搜索引擎系统学习与开发实践总结

总结人：相生昌

Email:superxsc@126.com

MSN:superxsc@hotmail.com

2005.4.18

中国知网数图研发部

目　　录

一、搜索引擎概述

搜索引擎的发展历史

在互联网发展初期，网站相对较少，信息查找比较容易。然而伴随互联网爆炸性的发展，普通网络用户想找到所需的资料简直如同大海捞针，这时为满足大众信息检索需求的专业搜索网站便应运而生了。

现代意义上的搜索引擎的祖先，是 1990 年由蒙特利尔大学学生 Alan Emtage 发明的 Archie。虽然当时 World Wide Web 还未出现，但网络中文件传输还是相当频繁的，而且由于大量的文件散布在各个分散的 FTP 主机中，查询起来非常不便，因此 Alan Emtage 想到了开发一个可以以文件名查找文件的系统，于是便有了 Archie。

Archie 工作原理与现在的搜索引擎已经很接近，它依靠脚本程序自动搜索网上的文件，然后对有关信息进行索引，供使用者以一定的表达式查询。由于 Archie 深受用户欢迎，受其启发，美国内华达 System Computing Services 大学于 1993 年开发了另一个与之非常相似的搜索工具，不过此时的搜索工具除了索引文件外，已能检索网页。

当时，“机器人”一词在编程者中十分流行。电脑“机器人”（Computer Robot）是指某个能以人类无法达到的速度不间断地执行某项任务的软件程序。由于专门用于检索信息的“机器人”程序象蜘蛛一样在网络间爬来爬去，因此，搜索引擎的“机器人”程序就被称为“蜘蛛”程序。

世界上第一个用于监测互联网发展规模的“机器人”程序是 Matthew Gray 开发的 World wide Web Wanderer。刚开始它只用来统计互联网上的服务器数量，后来则发展为能够检索网站域名。

与 Wanderer 相对应，Martin Koster 于 1993 年 10 月创建了 ALIWEB，它是 Archie 的 HTTP 版本。ALIWEB 不使用“机器人”程序，而是靠网站主动提交信息来建立自己的链接索引，类似于现在我们熟知的 Yahoo。

随着互联网的迅速发展，使得检索所有新出现的网页变得越来越困难，因此，在 Matthew Gray 的 Wanderer 基础上，一些编程者将传统的“蜘蛛”程序工作原理作了些改进。其设想是，既然所有网页都可能有连向其他网站的链接，那么从跟踪一个网站的链接开始，就有可能检索整个互联网。到 1993 年底，一些基于此原理的搜索引擎开始纷纷涌现，其中以 JumpStation、The World Wide Web Worm（Goto 的前身，也就是今天 Overture），和 Repository-Based Software Engineering (RBSE) spider 最负盛名。

然而 JumpStation 和 WWW Worm 只是以搜索工具在数据库中找到匹配信息的先后次序排列搜索结果，因此毫无信息关联度可言。而 RBSE 是第一个在搜索结果排列中引入关键字串匹配程度概念的引擎。

最早现代意义上的搜索引擎出现于 1994 年 7 月。当时 Michael Mauldin 将 John Leavitt 的蜘蛛程序接入到其索引程序中，创建了大家现在熟知的 Lycos。同年 4 月，斯坦福（Stanford）大学的两名博士生，David Filo 和美籍华人杨致远（Gerry Yang）共同创办了超级目录索引 Yahoo，并成功地使搜索引擎的概念深入人心。从此搜索引擎进入了高速发展时期。目前，互联网上有名有姓的搜索引擎已达数百家，其检索的信息量也与从前不可同日而语。比如最

近风头正劲的 Google，其数据库中存放的网页已达 30 亿之巨！还有百度其存放的网页也有 6 亿多。

随着互联网规模的急剧膨胀，一家搜索引擎光靠自己单打独斗已无法适应目前的市场状况，因此现在搜索引擎之间开始出现了分工协作，并有了专业的搜索引擎技术和搜索数据库服务提供商。象国外的 Inktomi（已被 Yahoo 收购），它本身并不是直接面向用户的搜索引擎，但向包括 Overture（原 GoTo，已被 Yahoo 收购）、LookSmart、MSN、HotBot 等在内的其他搜索引擎提供全文网页搜索服务。国内的百度也属于这一类，搜狐和新浪用的就是它的技术。因此从这个意义上说，它们是搜索引擎的搜索引擎。

