了算法基础。在

此基础之上可以逐步加入其它组件，包括：为提前候选对象删除而设的概率得分预测器，基于概率模型的顺序和随机访问调度决策器，使用专门查询操作符的动态查询扩展以及对 XML 全文本搜索的有效支持等。

2.2 TopX 的模型和索引结构

本文的重点在于阐述 TopX 查询处理过程中的若干问题。在正式进入查询处理过程的介绍之前，需要对 TopX 的数据模型、查询模型和索引结构等预备知识有一个大概的认识

2.2.1 TopX 的数据模型

TopX 使用半结构化数据的树型模型，遵循 W3C XML 1.0 和 1.1 标准，但是不考虑任何前缀标记(<!...>)和 XLink 或 ID/IDRef 属性形式的链接[6]。属性被当作是对应元素结点的子结点，文本结点则直接关联到它们的前位元素父结点。图 2.2 展示了一个非常简单的遵循 TopX 数据模型的示例 XML 文档，当前，所有 TopX 的索引结构都建立在文档树模型的基础上。

```
<article id="ieee/w4043">
<title>XML Data Management
and Retrieval</title>
<abs>XML data management systems vary
widely in their expressive power.
</abs>
<sec>
<st>Native XML Databases</st>
<par>Native XML databases
can store schemaless data.</par>
</sec>
<bib>
<item>
XML Path Language (XPath) 1.0
<url>www.w3c.org/TR/xpath</url>
```

图 2.2 XML 文档示例

2.2.2 TopX 的查询模型

```
//article[//bib[.//,about(.//item,W3C)]]
  //sec[about(.//title,XML retrival)]
    //par[about(.,native XML database)]
```

图 2.3 NEXI 风格的样例查询

图 2.3 展示了一个用 NEXI 句法写成的示例查询。

根据 XPath 和 NEXI 二者的规定，一个路径的最右边的高层结点称为查询的目标结点，所有在路径中进行结点检测的其它结点称为支持结点。其中，查询的目标结点定义了结果的粒度，如果对查询的结构进行严格解释的话，只有那些匹配查询目标结点的元素才可能成为有效结果。在示例中，par 元素是目标元素，标记为 article,bib,item,sec 和 title 的结点都是支持元素。

另外，如果一个查询中没有包含明确的标记名，而是以一个通配符“*”代替，这个通配符“*”可以匹配任意标记名。一个只包含通配符和一些内容条件的查询如 //*[about(.,native XML databases)] 对应于一个单纯的关键字查询，按照 INEX 中的说法，这样的查询称为 content-only(CO)查询，相对的，那些包含结构性限制条件的查询称为 content and structure(CAS)查询。示例中的查询是一个 CAS 查询。

2.2.3 TopX 的索引结构

TopX 中分别为内容条件和结构条件使用了两种主要的索引结构，另外还包括一个位置索引以检测文本的短语条件。所有的索引都使用一个关系型数据库管理系统作为存储后台并利用它自带的 B+树结构来实现。下面从概念组织层次来分别看一下这三种索引：

一、标记－语词索引　标记－语词对是在 TopX 中常用的一个概念，它由标记名及起始标记结束标记之间包含的任意一个语词组成，例如图 2.2 的示例文档就包含标记－语词对(title，XML)。TopX 为每一个标记－语词对设定一个对应的倒排链表，格式为：(tag, term, docid, pre, post, level, score, maxscore)，其中的 pre 和 post 是元素的的前序和后序编码(同时 pre 还被作为文档中某个元素的唯一标识符)，level 是元素在树中的深度，score 是元素对应此标记－语词条件的得分，maxscore 是给定文档中所有元素对应此标记－语词条件的最高得分。一个(tag,term)对的倒排链表中的各项按照以下排序规律进行排序以为查询处理器提供便利，即按照(maxscore,docid,score)的降序排列进行排序(即使用 maxscore 作为主排序标准，docid 作为第二排序标准，score 作为第三排序标准)。

二、结构索引　TopX 对文档中的元素使用一种能够有效检测各种 XPath 轴

的编码方式，为每个标记索引项预先计算出(tag, docid, pre, post, level)，其中 pre 和 post 编码了一个元素的标识符和导航位置，level 是元素在其对应的文档树中的深度，这些索引项只会在对特定元素进行随机访问时用到。

三、位置索引　对应每个语词项有一个索引项形式为(term, docid, pos)，其中 pos 是语词在文档中出现的偏移位置，这个索引只被随机访问使用来检测位置以进行短语匹配。

以上介绍了 TopX 中使用到的一些模型和索引结构，这些都是为 TopX 的查询处理服务的，第三章将正式进入 TopX 的查询处理过程分析。

第三章　　TopX 的查询处理

TopX 查询处理最核心的思想是通过一系列对查询资料库索引结构的访问和查找，按照特定的相似度评价算法，找到最满足查询条件的前 k 个结果，并按照相似度从高到低将其排序。TopX 的查询处理是一种典型的 top-k 查询处理。本章首先介绍几个基础的 top-k 查询处理算法，然后阐述 TopX 的基本查询处理过程。

3.1　几种 Top-k 查询处理算法

3.1.1　TA 算法

TA 算法是推动在大型磁盘索引结构上进行 top-k 查询处理的先驱[5]。它以一种交错的方式顺序扫描所有查询相关的索引链表，通过一个单调函数(如加和、最大值等)来计算检测到的数据项的“全局”得分。TA 算法记录当前 top-k 候选项的最低分和未检测到或未完全检测的候选对象的最高可能得分，后者作为一个阈值条件，当没有候选对象得分能够超过当前 top-k 中的最低得分的时候，就可以停止索引扫描。

TA 算法有几种变形形式，其最原始的形式是在每次顺序扫描检测到新候选项的时候，即刻调用随机扫描来获取这个对象所有的局部得分以计算其整体得分。假设存在 m 个索引，每个索引包含了一个查询条件的得分，而且一个索引的所有索引项按照得分降序排列。TA 算法执行的是一种称为一次一个文档的策略，即一旦在顺序访问中检测到新的候选对象，便通过随机访问在所有余下的输入链表中即时获取它的累计得分。通过这种方式，算法不会产生任何得分的不确定性，而且除了存储 top-k 中间结果的候选项队列之外无需其它队列。用一个初始的阈值表示各个链表当前位置的 high 得分之和，并以此作为尚未访问到的候选项的得分上界。如果这个阈值低于已经得到的 top-k 结果项中的最低得分，那么这个算法就可成功终止了，因为已经没有任何尚未访问到的候选项得分会超过这个阈值。图 3.1 所展示就是 TA 算法的伪码。

3.1.2　NRA 和 CA

这两种算法都是 TA 算法的改进形式。TA 算法在检测到一个新候选对象时即调用随机访问来获取其全局得分，因为随机访问的代价较大且在某些环境中可能无法实现，因此产生出 NRA(无随机访问算法)算法。NRA 假设只对索引链表进行顺序扫描且不需要所有的候选对象都被完全地按照查询条件评估一遍。与 TA 算法相反，NRA 执行的是称为一次一个语词的策略，它单独的为每个文档评估不同的查询条件，并在一个中间数据结构中记录此部分结果。因此，它给一个候选文档 d 提供两个得分预测，即一个下界 worstscore(d)(在已经评估过的查询条

件中的得分和)和一个上界 bestscore(d)(worstscore(d)加上文档 d 未评估过的查询条件集 $\overline{E(d)}$ 上的得分上界 $\sum high_i$)，NRA 在 TA 的基础上扩展了候选对象记录的功能，算法终止的条件是：未完全检测(包含未检测)的对象的最高 bestscore(d)已经不可能超过当前 top-k 对象中的最低 worstscore(d)。

NRA 算法完全丢弃了随机访问，但是一定数量精心调度的随机访问对于提高查询处理效率是非常有意义的。它能够导致大量的候选对象提前删除，从而节约内存空间和处理时间。CA(组合算法)的思想就来源于此，它在 NRA 的基础上增加了一些有限但非常有效的随机访问来获取当前最高得分候选对象的最终得分，从而在每一轮的顺序访问之后可以谨慎的删除一些不符合条件的最佳候选对象。图 3.2 展示的就是 CA 算法的伪码。

```
IndexList L[m];    // (L1,...Li,...Lm)
QueryItem t[m];    //(t1,...ti,...tm)
TopkList = null;
CandidatesQueen = null;
min-k = 0;
for(int i=0;i<m;i++){
  //peform next sorted access to L[i] in rount-robin mode
<d,s(d)[i]> = L[i].getNext();
high[i]=s(d)[i];
  threshold = 0;
    for(int j=0;j<m;j++){
       if(j!=i){
         //peform random access for d's soore s(d)[j] in    L[j]
           sccore(d)+=s(d)[j];
       }
     threshold += high[j];
    }
    if(score(d)>min-k){
       TopkList.removeMinkItem();
       TopkList.insert(d);
       min-k = TopkList.getMink();
    }else if(threshold <= min-k)
        return TopkList;    }
```

