摘 要

校园网建设迅速发展的今天，多数学校都架设了多台FTP服务器为师生提供大量的数字资源，但从大量数字资源中，找到指定的资源往往需要耗费大量时间。因此我设计研究一种高性能的FTP搜索引擎来帮助用户更快速、更方便的定位到所需的数字资源是非常有必要的，对节省查询成本、提高工作效率有积极意义。

本文论述了FTP搜索引擎开发的背景及目前关于FTP搜索引擎的研究现状。并研究讨论了相关技术的核心原理、机制，对模糊搜索和相关性排序做了研究。详细说明了FTP搜索引擎系统的开发过程和方法。论文首先从功能需求和性能需求两个方面对FTP搜索引擎的需求进行了分析，然后根据需求分析，设计一个基于B/S架构，提供多种查询方式，实现模糊搜索功能，且查询速度较快的FTP搜索引擎。

关键词：Lucene，模糊搜索，相关性排序，FTP搜索引擎

ABSTRACT

The rapid development of the campus network construction today, most schools set up FTP servers to provide a lot of digital resources for the teachers and students, but it often cost a lot of time to find the specified resource from a large number of digital resources. Therefore I design a high performance FTP searching engine to help users more quickly and more convenient get the required digital resources is necessary, and it have a positive significance to save costs of query , improve work efficiency .

This paper discusses on the development background and current research status of FTP searching engine, and introduces related technology of system implementation, and does some research on fuzzy search and correlation ranking And describe the development process and method of FTP searching engine in detail. First, the paper analysis the function demands and performance demands of the FTP searching engine. And then based on the needs analysis, design a FTP searching engine based on B/S structure, which offers a variety of inquiry mode, fast response and fuzzy search.

Keywords: Lucene, Fuzzy search, Correlation ranking, FTP searching engine

目 录

第1章 引言

1.1 研究背景及意义

FTP服务器是Internet上所使用的最主要的服务器之一，在FTP服务器上保存有大量的共享软件、技术资料、多媒体数据等各种各样的文件[1]。每个FTP服务器都有其各自的若干目录，且其目录结构往往都比较的复杂，所以要在FTP服务器找到所需的特定资源时，用户往往需要一个个目录的查看，这并不是一件容易的事情，而要在多个FTP服务器上查找文件则更加困难。FTP搜索引擎可以很好的解决上述的问题。通过FTP搜索引擎将各个服务器的文件信息整合在一起，用户就可以通过FTP搜索引擎快速的找到自己想要的资料。

近几年来，随着Web搜索引擎技术的不断发展成熟，人们对FTP搜索引擎进行了一定的研究和探讨，已经有成果投入使用。目前研究的热点主要集中于FTP搜索引擎体系结构的设计，并借助一些开发环境提出了一些简单的搜索原型[1]：基于PHP和MySQL数据库的简单FTP搜索引擎，该原型直接采用SQL语言查询数据库，并返回结果。优点是设计成本低；缺点是执行效率不是很高，不支持分词。

基于XML Web Service的FTP搜索技术，该方案大胆提出了一种新的FTP搜索引擎解决方案，通过在FTP服务器和检索服务器分别部署Web服务，FTP服务器向检索服务器提供符合XML标准的实时FTP文件信息，可以保证检索结果的时效性，基于Web服务的系统框架为大范围的分布式部署建立了良好的基础[2]。但在现行的网络环境下，大范围的在FTP服务器部署Web服务较为困难。

基于IA(Intelligent Agent，集成代理)和Web Service的FTP搜索引擎设计[3]。该设计方案主要采用在一个自治系统内部(这个系统的划分不一定以单位实体划分，可以使多个互联速度较快的网络一起构成一个自治系统，如中国教育科研网就可以使一个大的自治系统，其内部各成员单位的互联速度是比较快的)部署Search Server，并启动信息搜集Agent，信息处理Agent和信息通信Agent，信息搜集Agent的功能由爬虫(Spider)扮演，各个自治系统Search Server交互各自管辖范围的数据搜集，呈现Peer to Peer的架构。Search Server提供统一的用户接口，用户用客户端软件登陆FTP Search，也可以使用Web浏览器登陆任意的Search Server，由Search

Server为用户指派最近的节点。最大的优点是在搜索的数据量非常大时候，这种设计仍有高效的检索效率，维持着较高的性能。缺点是：实现的技术难度较高。

基于Lucene的FTP搜索引擎的设计。该方案针对目前中文FTP搜索引擎不支持中文分词及动态数据更新的缺陷，采用Lucene这个最大的全文搜索引擎工具包构建FTP搜索引擎，该搜索引擎能够生成标准的XML数据文档，并采用基于字典的前向最大匹配中文分词法在Lucene中动态更新全文索引，该引擎还可以对中英文混合分析和检索[4]。该方案更适合为校园内部提供FTP搜索服务。优点：搜索引擎的系统开发速度快。缺点：在数据量非常大时，性能会受到影响。

1.2 本文的主要工作

本文在研究了搜索引擎相关实现技术的基本原理、处理流程的基础上，结合对FTP搜索引擎的具体需求分析，设计并实现了基于Lucene的FTP搜索引擎。下面是本文的主要工作：

在需求分析阶段，从系统功能需求和非系统功能需求两个方面对FTP搜索引擎的需求进行描述。确定了需要实现如下功能：

1. 搜索功能。在Web前端为用户提供查询功能，能够使用户快捷方便的找到自己指定的资料，支持模糊搜索。

2. 数据采集、更新功能。从FTP服务器采集文件信息，并以一定的策略对数据库进行更新。

3. 数据处理功能。基于Lucene设计索引的数据结构和索引方式，对采集到的文件信息进行索引，使得搜索更为迅速快捷。

4. 后台管理功能。对FTP搜索引擎搜集的FTP服务器信息、文件信息、更新策略等做相关的管理。

在总体设计阶段，根据需求分析，设计并描述了系统的整体功能和总体流程，然后根据系统的特点将其划分为四个主要模块。这四个主要模块是：数据搜集模块、文件信息对比模块、数据索引模块、搜索模块。并分别对每个模块进行分析和设计。

在详细设计阶段，本文具体描述了各个功能模块的设计。并给出了关键模块的实现代码。

1.3 论文结构及安排

本文内容共分六章，章节结构安排如下：

第一章是绪论。主要阐述了FTP搜索引擎系统的研究背景和意义、技术的发展现状以及本文的主要工作和组织结构等内容。

第二章是相关技术及其核心原理。主要对实现FTP搜索引擎所使用到相关技术进行了介绍，并研究了它们的工作原理。

第三章是系统需求分析及总体设计。根据搜索引擎的工作原理结合功能性和性能需求分析，建立系统的基本功能架构。

第四章是系统详细设计与实现。根据需求分析和总体设计，提出了具体的实现方案，描述了本文具体描述了各个功能模块的设计，并详细描述了程序的实现。

第五章是系统测试与运行。首先介绍了整个系统的实现状况，对系统进行了测试，给出了系统的测试结果。

第六章是结束语。对全文内容进行了总结，指出系统待完善之处，并对需要进一步研究和解决的问题做了展望，阐述了下一步需要进行的工作。

第2章 相关技术及其核心原理

2.1 Lucene简介

Lucene是Apache旗下的一个开源的全文索引工具包，用Java语言编写，它为应用中实现全文索引功能提供了基础[5]。Lucene提供了一套具有强大功能的API，可以比较方便的嵌入到各种应用中，实现应用中的全文索引、检索功能。在Java开发环境下，Lucene是一个开放源代码的工具包，就其自身来说，Lucene是最近这几年，比较受欢迎的开源java全文索引引擎工具包。

Doug Cutting是Lucene的作者，也是一名全文索引、检索方面的专家，曾经是Apple的Copland操作系统的成就之一“V-Twin”搜索引擎的主要研发人员，之后在担任Excite高级系统架构设计师一职，目前着手于Internet底层架构方面的研究。Lucene的目标是为各中小型应用程序提供全文检索功能。

2.1.1 Lucene索引过程

Lucene的索引过程如图2-1所示。

1.提取文本:先从数据源里提取出纯文本格式的信息，让Lucene识别该文本信息并建立相对应的Lucene文档。Lucene没有提供有效的方法来从数据源中提取出有效的纯文本信息，这需要开发者自行实现提取纯文本信息的方法。

2.建立Lucene文档:使用提取出的纯文本信息来创建对应的Document实例。该Document实例包含若干个Field实例，Field实例中有两个比较重要的属性：域名和域值，域值的作用是保存纯文本信息中与该域对应的那部分。例如，对于一个公司的Document实例，应该包括公司名、拥有者、股东为域名的各种Field实例。每个Field实例的域值应该存储对应的内容，比如，某个公司名Field实例的域值为“XXXX有限公司”。

3.分析文档:分析(Analysis)是指将域(Field)文本转换成最基本的索引单元——项(term)的过程。在搜索的过程中，项决定了何种文档可以匹配查询条件。分析器封装了分析操作，通过执行这些操作，将纯文本信息转换成词汇单元，这些操作包括了：提取单词，去除标点符号、字母的规范化、去除常用词汇、词干

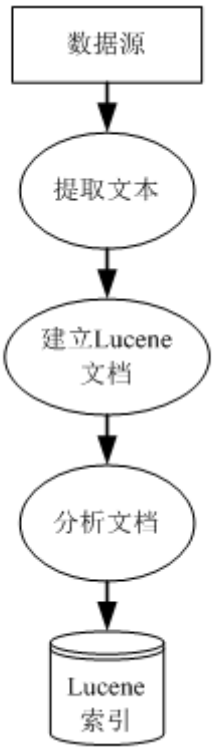


图 2-1 Lucene索引过程

还原，或词干归并。整个处理过程也叫做词汇单元化，从文本流提取出的文本块称为语汇单元（token）[6]。语汇单元和它的域名结合之后就形成了项。建立好Lucene的Document实例之后，就可以进行索引操作了，方法是调用IndexWriter对象的addDocument方法将数据传递给Lucene。对文本进行分析的过程会将域值处理成大量的语汇单元。

对输入的数据分析完成之后，会将分析结果写入索引文件中。Lucene会把输入数据以一种倒排索引（Inverted Index）的数据结构进行存储。那么倒排索引是什么呢？简单来说，倒排索引是经过优化后用来快速回答“哪些文档包含单词X？”而不是回答“这个文档包含了哪些单词？”现在几乎所有的Web搜索引擎核心采用的都是倒排索引技术。

在本系统中，数据源由FTP信息搜集模块搜集的FTP文件信息文本提供。

2.1.2 倒排索引

首先先介绍另一种搜索方式——顺序扫描，用于和倒排索引进行对比。

顺序扫描：当你要搜索包含某个字符串的文档时，遍历所有需要搜索的文档。对于每个文档，遍历文档的内容，如果该文档包含了所要搜索的字符串，那么该文档即为所需要搜索的文档，接下来遍历下一个文档，直到完成对所有文档的遍历。对于存储空间比较大的介质，如果要在其中找到一个包含某指定字符串的文档需要的时间会相对较长。这种搜索策略比较原始，适用于小数据量文档的搜索，方便快捷。但是对于大数据量的文档，这样的方法效率就相对较低。

倒排索引：由于从字符串到文档的映射是文档到字符串映射的逆向过程，所以将这种保存信息的索引方式称作倒排索引。

倒排索引保存的信息大致如下：假定文档集合里面有200篇文档，为了方便表示，为文档编号为1到200，得到图2-2的结构：

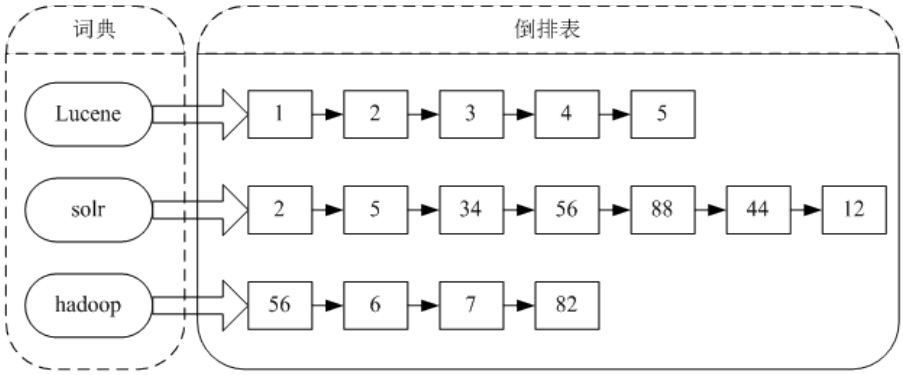


图 2-2 倒排索引基本结构

左边保存的是字符串信息，称为词典。每个字符串都指向包含此字符串的文档链表，此文档链表叫做倒排表(Posting List)。倒排索引保存的信息和需要搜索的信息是一致的，可以加快搜索的效率。假设要寻找既包含字符串“solr”又包含字符串“lucene”的文档，需要如下几步：

1. 取出包含字符串“solr”的文档链表。

2. 取出包含字符串“lucene”的文档链表。

3. 通过合并链表，找出既包含“solr”又包含“lucene”的文件——5。

图2-3示意了上述过程。

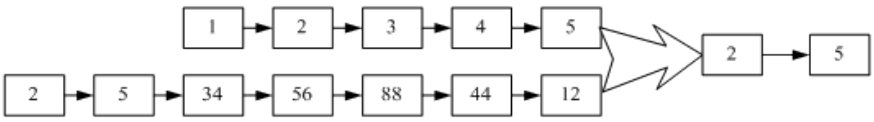


图 2-3 处理查询过程

2.1.3 词典实现原理

进行查询时会使用到其提供的字典功能[7]，即根据给定的term找到该term所对应的倒排文档id列表等信息。关于字典的实现，图2-2给出了一种最简单的方案——排序数组，即term字典是一个已经按字母顺序排序好的数组，数组每一项存放着term和对应的倒排文档id列表。每次载入索引的时候只要将term数组载入内存，通过二分查找即可。这种方法查询时间复杂度为Log(N)，N指的是term数目，占用的空间大小是O(Nstr(term))。此的缺点是内存消耗较大，即需要完整存储每一个term，当term数目多达上千万时，占用的内存将十分大。

1、常用字典数据结构

很多数据结构都能完成字典功能，如表2-1所示:

表 2-1 数据结构介绍

数据结构	优缺点
排序列表Array/List	使用二分法查找，不平衡。
HashMap/TreeMap	性能高，内存消耗大，几乎是原始数据的三倍。
Skip List 跳跃表	可快速查找词语，在lucene、redis、Hbase等均有实现。相对于TreeMap等结构，特别适合高并发场景。
Trie	适合英文词典，如果系统中存在大量字符串且这些字符串基本没有公共前缀，则相应的trie树将非常消耗内存。
Double Array Trie	适合做中文词典，内存占用小，很多分词工具均采用此种算法。
Ternary Search Tree	三叉树，每一个node有3个节点，兼具省空间和查询快的优点。
Finite State Transducers (FST)	一种有限状态转移机，内存开销较小。

2、FST原理简析

FST有两个优点：

1）空间占用小。通过对词典中单词前缀和后缀的重复利用，压缩了存储空间[8]；

2）查询速度快。O(len(str))的查询时间复杂度。

下面简单描述下FST的构造过程。我们对“cat”、“deep”、“do”、“dog” 、“dogs”这5个单词进行插入构建FST。

1）插入“cat”

插入cat，每个字母形成一条边，其中t边指向终点，如图2-4所示：

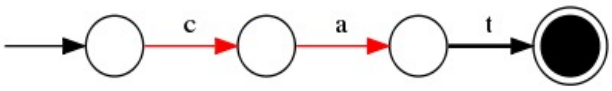


图 2-4 插入cat

2）插入“deep”

与前一个单词“cat”进行最大前缀匹配，发现没有匹配则直接插入，P边指向终点，如图2-5所示：

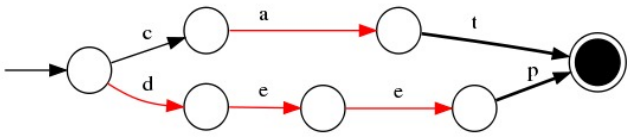


图 2-5 插入“deep”

3）插入“do”