现在一提到搜索引擎，人们往往想到的是 Google、百度、雅虎、搜狐等。那么究竟什么是搜索引擎呢？“搜索引擎”实际上是为人们提供在 internet 网上利用关键词来进行全文检索的一种网页检索工具。

搜索引擎分类

搜索引擎按其工作方式主要可分为三种，分别是全文搜索引擎（Full Text Search Engine）、目录索引类搜索引擎（Search Index/Directory）和元搜索引擎（Meta Search Engine）。全文搜索引擎是最广泛也是用得最多的一种，一般所说的搜索引擎都指的是全文搜索引擎。

全文搜索引擎

全文搜索引擎是名副其实的搜索引擎，国外具代表性的有 Google、Fast/AllTheWeb、AltaVista、Inktomi、Teoma、WiseNut 等，国内著名的有百度（Baidu）、中国搜索等。它们都是通过从互联网上提取的各个网站的信息（以网页文字为主）而建立的数据库中，检索与用户查询条件匹配的相关记录，然后按一定的排列顺序将结果返回给用户，因此他们是真正的搜索引擎。

从搜索结果来源的角度，全文搜索引擎又可细分为两种，一种是拥有自己的检索程序（Indexer），俗称“蜘蛛”（Spider）程序或“机器人”（Robot）程序，并自建网页数据库，搜索结果直接从自身的数据库中调用，如上面提到的 7 家引擎；另一种则是租用其他引擎的数据库，并按自定的格式排列搜索结果，如 Lycos 引擎。

目录索引

目录索引虽然有搜索功能，但在严格意义上算不上是真正的搜索引擎，仅仅是按目录分类的网站链接列表而已。用户完全可以不用进行关键词（Keywords）查询，仅靠分类目录也可找到需要的信息。目录索引中最具代表性的莫过于大名鼎鼎的 Yahoo 雅虎。其他著名的还有 Open Directory Project（DMOZ）、LookSmart、About 等。国内的搜狐、新浪、网易搜索也都属于这一类。

元搜索引擎 (META Search Engine)

元搜索引擎在接受用户查询请求时，同时在其他多个引擎上进行搜索，并将结果返回给用户。著名的元搜索引擎有 InfoSpace、Dogpile、Vivisimo 等（元搜索引擎列表），中文元搜索引擎中具代表性的有搜星搜索引擎。在搜索结果排列方面，有的直接按来源引擎排列搜索结果，如 Dogpile，有的则按自定的规则将结果重新排列组合，如 Vivisimo。

除上述三大类引擎外，还有以下几种非主流形式：

1、集合式搜索引擎：如 HotBot 在 2002 年底推出的引擎。该引擎类似 META 搜索引擎，但区别在于不是同时调用多个引擎进行搜索，而是由用户从提供的 4 个引擎当中选择，因此叫它“集合式”搜索引擎更确切些。