图 3.1　TA 算法伪码

```
IndexList L[m];    // (L1,...Li,...Lm)
QueryItem t[m];    //(t1,...ti,...tm)
TopkList = null;
CandidatesQueen = null;
min-k = 0;
for(int i=0;i<m;i++){
  //peform next sorted access to L[i] in rount-robin mode
<d,s(d)[i]> = L[i].getNext();
Worstscore(d) += s(d)[i];
E(d) = E(d)∪{i};
high[i] = s(d)[i];
CA:consider RA on Lj for j∈E(d)  according  to  cost  model
for j∈E(d) {
    Worstscore(d) += s(d)[i] ; //perform RA for d in Lj
    E(d) = E(d)∪{j};
}
bestscore(d) = worstscore(d) + Σ_{j∈E(d)} high_j
if(worstscore(d)>min-k){
      d'=TopkList.removeMinkItem();
      TopkList.insert(d);
      candidate.remove(d);
      min-k = TopkList.getMink();
if(bestscore(d)> min-k){
          Candidate.insert(d');
}else if(bestscore(d)>min-k){
      candidate.update(d);}
else candidate.remove(d);
if (TopkList.size() = = k & bestscore(candidate.top())<=min-k)
    return TopkList;
}
```

图 3.2　CA 算法

3.2 基本的评分规则

在解决 XML 数据的 top-k 查询处理问题之前，首先要了解对各个候选项的相似度评价规则。对于某两个 XML 元素或文档之间的相似度评价，与 XML 查询条件相对应，包括内容条件和结构性限制条件两方面。

一、内容条件

首先来定义当一个元素 e 满足了在查询语句 about 符号中出现的关键字条件时获得的局部得分。当且仅当一个元素 e(即在 XML 文档树中的一个结点)匹配标记名称且以 e 为根结点的子树包含查询关键字的时候，称此元素满足这个标记－语词内容条件。以某个元素为根结点的子树中包含的所有语词称为这个元素的全内容文本。更准确地说，一个元素的全内容文本包含它自身以及所有后代的文本内容。另外，定义以下统计数据：

• 全内容语词频率，ftf(t,e)，即语词 t 在元素 e 的全内容文本中出现的次数。

• 标记频率，N_A，具有标记名 A 的元素在整个查询资料库中出现的次数。

• 元素频率，$ef_A(t)$，具有标记名 A 且其全内容文本中包含语词 t 的元素在整个查询资料库中出现的次数。

规定一个语词内容条件的形式为 A=t，A 是一个标记名，t 是一个需要出现在该元素的全内容文本中的语词。根据 Okapi BM25 模型[10]，可以得到内容条件评分函数：

$$score(e, A=t) = \frac{(k_1+1)\, ftf(t,e)}{K + ftf(t,e)} \log\left(\frac{N_A - ef_A(t) + 0.5}{ef_A(t) + 0.5}\right) \qquad \text{(公式 3-1)}$$

其中 $K = k_1((1-b) + b\dfrac{\sum_{s \in full\ \ content\ \ of\ \ e} ftf(s,e)}{avg\{\sum_{s'} ftf(s',e') \mid e'\ \ with\ \ \ tag\ \ \ A\}})$ (公式 3-2)

当查询语句中的 about 操作符中有多个关键字关联到元素 e 的时候，元素 e 的累加得分可以通过所有单个标记－语词条件得分的加和得到：

$$score(e,q) = score(e, A[about(.,t_1,...t_m)]) = \sum_{i=1}^{m} score(e, A=t_i) \qquad \text{(公式 3-3)}$$

二、结构条件

评价一个候选元素与查询语句中结构性约束条件的相似度的方法是，统计这个元素满足导航约束条件的数目，为每一个导航性约束条件的满足赋予一个常量得分 c。定义满足一个导航约束条件的含义为，如果文档 d 中的某个元素 e 具备在查询解析并完成传递扩展之后得到的有向无环图中所有从 e 引出的边，则称元素 e 满足这个导航条件。其中的常量得分 c 是一个可以设置的常量，可以通过改变 c 的相对大小来改变查询处理中对结构性约束的重视程度。

以上分别讲述了一个元素对于查询语句中内容和结构条件的评分规则。当评

价一个元素的相似度的时候，需要将内容得分和结构性得分相加。此外，一个文档对于某个查询语句的相似度评分等于这个文档中所有元素的相似度评分的最大值。

3.3 基本的 top-k 查询处理

具有以上的基础知识之后，本节开始分析 TopX 的基本查询处理过程。为了突出重点，只阐述对 CAS 查询(同时包含内容和结构查询条件)的处理，因为它是 XML 信息检索中具有代表性的查询类型。不失一般性，假设查询语句中包含 m 个内容条件、n 个结构限制条件，L_i 到 L_m 为标记—语词项的索引链表。

TopX 针对分解后的查询对所有表示内容条件的标记—语词索引链表进行交错式的顺序扫描，每一步，搜索引擎首先读取 b 个连续的索引项(b 是可以设置的参数，通常在 100 到 1000 之间)。这 b 个索引项属于同一文档，在同一索引链表中，通常包括一到多个元素块。然后将这些元素块与之前在其它索引链表中扫描过的相同文档中的部分结果进行哈希连接。哈希连接在内存中进行，并即时使用元素的 pre/post 编码来检测查询中的导航限制条件。最后，对于元素的得分进行累加并更新到一个全局的得分变量中。值得注意的是，由于侧重于代价较低的顺序扫描，导致不能确知一个候选项的最终得分，所以 TopX 不仅对于中间 top-k 结果，也对于所有可能进入 top-k 结果的候选项采取了一些形式的内部记录和优先队列管理。

对 m 个索引链表进行扫描的时候，查询处理器搜集查询结果的候选项并将它们记录在两个优先队列中：一个存当前的 top-k 结果项，队列中的项按照 worstscore 的降序排序，另一个存那些可能进入最终 top-k 结果的候选项，队列项按照 bestscore 的降序排序。前一个队列中只包含 worstscore(d)>=min-k 的项，后一个队列包含 worstscore(d) <=min-k 但 bestscore(d)>min-k 的项。TopX 核心查询处理器记录着以下状态信息：

• 每个索引链表 L_i 中的当前指针位置 pos_i，

• 当前指针位置的得分 $high_i$，作为链表当前项之后对象未知得分的上界，

• 一组当前 top-k 项，d_1 到 d_k(重新编号以反映它们当前的排序关系)和一组当前候选队列 Q 中的文档集合 d_j,其中 j=k+1…k+q，以及一个为每一个对象建立的以下信息的数据结构：

• 一组已经评价过的查询维度(即标记－语词索引链表)E(d)，其中的 d 已经在顺序扫描或随机查找中出现过，

• 一组未评价的查询维度 $\overline{E(d)}$，其中 d 的得分暂且未知，

• d 的总得分的一个下界 worstscore(d)，以及一个上界 bestscore(d),其中

$$bestscore(d) = worstscore(d) + \sum_{i \in \overline{E(d)}} high_i \qquad \text{(公式 3-4)}$$

注意，bestscore(d)并没有真的存在内存中，而是在需要的时候根据 worstscore(d)和当前的 highi 计算出来的。

与原始 CA 算法中考虑的纯文本不同，一个文档 d 的 worstscore(d)界不能简单的从已评估过的内容条件中获得，另外还需要考虑结构上的限制条件，这就使得 worstscore(d)的计算更加复杂，后面的章节中将涉及到这个问题。

除了以上的信息之外，在每一步的扫描动作之后都需要获得以下信息：

除了以上的信息之外，在每一步的扫描动作之后都需要获得以下信息：

• 当前 top-k 结果中的最小 worstscore，称为 min-k，作为算法的终止阈值，

• 对应每一个候选项有一个得分差 $\delta(d) = \min-k - worstscore(d)$，表示 d 若要进入当前的 top-k 结果需要再增加的得分。

区分 top-k 队列与候选项队列的条件依然是 top-k 队列的第 k 项的最低分至少不低于候选项队列中所有项的最高 worstscore。当候选项队列中的最大 bestscore 不高于当前 top-k 队列中的最小 worstscore 的时候，算法就可以安全终止，并得到正确的 top-k 结果，即：

$$\underbrace{\min_{d \in top-k} \{worstscore(d)\}}_{=\min-k} \geq \max_{d \notin top-k} \{bestscore(d)\} \qquad \text{(公式 3-5)}$$