与前一个单词“deep”进行最大前缀匹配，发现是d，则在d边后增加新边o，o边指向终点，如图2-6所示：

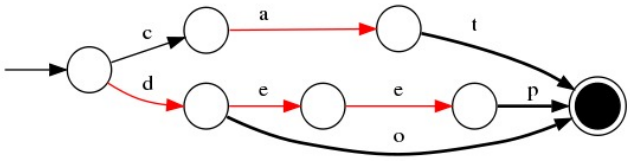


图 2-6 插入“do”

4）插入“dog”

与前一个单词“do”进行最大前缀匹配，发现是do，则在o边后增加新边g，g边指向终点，如图2-7所示：

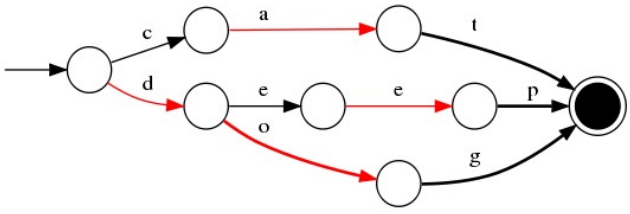


图 2-7 插入“dog”

5）插入“dogs”

与前一个单词“dog”进行最大前缀匹配，发现是dog，则在g后增加新边s，s边指向终点，如图2-8所示：

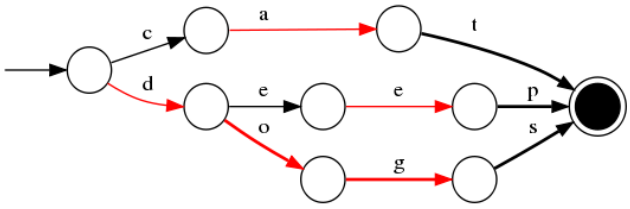


图 2-8 插入“dogs”

最终我们得到了如上一个有向无环图。利用该结构可以很方便的进行查询，如给定一个term“dog”，我们可以通过上述结构很方便的查询该term存不存在，甚至我们在构建过程中可以将单词与某一数字、单词进行关联，从而实现key-value的映射。

3、FST使用与性能评测

我们可以将FST当成一种Key-Value数据结构来进行使用，特别是在对内存开销要求较少的应用场景。

FST压缩率一般在3~20倍之间，相对于TreeMap/HashMap的膨胀3倍，内存节省就有9倍到60倍，那么FST在性能方面的表现如何。下面是在mac book（i7处理器）进行的简单测试，性能虽不如TreeMap和
HashMap，但也算良好，能够满足大部分应用的需求，测试结果如表2-2所示：

表 2-2 FST性能比较

数据结构	HashMap	TreeMap	FST
100万次查询	12ms	9ms	226ms

全文检索的方式可以提升搜索的效率，但同时也增加了索引的时间开销，在小数据量的搜索时，总体的耗时可能和顺序扫描差不多，而当数据量很大的时候创建索引的时间也不低，但建立索引毕竟是个一次性过程，不需要每次搜索的时候都建立索引，只要是建立好了索引，以后就可以享受，这也是全文搜索的一个巨大优点。

2.1.4 Lucene索引结构

在Lucene中的检索是一种索引检索，使用更多的空间消耗来节省时间消耗，

对需要搜索的文件和字符流进行全文索引，在搜索时，通过对索引快速检索，得到检索的位置，该位置为记录关键词出现的文件的路径或者某个关键词[7]。

Lucene中存储全文索引的结构为层次结构，共分5个层次：索引、段、文档、域和项，他们之间的关系如图2-9所示：

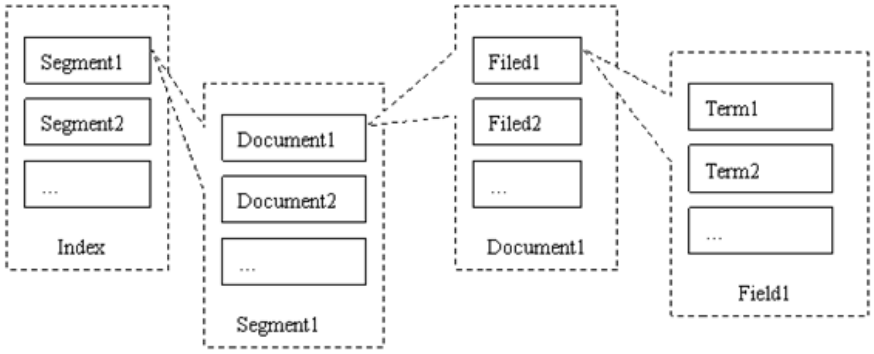


图 2-9 Lucene索引存储结构概念图

1.索引（Index）

每个目录都是一个索引，Lucene中的索引是放在文件夹中，如图2.5所示，同一个文件夹中的所有文件构成了一个索引。

2.段（Segment）

每个索引都包含了若干个段，每个段之间都是独立的，添加一个新的文档会生成一个新的段，不同的段之间可以合并。如图2-10所示，同一个段中的文件应该具有相同的前缀，图中共有"_1"、"_2"和"_3"三个段。segments.gen和segments_4是段的元数据文件，它们保存着段的属性信息。

3.文档（Document）

建立索引的基本单位就是文档，不同的段保存了不同的文档，一个段可以包含若干篇文档。新添加的文档单独保存在一个新生成的段中，随着段合并的过程，不同的文档会合并到同一个段中。

4.域（Field）

如果一篇文档中包含了不同类型的信息，我们可以分开对它进行索引，比如内容，作者，标题，时间等，这些可以在不同的域里保存，并且这些域的索引方式可以不同。

5.项（Term）

索引的最小单位即为项，项是经过了词法分析和语言处理后的字符串。

图2-10是Lucene生成索引的一个简单实例。Lucene的索引结构中，同时保存了正向信息和反向信息。

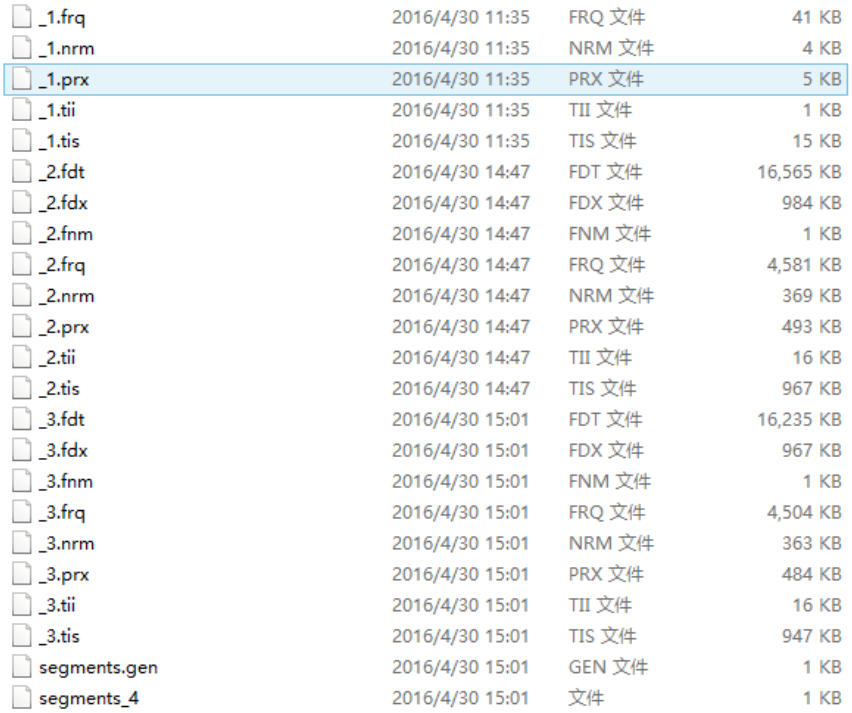


图 2-10 Lucene索引实例

正向信息：按照一定的层次保存了从索引到项的关系：索引→段→文档→域→项，也就是说XXX索引包含了哪些段，XXX段又包含了哪些文档，XXX文档又包含了哪些域，XXX域又包含了哪些项。

在这样的层次结构中，每个层次都会保存着本层次的信息以及下一层次中的属性信息，比如一本介绍宇宙的书，首先会介绍宇宙中有可观测宇宙空间和未知宇宙空间，再介绍可观测宇宙空间中有着各种不同的星系，每个星系又包含了各种不同的星体，再介绍各个星体的特征之类的。

包含正向信息的文件有：

1. segments_N，segments.gen保存着索引中包含了多少个段，每个段中又包含了多少篇文档。

2. AAA.fnm保存着这个段中包含了多少个域，每个域的名称和其索引方式。

3. AAA.fdx，AAA.fdt保存着这个段中包含的一系列文档，每个文档中包含多少个域，每个域中保存着哪些信息。

反向信息：保存了词典到倒排表的映射：项→文档包含反向信息的文件有：

1. AAA.tis，AAA.tii保存着词典：这个段包含的所有的词按照词典指定顺序的排序。

2. AAA.frq保存了倒排表：包含了每个项中文档的ID列表。

3. AAA.prx保存了倒排表中的每个项在其所对应的文档中的位置。

2.1.5 Lucene搜索过程

在本系统中，搜索的过程分成下列4个步骤：用户输入搜索关键词、处理关键词生成Query实例、使用IndexSearcher实例进行搜索、对返回的结果进行处理。在用户输入搜索关键后，根据具体的设定，使用Query类中的某些查询类型的具体子类或QueryParser来进行处理，并且生成Query实例。随后Query实例将会被传递给IndexSearcher的search方法中，进行搜索的过程，方法返回的结果会由TopDocs类的实例保存，使用ScoreDoc类实例来提供对TopDocs中每条结果的访问接口。

2.1.6 模糊搜索功能

要谈到模糊搜索就不得不提起与之对应的一个词“精确搜索”，下面简单介绍一下这两个词语的概念。假定搜索的关键词是由若干个字组成的词组或短语。

精确检索指的是输入的关键词在检索结果中的字间顺序与字间间隔完全一样，比如搜索“电子科大” 出现的结果可能为：

1 “电子科大XXX”

2 “XXX电子科大”

3 “XXX电子科大XXX”

而模糊检索指的是输入的关键词在检索结果中可以近似出现，字符之间的顺序、字符之间的间隔可以产生变化，同样搜索“电子科大”出现的结果可能大多数都是：“XXX电子科技大学XXX”，结果中关键词“电子科大”被拆分。模糊搜索的好处是可以扩大搜索范围以及搜索结果的条目数，使用者可能会从相关结果中找到自己想要的结果。

下面介绍一种算法Levenshtein Distance（LD）即编辑距离算法[9]，用以实现系统中模糊搜索的功能。该算法用于判断两个字符串的距离，或者叫模糊度。简单点理解就是差异程度。而这个差异程度的标准就是下列这些操作都会使距离值增加：

1 加一个字母(Insert),

2 删一个字母(Delete),

3 改变一个字母(Substitute)。

比如搜索的目标字符串为“word”，而源字符串为“world”，“world”与“word”之间的LD距离即为1，而“word”与“wild”之间的LD距离为2。

2.1.7 中文分词模块

对于Lucene查询来说，分析的操作会出现在两个地方：建立索引的时候和使用QueryParser构建Query对象实例的时候。如果在搜索结果中要以高亮的形式显示搜索内容，也可能会用到分析操作。如2.1.1提到的，分析操作被封装在分析器中。在本系统中，分析器主要工作是中文分词[10]。下面简单介绍一下3款智能分词工具。ICTCLAS 是一款基于隐马尔科夫模型(HMM)的智能分词工具，而imdict-chinese-analyzer 和 ictclas4j 都是基于同一模型开发的Java版分词开源分词工具。

三种智能分词工具的效率对比三者的分词效率对比如图2-11所示

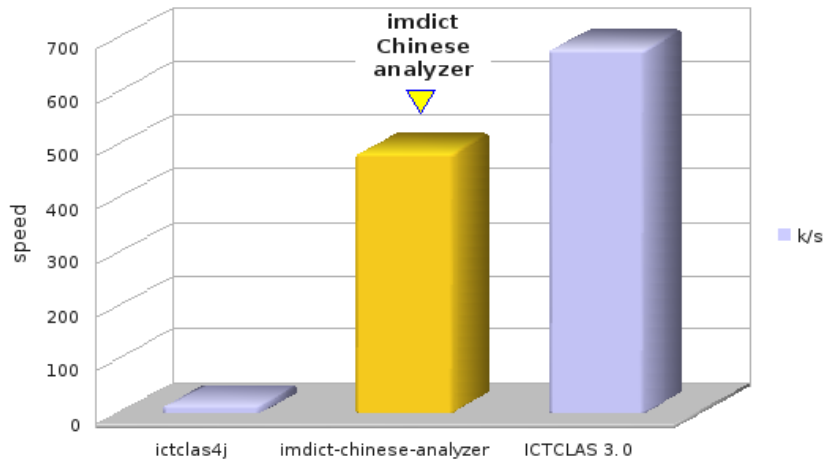


图 2-11 智能分词效率对比

具体数据参见表2-3：

表 2-3 具体数据对比

	ictclas4j	smart-chinese-analyzer	ICTCLAS 3.0
分词速度(字节/秒)	14.75	491.44	661.54
分词速度(汉字/秒)	7822	261235	348751

测试的数据为中文文件，大小为65432KB，内容长度为35126747字符，他们各自独立进行分词工作，并将分开的词语写到文件里。测试环境均相同为：I5 CPU，内存4G。

由图2-11、表2-3对比可以看出， smart-chinese-analyzer在分词的效率上与ICTCLAS的分词效率是同一个数量级的，为 ictclas4j 的 30多倍。本系统拟采用smart-chinese-analyzer作为分析器。主要原因是其开源免费且具有良好的性能。smart-chinese-analyzer具有如下特性：

1. 完全支持Unicode：分词核心模块采用Unicode编码，对汉字的编码无需转换，提升了分词的效率。

2. 提升了搜索的效率：在智能词典的实践中，如果有智能中文分词时，索引文件比没有中文分词的文件小了接近30%。

3. 提高了搜索准确度：smart-chinese-analyzer基于HHMM分词模型，极大提高了分词的准确率，在此基础上进行的搜索，比对汉字的逐个切分要更加准确。

4. 高效的数据结构：为了提高效率，针对常用中文检索的应用场景，smart-chinese-analyzer对功能进行了优化，例如人名识别、词性标注、时间识别等。另外还修改了原算法的数据结构，在内存占用量缩减到30%的情况下把效率提升了数倍。

2.2 Struts2框架

2.2.1 Struts2简介

Struts 2是基于 struts 和WebWork的技术基础，进行了合并之后的全新框架。它的体系结构与Struts 1差别巨大。Struts 2以WebWork为核心，采用拦截器机制来处理用户的Request请求，这样的设计方式也使得业务逻辑控制器能够与Servlet API完全分离，所以Struts 2可以理解成新一代的WebWork产品。所有Struts 2程序都是基于client/server的HTTP交换协议，同样提供了MVC模式的清晰实现，包含了拦截器，OGNL、堆栈等参与对请求进行处理的组件。

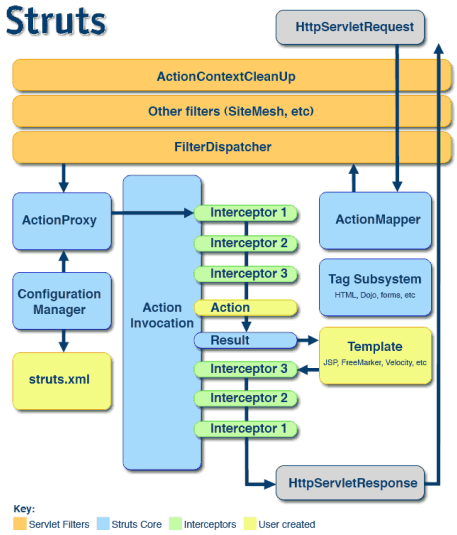


图 2-12 Struts2框架结构图

2.2.2 Struts2工作原理

如图2-12所示为一个请求在Struts2框架中的处理过程，大致分成以下几步：

1. 客户端初始化一个指向Servlet容器（例如Tomcat）的请求。

2. 请求经过一系列过滤器（Filter）。

3. FilterDispatcher被调用，FilterDispatcher询问ActionMapper该请求需不需要调用某个Action。FilterDispatcher是控制器的核心，也就是mvc中控制层c的核心。

4. 如果ActionMapper确定需要调用某个Action，FilterDispatcher就会把请求交给ActionProxy来处理。

5. ActionProxy通过Configuration Manager访问框架中的Config文件，找到需要调用的Action类。

6. 同时ActionProxy会创建一个ActionInvocation实例。

7. ActionInvocation实例使用Java命名模式来调用，在调用Action的时候，会涉及到相关拦截器（Intercepter）的调用。

8. Action执行完之后，ActionInvocation会从配置文件struts.xml中找到相应的返回结果。返回结果常常是一个需要被表示的FreeMarker或者JSP的模版。在表示的过程中会涉及使用Struts2 框架中继承的标签，以及ActionMapper。