2、门户搜索引擎：如 AOL Search、MSN Search 等虽然提供搜索服务，但自身即没有分类目录也没有网页数据库，其搜索结果完全来自其他引擎。

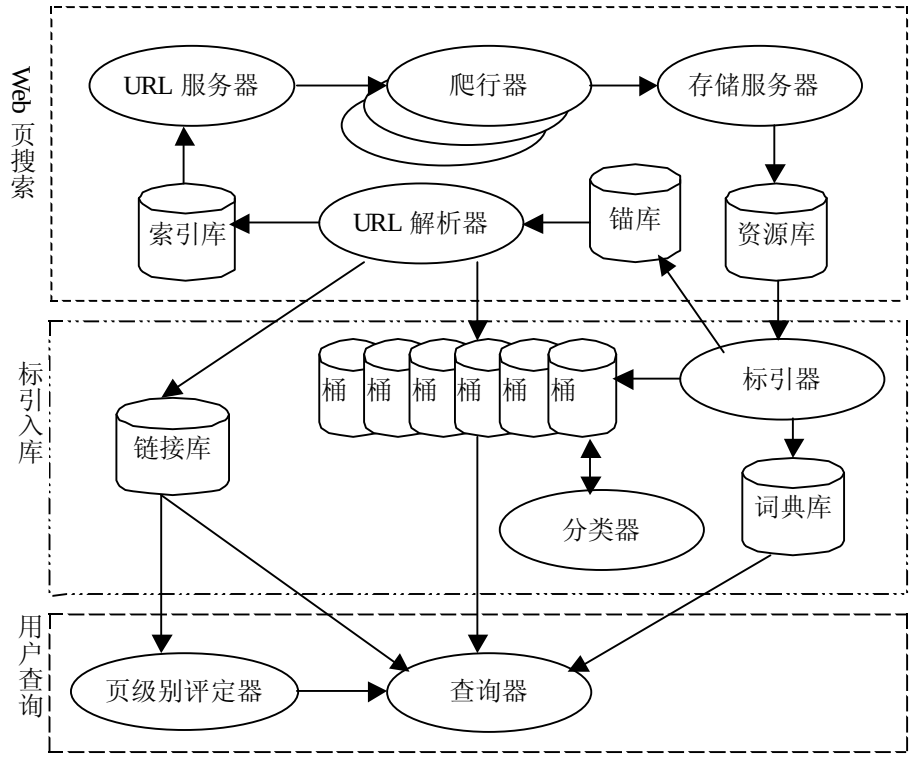
3、免费链接列表（Free For All Links，简称 FFA）：这类网站一般只简单地滚动排列链接条目，少部分有简单的分类目录，不过规模比起 Yahoo 等目录索引来要小得多。

由于上述网站都为用户提供搜索查询服务，为方便起见，我们通常将其统称为搜索引擎。

搜索引擎组成及工作原理

搜索引擎系统一般由蜘蛛（也叫网页爬行器）、切词器、索引器、查询器几部分组成。蜘蛛负责网页信息的抓取工作，一般情况下切词器和索引器一起使用，它们负责将抓取的网页内容进行切词处理并自动进行标引，建立索引数据库。查询器根据用户查询条件检索索引数据库并对检索结果进行排序和集合运算，如并集、交集运算，再提取网页简单摘要信息反馈给查询用户。

Google 搜索引擎从功能上同样分为三大部分：网页爬行、标引入库和用户查询。网页爬行主要负责网页的抓取，由 URL 服务器、爬行器、存储器、分析器和 URL 解析器组成，爬行器是该部分的核心；标引入库主要负责对网页内容进行分析，对文档进行标引并存储到数据库里，由标引器和分类器组成，该模块涉及许多文件和数据，有关于桶的操作是该部分的核心；用户查询主要负责分析用户输入的检索表达式，匹配相关文档，把检索结果返回给用户，由查询器和网页级别评定器组成，其中网页等级的计算是该部分的核心。其总体系统结构下图所示。



搜索引擎的主要工作流程是：首先从蜘蛛开始，蜘蛛程序每隔一定的时间（象 google 一般是 28 天）自动启动并读取网页 URL 服务器上的 URL 列表，按深度优先或广度优先算法，抓取各 URL 所指定的网站，将抓取的网页分配一个唯一文档 ID(DocId)，存入文档数据库。一般在存入文档数据库之前进行一定的压缩处理。并将当前页上的所的超连接存入到 URL 服务器中。在进行抓取的同时，切词器和索引器将已经抓取的网页文档进行切词处理，并按词在网页中出现的位置和频率计算权值，然后将切词结果存入索引数据库。整个抓取工作和索引工作完成后更新整个索引数据库和文档数据库，这样用户就可以查询最新的网页信息。查询器首先对用户输入的信息进行切词处理，并检索出所有包含检索词的记录，通过计算网页权重和级别对查询记录进行排序并进行集合运算，最后从文档数据库中提取各网页的摘要信息反馈给查询用户。