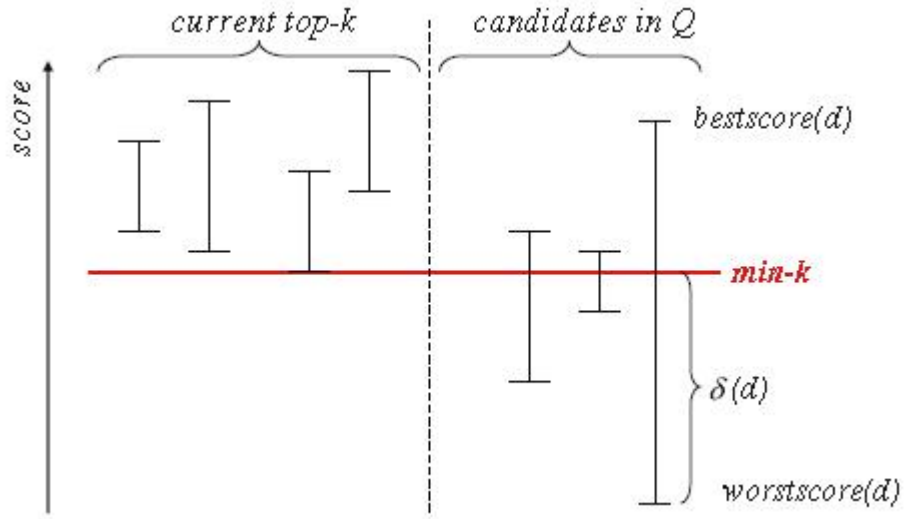


图 3.2 top-k 及候选项记录

公式 3-5 可以作为一个 min-k 阈值测试条件。更一般的情况是，无论什么时候当队列 Q 中有一个候选项的 bestscore 不大于 min-k 的时候，这个候选项就可以被安全的删除。当候选者队列为空的时候，提前终止是有效的 top-k 查询处理的一个目标，提前删除候选项以降低维持队列的内存消耗也是一个同样重要的目标。图 3.2 描述了中间 top-k 结果队列和候选项队列的对应记录图。

以上介绍了基本的 top-k 查询处理过程。另外需要说明是，这个过程在 TopX

中是以一种多线程的分层结构实现的。

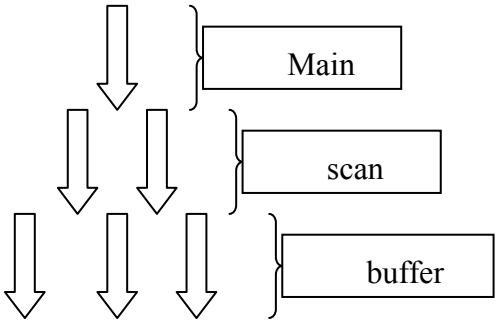


图 3.3　多线程分层结构图

图 3.3 显示的就是 TopX 多线程的分层结构，其中最上一层是主线程，它的主要作用是维持下层几个扫描线程的同步、维护队列(候选对象队列和 top-k 队列)以及进行访问调度和候选对象删除等。中间的扫描线程则主要负责对索引链表进行扫描，它的数目有可能对应着查询维度或根据顺序访问调度器来安排。以上这两层都没有进行实际的磁盘操作，扫描线程读取的索引链表由最底层的缓存线程提供，缓存线程可以在整个查询处理过程中进行持续的异步磁盘操作，以消除实际 I/O 性能对查询处理效率的影响。

本章分析了一个基本的 top-k 查询处理过程，第四章和第五章将深入探讨 XML 信息检索中的一些关键问题。

第四章　　对内容条件的处理

第四章和第五章将分别对查询语句中内容条件和结构条件的处理过程进行分析。虽然这种划分存在少许的不合理性，比如一个标记—语词项就同时包括这两种条件，但是对于整个查询处理过程的阐述来说，这样做是有益的

4.1　一般内容条件的处理

对于出现在查询语句 about 操作符中的一般关键字条件(内容条件)的处理过程，与传统的关键字查询搜索引擎有很多类似的地方，首先要经过一个去除停用词、词干提取的解析过程[8]，然后使用解析过的新查询语句对存在一个数据库管理系统中的标记－语词索引进行扫描比照，根据相似度评价模型得到各个候选项的相似度评分，并返回排序结果。

在结构化查询中出现的内容语词条件可以以两种模式进行评估，即强连接性模式和宽松模式。在前一种情况下，查询条件中所有的语词都必须出现，但是不同的匹配程度也会导致不同的得分。这样，一个匹配语词的重要性可以影响它的得分或最终排序，但是不包含某个特定语词的文档(或子树)会被立刻去除。在后一种情况下，一个结点只需在它的内容中至少出现一次 t1,t2…中的某个语词，就满足了一个”/t1,t2…”形式的内容条件。

为了评价内容条件的得分需要使用以下统计数据，即在第三章说明过的全内容语词频率 ftf(t,e)、标记频率 NA、和元素频率 efA(t)。现在来考虑一个 形式的内容条件，其中 A 是一个标记名，t1 到 tm 是需要出现在子树全文本内容中的语词。一个具有标记名 A 的结点 n 相对这样一个内容条件的得分应该包含以下特征：

- 语词 t_1 到 t_m 的 ftf 值的单调累加，以反映出语词对结点内容的相似度，
- 搜索语词的专门性，建立在所有结点标记的统计值 $ef_A(t_i)$和 N_A 基础上，
- 在其全内容中包含搜索语词的结点 n 的子树的紧密度。

之后就可以得到一个结点 n 相对于查询条件 $A//"t_1...t_m"$ 的得分公式 4-1：

$$score(n,//A[t_1...t_m]) = \frac{\sum_{i=1}^{m} relevance_i \cdot specificity_i}{compactness(n)}, \qquad \text{(公式 4-1)}$$

其中的 $relevance_i$ 反映出 ftf 的值， $specificity_i$ 的值从 $ef_A(t_i)$和 N_A 的值得到，紧密度考虑的则是子树或元素大小的正规化长度。值得注意的是 specificity 考虑的是 XML 特有的组合标记－语词频率而非全局语词统计数据，这样就可以根据一个特定元素类型的专门性来给单个标记－语词对赋与不同的权重，将 IDF 项的作用映射到 XML 数据的处理上，这也是概率信息检索中的常用技术。

可以将上面的公式细化为一个简单的 TF*IDF 风格的度量公式。但是，根据

从文本信息检索中得到的一个重要经验，语词频率和元素频率的值对相似度结果的影响应该被亚线性弱化，以防止相似度评价偏好那些罕见语词以及出现频率高的短元素。同样，上面公式中紧密度也应该使用元素大小的一个弱化形式。为了满足这些方面的考虑，TopX 为 XML 数据采用了被广泛使用的具有经验优良特性的 Okapi BM25 评分模型，见公式 3-1 和 3-2。

4.2 高代价谓词

4.2.1 高代价谓词的概念

以文本谓词的形式，如短语(“”)、强制语词(+)、否定(－)，出现的辅助查询提示能够显著的提高一个信息检索系统的检索结果质量。在一个基于 top-k 的查询处理器中加入这些谓词的难点在于，如何有效实现这些附加的查询限制并采用合适的顺序和随机访问调度规则来适应这些谓词的出现。

本节中主要阐述高代价谓词在 TopX 系统中的处理方法。首先来看一下高代价谓词的定义：如果一个查询谓词不能通过对索引链表单纯的顺序访问来解决，或是单纯顺序访问需要的代价很高，则称这个查询谓词为高代价谓词。

依据这个定义，可以很容易列举一些高代价谓词。例如，短语测试是高代价的，因为它不能够通过一个标准的倒排索引来完成测试。否定也是高代价的，因为作为一个严格的否定测试来说，需要扫描整个索引链表而不考虑文档在这个否定条件中的得分。将一个谓词条件与一个或更多的查询条件联系起来不可避免地要增加索引访问的代价，因此需要一个巧妙的方法来最小化为了解决未检测的谓词条件而调度的代价较高的随机访问的次数。

4.2.2 为高代价谓词进行的随机访问调度

我们将一个候选项的最终得分看作是普通(不受限制的)查询条件与高代价谓词条件的混合结果，它是一个局部得分与一个静态(已知的)谓词得分的组合，该静态谓词得分在候选项满足谓词条件的情况下可以额外获得。在特殊情况下，当谓词用作二元过滤器的时候(例如用作二元短语匹配)，这个额外的得分可能就是在谓词条件上的局部得分。

一方面，一些文档必须被完全探察，这其中至少包括 top-k 结果，以获得它们的相似度评分以及在查询结果中的排序。另一方面，因为 k 取值通常很小，最终只需要一些结果就可以了，没有必要对每个文档都进行完全探察。为了避免这些极高的代价花费，TopX 的目标在于，对每个候选对象的探察都要尽可能早的结束。实际上，大多数的对象根本不需要进行探察，因为它们不可能成为 top-k 结果。为此下面给出必要谓词探察的定义。

必要谓词探察：考虑一个具有评分函数 aggr、检索大小 k 的排序查询，如果根据 aggr 的 top-k 结果不能通过不进行探察的其它任何算法得到的话，则对一个

候选对象 d 的某个谓词 $i \in p(d)$ 的探察称为必要的，且与其它的探察结果无关。

为了进行与倒排链表查找异步的、专对高代价谓词进行探索的增量式测试和调度，需要在候选对象的数据结构中加入一个附加位向量：P(d)，一组 d 尚待测试的谓词维度。P(d)为查询结构所定义，初始值包含在倒排链表扫描中所有候选项遇到的谓词维度的相同子集，即 $p(d) \subseteq \{1,...,m\}$。定义一个得分差 gap(d)，这是一个候选对象若满足所有已知的谓词条件会得到的得分。

$$gap(d) = \sum_{i=1}^{m} s_i(d) \quad for \quad i \in E(d) \cap P(d) \qquad \text{(公式 4-2)}$$

同时，根据由于 p(d)域的存在而导致的不确定性，为保持候选项预测得分上下界更新时的单调性，推导出一个新的上下界公式：

$$worstscore(d) = \sum_{i \in E(d) \cap P(d)} s_i(d)$$

$$bestscore(d) = \sum_{i \in E(d)} s_i(d) + \sum_{i \in E(d)} high_i \qquad \text{(公式 4-3)}$$