2.3 其他技术简介

2.3.1 Mybatis简介

MyBatis是支持普通 SQL查询，存储过程和高级映射的优秀持久层框架。MyBatis消除了几乎所有的 JDBC 代码和参数的手工设置以及结果集的检索。MyBatis使用简单的 XML或注解用于配置和原始映射，将接口和 Java的POJOs（Plain Old Java Objects，普通的Java对象）映射成数据库中的记录。

相对Hibernate和Apache OJB等“一站式”ORM解决方案而言，MyBatis是一种“半自动化”的ORM实现。MyBatis需要开发人员自己来写SQL语句，这可以增加了程序的灵活性，在一定程度上可以作为ORM的一种补充。程序设计人员应该结合自己的项目的实际情况，来选择使用不同的策略。MyBatis和Hibernate都做了映射，但MyBatis是把实体类和SQL语句之间建立了映射关系，这种策略可以允许开发人员自己来写合适的SQL语句，而Hibernate在实体类和数据库之间建立了映射关系，SQL对于开发人员是不可见的，对于那些数据量非常大的应用，无法去优化SQL语句。

在本系统中，Mybatis框架完成了所有持久层的任务。

2.3.2 Apache Commons Net简介

Apache Commons Net™ 工具包实现了多种基本的网络协议的客户端。Apache Commons Net™ 工具包的目标是提供基本的协议访问，而非高级的抽象。因此某些设计违背了面向对象设计的原则。Apache Commons Net™ 工具包的理念是尽可能让一个协议的总体功能易于使用，同时在适用情况下提供基本协议的接口以便程序员构造自己定制的实现。

在FTP搜索引擎系统中，Apache Commons Net™ 工具包帮助系统完成了与FTP服务器的全部交互工作。

2.3.3 JDOM简介

JDOM是一种特殊的Java开发工具包，使用了 XML ，用于提高开发 XML 应用程序的效率。它的设计中包含了 Java 的语法和语义。

JDOM是Brett Mclaughlin 和 Jason Hunter 的研发成果，他们是两位接触的 Java 开发人员， 2000 年初在类似 Apache 协议的许可下，JDOM作为一个开源项目正式启动。时至今日，JDOM已经发展壮大，成为一个成熟的系统，每天接收着众多优秀的 Java 开发人员的改进、集中反馈及BUG修复，并致力建立一个基于 Java 平台的完整解决方案，通过 Java 接口来进行访问、操作并输出 XML 数据。

在FTP搜索引擎系统中，JDOM完成对系统配置XML文档的相关操作。

第3章 系统需求分析及总体设计

3.1 需求分析

FTP搜索引擎实现目标就是满足Internet用户的需求，系统以B/S结构为基础，用户通过使用Web浏览器访问系统，方便快捷的利用系统各个模块，获取到需要的特定资源。一般校内搜索的主要内容基本上以课件、教学、音乐、视频、软件等内容为主。搜索引擎需要对中英文混合的文件名有较好的分词支持,并支持模糊搜索。对于FTP搜索引擎而言，用户主要分为两类：Web前端的一般用户和后台的管理员。

3.2 系统设计原则

1.首先要完成需求分析中提到的功能并保证整个系统的完整性和一致性。
2.增强系统的可维护性。
3.降低系统各模块间的耦合度。
4.提高系统性能。

3.3 功能性需求

Web前端部分的用例图如图3-1所示。Web前端的一般用户，可以通过Web前端网页输入查询关键词，进行信息的搜索，从而通过FTP搜索引擎快速定位到指定的资源。系统同时提供高级搜索，能限定搜索结果的范围，如文件大小、日期、所在服务器等，使得搜索能够更精确的定位到所需的资源上。用户还可以为FTP搜索引擎推荐FTP服务器，这样，搜索引擎就能够更好的对更多的FTP服务器进行索引。系统还应提供用户反馈功能，接受用户意见，以便更好的改进用户体验。

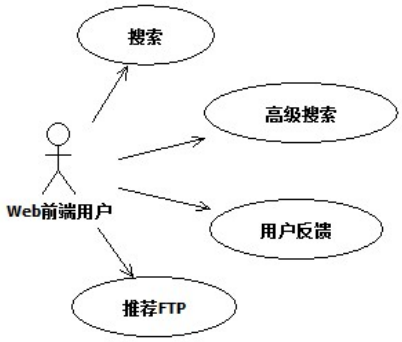


图 3-1 Web前端用例图

对于后台管理系统，如图3-2所示。首先，它必须有一个用于身份认证的登陆过程，以防止系统的相关设置被非法修改。在登陆之后，管理员可以通过后台管理系统来了解并设置当前系统的配置，系统的配置主要包括Lucene索引存放的目录、FTP信息搜集模块所搜集信息存放的目录等的配置工作；能够管理已经搜集FTP服务器信息及用户反馈。

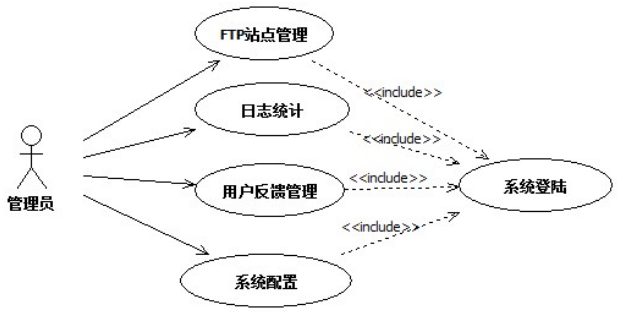


图 3-2 后台管理系统用例图

3.4 系统性能需求

1.搜索引擎应该有较好的跨平台性，能够在不同平台的服务器上较为良好的运行。

2.在短时间内快速返回搜索结果，并有较好的相关性排序。

3.需要具有清晰友好的用户界面，使之简单易用。

3.5 系统核心功能

如图3-3所示，整个FTP搜索引擎的核心架构部分——Lucene搜索的实现由四个部分构成：数据搜集、Web前端查询、FTP文件信息对比、Lucene。后端数据搜集使用Apache Commons Net^{TM} 工具包编写的搜集模块来遍历FTP服务器的指定目录，并将遍历的结果以文本的形式存储在计算机文件系统当中。对于同一个FTP服务器不同时间所搜集的文件信息，对新旧两个不同的文件信息文本进行对比，得到新旧两次文件信息不同之处，用于对Lucene索引的增量式更新。

而后，数据索引模块从存储的FTP文件信息文本中获取文件信息，在进行分析之后，建立相关Lucene文档实例，存入索引当中，为用户的搜索提供基础。

用户的查询过程将在Struts 2框架下完成，用户的搜索请求，经由定义在Struts2中的相关action来完成一系列Lucene的相关操作后为用户返回想要的结果，这些操作包括构建Query查询对象、使用IndexSearcher进行查询、构建查询结果集。

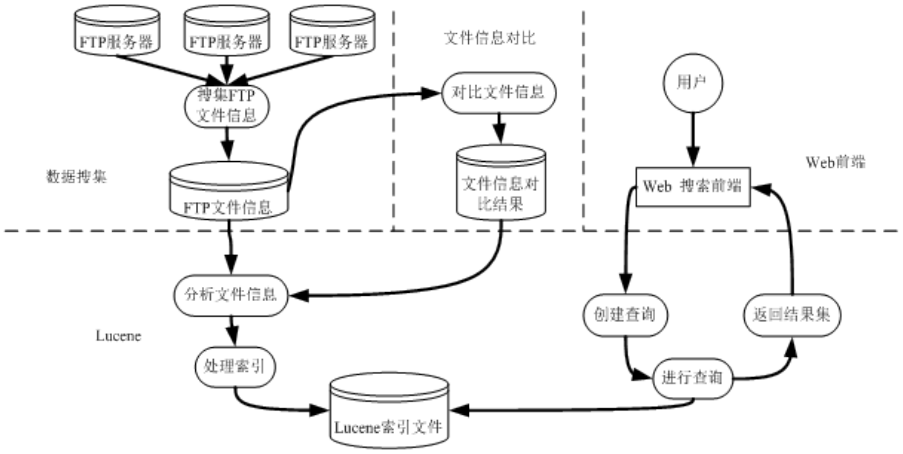


图 3-3 核心功能架构图

3.6 数据库设计

系统采用MySQL数据库存储FTP服务器信息、用户反馈、搜集模块日志、用户查询日志、查询统计等信息。图3-4 为整个数据库的E-R图，一个FTP服务器拥有多个不同时间的搜集日志，FTP信息文本记录了FTP服务器指定目录的完整结构及文件信息。用户反馈存储了用户通过Web前端输入的用户反馈信息。用户查询日志简单记录了用户每次搜索行为。查询统计记录了从FTP搜索日志中统计出各个关键词被搜索的次数。

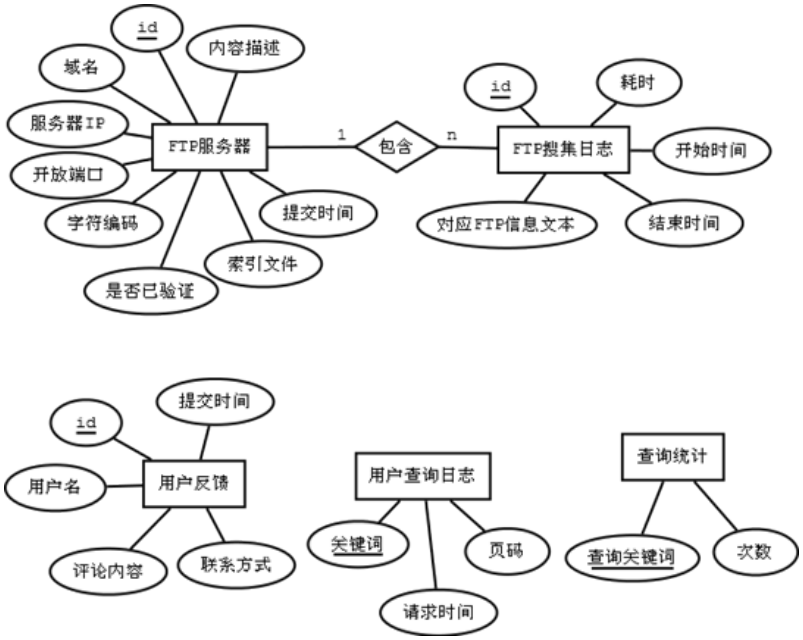


图 3-4 E-R图

下面的表格给出了数据库中各个表的详细逻辑结构设计（注：PK：primary key（主键）；FK：foreign key（外键））

表 3-1 FTP服务器信息（ftpserver）表

列名	类型	约束	备注
id	int	PK	每个FTP服务器在数据库中的唯一标识。
domain	varchar(100)	无	FTP服务器的域名，域名与IP不能同时为空。
ipv4	varchar(15)	无	FTP服务器的IP，域名与IP不能同时为空。
port	int	NOT NULL	服务器所开放的端口，默认值：21。
encoding	varchar(20)	NOT NULL	确保程序能正确编码FTP服务器所返回的目录信息。默认值：GBK。
verify	char(1)	NOT NULL	服务器验证信息，默认值：0，当verify=0时，说明其未被验证。
submittime	datetime	无	用户提交FTP服务器信息的时间。
description	text	无	FTP所包含的内容的简介。
indexfile	int	FK	当前Lucene索引的FTP信息文件。

表 3-2 用户反馈信息（Feedback）表

列名	类型	约束	备注
id	int	PK	每条反馈的唯一标识。
username	varchar(20)	NOT NULL	默认值：匿名用户。
contact	varchar(255)	无	联系方式，提供更一步交流的可能。
comment	text	NOT NULL	评论内容。
commentdate	datetime	NOT NULL	评论提交时间。

表 3-3 FTP搜集日志（ftpspiderlog）表

列名	类型	约束	备注
id	int	PK	每条反馈的唯一标识。
ftpid	int	FK	日志所对应的FTP服务器。
starttime	datetime	NOT NULL	搜集开始时间。
finishtime	datetime	NOT NULL	完成时间。
cost	int	NOT NULL	耗时，单位：秒。
ftpinfo	varchar(255)	NOT NULL	本次搜集对应的存储文本。

表 3-4 用户查询日志（userquerylog）表

列名	类型	约束	备注
id	int	PK	每条反馈的唯一标识。
querytime	datetime	NOT NULL	查询发生的时间。
queryword	varchar(255)	NOT NULL	查询的关键词。
querypage	int	NOT NULL	用户所处的页面。

表 3-5 查询统计（querycount）表

列名	类型	约束	备注
id	int	PK	每条反馈的唯一标识。
queryword	varchar(255)	NOT NULL	查询的关键词。
count	int	NOT NULL	被查询的次数。

第4章 系统详细设计与实现

4.1 数据搜集模块

数据搜集模块用于访问FTP服务器并获取FTP服务器上的所有文件信息。并将文件信息以文本的形式存储在本地的磁盘上，以供其他需要文件信息文本的模块使用。整个模块被封装在FTPSpider类中。FTPSpider类的结构如下：

```
1 public class FTPSpider {
2     private FTPClient     ftp;            // FTP客户端类
3     private String        server;         // ⤸
4     连接的FTP服务器
5     private int           port;           // ⤸
6     FTP服务器开放的端口
7     private String        controlEncoding; // 控制编码
8     private String        charEncoding;   // 字符集编码
9
10     private String        outputFile;     // ⤸
11     输出文件的位置
12     private Queue<String> pathQueue;      // ⤸
13     路径队列，遍历FTP服务器时使用
14     private int           count = 0;      // ⤸
15     记录遍历的文件数量
16
17     public int getCount()  //遍历文件数量
18     public FTPSpider(String server, int port, String ⤸
19     charEncoding,String outputFile)  //构造方法
20     public FTPSpider(String server, String outputFile)  ⤸
21     //构造方法
22     public void connectFTP()  //连接FTP服务器
23     public void getFTPInfo(String remotePath)  ⤸
```

```
24      //扫描指定目录
25      private void closeTheConnect()  //退出FTP服务器
26 }
27
```

4.1.1 数据存储格式

对于从FTP搜集下来的目录信息及文件信息，采用如表4-1所示规则存储：

表 4-1 数据存储格式表

文件头部
Server: FTP服务器的IP Port: FTP服务器所开放的端口 ----------
文件主体（FTP服务器的目录信息和文件信息）
目录行 文件信息行*n（n>=0） 目录行 文件信息行*n（n>=0） ……

目录行：以“/”开头，第一个“/”代表FTP服务器根目录。文件信息行：以数字开始，格式为“文件修改时间/文件类型/文件大小/文件名”。文件修改日期是以文件修改时间减去1970年1月1日00：00的时差，精确到秒。对于一般的文件来说，其类型为其文件名的后缀名，即最后一个“.”后的字符串。例如文件名“新建文本文档.txt”，它的类型为txt。但是最后一个“.”的位置不能是文件名的第一个字符，如“.classpath”这个文件名，它的文件类型并不是classpath。文件大小的单位为Byte。在本系统中，文件夹也被看作一种文件，它是类型为folder，大小为0。图4-1给出了一个FTP信息文本的实例的一部分，它包括了上面所说的全部定义：

```
Server:192.168.1.102
Port:21
--------------------------------
/
1437486660/folder/0/Ebook
1454760720/folder/0/Games
1425199560/folder/0/Movies
1440149520/folder/0/Study
1461549060/folder/0/Tools
/Ebook/
1405257600/folder/0/api
1411866600/pdf/3114408/CDH4-Installation-Guide
1411866600/pdf/1826215/CDH4-Quick-Start
1411866600/pdf/1768197/CDH4-Requirements-and-Supported-Versions
1404472320/pdf/59924445/Head First Java第2版中文版
1437483180/pdf/82520350/Java经典编程300例
1404476280/pdf/2866802/Java编程思想第4版_优化
1404471780/pdf/51470824/Linux命令行与Shell脚本编程大全第2版.布卢姆
1407826560/pdf/849841/netflow_whitePaper
1285488660/pdf/34679310/Python基础教程(第2版)
1389174420/chm/696925/Win32 API
1415061840/pdf/35727096/[Python灰帽子：黑客与逆向工程师的Python编程之道].
(Justin.Seitz).丁赟卿.扫描版(ED2000.COM)
1404832200/pdf/524567/周志华：数据挖掘与机器学习
1408979640/pdf/2494307/机器学习应用实践
1389174420/chm/2470326/汇编指令学习
```