二、网络蜘蛛

概述

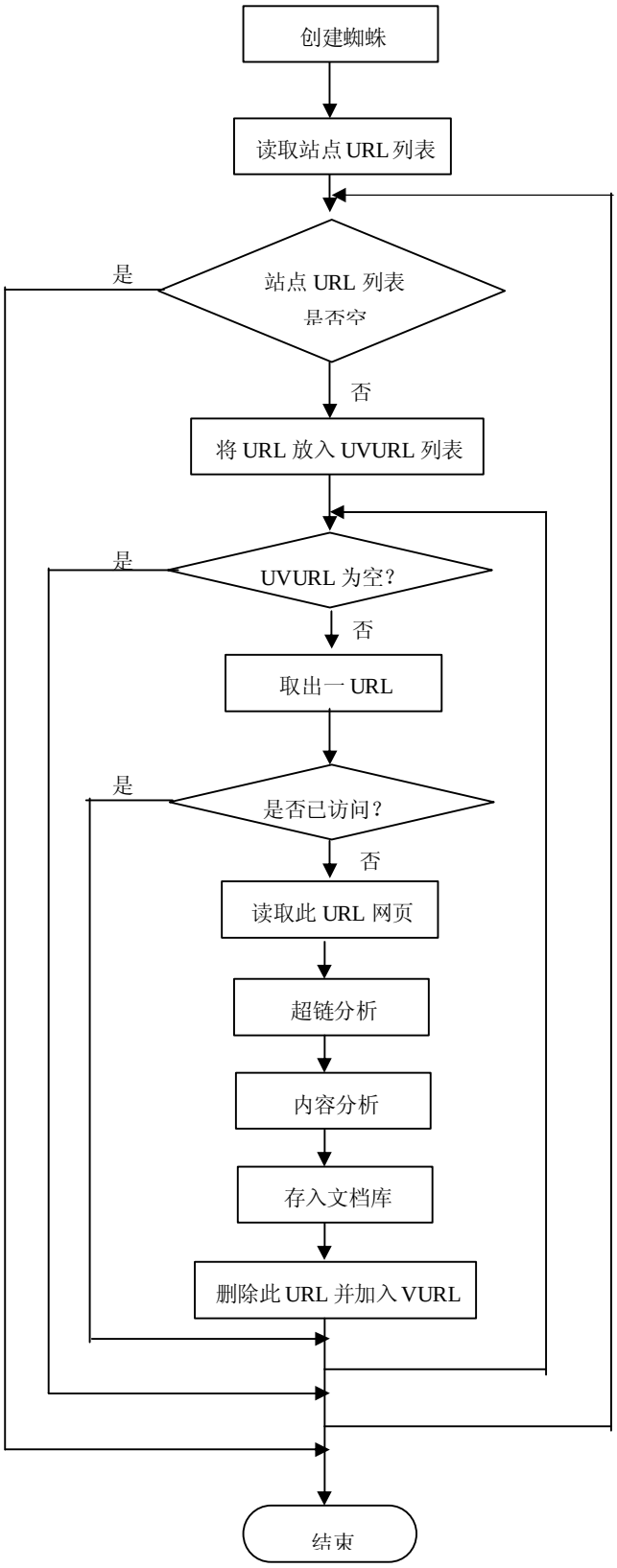
蜘蛛(即 Web Spider)，实际上是一个基于 HTTP 协议的网络应用程序。网络蜘蛛是通过网页的链接地址来寻找网页，从网站某一个页面（通常是首页）开始，读取网页的内容，并抽取出网页中的其它超链接地址，然后通过这些链接地址寻找下一个网页，这样一直循环下去，直到把这个网站所有的网页都抓取完为止。