在这种方式下，对于最低分的预测更加保守，这是因为，即使在某个倒排链表中发现某个候选对象的存在，但是这个对象对于谓词条件，例如一个短语，最终也不一定会匹配。对最高分的预测依然不变，因为即使还没有测试过某个谓词条件，候选对象的最高得分也不会超过之前的预期。同时，需要保证在查询处理过程中的每一步中，对上下界的更新都是单调的。

现在就可以定义一种调度规则，只有在以下条件满足时才对所有的 $i \in p(d)$ 调度随机访问，即 $worstscore(d) + gap(d) > \min - k$ (公式 4-4)

这个调度规则与之前的必需谓词探察的定义完全吻合，它尽可能将索引访问集中在代价较低的顺序访问上。

值得注意的是，当获得更多关于候选对象的信息 $i \in E(d)$ 的时候，gap(d)的值和 worstscore(d)的值都会增加，因此这个调度规则对大多数候选对象来说，倾向于在查询处理的后期阶段做出调度决策，也就是说，只有当确信随机访问的执行会使得这个候选对象进入 top-k 结果并同时增加 min-k 阈值的时候，才会为候选对象 d 调度代价较高的随机访问以获得未解决的谓词条件的得分。这样，只有那些最有可能进入 top-k 结果的候选对象的会被探察，对大多数候选对象来说 gap(d)+worstscore(d) 永远不会超过 min-k。还有一点，一旦满足 bestscore(d)<=min-k 条件的时候，对多个谓词条件进行测试的随机访问序列就会被中断，因为这个候选项已经不可能满足足够多的谓词条件，并且会被从队列中删除。而那些出现概率很低的谓词，可能会导致谓词测试的数量增多，从而降低

算法的性能。

4.2.3 几种高代价谓词的处理方式简述

一、强制语词

假设 $M \subseteq \{1,...,m\}$ 是一组用"+"标记的内容条件，表示对应的语词必须出现在结果当中。定义一个文档 d 中某个候选元素 e 的累加得分为：

$$score(d,q)=\begin{cases}\sum_{i=1}^{m}\alpha_i(s_i(d)+\beta_i) & d\in L_i\\ 0 & otherwise\end{cases} \quad \text{(公式 4-5)}$$

其中的 α_i 表示语词权重，由于在局部得分 $s_i(d)$上简单地乘以这个系数并不能充分强调文档中该语词的重要性，因此为每个强制语词的得分增加一个常量 β_i，引导在索引扫描时优先考虑强制语词的索引链表，并弱化累加函数在查询维度上的补偿功能。如果 β_i 的取值较大的话，例如对 $i\in M$ 有 $\beta_i=1$，包含强制语词条件的元素就可能进入最终的 top-k 结果而不论其局部得分 $s_i(d)$的大小，这表示可以通过改变 β_i 的取值来改变强制语词对于整个查询和结果排序的重要程度。

二、否定词

对于非强连接性的即宽松查询来说，否定词的意义并不是很大，通常的处理方式是一个否定语词的出现会导致该候选项的一个得分惩罚，但即使是在一个强连接性的查询中，也没有必要全部删除那些包含了某个否定语词的候选项。因此，一个否定语词的出现并不一定会导致该候选项与查询无关，如果该候选项对别的内容相关的查询条件匹配很好的话，它依然可以成为 top-k 结果。相反如果严格执行否定的话可能会陷入大幅降低查全率的危险。

因此，与强制语词搜索条件相反，否定语词的评分是固定且与该语词的实际内容评分 $s_i(d)$无关的，与 XML 评分模型中的结构条件评分规则类似，如果一个候选项不包含某个否定语词就会得到一个附加的静态得分。假设 $N(d)\subseteq P(d)\subseteq\{1,...,m\}$ 是一组标记了"－"的查询条件，则一个候选项 d 的累加得分定义为

$$score(d,t_i)=\begin{cases}\alpha_i(s_i(d)+\beta_i) & for\quad d\in L_i\\ \alpha_i\beta_i & for\quad d\in N(d)\wedge d\notin L_i\\ 0 & otherwise\end{cases} \quad \text{(公式 4-6)}$$

其中的 $\beta_i=1$ 对 $i\in M$ 或 $i\in N$。需要注意的是需要至少一个非否定查询条件

作为基础执行顺序扫描。

按照前面讲过的高代价谓词的调度规则，如果一个候选对象在一次成功的否定语词测试(即其不包含该否定语词，获得未匹配而给予的静态得分)之后就可以进入 top-k 结果，那么就为这个候选项调度对否定条件 $i \in N(d)$ 的倒排链表 Li 的随机查找。

三、短语匹配

为了进行短语匹配，TopX 在一个辅助的数据表中存储了所有语词的偏移位置。与在高代价谓词中描述的一样，短语条件只通过对偏移表的随机访问来解决，相应的调度规则之前已经说明。唯一不同的是，需要决定一个候选项是否将其对短语相关条件的附加得分累加到总体 worstscore(d)中，以便确定它在 top-k 中的排序。为了保持这些得分累加函数在预先计算内容得分中的单调性，在目前的实现中只将短语查找看作一种二元过滤器。

本章分析了 TopX 中内容条件的查询处理，其中省略了系统中包含的查询扩展等内容。第五章将分析结构条件的处理方式。

第五章　对结构条件的处理

5.1　对结构条件的处理方法

TopX 的基本原则就是尽可能推迟代价较高的随机访问，并只对最可能进入 top-k 结果的的候选项进行随机访问。然而，提前进行路径条件测试也是有益的，可以去除那些不满足条件但具有高最低分预测值的候选对象。而且，在一个规定不满足路径条件会导致得分处罚的查询模型中，积极地测试路径条件可以提高一个候选对象的最低分预测，从而潜在地改进 min-k 阈值并导致更多的候选者删除。在 TopX 中只在特定时刻即优先队列重建时考虑进行随机访问。在这个时刻，TopX 考察每个候选对象 d 并决定是否为其调度随机访问来检测待解决的路径条件，或查找其内容条件的缺失得分。

TopX 中为这个 XML 特定的调度规则开发了两种不同的决策方法。第一种，称为 Min-Probing。由于一个文档每满足一个结构性查询条件时会带来一个静态的得分 c，而这些结构性条件的检测只通过执行对 TagsRA 索引表(2.2.3 节中介绍的结构索引)的随机查找来实现，也就是说结构性条件根本不能通过顺序访问来解决。Min-Probing 调度器单纯在其已知的最低分基础上检测所有将要进入中间 top-k 结果的候选对象，不考虑这些这些对象的预期得分或其剩余结构查询条件的选择度。这是一种简单安全的方法，能够保证在算法执行的任何时刻所有的 top-k 对象的路径条件都被完全检测过，但是它不是面向代价的方法，如果结构部分的选择度很低的话可能会导致在候选项上调度过多的查找。

第二种方法，称为 Ben-Probing。这个算法的主要特点是面向代价，它考虑随机和顺序访问的 CR/CS 代价比率以致力于顺序访问和随机访问代价的平衡。这个代价模型可以与基于内容的调度器合并来触发对一个给定候选对象的结构和内容条件的查找。此算法的难点在于，如何根据候选对象的评估得分和未检测的查询内容和结构条件的组合选择度来安排合适的概率排序进行随机查找。

5.2　Min-Probing

Min-Probing 通过只对最可能进入 top-k 结果的候选对象进行探察来达到最少量的随机访问。由于不对结构性查询条件如标记序列、分支路径条件进行顺序访问，因此可以采用 4.2.1 节中提到的高代价谓词的概念来表示这些导航查询条件。满足如下条件的候选对象会被调度随机访问：

$$worstscore(d) + o_j \cdot c > \min - k \qquad \text{(公式 5-1)}$$

其中 o_j 是候选对象 d 未检测的结构条件的数量，c 是根据我们的评分模型推导的 d 在每一个结构条件的满足之后得到的静态得分，o_j*c 相当于高代价谓词中的得分差 gap(d)，即 d 在已知得到的最低分之外因满足谓词条件而累加的得分。

这样一来，如果 d 已经有足够高的最低分 worstscore(d)，只要结构条件满足就可以进入 top-k 结果队列的话，就为其调度一系列的随机查找。如果在随机查找之后发现 d 的最高分 bestscore(d)低于当前的 min-k 阈值，就可以安全的丢弃这个候选对象。这么做的正面效果在于，top-k 索引链表中只包含那些所有的结构条件都被验证过且被证明有足够高的得分使其进入当前的 top-k 链表中的对象。