图 4-1 数据存储实例

4.1.2 相关类及方法

在本系统中，使用FTPClient类的实例ftp来完成所有与FTP服务器的交互工作，这些工作包括：连接并登陆匿名的FTP服务器、改变当前FTP会话的工作目录、获取工作目录下的文件信息。这些工作的主要实现方法如下，他们都由FTPClient类来提供：

1. 连接匿名的FTP服务器：connect(server, port)，该方法需要提供服务器的域名或者IP作为server参数，提供所开放的FTP端口号作为port参数。

2. 登陆匿名的FTP服务器：login(String username, String password)，需提供用户名与密码来完成FTP服务器的登陆工作，由于本系统只对匿名的FTP服务器检索，所以在这里，username为“anonymous”，password为笔者的邮箱“269504518@qq.com”。

3. 改变当前FTP会话的工作目录：changeWorkingDirectory(String pathname)，每当获取并完成对一个工作目录下的文件信息的存储后，需要使用该方法来改变当前工作目录，从而能够获取其他工作目录的信息，以完成对FTP指定目录遍历的目的。

4. 退出FTP服务器：disconnect()，当搜集模块完成了遍历任务后，用它来释放FTP连接。

4.1.3 处理流程及实现

整个搜集的过程如图4-2所示。pathQueue是一个String队列，里面存储着提供给changeWorkingDirectory(String pathname)使用的pathname，对于一个文件夹类型的文件，它自身所在的目录加上它的文件名就构成了新的pathname。

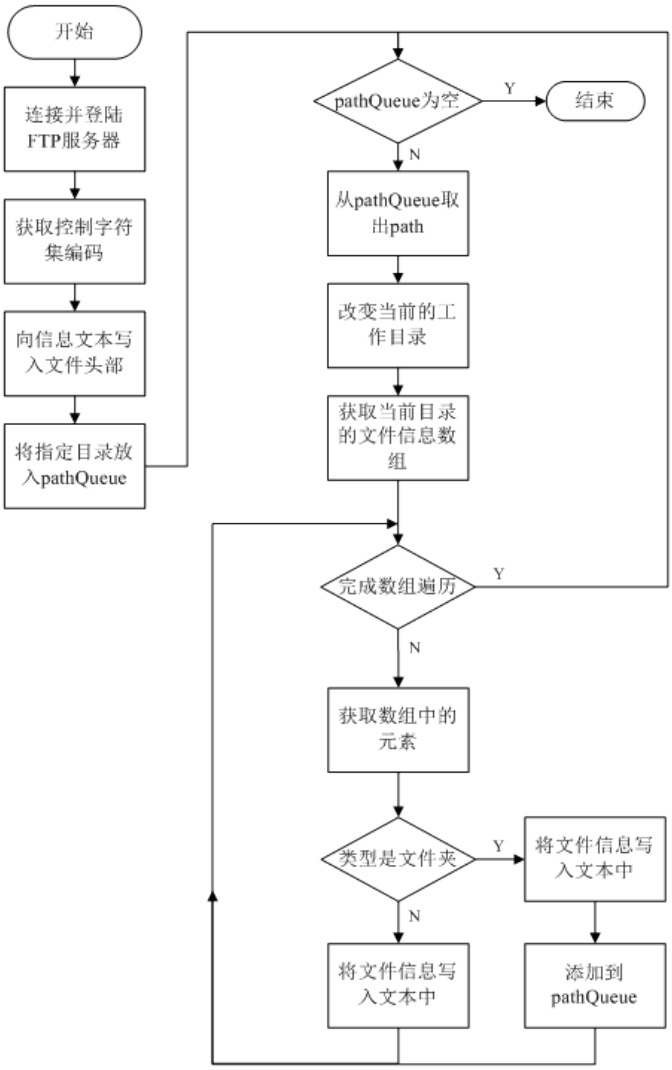


图 4-2 FTP数据搜集流程图

4.2 文件信息对比模块

对于一个在FTP服务器的文件，文件的文件大小或修改日期发生变化时，称该文件被更改了。FTP服务器随时都有可能更改、删除某个文件，或者添加新的

文件。文件信息对比模块将新搜集的文件信息文本与当前被Lucene索引的文件信息文本进行对比，找出FTP服务器文件信息的变化之处，为索引模块更新索引提供依据。文件信息对比模块所有的操作被封装在FileComparer类中，它与其他外部类的关系如图4-3所示。

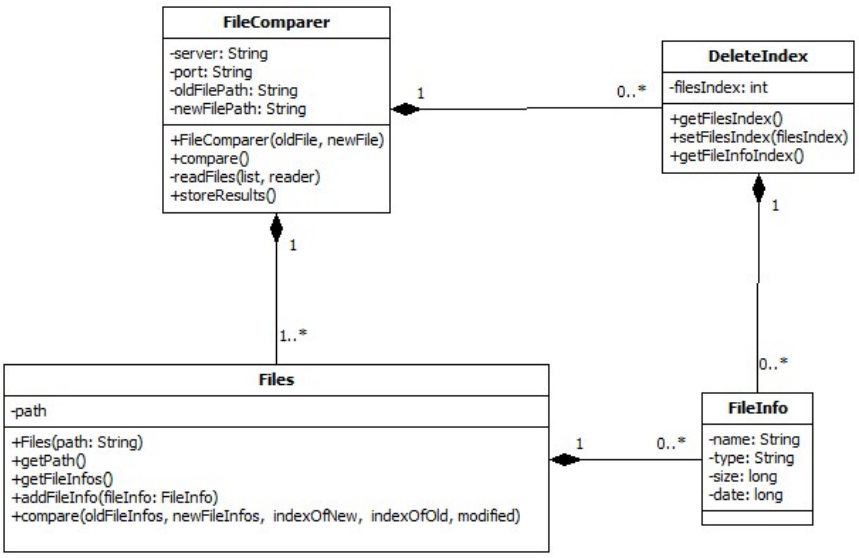


图 4-3 FileComparer类图

4.2.1 数据结构设计

FileInfo类用于4.1.1数据存储格式中文件信息的实例化，每一条文件信息构建一个FileInfo实例。 Files类将工作目录和目录下的文件信息结合在一起，他包含path，以及在这个工作目录下的所有文件信息。这样就很好的与4.1.1数据存储格式对应了起来。由于ArrayList在遍历时，其中的元素不能删除，故用DeleteIndex记录每个工作路径下需要被删除的文件信息。最后统一删除。

4.2.2 处理流程及实现

对比流程如图4-4所示。新搜集的FTP文件信息文本被读取并构建为Files实例，存入newFiles数组列表；之前被Lucene 索引的FTP文件信息文本被读取并构建为Files实例，存入oldFile数组列表，整个处理流程通过对比找出并删除两个数组中相同的FileInfo实例，即删除FTP中文件信息没有变化的文件。在完成整个处理流程后，newFiles数组列表中剩下的元素就是FTP服务器上新添加的文件。oldFiles中剩下的元素就是FTP服务器上被删除的文件。

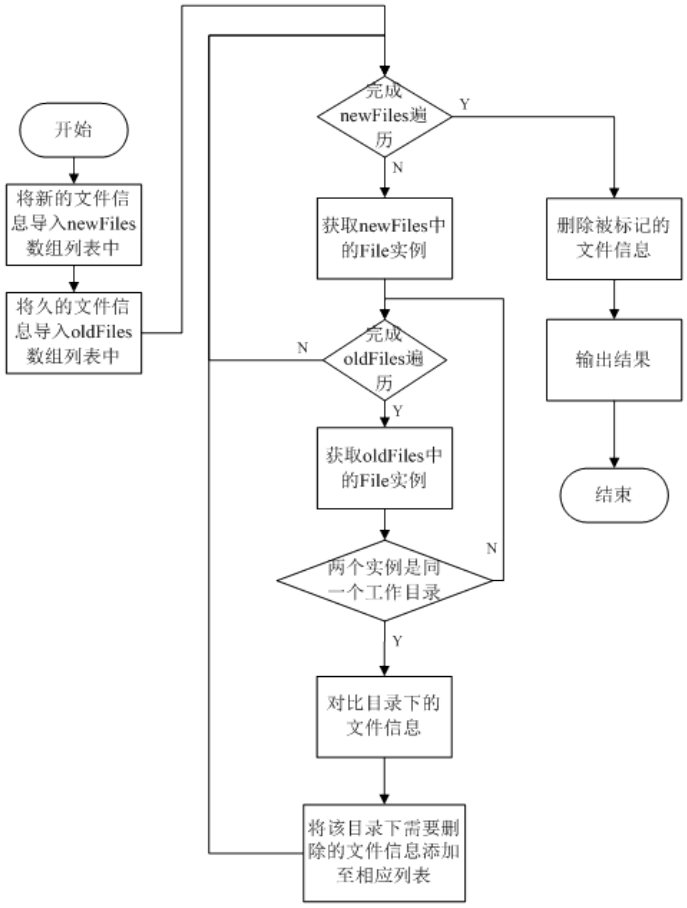


图 4-4 文件信息对比模块流程图

关键代码如下：

```
1 for (Files newFile : newFiles) {
2     for (Files oldFile : oldFiles) {
3         if (newFile.getPath().equals(oldFile.getPath())) {
4             modified = new Files(newFile.getPath());
5             indexOfNew = new DeleteIndex();
6             indexOfOld = new DeleteIndex();
7             indexOfNew.setFilesIndex(newFiles.indexOf(
8             newFile));
9             indexOfOld.setFilesIndex(oldFiles.indexOf(
10             oldFile));
11
```

```
12              newFile.compare(oldFile.getFileInfos(),
13                                      indexOfNew.getFileInfoIndex(),
14                                      indexOfOld.getFileInfoIndex(),
15                                      modified.getFileInfos());
16
17              modifiedFiles.add(modified);
18              deleteInNew.add(indexOfNew);
19              deleteInOld.add(indexOfOld);
20
21              break;
22          }
23      }
24 }
25
```

图4-5为图4-4中“对比目录下的文件信息”过程的细化。这个过程被封装在Files类的compare()方法中。

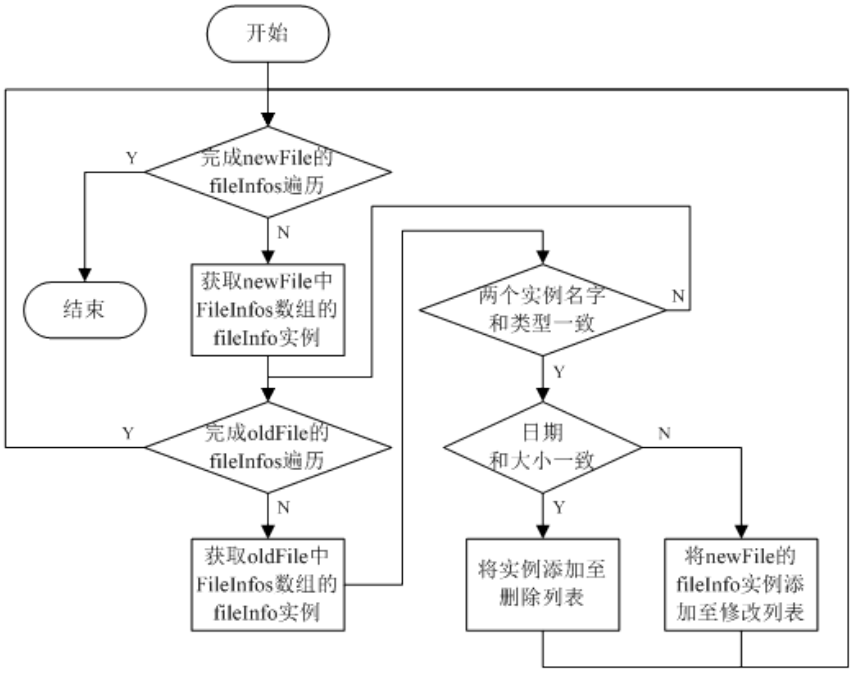


图 4-5 文件信息对比流程图

关键代码如下：

```
1 public void compare(ArrayList<FileInfo> oldFileInfos,
2                     ArrayList<FileInfo> indexOfNew, ↵
3                     ArrayList<FileInfo> indexOfOld,
4                     ArrayList<FileInfo> modified) {
5     for (FileInfo newOne : this.fileInfos) {
6         for (FileInfo oldOne : oldFileInfos) {
7             if (newOne.getName().equals(oldOne.getName())
8                     && newOne.getType().equals(oldOne.↵
9                     getType())) {
10                if (newOne.getSize() == oldOne.getSize()
11                        && newOne.getDate() == oldOne.↵
12                        getDate()) {
13                    indexOfNew.add(newOne);
14                    indexOfOld.add(oldOne);
15                    break;
16                } else {
17                    modified.add(newOne);
18                    break;
19                }
20            }
21        }
22    }
23 }
24
```

4.2.3 结果存储格式

对比结果文本分为4部分，如表4-2所示，目录行及文件信息行的定义与4.1.1中一致。文件头部：指明了文件信息所属的服务器及该服务器所开放的端口。被删除的条目：FTP服务器上被删除的文件，被索引模块用于删除Lucene索引中过时的条目。新添加的条目：FTP服务器上新添加的文件，被索引模块用于向lucene索引添加新的条目。发生变化的条目：可用于统计FTP每个目录发生变化的次数。

表 4-2 文件信息对比结果存储格式表

Server: FTP服务器的IP Port: FTP服务器所开放的端口 oldFile:两个文件信息文本中较旧的那个 newFile:两个文件信息文本中较新的那个 ----------
被删除的条目： 目录行 文件信息行*n（n>=0） 目录行 文件信息行*n（n>=0） …… ----------
新添加的条目： 目录行 文件信息行*n（n>=0） 目录行 文件信息行*n（n>=0） …… ----------
发生变化的条目： 目录行 文件信息行*n（n>=0） 目录行 文件信息行*n（n>=0） ……

图4-6给出了一个FTP信息文本的实例的一部分，它包括了上面所说的全部定义。

```
Server:192.168.1.102
Port:21
oldFile:1462943600_192.168.1.102.txt
newFile:1462944259_192.168.1.102.txt
----------
被删除的条目：
/
1462943520/folder/0/Ebook
1425199560/folder/0/Movie
1462762380/folder/0/Tools
/Ebook/
1439741640/txt/9/BugReport
1389174420/chm/696925/Win32 API
/Games/
/Movie/
/Study/
/Tools/
/Ebook/api/
```

图 4-6 文件信息对比结果实例

4.3 数据索引模块

数据索引模块将执行Lucene的索引过程，对搜集到的FTP文件信息文本进行索引。数据只有经过索引之后才能够被用于搜索。

4.3.1 相关类及方法

IndexWriter类：IndexWriter类是org.apache.lucene.index包中的类，他主要负责创建并维护一个lucene索引。在数据索引模块中，使用indexWriter实例的addDocument(Document doc)方法来将相关数据构建成的Document实例添加至Lucene索引中。

4.3.2 构建IndexWriter

构建indexWriter实例需要经过以下的过程：

1.指定lucene索引的存放位置。

需要实例化一个Directory对象来完成这一工作：Directory dir = FSDirectory.open(new File(configer.getIndexDir()))其中，路径由从configer实例的getIndexDir()方法给出，在4.7节将给出configer的具体设计。