在抓取网页的时候，网络蜘蛛一般有两种策略：广度优先和深度优先。广度优先是指网络蜘蛛会先抓取起始网页中链接的所有网页，然后再选择其中的一个链接网页，继续抓取在此网页中链接的所有网页。这是最常用的方式，因为这个方法可以让网络蜘蛛并行处理，提高其抓取速度。深度优先是指网络蜘蛛会从起始页开始，一个链接一个链接跟踪下去，处理完这条线路之后再转入下一个起始页，继续跟踪链接。这个方法有个优点是网络蜘蛛在设计的时候比较容易。

主要组成

根据抓取过程蜘蛛主要分为三个功能模块，一个是网页读取模块主要是用来读取远程 Web 服务器上的网页内容，另一个是超链分析模块，这个模块主要是分析网页中的超链接，将网页上的所有超链接提取出来，放入到待抓取 URL 列表中，再一个模块就是内容分析模块，这个模块主要是对网页内容进行分析，将网页中所有超标志去掉只留下网页文字内容。蜘蛛的主要工作流程如下图所示：

首先蜘蛛读取抓取站点的 URL 列表，取出一个站点 URL，将其放入未访问的 URL 列表（UVURL 列表）中，如果 UVURL 不为空刚从中取出一个 URL 判断是否已经访问过，若没有访问过则读取此网页，并进行超链分析及内容分析，并将些页存入文档数据库，并将些 URL 放入已访问 URL 列表（VURL 列表），直到 UVRL 为空为止，此时再抓取其他站点，依次循环直到所有的站点 URL 列表都抓取完为止。



说明

UVURL：为当前站点未访问的 URL

VURL：为当前站点已访问的 URL

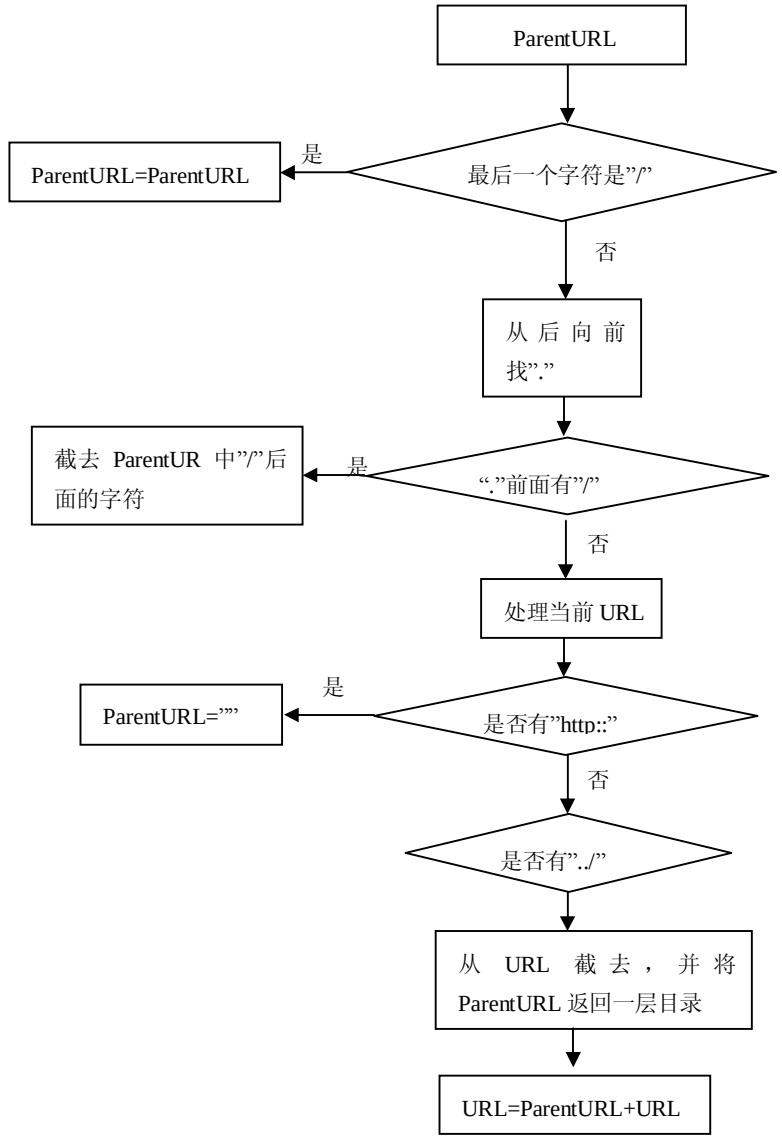
关键技术

1、 多线程技术：由于抓取的站点 URL 相当多，采用单线程蜘蛛抓取时速度不够，也不能满足实际的需要。因而需要多线程技术来创建多个蜘蛛线程来同时抓取，以提高速度。