这种调度策略最适合用于查询目标元素的选择度低且支持元素的选择度高的组合，也就是说，排序主要取决于内容相关的查询条件，对结构条件的随机访问不会经常失败。实际上，在 INEX 中的大多数查询都接近这种类型。同时，Min-Probe 方法也适用于基于内容的 SA 和 RA 调度，这种评估方法代价较低，可以用来在每次顺序的块访问之后对每个将要进入 top-k 结果的候选元素进行测试，因此它是 TopX 核心查询处理器的一部分。图 5.1 就是 Min-Probe 方法的伪代码。

```
IndexList L[i];    Batchsize b[i] ;    Pos[i] = 0;
While(L[i].hasNext()){
    //perform next sorted access to L[i]
    <docid,score> = L[i].getNext();
    d = getItem(docid);
    s(d)[i]    = score;
    E(d)    = E(d) + {i};
    high[i] = score;
    pos[i]++;
    //update worst- and bestscore bounds
      worstscore(d) = ∑_{i∈E(d)} α_i(β_i + s_i(d))
      bestscore(d) = worstscore(d) + ∑_{j∈E(D)} α_j(β_j + high_j)

 worstscore(d) + o_j□c > min−k

    if( worstscore(d) + o_j□c > min−k ){

      d'=TopkList.removeMinkItem();
      TopkList.insert(d);
      candidate.remove(d);
      min-k = TopkList.getMink();
if(bestscore(d)> min-k){
          Candidate.insert(d');
}else if(bestscore(d)>min-k){
```

```
        candidate.update(d);}
    else candidate.remove(d);
    if (TopkList.size() = = k & bestscore(candidate.top())<=min-k)
        return TopkList;
    }
```

图 5.1 Min-Probe 算法伪代码

5.3 Ben-Probing

在介绍对 XML 数据结构性条件随机访问调度的 Ben-Probing 算法之前，需要先简单阐述一下在纯文本数据检索上使用过的一些算法作为预备知识。一、Last-Probing 算法，其核心思想是将查询处理过程中对索引链表的访问过程分为严格的两个阶段，第一阶段只进行顺序扫描而完全不使用随机扫描，然后在第二阶段执行一些必需的随机访问来识别出最终的 top-k 结果，并完成算法。二、Ben-Probing，此算法在 Last-Probing 算法的基础上扩展了一个概率代价模型来评估调用 RAs 对一组最可能进入 top-k 结果的候选对象进行访问与继续在索引链表上执行 SAs 二者之间的优劣，这种代价的比较以每 b 步的顺序扫描为周期执行，即当需要决定是否执行下一个批量的顺序扫描的时候，为候选队列或当前 top-k 队列中的每个文档 dj 计算这些代价，然后根据代价比较的结果选择为单个的文档调用随机访问或继续批量为多个候选对象执行顺序扫描并简单累加各个候选对象的 RA 代价。

Ben-Probing 使用一个代价分析模型来支持对访问调度的决策。假设一个候选对象 d 有 o_j 个未检测的结构查询条件(至少包含一个路径或分支限制条件)，在第 i 个索引链表中仍需要进行扫描的索引项数目为 li，相对应的存在于明确相关统计数据中的重叠链表为 l_{ij}，优先级队列中的文档数目为|Q|，在索引链表上执行的一轮批量顺序访问的大小为 $b = \sum_i^m b_i$。一个候选文档对象 d，已在标记—索引链表集 E(d)中被检测过，尚未在链表集 $\overline{E(d)}$= [1..m]－E(d)中被检测，它进入最终 top-k 结果的概率 p(d)通过组合的得分预测器和选择度评估器来进行预测。其中得分预测器基于标记—语词得分分布和标记—语词对的索引结构，这二者同时建立在预先计算的全内容评分模型上。选择则是进一步在文档级别被确定。这种方法将已有的对纯文本倒排链表的调度和概率删除假设直接转化到 XML 运用中，使得不同的软件组件之间具有兼容性。为了将已存在的方法应用到一个真实的已知结构的调度器中，需要重新定义选择度评估器的含义以包含 XML 特有的模式。

5.3.1 结构查询条件的选择度评估器

按照前面的选择度评估器假设，需要将 XML 风格的查询结构分解为以下几种基本的结构查询模式：

- 标记－语词对：针对内容相关条件的合并的标记－语词对，
- 子结点(后代)：具有传递扩展的后代关系的标记对，
- 分支：具有传递扩展的后代关系的的分支路径元素的标记三元组。

现在对余下的 oj 个关联一到多个路径或分支模式的结构查询条件的选择度进行评估。一种方法是根据预先计算的祖先－后代结点和分支路径元素在语料库中的频率来执行评估。这是一种简单形式的 XML 摘要表示，但是实验表明这种方法对于候选对象删除以及随机访问调度的效果都非常明显。以下通过一个实例来说明结构查询条件选择度评估器的工作过程。假设存在一个包含低选择度结构条件的查询语句：

//article[//sec[about(.//,"XML retrival") and about(.//bib,"W3C")]]

//par[about(.//,"native XML database")]

对应的查询解析得到的 DAG(有向无环图)如图 5-2 所示。

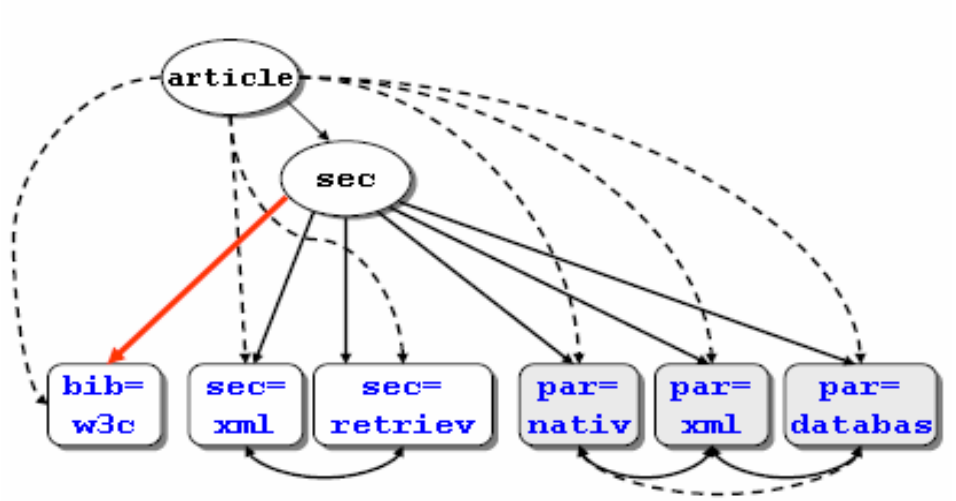


图 5.2 查询 DAG

每一个查询条件(或查询 DAG 图中的结点)都可以与一或多个以上列举的三种结构性查询模式之一相对应，例如内容相关的标记－语词条件 par=xml 与两个路径 article//par 和 sec//par 及一个分支 article[//bib]//par 相对应。为了做出基于结构的调度决策，通常考虑一个无重叠结构的候选对象的未检测结构路径和分支模式的最大子集。例如，若一个部分检测过的候选对象存在三个未解决的查询条件 sec，bib=w3c，p=xml，并不将这几个条件对应的结构性查询模式的选择度相乘，而只取 sec[//bib]//par 的选择度作为选择度评估使用。图 5.3 是一个 INEX 测试数据集合相对实例查询的基本结构性查询模式及其对应选择度表。

Type	Pattern	Selectivity
Twigs	article[//bib]//par	S1=0.682
	article[//bib]//sec	S2=0.613
	sec[//bib]//par	S3=0.002
Descendants	article//bib	S4=0.614
	article//par	S3=0.982
	article//sec	S4=0.982
	sec//bib	S5=0.002
	sec//par	S6=0.964
Tag-Term Pairs	bib=w3c	S7=0.007
	sec=xml	S8=0.055
	sec=retriev	S9=0.171
	par=native	S10=0.044
	par=xml	S11=0.055
	par=databas	S12=0.319

图 5.3　基本结构性查询模式及其对应选择度表

在已知以上数据的基础上，尽量以一个低选择度的查询条件开始对一个部分检测的候选对象的随机访问序列，这样该候选对象就可能因为随机访问后得到的 bestscore(d)<=min-k 而被删除，从而节省了对其进行更多查找的花费。

5.3.2　对结构条件进行随机查找的代价模型

BenProbe 的基本思想是对比对标记－索引链表或结构路径条件的索引进行随机访问与继续执行顺序索引扫描之间的代价。对所有这三种类型的代价，都只考虑一个预期浪费代价(EWC)值，它的含义是，根据我们的调度决策会执行但是不会被一个最优的调度决策执行的的预期随机(或顺序)访问的数量，该最优调度决策只为最终的 top-k 结果执行随机访问。

对一个在索引链表集 $\overline{E(d)}$ 中的得分均未知的候选项 d，如果这个候选对象不能最终进入 top-k 结果，会发生 $|\overline{E(d)}|$ 次随机访问的浪费。通过计算 $\overline{E(d)}$ 的卷积直方图，可以得到如下的概率：

$$P[d \notin top-k] = 1 - p(d) = 1 - p_S(d) \square q(d) \quad \text{(公式 5-2)}$$

其中 $p_S(d) = P[\sum_{i \in E(d)} S_i > \delta(d) \mid S_i \le high_i]$ 是一个著名的得分评估公式，但是此处的的 δ (d) = min-k−worstscore(d)−o_j ·c，q(d)是选择度评估器：

$$q(d) = (1 - \prod_{i \in E(d)} (1 - \max_{j \in E(d)} \frac{l_{ij}}{l_j}))$$ (公式 5-3)