2. 构建IndexWriterConfig实例。

IndexWriterConfig的实例用于修改IndexWriter的一切配置，以使得新建的IndexWriter能够符合需求。例如：IndexWriterConfig iwconf = new IndexWriterConfig(Version.LUCENE_35, new SmartChineseAnalyzer(Version.LUCENE_35));在构建IndexWriterConfig实例，需要确认使用的Lucene的版本号，以及IndexWriter默认的分析器是什么。在例子中，使用的Lucene版本为3.5,使用的是SmartChineseAnalyzer分析器。

3. 构建IndexWriter实例。

在完成1、2两个步操作后，利用1、2所构建的对象实例来构建出indexWriter实例，例如：indexWriter = new IndexWriter(dir, iwconf)。

4.3.3 构建Document

在FTP搜索引擎中，Lucene的Document实例用于存储文件信息，一个document实例对应一个文件的文件信息。结构如表4-3所示。

表 4-3 FTP搜索引擎Lucene Document结构

域名（field name）	内容	类型
Name	文件名	String
Type	文件类型	String
Size	大小，单位：Byte	Long
date	修改时间，精确到天	Int
url	文件的URL	String

Document实例构建过程如下：

1. 使用构造方法，构建一个Document实例，例如：Document doc = new Document();

2. 使用Document实例的add(Fieldable field)方法向Document实例中添加实现了Fieldable接口的实例。

4.3.4 Fieldable接口

对于字符串类型的索引，使用Field(String name, String value, Field.Store store, Field.Index index)来构建。各参数的含义见表4-4。

表 4-4 Field实例构造方法个参数意义

参数	意义
String name	域名
String value	域值
Field.Store store	值为Field.Store.NO时，该域值不会被存储；值为Field.Store.YES时，该域值将被存储；
Field.Index index	值为Field.Index.ANALYZED时该域值会被分析器分析并存入索引,使得该域能被用于搜索。值为Field.Index.NOT_ANALYZED，则该域值将直接作为一个单一的语汇单元存入索引中。并能被搜索。

对于数字型的索引，使用NumericField(String name, Field.Store store, boolean index)来构建，各个参数的含义见表4-5。

表 4-5 NumericField实例构造方法各参数意义

参数	意义
String name	域名
Field.Store store	值为Field.Store.NO时，该域值不会被存储；值为Field.Store.YES时，该域值将被存储；
boolean index	True:该域将被存入索引中，使之能被搜索，False：进行索引，不能用于搜索

当完成了NumericField实例的构造后，还需要调用其实例的一些方法，初始化实例。使用如表4-6所示的方法即可。

表 4-6 NumericField实例初始化数值的相关方法

方法	初始化方式
setDoubleValue(double value)	使用双精度浮点数初始化
setFloatValue(float value)	使用浮点数初始化
setIntValue(int value)	使用整数初始化
setLongValue(long value)	使用长整形初始化

4.3.5 数据索引流程

整个数据索引流程如图4-7所示。索引模块先读取FTP文件信息文本中的FTP服务器信息及文件信息，利用文件信息建立Lucene文档并将其添加到Lucene索引。

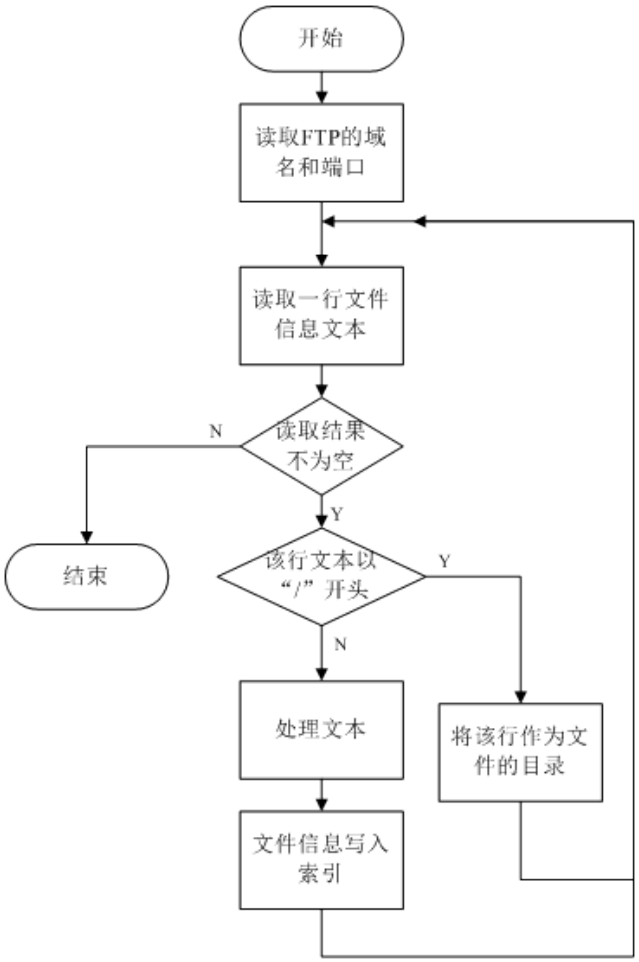


图 4-7 数据索引流程图

将文件信息写入索引的关键代码如下：

```
protected void addIntoIndex(String name, String type, long
size, int date,
                            String path) throws Exception {
    Document doc = new Document();
    //初始化Lucene文档
    doc.add(new Field("name", name, Field.Store.YES, Field.
    Index.ANALYZED));
    doc.add(new Field("type", type, Field.Store.YES, Field.
    Index.NOT_ANALYZED));
     doc.add(new NumericField("size", Field.Store.YES, true).
```

```
11      setLongValue(size));
12      doc.add(new NumericField("date", Field.Store.YES, true).
13      setIntValue(date));
14      doc.add(new Field("path", path, Field.Store.YES, Field.
15      Index.NOT_ANALYZED));
16      //将文档添加到Lucene索引
17      indexWriter.addDocument(doc);
18  }
19
```

4.3.6 索引中文档的更新

对于Lucene来说，它只能够删除整个旧的文档，然后向索引添加新文档。IndexWriter中的updateDocument(Term,Document)方法是通过调用deleteDocuments(Term)和addDocument(Document)方法来实现的。所以在本系统中，我们采用如下方法更新索引中的文档：

1.使用IndexWriter的deleteDocuments(Query query);删除指定FTP服务器的所有在索引中的文件信息。

2.调用4.3.5的索引过程，将新的FTP服务器文件信息添加到索引中。

4.4 搜索模块

给用户提供搜索功能。所有的搜索操作的被封装在FTPSearcher类中。

4.4.1 Levenshtein Distance算法设计

算法设计如下：

Step 1

Set n to be the length of s.Set m to be the length of t.

If n = 0, return m and exit.If m = 0, return n and exit.

Construct a matrix containing 0..m rows and 0..n columns.

Step 2

Initialize the first row to 0..n.

Initialize the first column to 0..m.

Step 3

Examine each character of s (i from 1 to n).

Step 4

Examine each character of t (j from 1 to m).

Step 5

If s[i] equals t[j], the cost is 0.

If s[i] doesn’t equal t[j], the cost is 1.

Step 6

Set cell d[I,j] of the matrix equal to the minimum of:

a. The cell immediately above plus 1: d[i-1,j] + 1.

b. The cell immediately to the left plus 1: d[i,j-1] + 1. S c. The cell diagonally above and to the left plus the cost: d[i-1,j-1] + cost.

Step 7

After the iteration steps (3, 4, 5, 6) are complete, the distance is found in cell d[n,m].

1、得到源串s长度n与目标串t的长度m，如果一方为的长度0，则返回另一方的长度。

2、初始化(n+1)*(m+1)的矩阵d，第一行第一列的值为0增至对应的长度。

3、遍历数组中的每一个字符(i,j从1开始)。如果s[i]与t[j]的值相等，cost值为0，否则为1。D[i][j]的值为d[i-1,j] + 1(左边的值加1)、d[i,j-1] + 1.(上边的值加1)、d[i-1,j-1] + cost (斜上角的值加cost) 中的最小者。

4、等第三步遍历完后，右下角d[n,m]的值就为两个字符串的LD距离，如图4-8所示。

		W	O	R	L	D
	0	1	2	3	4	5
W	1	0	1	2	3	4
O	2	1	0	1	2	3
R	3	2	1	0	1	2
D	4	3	2	1	1	1

图 4-8 LD算法示意图

如果单纯靠编辑距离来匹配的话，搜索结果常常比较生硬，所以进行如下处理，得到distance之后，计算出匹配得分score，公式如下：

$$score = 1 - \frac{distance}{min(m,n)}$$

将score作为一个变参数，在使用函数的时候传入，可比较灵活的定义匹配的模糊程度，简单地说score越大这两个项之间的相关性越大，score越小，相关性越小。

4.4.2 相关类及方法

IndexReader类：这是一种抽象类，为访问Lucene索引提供了一个接口，任何对索引的搜索操作都需要通过IndexReader类的这个接口。它为IndexSearcher的搜索提供基础。使用该类的静态方法IndexReader.open(Directory dir)来获取IndexReader的实例，和4.3.2中构建IndexWriter的对象实例一样，它同样需要指定lucene索引的存放位置。

IndexSearcher类：该类是搜索索引的门户，所有的搜索都通过IndexSearcher的实例来完成。当构建好了IndexReader对象实例后，使用IndexSearcher带有参数的构造方法IndexSearcher(reader)来构建IndexSearcher对象实例，这个参数是IndexReader的对象实例。

QueryParser类：将用户输入的查询语句解析为Query对象，使得IndexSearcher的实例能够通过Query对象完成查询任务。通过QueryParser的parse(String query)方法

来解析一个查询字符串并返回一个Query对象。对于构建QueryParser类的对象实例，使用其带参数的构造方法：QueryParser(Version matchVersion, String fieldname, Analyzer defaultAnalyzer)。参数含义见表4-7。

表 4-7 QueryParser构造方法参数含义

参数	含义
Version matchVersion	QueryParser对应的Lucene版本，指定版本能有保证向后兼容性，在本系统中，使用的是lucene 3.5版本
String fieldname	所有被搜索项所对应的默认域
Analyzer defaultAnalyzer	在解析查询语句时所用的默认分析器

TopDocs类：存储IndexSearcher实例返回的结果。ScoreDoc类：提供对TopDocs中每条搜索结果的访问接口。在构建了IndexSearcher的对象实例后，通过调用它的search (Query query, int n)方法来进行搜索了，其中，第一个参数是Query的对象实例，第二个参数限制返回结果的数量，n大于0，也就是说search方法返回的结果数会小于等于n，这n个结果是与查询条件相关性最大的前n个。执行结果会返回一个TopDocs对象。在TopDocs.totalHits中存放着实际符合要求的文档的数量。当希望能够让IndexSearcher实例返回所有符合条件的文档时，可以运用如下编程方式来实现：

```
1 TopDocs hits = searcher.search(query,1);
2 if (hits.totalHits > 0)
3     hits = searcher.search(query,hits.totalHits);
4 else
5     hits = searcher.search(query,1);
6
```

第一次调用search()方法用于获知共有多少个文档符合查询语句的要求，由于search(Query query, int n)中的参数n需要大于0，故需要进行判断hits.totalHits是否大于0，当hits.totalHits 等于0时，search方法会抛出错误。

返回的TopDocs对象包含了一个ScoreDoc数组，ScoreDoc数组中是搜索结果对应文档的文档ID。使用IndexSearcher的doc(int docID)方法，就能把结果文档从Lucene索引目录中取出。

4.4.3 处理流程及实现

在本系统中，整个搜索过程如图4-9所示，

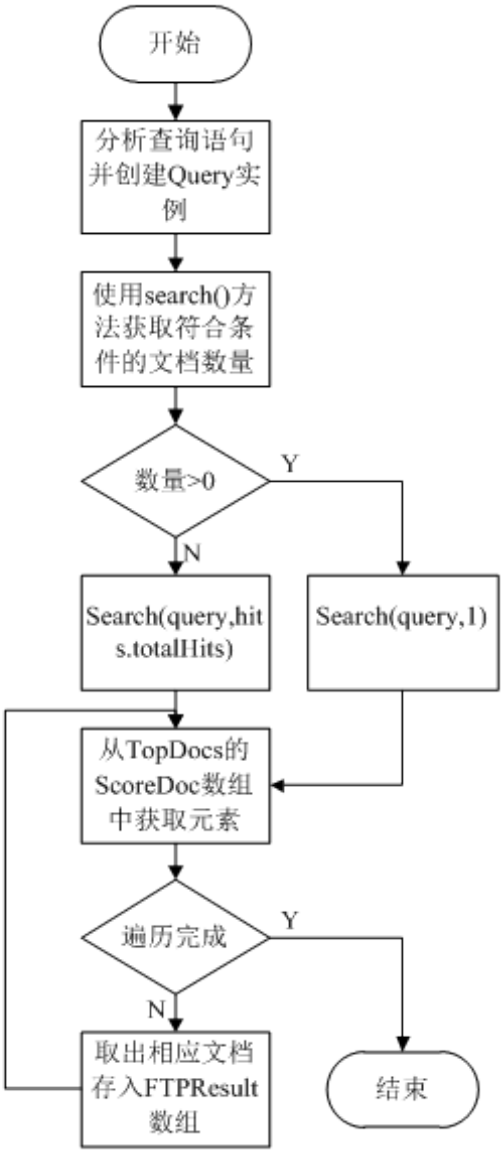


图 4-9 搜索流程图

在流程的最后会生成个FTPResults类的数组，该数组用于存储搜索结果，用于向Web前端输出。其结构如下：

```
1 public class FTPResults {
2     private String name,type,size,date,path,url;
3     ……
4     省略了部分getter和setter
5     ……
```

```
6     //将Byte转换到其他单位
7     public void setSize(String size) {
8         long numofsize = Long.valueOf(size);
9         int level = 0;
10         while (numofsize > 1023) {
11             numofsize = numofsize / 1024;
12             level++;
13         }
14         if (level == 0) {
15             this.size = "大小: " + numofsize + "B";
16         }
17         if (level == 1) {
18             this.size = "大小: " + numofsize + "KB";
19         }
20         if (level == 2) {
21             this.size = "大小: " + numofsize + "MB";
22         }
23         if (level == 3) {
24             this.size = "大小: " + numofsize + "GB";
25         }
26     }
27 //构造文件URL地址
28     public void setUrl(String path, String name, String type)
29     {
30         if (type.equals("folder")) {
31             this.url = path+name+"/";
32         } else {
33             this.url = path+name+"."+type;
34         }
35     }
36 }
37
```

4.5 基于MyBatis的数据库操作

4.5.1 MyBatis程序结构

首先，Mybatis需要有一个全局配置文件。在全局配置文件中需要配置的信息主要包括如下几个方面[11]：

Properties：用于提供一系列的键值对组成的属性信息，该属性信息可以用于整个配置文件中。

Settings：用于设置 MyBatis 的运行时方式，比如是否启用延迟加载等。

typeAliases ：为 Java 类型指定别名，可以在 XML 文件中用别名取代 Java 类的全限定名。

typeHandlers：在 MyBatis 通过 PreparedStatement 为占位符设置值，或者从 ResultSet 取出值时，特定类型的类型处理器会被执行。

objectFactory：MyBatis 通过 ObjectFactory 来创建结果对象。可以通过继承 DefaultObjectFactory 来实现自己的 ObjectFactory 类。

Plugins：用于配置一系列拦截器，用于拦截映射 SQL 语句的执行。可以通过实现 Interceptor 接口来实现自己的拦截器。

Environments：用于配置数据源信息，包括连接池、事务属性等。

Mappers：程序中所有用到的 SQL 映射文件都在这里列出，这些映射 SQL 都被 MyBatis 管理。

上面提及的大多数元素都不是必需的，通常 MyBatis 会为没有显式设置的元素提供缺省值。下面给出一个简单的全局配置代码：

```
1 <?xml version="1.0" encoding="UTF-8" ?>
2 <!DOCTYPE configuration
3   PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
4   "http://mybatis.org/dtd/mybatis-3-config.dtd">
5 <configuration>
6   <environments default="development">
7     <environment id="development">
8       <transactionManager type="JDBC"/>
9       <dataSource type="POOLED">
10          <property name="driver" value="com.mysql.jdbc.
```

```
11          Driver"/>
12          <property name="url" value="jdbc:mysql://localhost:↩
13          3306/test?useUnicode=true&amp;↩
14          characterEncoding=UTF8"/>
15          <property name="username" value="root"/>
16          <property name="password" value="940920"/>
17        </dataSource>
18      </environment>
19    </environments>
20    <mappers>
21      <mapper ↩
22      resource="ftpSearcher/mybatis/persistence/FeedbackMapper.↩
23      xml"/>
24      <mapper ↩
25      resource="ftpSearcher/mybatis/persistence/FTPServerMapper↩
26      .xml"/>
27      <mapper ↩
28      resource="ftpSearcher/mybatis/persistence/UserQueryLogMap↩
29      per.xml"/>
30      <mapper ↩
31      resource="ftpSearcher/mybatis/persistence/PathPriorityMap↩
32      per.xml"/>
33      <mapper ↩
34      resource="ftpSearcher/mybatis/persistence/QueryCountMappe↩
35      r.xml"/>
36      <mapper ↩
37      resource="ftpSearcher/mybatis/persistence/FTPSpiderLogMap↩
38      per.xml"/>
39    </mappers>
40 </configuration>
```