2、 网页抓取：网页抓取是基于 HTTP 协议之上的，网页上的资源有多种，有网页，有 Word 文档也有其他类型的文件，这样抓取时需要判断 URL 所指向资源的类型。

3、 超链分析：超链分析是一个比较重要的环节，需要对 HTML 的各种标志（tag）有一个很全面的了解。需要反复测试，考虑各种情形的发生。

超链分析时从网页里提取出来的是相对于当前页的相对 URL，因而需要根据当前页的绝对 URL 将提取的这个 URL 转换成绝对 URL。在此过程中需要根据 ParentURL（就是当前页的 URL）作出各种判断。各种情况判断如下图所示：



经验总结

商业化的蜘蛛需要抓取上亿的网页，因而抓取速度是一个关键，另外蜘蛛需要自动运行，尽是减少人工的参与，因而系统的性能也是一个很重要的关键，系统能够在发生异常的时候自动进行处理，防止程序的退出和死机。本人认为有一些细节需要注意：

1、 系统应该使用多线程，使用多个蜘蛛同时抓取，在可能的情况下，最好是做成分布式的蜘蛛程序，蜘蛛应该分布地网络上多台服务器上协同抓取网页，这样速度会更快，更符合我们的实际应用。

2、 对于同一网站的网页应该采用同一个 HttpConnection 这样有效地节省创建一个连接的时间，另外对于抓取的 URL 采用域名缓冲机制（可在网关一级上实现），这样抓取时减少由域名到 IP 地址的转换时间以及重复的域名转换。若能做到这一步将会大大减少抓取时间，因为访问一 URL 时每次都要进行域名到主机 IP 地址的转换。

3、 最好是能够将读取网页、超链分析及网页内容分析三部分分开来做，让它们并行协同工作，这样效率会更高。因为在这三个过程中网页读取比起其他两个功能来说是一个长任务，最耗时间。当抓取完一网页后，在抓取下一网页的时候让去执行超链分析和内容分析。这样在下一网页抓取完成之前超链分析和内容分析任务就能完成，抓取任务不会延迟，这样节省了一些时间。

三、切词器

概述

1、 概述

众所周知，英文是以词为单位的，词和词之间是靠空格隔开，而中文是以字为单位，句子中所有的字连起来才能描述一个意思。例如，英文句子 I am a student，用中文则为：“我是一个学生”。计算机可以很简单通过空格知道 student 是一个单词，但是不能很容易明白“学”、“生”两个字合起来才表示一个词。把中文的汉字序列切分成有意义的词，就是中文分词，有些人也称为切词。我是一个学生，分词的结果是：我 是 一个 学生。

2、切词算法

现有的分词算法可分为三大类：基于字符串匹配的分词方法、基于理解的分词方法和基于统计的分词方法。

1)、基于字符串匹配的分词方法

这种方法又叫做机械分词方法，它是按照一定的策略将待分析的汉字串与一个“充分大的”机器词典中的词条进行匹配，若在词典中找到某个字符串，则匹配成功（识别出一个词）。按照扫描方向的不同，串匹配分词方法可以分为正向匹配和逆向匹配；按照不同长度优先匹配的情况，可以分为最大（最长）匹配和最小（最短）匹配；按照是否与词性标注过程相结合，又可以分为单纯分词方法和分词与标注相结合的一体化方法。常用的几种机械分词方法如下：