我们可以结合所有可获得的信息，包括得分卷积、不同索引链表的选择度及标记－语词对的相关性来计算 p(d)的值。通过分析计算可得到以下的 3 个公式：

• 为检测丢失的标记－语词得分而进行的随进访问具有预期浪费代价：

$$EWC_{RA\text{-}C}(d) = |E(d)| \cdot (1 - p_S(d) \cdot q(d)) \frac{C_R}{C_S}$$ (公式 5-4)

• 对路径和分支条件的随机访问具有预期浪费代价：

$$EWC_{RA\text{-}S}(d) = o_j \cdot (1 - p_S(d) \cdot q'(d)) \frac{C_R}{C_S}$$ (公式 5-5)

• 顺序访问的分摊预期浪费代价：

$$EWC_{SA}(d) = \frac{b}{|Q|} \cdot \sum_{d \in Q} (1 - p_S(d) \cdot q^b(d))$$ (公式 5-6)

在这些公式的基础可以得到我们的调度规则，即当且仅当满足条件

$EWC_{RA\text{-}C}(d) < EWC_{SA}(d) \wedge EWC_{RA\text{-}S}(d) < EWC_{SA}(d)$ 的时候开始对标记－语词得分查找和结构条件的随机访问，其中随机访问相对顺序访问的权重是根据代价比率 c_R/c_S 得到的。实际执行时按内容和结构条件的选择度降序排序，每次执行一次随机访问，那些不能够满足 top-k 的候选对象会尽可能早的被丢弃，而不会对它们执行更多无用的随机访问。

第六章　TopX 的改进

6.1　TopX 的架构改良

TopX 中采用了 xhtml 页面和 java servlet 组合的运行机制。在这种运行机制中，由 servlet 生成 html 页面很麻烦，而且程序结构混乱不利于分析程序运行逻辑和修改程序。因此，本文将其改进为 JSP+Servlet 且采用 Struts 框架的 MVC 架构。

MVC 是模型（Model）、视图(View)和控制(Controller)这三个单词的缩写，MVC 模式的目的就是实现 Web 系统的职能分工[14]。模型是应用对象，没有用户界面。视图表示它在屏幕上的显示，代表流向用户的数据。控制器定义用户界面对用户输入的响应方式，负责把用户的动作转成针对 Model 的操作。Model 通过更新 View 的数据来反映数据的变化。

在原 TopX 系统中由 TopxServlet.java 函数接受请求、处理并返回结果页面，程序逻辑比较混乱。在改进后的架构中，由 index.jsp 接受用户输入的查询语句和查询限制，在 CNTopX.java 有一个 Action 来接受参数，并调用处理函数 TopService.java 进行查询处理返回查询结果，结果映射到页面 CNTopX.jsp 中。整个程序的流程图如图 6-1。

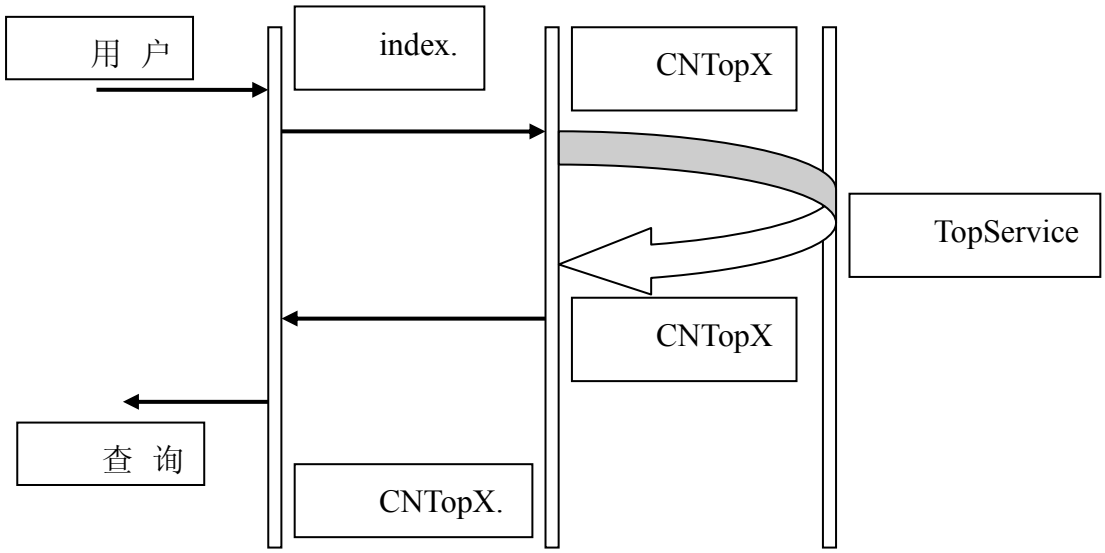


图 6.1　改进后的程序流程图

6.2　TopX 字符编码的国际化和本地化

在原 TopX 系统中，字符统一采用 ISO-8859-1 编码。ISO-8859-1 属于单字节编码，最多能表示的字符范围是 0-255，应用于英文系列。比如，字母 a 的编码为 0x61=97。很明显，ISO-8859-1 编码表示的字符范围很窄，无法表示中文字符。

但是，由于是单字节编码，和计算机最基础的表示单位一致，所以很多时候，仍旧使用 ISO-8859-1 编码来表示[15]。而且在很多协议上，默认使用该编码。TopX 的运行服务器 Tomcat 也是默认使用这一编码。

在对 TopX 字符编码进行国际化和本地化的改进中，首先要解决的就是它的中文编码问题，我们所选择的编码必须能很好的支持中文显示和处理，同时也要兼容英文等其它字符的处理。经过考虑，最终选择的是 UTF-8 编码。UTF-8 是 UNICODE 的一种变长字符编码，已经标准化为 RFC－3629。UTF-8 用 1 到 6 个字节编码 UNICODE 字符。下面介绍 TopX 的编码改进流程。

在程序初始页面 index.jsp 中设定 pageEncoding="UTF-8"，参数读取是在 CNTopX.java 中进行的，需要在此执行以下几个步骤来获取正确编码的参数。

```
String query = request.getParameter("query");
//取得 ISO-8859-1 编码的字节，因为 Tomcat 默认是此编码
byte[]    bytesUTF = query.getBytes("ISO-8859-1");
//用指定的编码将字节转换成字符串，即获得 UTF-8 编码的参数
query = new String(bytesUTF, "UTF-8");
```

其它参数的获取类似，获得正确编码的参数之后，调用查询处理函数，进行查询处理，得到结果后，设定 response.setCharacterEncoding("UTF-8")，跳转到结果页面 CNTopX.jsp，这样查询结果就以 UTF-8 编码的形式展示给用户了。

6.3 TopX 对中文 XML 文档的支持

改进 TopX 使其支持中文 XML 文档处理是最有意义的一项改进，这是 TopX 本地化的重要标志。纵观 TopX 的系统设计和程序流程，除了一些编码的限制(可以改进，已在 6.2 节阐述过)外，并没有针对特定的语言设计，也就是说，查询处理过程本身就支持多种语言，将 TopX 应用在中文上唯一的问题在于，TopX 中没有良好的中文分词程序，无法识别出有意义的中文单词，导致不能得到有效的查询结果，因此如果我们能在 TopX 的查询处理之前加入一个中文分词过程，对其中的中文语词进行有效的划分，就能够拓展 TopX 使其具备中文处理能力。下面就来着重讨论这个问题。

词是最小的能够独立活动的有意义的语言成分。在汉语中，词与词之间不存在分隔符，词本身也缺乏明显的形态标记，因此，中文信息处理的特有问题就是如何将汉语的字串分割为合理的词语序列，即汉语分词。在此，我们使用的分词程序是由中国科学院计算技术研究所在多年研究基础上，耗时一年研制出的基于多层隐马模型的汉语词法分析系统 ICTCLAS(Institute of Computing Technology, Chinese Lexical Analysis System)，该系统的功能有：中文分词；词性标注；未登录词识别。分词正确率高达 97.58%(最近的 973 专家组评测结果)，基于角色标注的未登录词识别能取得高于 90%召回率，其中中国人名的识别召回率接近 98%，

分词和词性标注处理速度为 543.5KB/s[19]。

关于 ICTCLAS 的更多细节，本文不加讨论，在程序改进中只是将 ICTCLAS 当作一个底层调用函数库，我们使用的是使用 JAVA 调用动态链接库的方式调用一个称为 FREEICTCLAS.DLL 的 C++动态链接库(该动态链接库由我和王乐共同编写)，其中用到了 Java 本地接口(JNI)技术，Java Native Interface 技术允许在 Java 应用中集成 C 或 C++构建的模块。现在假设已经存在这样一个函数 ParagraphProcess()，它能够接受包含中文语句的一个中英文单词序列，返回一个去除了中文停用词的由空格分隔的中英文单词序列。我们可以在原程序中加入以下程序段(图 6.2)的处理得到一个经过中文分词的查询语句。

```
System.loadLibrary("FreeICTCLAS");
//加载停用词
StopWordsFilter stopWordsFilter = new StopWordsFilter();
stopWordsFilter.importStopWords("e:\\stopWords.txt");
ParagraphProcess pp = new ParagraphProcess(true,false,true,stopWordsFilter);
FreeICTCLAS.FreeICTCLAS_Init(0, 0);
String inter = data.replace("/", "&"); //因为“//”在下面的调用函数中无法使用，因此暂时替换一下
try {
inter = pp.getResultOfPara(inter);
} catch (Exception e) {
        e.printStackTrace();
    }
String res = inter.replace("&","/");
```