有了这些信息，MyBatis[12]便能够和数据库建立连接，并应用给定的连接池信息和事务属性。MyBatis 封装了这些操作，最终暴露一个 SqlSessionFactory 实

例供开发者使用，从名字可以看出来，这是一个创建 SqlSession 的工厂类，通过 SqlSession 实例，开发者能够直接进行业务逻辑的操作，而不需要重复编写 JDBC 相关的样板代码。根据全局配置文件生成SqlSession的代码如下：

```
1 String resource = "/mybatis-config.xml";
2 InputStream inputStream = Resources.getResourceAsStream(
3 resource);
4 SqlSessionFactory sqlSessionFactory = new 
5 SqlSessionFactoryBuilder().build(inputStream);
6 SqlSession session = sqlSessionFactory.openSession();
7
```

可以把上面的代码看做是MyBatis创建 SqlSession 的样板代码。其中第二行代码在类路径上加载配置文件，Resources 是 MyBatis 提供的一个工具类，它用于简化资源文件的加载，它可以访问各种路径的文件，不过最常用的还是示例中这种基于类路径的表示方式。

在完成全局配置文件，并通过 MyBatis 获得 SqlSession 对象之后，便可以执行数据访问操作了。对于 MyBatis 而言，要执行的操作其实就是在映射文件中配置的 SQL 语句。配置代码如下：

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
3 "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
4
5 <mapper namespace="ftpSearcher.mybatis.mapper.
6 FeedbackMapper">
7
8     <insert id="insertFeedback" parameterType="ftpSearcher.
9     model.Feedback">
10         INSERT INTO feedback
11         (username, comment,commentdate) VALUES
12         (#{username},#{comment},#{commentdate})
13     </insert>
14     <delete id="deleteByID" parameterType="int">
15         delete from feedback where id = #{id}
```

```
16     </delete>
17 </mapper>
```

在 MyBatis 中，namespace使得映射文件与接口绑定变得非常自然。下面将展示，XML映射文件与接口绑定之间的关系，与上述代码对应接口的定义代码如下：

```
1 package ftpSearcher.mybatis.mapper;
2 import java.util.List;
3 import ftpSearcher.model.Feedback;
4
5 public interface FeedbackMapper {
6     public void insertFeedback(Feedback feedback);
7     public List<Feedback> getAllFeedback();
8     public void deleteByID(int id);
9 }
10
```

表4-8给出了XML映射文件与接口直接的对应关系，以insertFeedback和getAll-Feedback属性为例[11]。

表 4-8 XML映射与接口对应关系

XML文件	接口
namespace="ftpSearcher.mybatis.mapper.FeedbackMapper"	接口的位置
id="insertFeedback"	接口对应的方法名insertFeedback
parameterType="ftpSearcher.model.Feedback"	insertFeedback的参数类型Feedback
resultType="ftpSearcher.model.Feedback"	getAllFeedback()返回的结果列表中对象的类型

在MyBatis框架中，只需要声明而不需要实现接口。完成了映射文件配置SQL语句及接口定义后，就可以用Mybatis的方式来进行相关数据库操作了。一个删除用户反馈的示例代码如下：

```
1 //假定session对象实例已被构建
2 FeedbackMapper feedbackMapper = session.getMapper(
3 FeedbackMapper.class);
4 feedbackMapper.deleteByID(feedbackID);
```

```
5 session.commit();
6 session.close();
7
```

在进行了删除操作之后一定要使用session.commit()将事务提交到数据库，否则删除、插入、修改这类操作是无法生效的。完成操作后，使用session.close()关闭与数据库的连接。

4.5.2 数据库操作

下面列举了一些系统中对数据库的操作。

1. FTP服务器表相关操作：

```
1 添加新的FTP服务器信息：INSERT INTO ftpserver (⤸
2 domain, ipv4, port, submittime, description) VALUES (#{⤸
3 domain},#{ipv4},#{port},#{submittime},#{description})
4 获取所有FTP服务器信息：SELECT * FROM ftpserver
5 获取指定FTP服务器信息：SELECT * FROM ftpserver ⤸
6 WHERE id = #{id}
7 更改指定FTP服务器信息：UPDATE ftpserver SET domain ⤸
8 = #{domain}, ipv4 = #{ipv4},port = #{port}, encoding = #{⤸
9 encoding}, verify= #{verify} WHERE id = #{id}
10 删除指定FTP服务器信息：DELETE FROM ftpserver WHERE ⤸
11 id = #{serverID}
```

2. 用户反馈表相关操作：

```
1 添加用户反馈：INSERT INTO feedback (username, comment,⤸
2 commentdate) VALUES (#{username},#{comment},#{commentdate})
3 读取所有用户反馈：SELECT * FROM feedback ORDER BY ⤸
4 commentdate DESC
5 删除用户反馈：DELETE FROM feedback WHERE id = #{id}
```

3. 用户搜索日志统计：SELECT queryword, COUNT(*) AS count FROM user-querylog WHERE querypage = 1 GROUP BY queryword ORDER BY COUNT(*) DESC

4.6 基于JDOM的XML文档操作

4.6.1 关键类及关键方法介绍

在本系统中，使用 SAXBuilder 对ftpSearcher-config.xml 文件以SAX的方式进行语法分析，构建JDOM的Document实例。 Document实例是执行XML文档相关操作的基础。下面是构建JDOM文档的过程：

```
1 SAXBuilder buider = new SAXBuilder();
2 //获得ftpSearcher-config.xml的位置
3 URL path = this.getClass().getClassLoader()
4             .getResource("ftpSearcher-config.xml");
5 File file = new File(path.getPath())
6 Document doc = buider.build(file);
7
```

在构建好Document实例后，就能通过JDOM提供的方法获取XML里中所需要的内容了。

4.6.2 FTP搜索引擎配置文件结构及文档操作

FTP搜索引擎的配置代码如下：

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <ftpSearcher>
3     <fileInfoDir>D:/FTPSearcher/FTPFileInfo</fileInfoDir>
4     <indexDir>D:/FTPSearcher/LuceneIndex</indexDir>
5     <logDir>D:/FTPSearcher/FTPSearcherLog</logDir>
6     <username>21232f297a57a5a743894a0e4a801fc3</username>
7     <password>21232f297a57a5a743894a0e4a801fc3</password>
8 </ftpSearcher>
```

fileInfoDir：存放FTP服务器搜集模块所搜集的FTP文件信息文本。

indexDir：存放Lucene索引的目录。

logDir:存放对比模块产生的文件对比文本。

username：后台管理系统用户名字符串的MD5值。

password：后台管理密码字符串的MD5值。

存放MD5值相较于原始值来说，更为安全，但是如果密码的原始字符串是常见密码，MD5值也并不安全。可以通过暴力破解来非法登陆系统。

1. 获取内容

以4.6.1中构建好的Document对象实例doc为基础，获取XML文件中的内容需要通过一下的方法：

(1) 使用Document类的getRootElement()方法获取了XML的根元素的Element对象实例。

(2) Element类的getChildText(”子元素名”)来获取子元素的内容。

具体的核心代码如下：

```
1 Element root = doc.getRootElement();
2 indexDir = root.getChildText("indexDir");
3 fileInfoDir = root.getChildText("fileInfoDir");
4 logDir = root.getChildText("logDir");
5 username = root.getChildText("username");
6 password = root.getChildText("password");
7
```

2. 修改内容

修改XML文件需要使用以下的类及其方法。

(1)使用root的root.getChild(”子元素名”)获取子元素的Element实例，如Element eIndexDir = root.getChild(”indexDir”)。

(2)使用Element类的setText(字符串)方法修改元素内容，如eIndexDir.setText(indexDir)。

(3)使用XMLOutputter().outputString(doc)方法将Document对象输出为一个字符串。

(4)创建FileWriter对象实例。使用FileWriter类的write(字符串)方法把修改结果写入文件。

具体的核心代码如下：

```
1 Element eIndexDir = root.getChild("indexDir");
2 eIndexDir.setText(indexDir);
3
4 Element eFileInfoDir = root.getChild("fileInfoDir");
5 eFileInfoDir.setText(fileInfoDir);
```

```
6
7 Element eLogDir = root.getChild("logDir");
8 eLogDir.setText(logDir);
9
10 if (password.length() != 0) {
11     Element ePassword = root.getChild("password");
12     ePassword.setText(getMD5(password));
13 }
14
15 String des = new XMLOutputter().outputString(doc);
16 FileWriter fileWriter = new FileWriter(file);
17 fileWriter.write(des);
18 fileWriter.close();
19
```

4.7 用户搜索日志记录

4.7.1 用户搜索日志的意义

单个来看，每条搜索日志都是平淡无奇的，但是搜集每个用户、每次查询、每次应答的日志，将它们放在一起，就会成为一个内含丰富、深藏玄机的资料集。例如将一个用户在一个时间段内，针对某个搜索目标所进行的一系列查询和点击记录放在一块，就可以大致推测，对这样的用户、这样的需求，哪些结果是他们希望看到的，从而优化结果的排序。此外，综合众多用户的查询日志，可以知道哪些查询词是用户搜得最多的，哪些查询词是某一段时间内搜索量增长最快的，从而分析搜索的趋势。例如百度风云榜（http://top.baidu.com/），某个时期，百度用户最为关心的、搜索频率最高的事件，都在榜单里有所体现。榜单的生成就离不开千千万万平凡的搜索日志。同样，Google也有类似的资料发布在互联网上，在http://www.google.com/intl/en/press/zeitgeist/index.html可以看到相关的内容，顺便领略一下，在Google的用户群中，大家关心的是什么内容。

这些只是日志中所体现的非常基本的内在信息。对搜索引擎日志的分析，还能帮助开发者理解用户的查询意图、理解搜索结果的内容、评判搜索结果质量、

改进搜索系统等一系列有用的事情，可以为人们不断改进获取有效信息的便捷性提供有力的帮助。

4.7.2 记录用户搜索的方法

在本系统中，用户搜索日志记录的模块作为一个Struts2架构中的拦截器来完成工作。原理如图4-10所示：

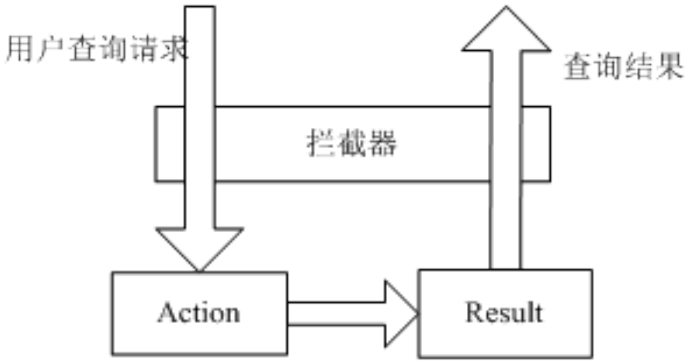


图 4-10 拦截器原理

拦截器将用户请求拦截下来，将与用户查询相关的信息写入数据库，然后再把请求交给Action处理。代码实现如下：

```
1 public String intercept(ActionInvocation actionInvocation) ⤸
2 throws Exception {
3     UserQueryLog userQueryLog = new UserQueryLog();
4     int page = 1;
5     if (ServletActionContext.getRequest().getParameter(⤸
6     "page") != null) {
7         page = Integer.valueOf(ServletActionContext.⤸
8         getRequest()
9         .getParameter("page"));
10     }
11     userQueryLog.setQuerypage(page);
12     userQueryLog.setQuerytime(new Date(System.⤸
```

```
13      currentTimeMillis()));
14      userQueryLog.setQueryword(ServletActionContext.
15      getRequest()
16      .getParameter("keywords"));
17
18      try {
19          String resource = "/mybatis-config.xml";
20          InputStream inputStream = Resources.
21          getResourceAsStream(resource);
22          SqlSessionFactory sqlSessionFactory = new
23          SqlSessionFactoryBuilder()
24          .build(inputStream);
25          SqlSession session = sqlSessionFactory.openSession();
26
27          UserQueryLogMapper userQueryLogMapper = session
28          .getMapper(UserQueryLogMapper.class);
29          userQueryLogMapper.inserUserQueryLog(userQueryLog);
30
31          session.commit();
32          session.close();
33      } catch (Exception e) {
34          e.printStackTrace();
35      }
36      return actionInvocation.invoke();
37  }
38  }
39
```

为使得拦截器能够生效，在struts2配置文件struts.xml中编写代码如下：

```
1 <action name="search" class="ftpSearcher.action.FTPSearch">
2     <interceptor-ref name="defaultStack" />
3     <interceptor-ref name="userQueryLoginterceptor" />
```

```
4     <result name="success">/results.jsp</result>
5 </action>
```

第5章 系统测试与运行

测试工程是软件开发的一个重要阶段，也是软件开发的最后阶段。通过测试工程对软件进行测试，验证软件的有效性，发现并修补软件系统的缺陷，以提高软件质量，确保开发出用户满意得软件产品。

5.1 Web前端测试

打开搜索主界面，如图5-1所示:

图 5-1 Web前端主页

5.2 系统后台管理测试

后台系统登录界面如图5-2所示：

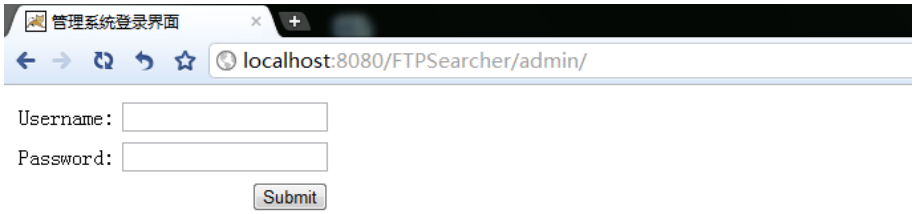


图 5-2 系统后台登陆界面

输入用户名：admin 密码：admin并登陆后如图5-3所示：

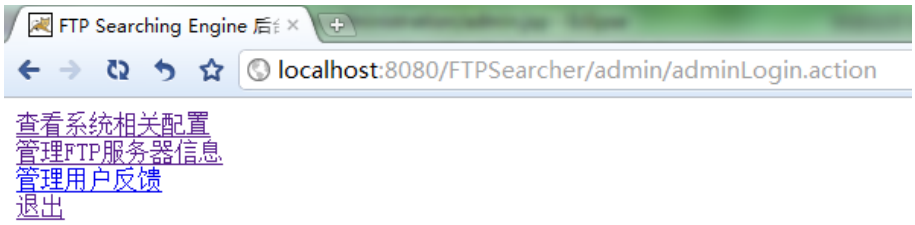


图 5-3 后台管理界面

点击“查看系统相关配置”，如图5-4所示：

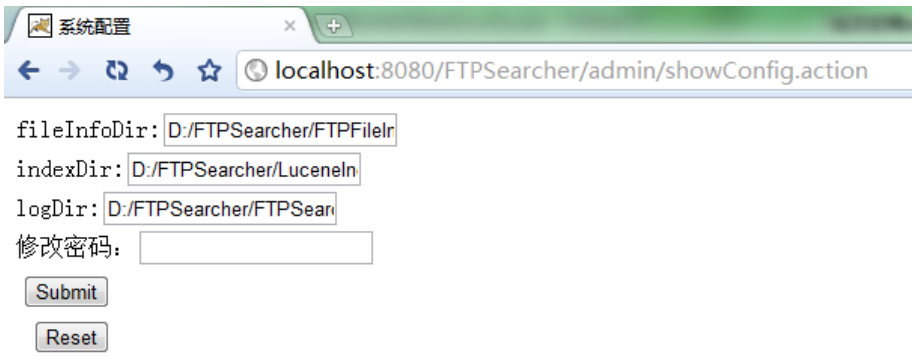


图 5-4 系统相关配置

5.3 搜索测试

键入想要查询的关键词，如图5-5所示：

图 5-5 关键词输入

点击“搜一下”，得到搜索结果，如图5-6所示：

图 5-6 搜索结果

点击第二页链接，进行翻页测试，如图5-7所示：

图 5-7 翻页测试

翻页结果如图5-8所示：

图 5-8 翻页结果

模糊搜索测试结果如图5-9所示：

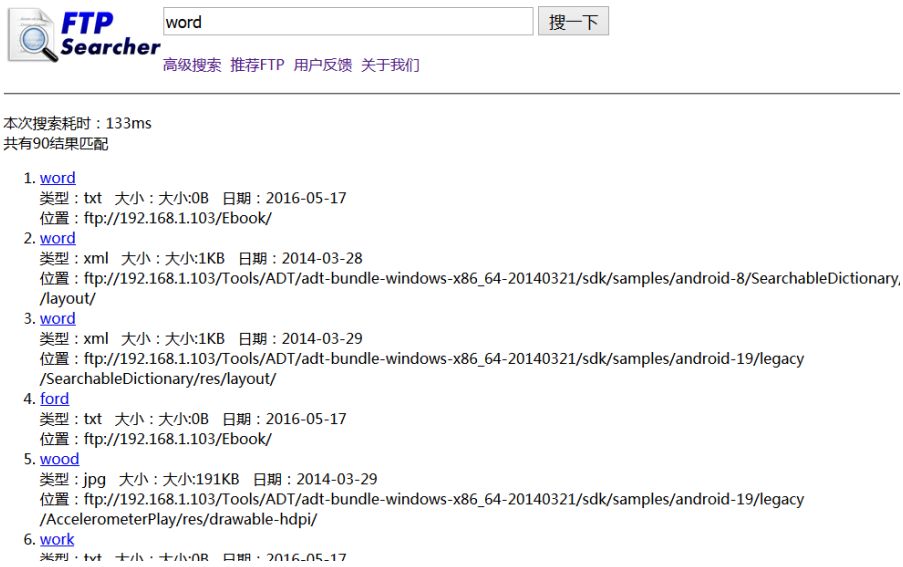


图 5-9 模糊搜索测试

从图5-9中可以看出搜索结果中包含与键入关键词”word”相近的几个关键词。

5.4 测试结论

本系统成功完成了FTP搜索引擎的设计与实现，实现了模糊搜索功能，并且在可控范围内对相关性进行了良好的排序，性能方面良好，在测试所用的10GB数据量下搜索延迟不超过0.5s。

第6章 结束语

6.1 总结

搜索引擎作为Web浏览的主要工具，已经受到业界越来越广泛的关注。整个FTP搜索引擎的实现是基于Lucene全文检索工具包与Spider程序来完成数据的搜索工作，使用Struts2框架来实现B/S架构程序开发，使用Mybatis作为持久层，完成与数据库的交互工作。本文的工作可归纳为4个方面：

1. 研究并阐述了FTP相关技术的核心原理。
2. 基于Lucene完成了对源数据索引的改进、构建与中文分词的设计。
3. 以软件工程的设计为顺序，介绍了整个项目从零到整的开发过程。
4. 重点介绍了核心模块的实现方法。

6.2 展望

为了进一步完善搜索引擎系统，本课题还需要在多个方面进一步进行研究。

1. 关键词自动纠错：有些时候，用户输入的关键词会出现拼写错误，例如“哈利波特”拼写成“哈力波特”，当用户输入“哈力波特”，能够向用户提示“哈利波特”。

2. 多线程的数据搜集模块：对于一个普通的FTP服务器来说，遍历并获取整个服务器的文件信息，耗时会受到数据量及网络状况等因素的影响。当需要遍历的FTP服务器数量增多时，时间会相对较长。通过多线程的方式，同时遍历同一服务器的不同目录或同时遍历不同的FTP服务器可以缩短数据采集的时间，需要好的策略去设计、实现、管理并发的搜集过程。

3. 性能优化：当网站访问量变大后，整个系统的性能可能会遇到瓶颈，构建分布式的搜索引擎可以解决瓶颈问题，提高搜索的速度。

4.界面优化：应该重新设计用户界面，使得其更加美观。

参考文献

[1] 黎冬. 基于Linux平台Ftp搜索引擎的研究[D]. 湖北: 湖北工业大学, 2009, 50–60

[2] 王爱国, 杨波, 柴乔林. 基于XML Web Service的FTP搜索技术[J]. 济南大学学报（自然科学版）, 2005, 19(3):230–234

[3] 龚达. 基于IA和Web Service的FTP搜索引擎设计[J]. 江南大学学报(自然科学版), 2003, 2(1):59–65

[4] 赵珂, 逯鹏, 李永强. 基于Lucene的搜索引擎设计与实现[J]. 计算机工程, 2011, 37(16):39–41

[5] 管建和, 甘剑峰. 基于Lucene全文检索引擎的应用研究与实现[J]. 计算机工程与设计, 2007, 28(2):489–491

[6] 于天恩. Lucene搜索引擎开发权威经典[M]. 北京: 中国铁道出版社, 2008

[7] 詹礼军. Lucene字典实现原理[EB/OL]. http://lucene.apache.org/core/old_versioned_docs/versions/3_5_0/fileformats.html, 2012

[8] E. Brown. FST for Natural Language Processing[EB/OL]. http://infolocata.com/mirovia/finite-state-transducers-for-natural-language-processing/, 2014

[9] Pandora. Levenshtein Distance 算法详解[EB/OL]. http://www.2cto.com/kf/201407/314271.html, 2014

[10] L. Xiang-yang, Z. Ya-fei. Fast Hash algorithm for Chinese word segmentation[J]. Natural Science Edition, 2004, 5(2):40–42

[11] 张建平. 凤凰涅槃：从iBatis到MyBatis[EB/OL]. http://www.ibm.com/developerworks/cn/opensource/os-cn-mybatis/, 2011

[12] MyBatis.org. Mapper XML Files[EB/OL]. http://www.mybatis.org/core/sqlmap-xml.html, 2012

[13] H. Chang-ning. Chinese Word Segmentation in Chinese Information Processing[J]. Applied Linguistics, 1997, 21(1):72–78

致 谢

历时将近两个月的时间终于将这篇论文写完，在论文的写作过程中遇到了无数的困难和障碍，都在同学和老师的帮助下度过了。尤其要强烈感谢我的论文指导老师—赵勇老师，他对我进行了无私的指导和帮助，在设计过程中解决了我许多困惑的地方和技术上的问题，在论文撰写过程中帮助我对论文进行修改和改进。另外，在校图书馆查找资料的时候，图书馆的老师也给我提供了许多方面的支持与帮助。在此向帮助和指导过我的各位老师表示最衷心的感谢!

感谢这篇论文所涉及到的各位学者。本文引用了众多专家学者的研究文献，如果没有各位学者的研究成果的帮助和启发，我将很难完成本篇论文的写作，同时也要感谢各大开源论坛上无私奉献的科技工作者，为科技工作无私奉献着。

整个毕业设计的过程中我学到了很多东西，不仅仅是毕业设计所研究的专业内容、，更多的是对待工作的认真仔细，大量写代码时的专注投入，还有工作完成后的满足感，这一切都已经深深印入了我的心中，受益终身。

最后，向在百忙中抽出时间对本文进行评审并提出宝贵意见的各位专家表示衷心地感谢!

Finite-State-Transducers for Natural Language Processing

Finite State Transducers provide a method for performing mathematical operations on ordered collections of context-sensitive rewrite rules such as those commonly used to implement fundamental natural language processing tasks. Multiple rules may be composed into a single pass, mega rule, significantly increasing the efficiency of rule-based systems.

1.1 Quick Primer on Finite State Machines

A finite-state machine (FSM) or automata is an abstract mathematical model of computation that is capable of storing a status or state and changing this state based on input. While not as powerful as other computational models such as Turing machines due to their limited memory, FSMs are applicable to a number of electronic modeling, engineering, and language processing problems. An FSM can be represented as a set of nodes representing the various states of the system and labeled edges between these nodes where the edges represent transitions from one state to another and the labels represent conditions on these transitions. A stream of input (an input tape containing a string) is then ‘processed’ by the FSM, potentially causing a number of state transitions. The simple FSM example below is a remarkably accurate computational model for a ferret:

The daily behavior of my ferret, Pangur Bán, would best be represented by an input tape containing the string, ‘tired,tired,tired,tired,hungry,tired,tired,tired,bored,tired,tired, tired’.

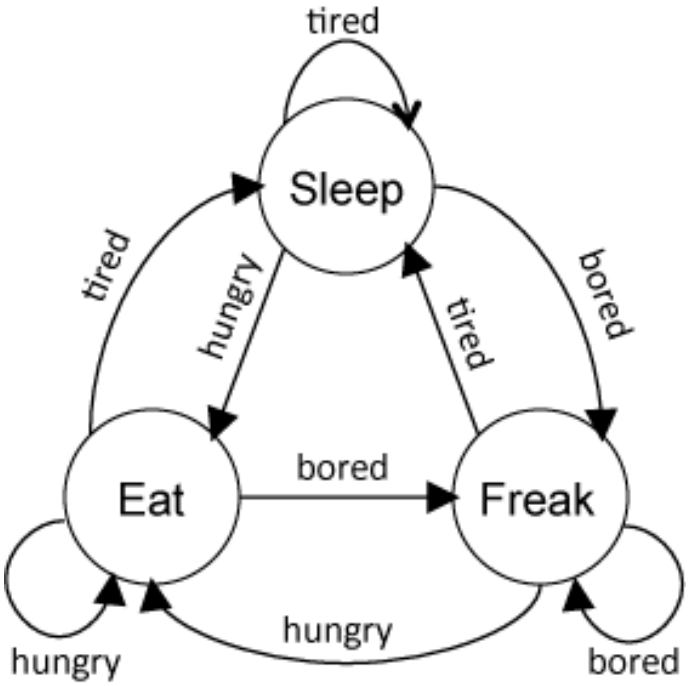


图 A-1 FSM example for a ferret

1.2 FSMs for Language Processing

In language processing, an FSM containing a start state (node) and an end state can be used to generate or recognize a language defined by all possible combinations of symbols (conditional labels) on each of the edges generated by traversing the FSM from the start state to the end state. The class of languages generated by finite automata is known as the class of regular languages.

1.3 Finite State Transducers

A finite state transducer (FST) is a special type of finite state machine. Specifically, an FST has two tapes, an input string and an output string. Rather than just traversing (and accepting or rejecting) an input string, an FST translates the contents of its input string to its output string. Or put another way, it accepts a string on its input tape and generates another string on its output tape.

1.4 Context-Sensitive Rules

FSTs are particularly useful in the implementation of certain natural language processing tasks. Context-sensitive rewriting rules (ex: ax→bx) are adequate for implementing certain computational linguistic tasks such as morphological stemming and part-of-speech tagging. Such rewrite rules are also computationally equivalent to finite-state transducers, providing a unique path for optimizing rule based systems.

Take, for example, the following set of ordered context-sensitive rules:

1) change ‘c’ to ‘b’ if followed by ‘x’ cx→bx

2) change ‘a’ to ‘b’ if preceded by ‘rs’ rsa→rsb

3) change ‘b’ to ‘a’ if preceded by ‘rs’ and followed by ‘xy’ rsbxy→rsaxy

Given the following string on the input tape:

rsaxyrscxy

the application of the given rule set would proceed as follows:

1) rsaxyrscxy→rsaxyrsbxy

2) rsaxyrsbxy→rsbxyrsbxy

3) rsbxyrsbxy→rsaxyrsaxy

The time required to apply a sequence of context-sensitive rules is dependent upon the number of rules, the size of the context window, and the number of characters in the input string. The inefficiencies in such an implementation are highlighted in this example by the multi-step transformation required to translate ‘c ‘ to ‘b ‘ then ‘b ‘ to ‘a ‘ by rules 1 and 3, and the redundant transformation of ‘a ‘ to ‘b ‘ and back to ‘a ‘ by rules 2 and 3. Finite State Transducers provide a path to eliminate these inefficiencies. But first we need to convert the rules to State Machines.

1.5 Converting Rules to FSTs

To do this, we simple represent each rule as an FST where each link between states represent the acceptance of an input character and the expression of the corresponding output character. This input / output combination is denoted within an FST by labeling edges with both the input and output character separated by the ‘/’ character. Following is an FST for each of the above rules:

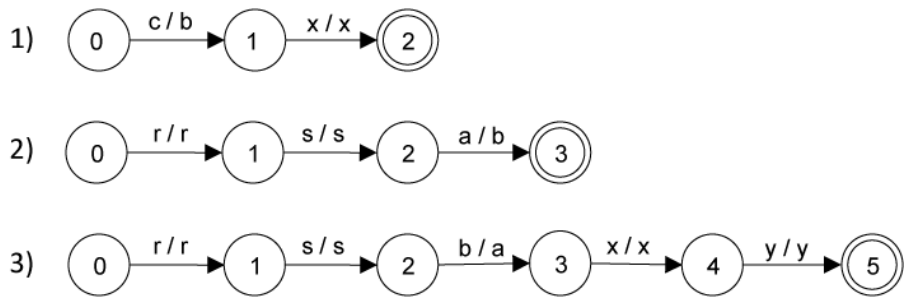


图 A-2 FST

1.6 Extending the FSTs

While the FSTs above represent our set of context-sensitive rules, they would be of little use in matching against an input string as each is designed to process exactly the context window described in its corresponding rule. To make each FST applicable to a string of arbitrary length and perform the necessary translation each time the rule is fired, we will need to extend each of the Transducers. We do this by allowing for every possible input in our language at each state. For example, rule 1 must be able to handle the string rsaxyrscxy. Since rule 1 matches the string ‘cx‘ and outputs the string ‘bx‘, it must handle the characters ’ r ‘, ’ s ‘, ’ a ‘, ’ x ‘, ’ y ‘, ’ r ‘, and ‘s‘ before finally encountering ‘c‘ and ’ x ‘. Then it must handle the final ‘y‘ character. Each of these characters (and all other possible characters) could be explicitly listed on its own individual edge, but to simplify, we can create a single edge labeled with ‘?/?‘, to match and output any character not already represented on another edge leading from that state. FST extension of rules with a trailing context will also require the use of edges with an input but no output (labeled with ‘ε‘ for output) and edges with multiple outputs. Following is the extension for each of the above FSTs:

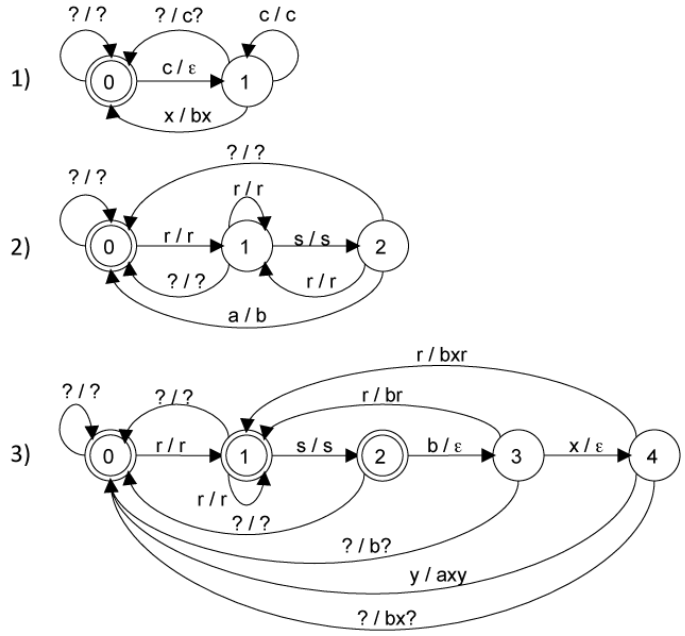


图 A-3 Extending FSTS

1.7 Composing a single FST

One of the operations that can be performed on a pair of FSTs is composition. The composition operation will take two deterministic FSTs, A and B, and combine their nodes and edges into a single deterministic FST. The resulting Transducer will accept an input string of arbitrary length and output the equivalent of applying Transducer A followed by Transducer B. A full explanation of the FST composition algorithm is beyond the scope of this write-up. Following is the FST resulting from the composition of FSTs 1 and 2 followed by the composition of the resulting FST and FST 3:

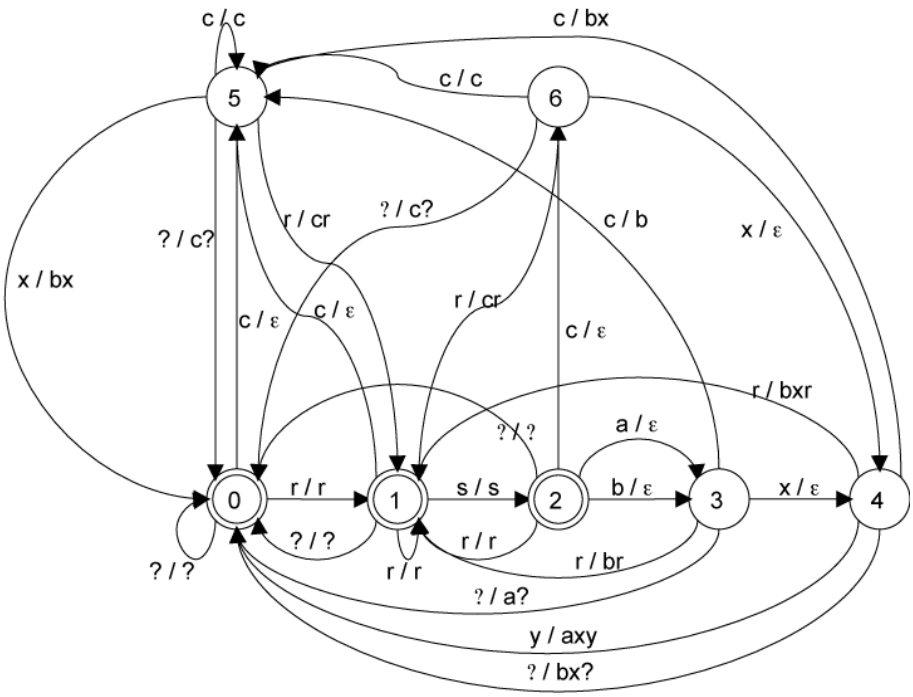


图 A-4 Final FST

Note that while the output of applying the final FST to the input string ‘rsaxyrscxy’ is exactly equivalent to the output of applying each of the individual context-sensitive rules (‘rsaxyrsaxy‘), the transformation required only a single pass through the FST and did not result in any inefficient transformations. While the time required to apply the original rules was dependent upon the number of rules, the size of the context window, and the number of characters on the input tape, application time of the final FST is dependent only upon the number of characters on the input tape.

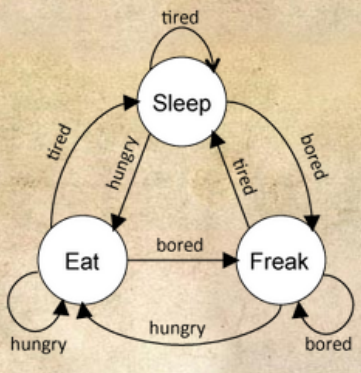
Finite-State-Transducers for Natural Language Processing

top

Finite State Transducers provide a method for performing mathematical operations on ordered collections of context-sensitive rewrite rules such as those commonly used to implement fundamental natural language processing tasks. Multiple rules may be composed into a single pass, mega rule, significantly increasing the efficiency of rule-based systems.

Quick Primer on Finite State Machines

A finite-state machine (FSM) or automata is an abstract mathematical model of computation that is capable of storing a status or state and changing this state based on input. While not as powerful as other computational models such as Turing machines due to their limited memory, FSMs are applicable to a number of electronic modeling, engineering, and language processing problems. An FSM can be represented as a set of nodes representing the various states of the system and labeled edges between these nodes where the edges represent transitions from one state to another and the labels represent conditions on these transitions. A stream of input (an input tape containing a string) is then 'processed' by the FSM, potentially causing a number of state transitions. The simple FSM example below is a remarkably accurate computational model for a ferret:



The daily behavior of my ferret, Pangur Bán, would best be represented by an input tape containing the string, *'tired,tired,tired,tired,hungry,tired,tired,tired,bored,tired,tired,tired'*:

FSMs for Language Processing

In language processing, an FSM containing a start state (node) and an end state can be used to generate or recognize a language defined by

图 A-5

FST在自然语言处理上的应用

有限状态传感器提供了一种方法，用于执行上下文相关的重写规则方面的数学运算，这种方法通常用于实现基础的自然语言处理任务。多种规则可能会组成一个单通的复杂规则，可以显着提高基于规则的系统的效率。

1.1 有限状态机的快速入门

一个有限状态机或者自动机是一种抽象的数学计算模型，可以存储当前的状态或者根据输入来改变这种状态。有限状态机由于本身有限的存储，虽然没有像图灵机之类的其他计算模型一样强大的能力，但也适用于许多电子建模、工程和语言处理方面的问题。一个有限状态机由一组各种各样表示系统状态的节点和这些节点之间代表从一种状态转移到另一种状态的带有标记的边组成，这些标记表示了这些状态转换的条件。一个输入流（一次输入包含一个串）会被有限状态机“处理”，造成一系列的状态转换。下面的一个关于雪貂的精确计算模型就是一个简单的有限状态机实例：

我的雪貂“Pangur Bán”每天的行为，可以很好的被表示为这样一个串“累了，累了，累了，饿了，累了，累了，累了，无聊了，累了，累了，累了”。

1.2 语言处理的有限状态机

在语言处理中，一个有限状态机包含了个起始状态（节点）和一个最终状态可以用来生成或者识别一种由所有可能符号（条件标签）组合而成的语言，这些符号（条件标签）在每次从有限状态机初始状态到最终状态的过程中产生。有限自动机产生的语言类称为正则语言类。

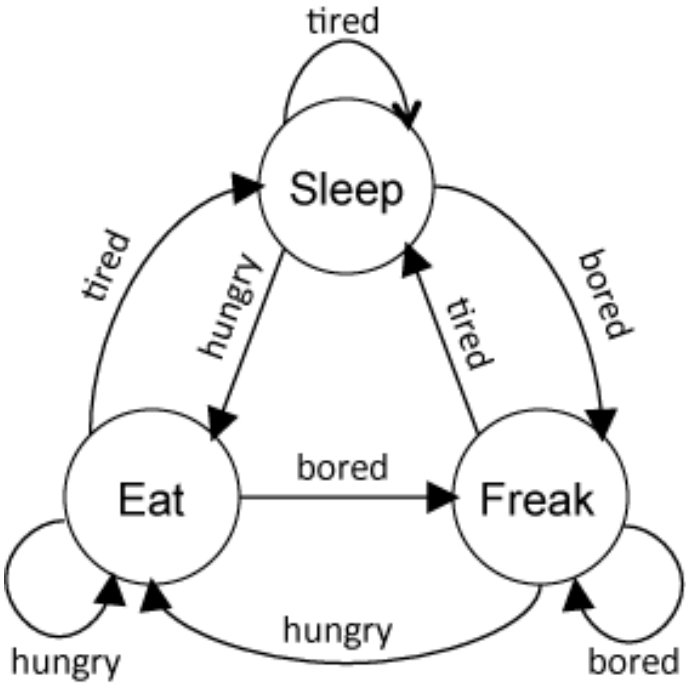


图 A-1 雪貂FSM实例

1.3 有限状态转移机

有限状态转移机（FST）是一种特殊形式的有限状态机。特别的事，一个有限状态转移机有两种串，一个输入串和一个输出串。相比起对于某种输入串要么不改变，要么就接受条件做出转移的有限状态机来说，有限状态转移机将这种输入串翻译成了输出串。或者换种说法，它接受一个字符串的输入，然后输出另一个字符串。

1.4 上下文相关规则

FST在实现一些自然语言处理的任务中是十分有用的。上下文重写规则（比如ax→bx）足够实现许多计算语言方面的任务，例如形态产生和词性标注。这些重写规则也等同于为FST提供了一种独特的路径用于优化基于规则的系统。

那么，举个例子，下面是几个有序的上下文规则：

1) 如果紧跟在后面的是‘x’，把‘c’变成‘b’ cx→bx

2) 如果在前面的是‘rs’，把‘a’变成‘b’ rsa→rsb

3) 如果跟在前面的是‘rs’，跟在后面的是‘xy’，把‘b’变成‘a’ rsbxy→rsaxy

所以当给定的输入字符串是rsaxyrscxy时，根据以上三个规则，我们就可以做出以下的变换：

1) rsaxyrscxy→rsaxyrs b xy

2) rsaxyrsbxy→rs b xyrsbxy

3) rsbxyrsbxy→rs a xyrs a xy

提供一个上下文相关的规则所需要的时间取决于规则的数目，上下文容量的大小和输入字符串的字符数量。在这个转换的例子中效率明显非常低，通过多步变化需要将‘c’转变为‘b’然后将‘b’转变为‘a’通过规则1和3，通过规则2和3把‘a’转换成‘b’，又把‘b’转换成‘a’是多余的转换。有限状态转移机提供了一种消除这种低效工作的路径。不过首先我们需要将规则转变为状态机。

1.5 将规则转换成有限状态转移机

为了达到目的，我们首先讲过每个规则表示成一个有限状态转移机，每个状态之间的链接表示接受了输入字符，并输出相应的字符。这种输入/输出的组合是指在有限状态转移机标签杓袁的输入和输出特性之间用字符‘/’分隔。下面就是上述三种规则转变之后的有限状态转移机：

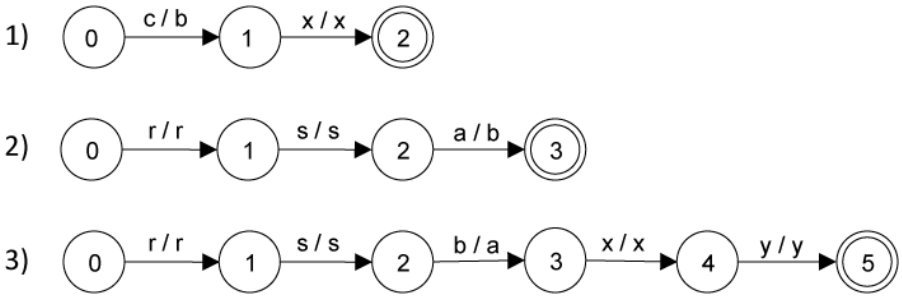


图 A-2 FST

1.6 扩展有限状态转移机

当上面的有限状态转移机表示了我们上下文相关规则的集合时，他们将会很少用于匹配一个输入字符串，而是用于精确处理上下文窗口，描述他们相对应

的规则。为了使每个有限状态转移机适用于任意长度的字符串并且在每次满足规则时都执行必要的转换，我们将需要扩展这些转移机。我们通过允许在每个状态接受任意可能的输入来达到这个目的。比如说，规则1必须能够处理字符串‘rsaxyrscxy’。当规则1匹配了字符串‘cx’而且输出了字符串‘bx’之后，他必须在最后遇到‘c’和‘x’之前处理字符‘r’，‘s’，‘a’，‘x’，‘y’，‘r’和‘s’。然后在最后处理最后字符‘y’。每一个这些可能的字符（还有其他所有可能的字符）都应该明确的被列在它自己独立的边上，不过为了简化过程，我们可以建立一条独立的边，标注为‘?/?’，用于匹配和输出任何此状态下尚未被其他边表示的字符。有限状态转移机扩展规则中的尾部将需要使用只有输入没有输出的边（标注为‘ε’用于输出）和拥有多个输出的边。下面就是对上述有限状态转移机的扩展：

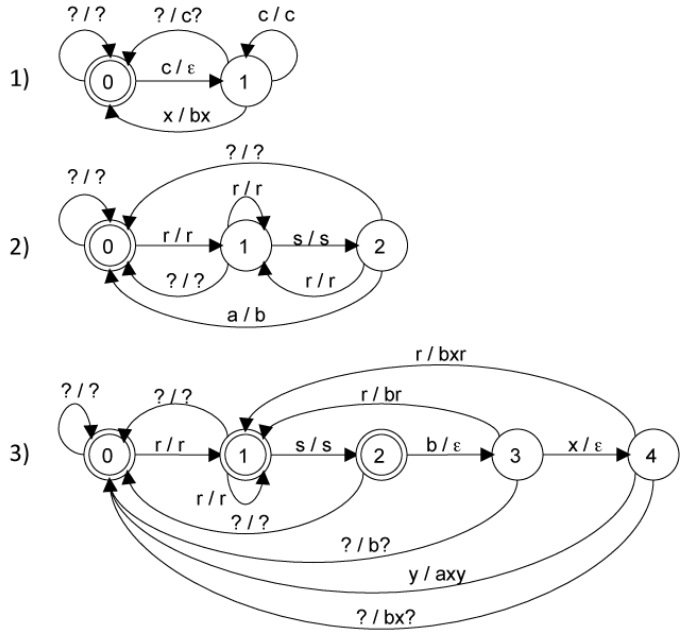


图 A-3 扩展FST

1.7 合并成单一的有限状态转移机

合并运算是在两个有限状态转移机中可以执行的操作之一。合并操作需要使用两个确定的FST，A和B，然后将他们的节点和边联合成一个确定的有限状态转移机。有此产生的转移机能够接受一个任意长度的输入字符串然后输出一个字符串，这个字符串与先经过转移机A处理再经过转移机B处理得到的字符串一致。

一个完整的有限状态转移机合并运算算法已经超出本文描写的范围了。下面是将FST1和FST2合并运算之后得到的FST再和FST3合并运算得到的FST：

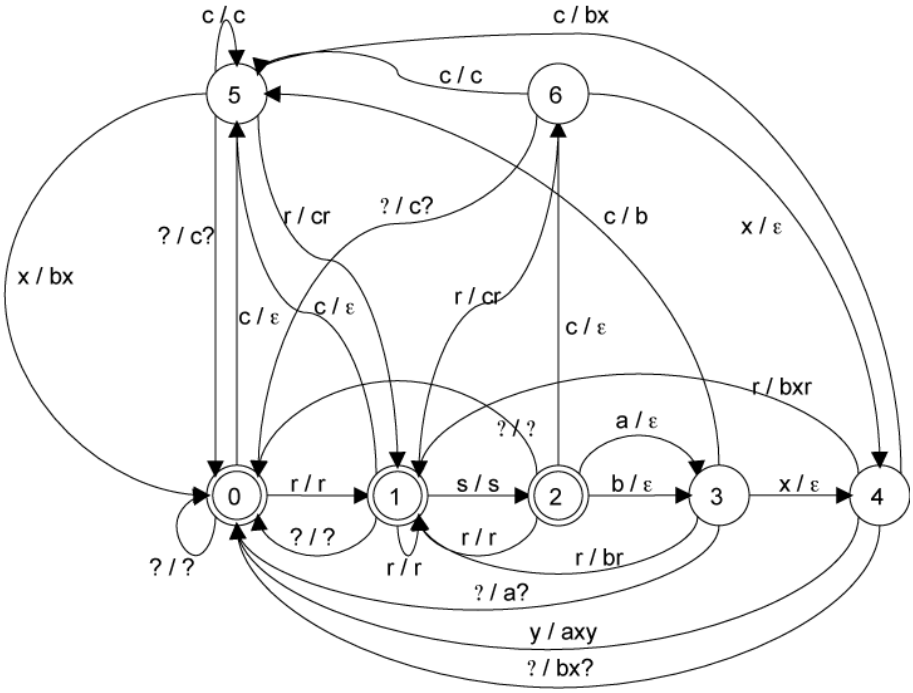


图 A-4 最终FST

注意，当应用最终有限状态转移机来处理输入字符串‘rsaxyrscxy’时，得到的输出结果已经完全等同于使用独立的上下文相关规则来处理同样字符串时得到的结果，转换只需要通过一次最终有限状态转移机，没有造成任何低效的转换。与耗时取决于规则的数目，上下文容量的大小和输入字符串字符数量的原始规则想比，最终有限状态转移机的耗时仅仅取决于输入字符串字符数量。