a）正向最大匹配法（由左到右的方向）；

b）逆向最大匹配法（由右到左的方向）；

c）最少切分（使每一句中切出的词数最小）。

还可以将上述各种方法相互组合，例如，可以将正向最大匹配方法和逆向最大匹配方法结合起来构成双向匹配法。由于汉语单字成词的特点，正向最小匹配和逆向最小匹配一般很

少使用。一般说来，逆向匹配的切分精度略高于正向匹配，遇到的歧义现象也较少。统计结果表明，单纯使用正向最大匹配的错误率为 1/169，单纯使用逆向最大匹配的错误率为 1/245。但这种精度还远远不能满足实际的需要。实际使用的分词系统，都是把机械分词作为一种初分手段，还需通过利用各种其它的语言信息来进一步提高切分的准确率。

一种方法是改进扫描方式，称为特征扫描或标志切分，优先在待分析字符串中识别和切分出一些带有明显特征的词，以这些词作为断点，可将原字符串分为较小的串再来进行机械分词，从而减少匹配的错误率。另一种方法是将分词和词类标注结合起来，利用丰富的词类信息对分词决策提供帮助，并且在标注过程中又反过来对分词结果进行检验、调整，从而极大地提高切分的准确率。

2)、基于理解的分词方法

这种分词方法是通过让计算机模拟人对句子的理解，达到识别词的效果，但这种方法需要大量的词法、句法、语义知识。其基本思想就是在分词的同时进行句法、语义分析，利用句法信息和语义信息来处理歧义现象。它通常包括三个部分：分词子系统、句法语义子系统、总控部分。在总控部分的协调下，分词子系统可以获得有关词、句子等的句法和语义信息来对分词歧义进行判断，即它模拟了人对句子的理解过程。这种分词方法需要使用大量的语言知识和信息。由于汉语语言知识的笼统、复杂性，难以将各种语言信息组织成机器可直接读取的形式，因此目前基于理解的分词系统还处在试验阶段。

3)、基于统计的分词方法

从形式上看，词是稳定的字的组合，因此在上下文中，相邻的字同时出现的次数越多，就越有可能构成一个词。因此字与字相邻共现的频率或概率能够较好的反映成词的可信度。可以对语料中相邻共现的各个字的组合的频度进行统计，计算它们的互现信息。定义两个字的互现信息，计算两个汉字 X、Y 的相邻共现概率。互现信息体现了汉字之间结合关系的紧密程度。当紧密程度高于某一个阈值时，便可认为此字组可能构成了一个词。这种方法只需对语料中的字组频度进行统计，不需要切分词典，因而又叫做无词典分词法或统计取词方法。但这种方法也有一定的局限性，会经常抽出一些共现频度高、但并不是词的常用字组，例如“这一”、“之一”、“有的”、“我的”、“许多的”等，并且对常用词的识别精度差，时空开销大。实际应用的统计分词系统都要使用一部基本的分词词典（常用词词典）进行串匹配分词，同时使用统计方法识别一些新的词，即将串频统计和串匹配结合起来，既发挥匹配分词切分速度快、效率高的特点，又利用了无词典分词结合上下文识别生词、自动消除歧义的优点。

到底哪种分词算法的准确度更高，目前并无定论。对于任何一个成熟的分词系统来说，不可能单独依靠某一种算法来实现，都需要综合不同的算法。笔者了解，海量科技的分词算法就采用“复方分词法”，所谓复方，相当于用中药中的复方概念，即用不同的药才综合起来去医治疾病，同样，对于中文词的识别，需要多种算法来处理不同的问题。

3、关键问题

1）通用词表和切分规范

汉语的语素和单字词，合成词和短语之间没有清晰的界限。语言学界虽然对于词在概念上有一个十分清晰的定义，即，“词是最小的能够独立活动的有意义的语言成分。”但从一些词典的编撰中，我们仍然可看出一些上述界限难以区分的问题。比如：“听见”“看见”在很多词典中都有收录，但是有类似结构的“闻见”却没有收录。在建立分词系统词表时，仍然对于收词的标准难以把握，例如：“鸡蛋”是词，那么“鸭蛋、鹌鹑蛋”是否也作为词收入词表？至今为止，分词系统仍然没有一个统一的具有权威性的分词词表作为分词依据。这不能不说是分词系统所面临的首要问题。除了分词词表，还有一个概念值得我们注意，即“分词单位”。从计算机进行分词的过程来看，其输出的词串我们称之为“切分单位”或“分词单位”。《信息处理用现代汉语分词规范》中对于“分词单位”也有一个定义：“汉语信息处