图 6.2　在 StemmerDriver.java 中加入的程序片段

经过正确的中文分词，再加上 6.2 中叙述的字符编码的改进之后，TopX 便可以流畅的处理中文或中英文混合的查询语句了。

第七章　结论

7.1　结论

伴随着 XML 数据量的不断增加，XML 信息检索已经成为一门极具发展前景的领域，XML 信息检索系统的研究也在国际上迅速开展起来。

在介绍了信息检索和 XML 数据管理相关的基础知识之后，本文结合源代码分析了一个 XML 信息检索系统 TopX 的查询处理过程，主要着重于回答以下问题：系统如何确定内容和结构条件的相似度评价体系，系统如何组合内容条件和结构条件的评价，系统如何制定对索引的访问策略来渐进地完成候选对象评估，系统如何得到排序的 top-k 查询结果以及系统如何利用一些改进的技术来提高查询质量和查询效率。

本文在这些细节问题的分析之上描述了 TopX 查询处理的工作流程，阐明了其中的要点、关键技术和实现机制，并对原系统做了一些简单的改进，包括系统架构、字符编码和中文处理的改进。论文工作的主要目标是帮助读者对 XML 信息检索领域世界先进水平的工作有所了解和认识，为自己将来的研究和创新打下基础。

7.2　展望

本文的重点在于分析 XML 信息检索系统 TopX 的查询处理过程。在今后的学习仍需要进一步完成的工作包括：

1、完整的 TopX 系统的分析：在查询处理之外，另外还有建立索引，性能评价等更多的分析工作需要完成，短期的目标就是要掌握一个完整的 XML 信息检索系统的工作流程和其中的关键技术，了解 XML 信息检索设计中的主要问题和常用解决方案，熟悉一些经典的性能评价体系。

2、设计自己的 XML 信息检索系统：分析 TopX 系统源代码的最终目标在于设计一个自己的 XML 信息检索系统，这需要分析和阅读更多的系统源代码和相关论文，汲取精华，加以改进和思考，并动手编程实现一个能够良好工作的 XML 信息检索系统，在此基础上不断加以改进和完善，以追赶国际先进水平。

参考文献

[1] 陈峰．文本数据的分类[EB/OL]．http://www.ieee.org.cn．2004

[2] Pineapplesoft．使用 XML:XPath 2.0 入门[EB/OL]．http://www.ibm.com．2006..

[3] W3C school．XQuery 简介[EB/OL]．http://www.w3school.com.cn/xquery/．2008．

[4] Sukomal Pal．XML Retrieval：A Survey[P]．Computer Vision and Pattern Recognition Unit Indian Statistical Institute，Kolkata．June 30，2006．

[5] Matin Theobald，Holger Bast，Debapriyo Majumdar et al．TopX：efficient and versatile top-k query processing for semistructured data[J]．The VLDB Journal，2008，17：81-115．

[6] W3C．Extensible Markup Language (XML)[EB/OL]．http://www.w3.org/XML/．2008．

[7] DMman．信息检索介绍[EB/OL]．http://bbs.xml.org.cn．2007．

[8] Ricardo Baeza-Yates 等．现代信息检索[M]．北京：机械工业出版社．2005．1-300．

[9] Norbert Fuhr．XML Information Retrieval Achievements and Challenges[R]．Germany：University of Duisburg-Essen．2004．

[10] K.Spark.Jones，S.Walker，S.E.Robertson et al.．A probabilistic model of information retrival : development and comparative experiments[J]．Information Processing and Management 36，2000．779-808．

[11] 王国仁 等，XML 数据管理技术[M]．北京：电子工业出版社．2007．1-150．

[12] Norbert Fuhr．XML 信息检索与评价进展[M]．湖北：湖北辞书出版社．2006．1—30．

[13] Sihem AmerYahia．．XML Retrieval:DB/IR in Theory，Web in practice[Z]，USA Yahoo! Research．2006．

[14] cinc．了解 MVC 架构对于用 Struts 构建的强大的 Web 应用程序很重要[EB/OL]．http://www.chinaunix.net．2002．

[15] 凌峰．Unicode、UTF－8 和 ISO8859-1 到底有什么区别[EB/OL]．搜狐博客>Java 视点．2007．

[16] 秦领．从原理上解决 Tomcat 中文问题[EB/OL]．Linux 联盟．2006．

[17] 段明辉．Java 编程技术中汉字问题的分析及解决[EB/OL]．www.ibm.com /developerworks/cn/java/java_chinese/．2000．

[18] huapingZhang，hongkuiYu，deyiXiong．HHMM-based Chinese Lexical Analyzer ICTCLAS[J]．2end SIGHAN workshop affiliated with 41th ACL，2003 pp.184-187．

[19] 中国科学院计算技术研究所．计算所汉语词法分析系统 ICTCLAS[EB/OL]．http://www.nlp.org.cn/．2002．

[20] 维基百科．XML[EB/OL]．http://wiki.ccw.com.cn/XML．2008．

[21] 那罡．信息检索的黄金时代发布日期[EB/OL]．赛迪网．2006．

附　　录

一、程序运行和安装说明书

运行环境：

1．Java 2 SDK 1.5 或以上版本

2．Apache Tomcat 5.0 或以上版本

3．Oracle database server, Release 9i 或更新版本

4．浏览器最好使用 Firefox 或 SeaMonkey，程序在 IE 中运行有问题

必需 JAVA 包：

1．classes12.jar (the Oracle JDBC driver)

2．jWN.jar (the JWordNet package)

3．colt.jar

4．trove.jar

5．lucene-snowball-2.0.0.jar

6．xmlparserv2.jar

安装过程：

1．编译 TopX 源代码

2．在 Tomcat 的 webapps 目录下创建一个目录 TopX 以及子目录 WEB-INF，WEB-INF 下再创建子目录 classes 和 lib.

3．将源文件中 topxindexer.bat 拷贝到 TopX 目录下

4．将源文件中的 web.xml 拷贝到 WEB-INF 目录下

5．将源文件中的整个 xhtml 目录和内容拷贝到 classes 目录下

6．将编译好的 JAVA 类拷贝到 classes 目录下

7．将 topx_config.xml 和 topx_schema.xml 拷贝到 classes 目录下

8．将所需的 JAVA 包拷贝到 lib 目录下

9．双击 topxindexer.bat，启动建索引程序，建立索引

10．启动 Tomcat，打开 Firefox，输入 http://***:8080/topx 即可开始查询

二、程序中主要代码分布说明

1．程序主函数

de.mpii.topX.servlets.TopXService.java

de.mpii.topX.servlets.TopXServlet.java

2．查询解析主函数：

de.mpii.topX.query.xml.nexi.NEXI_QueryImporter.java

3．提取词干函数

de.mpii.util.StemmerDriver.java

4．top-k 主函数：

de.mpii.topX.qe.Topk.java

其中包括以下函数：

class Topk-->init()

设置初始 min-k,启动多线程，定义多个索引链表，检查谓词，设置调度规则。按结构性选择度的升序排序调度随机访问，按内容条件的升序排序调度随机访问。创建新 top-k 链表和候选对象队列

class Topk-->class TopkThread-->checkMink()

CA 算法，Min-Probing 算法

返回需要调用 RA 访问其结构条件的索引项,更新 top-k 队列和 candidate 队列

class Topk-->class TopkThread-->checkPredicates

检查高代价谓词

class Topk-->class TopkThread-->checkRandom()

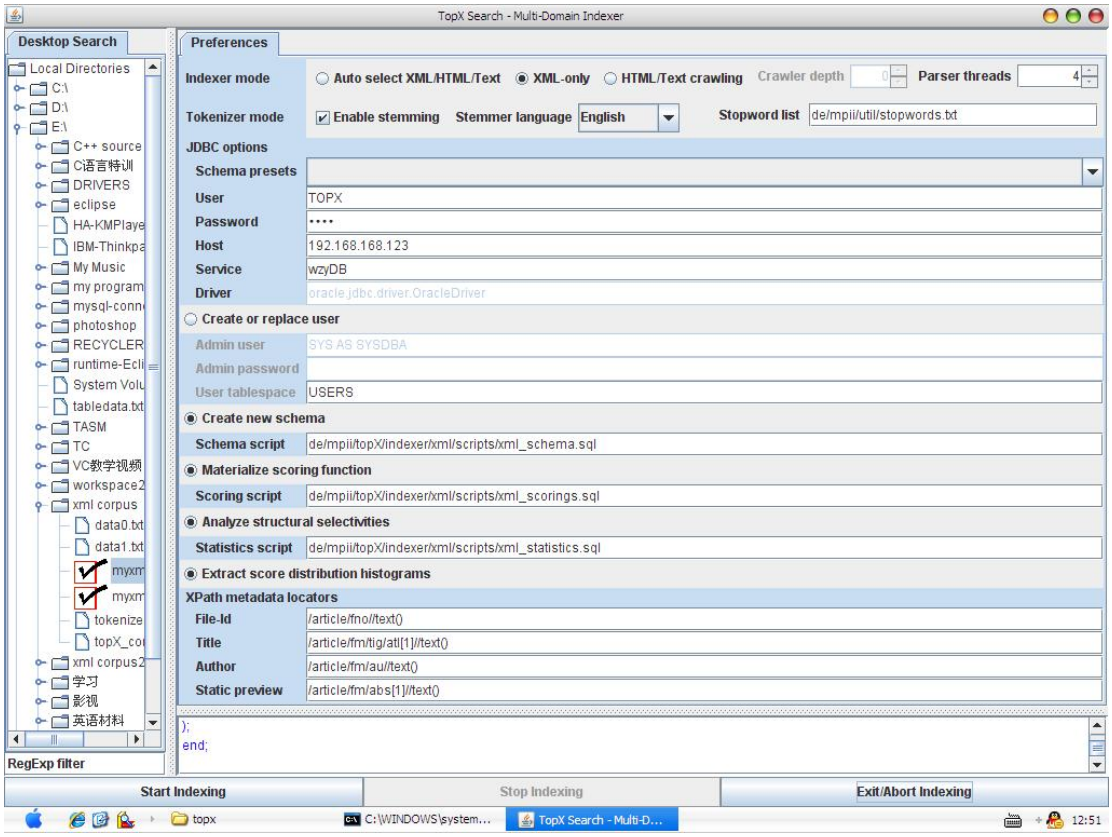
检查是否调度随机访问

5．de.mpii.topX.qe.IndexPruning

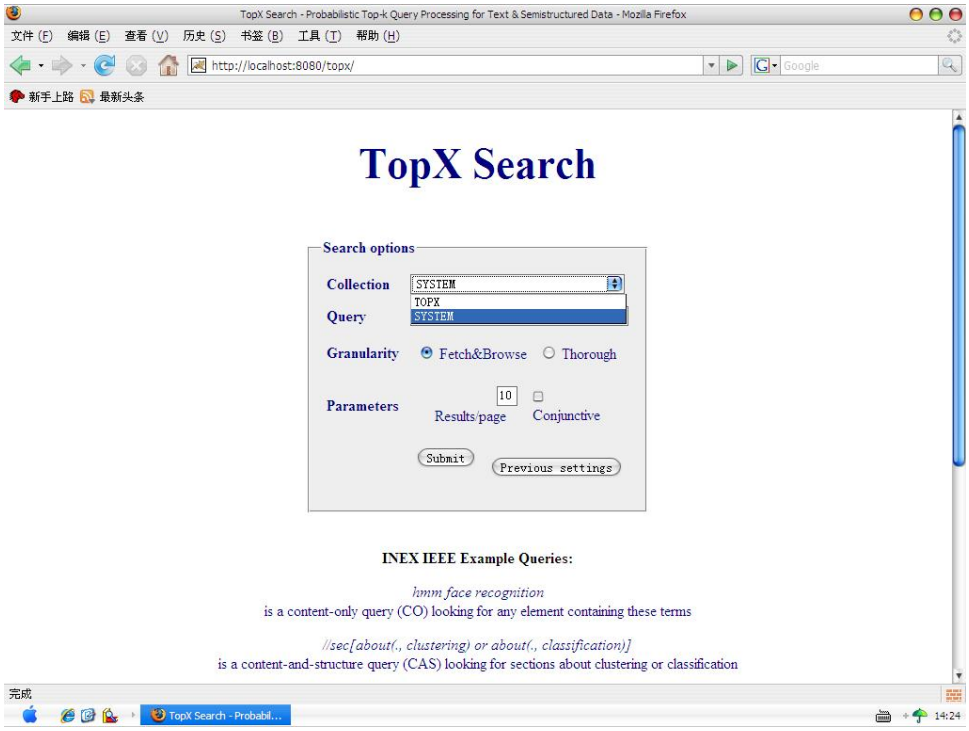
Ben-Probing 算法 提前候选对象删除

三、程序运行过程界面截图

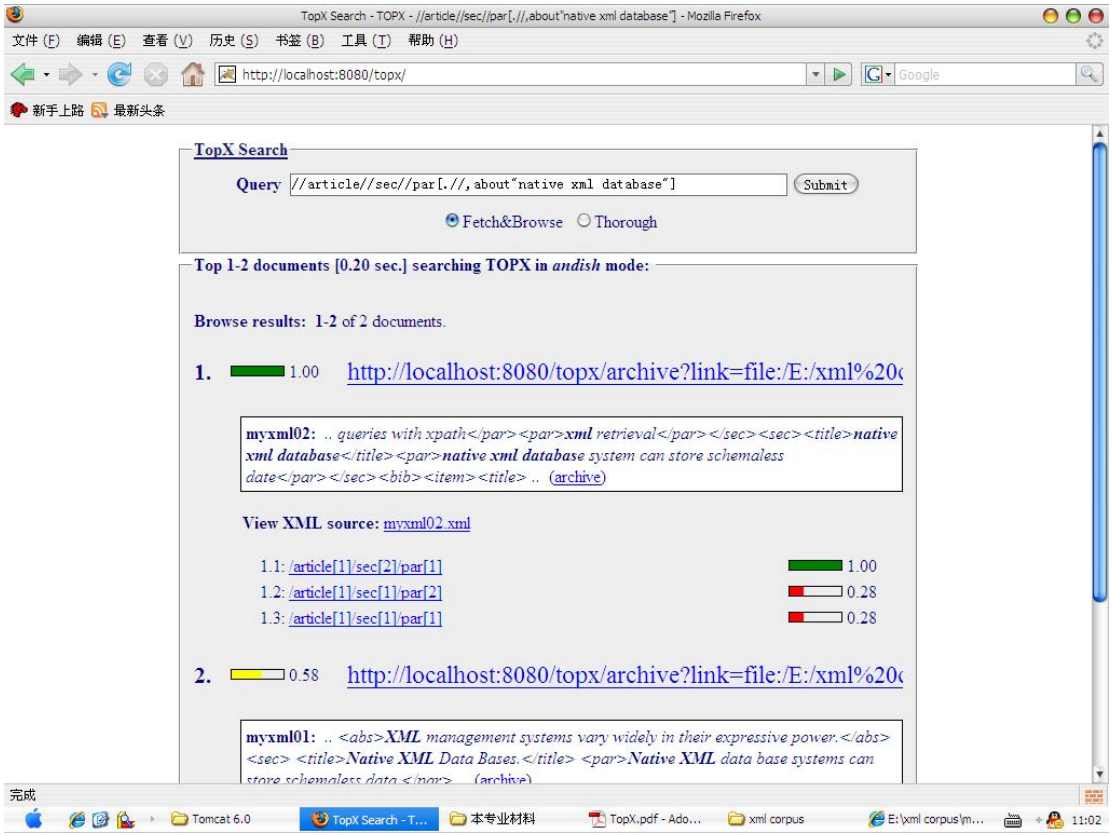
1．建索引



2．选择查询库，输入查询语句



3．查询结果


TopX Search - TOPX - //article//sec//par[.//,about"native xml database"] - Mozilla Firefox
文件 (F) 编辑 (E) 查看 (V) 历史 (S) 书签 (B) 工具 (T) 帮助 (H)
http://localhost:8080/topx/
新手上路 最新头条
TopX Search
Query //article//sec//par[.//,about"native xml database"]
Submit
Fetch&Browse Thorough
Top 1-2 documents [0.20 sec.] searching TOPX in andish mode:
Browse results: 1-2 of 2 documents.
1. 1.00 http://localhost:8080/topx/archive?link=file:/E:/xml%20
myxml02: .. queries with xpath</par><par>xml retrieval</par></sec><sec><title>native xml database</title><par>native xml database system can store schemaless date</par></sec><bib><item><title> .. (archive)
View XML source: myxml02.xml
1.1: /article[1]/sec[2]/par[1] 1.00
1.2: /article[1]/sec[1]/par[2] 0.28
1.3: /article[1]/sec[1]/par[1] 0.28
2. 0.58 http://localhost:8080/topx/archive?link=file:/E:/xml%20
myxml01: .. <abs>XML management systems vary widely in their expressive power.</abs> <sec> <title>Native XML Data Bases.</title> <par>Native XML data base systems can
完成
Tomcat 6.0
TopX Search - T...
本专业材料
TopX.pdf - Ado...
xml corpus
E:\xml corpus\m...
11:02

致　　谢

本文的写作过程中得到了许多老师、同学的指导和帮助。

首先要感谢张坤龙老师，在张老师的指引下我确定了论文的写作方向，理解了我的工作重点。在写作过程中，受到张老师的言传身教，张老师对于科研工作认真的态度，饱满的热情，严谨的学风，都给我留下了深刻的印象，也使得我的论文能够按时按质的顺利完成。

其次要感谢王乐同学，他与我一同学习研究信息检索的问题，经常在探讨中给予我启发，并毫无保留的分享他收集到的资料。

还要感谢实验室的师兄师姐们，在我的实验和论文写作过程中，他们为我提供机器，提供材料，并热情的给予我论文写作的建议。

再次对所有帮助过的人表示感谢！