理使用的、具有确定的语义或语法功能的基本单位。包括本规范的规则限定的词和词组。”由此可见，信息处理中分词单位的定义比传统意义上的词更宽泛些。这也就避开了理论上对于词的界定难以把握的困扰。分词系统可以面向解决实际问题的需求和真实语料中使用的频繁程度来规定“分词单位”。分词单位可以是同词表中词完全一致，也可以是包含未登录词识别以及一些词法分析的切分单位， 例如，一些人名、地名、机构名、外国人译名，应予以识别和切分。一些动词和形容词重叠结构，如“高高大大”、“甜甜蜜蜜”等；一些附加词，如后缀，“亲和性”、“热敏性”等；都可以作为分词单位予以识别和切分。因此，对于一个分词系统而言，制定一个一致性的分词单位切分规范无疑也是一个重要的问题。

2）歧义切分字段

分词系统要处理的第二个关键问题是文本中歧义切分字段的判别。汉语中歧义切分字段最基本有以下两种类型：

交集型歧义字段，据统计，这种歧义字段占全部歧义字段的８５％以上。[4] 所以这也是分词系统所要重点解决的问题。在字段 ABC 中，这里，A,B,C 分别代表有一个或多个汉字组成的字串。A,AB,BC,C 分别都是词表中的词，则称该字段为交集型歧义字段。如：“中国/人”，“中/国人”两种切分结果。

组合型歧义在字段 ABC 中， A,B,AB 分别都是词表中的词，则称该字段为交集型歧义字段。如：他/具有/非凡/的/才能/。/ 只有/他/才/能/举起/这/个/重物/。/

3）未登录词识别

我们知道，词表中不能囊括所有的词。一方面是因为语言在不断的发展和变化，新词会不断的出现。另一方面是因为词的衍生现象非常普遍，没有必要把所有的衍生词都收入辞典中。

特别是人名、地名等专有名词，在文本中有非常高的使用频度和比例。而且由于未录词引入的分词错误往往比单纯的词表切分歧义还要严重。这就要求分词系统具有一定的未登录词识别能力，从而提高分词的正确性。 除了人名、地名的识别，我们认为，分词系统还需要有一定的词法分析能力，从而解决衍生词和复合词等词汇平面上的问题，为进一步的中文信息处理提供坚实的基础。

切分原理

1、 词库组织

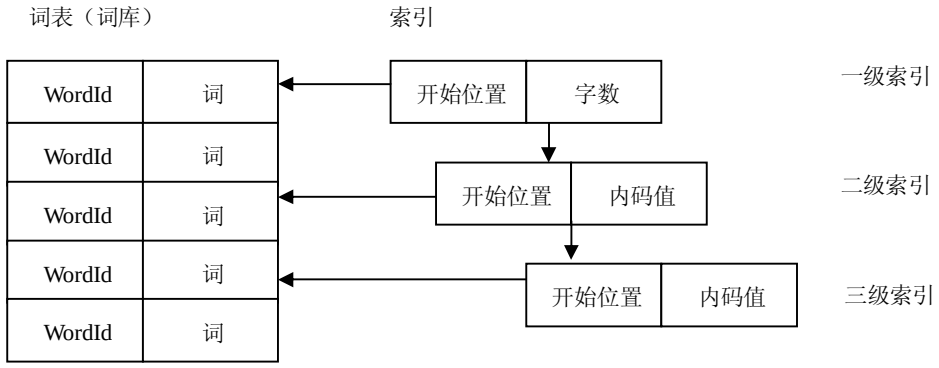
本人采用的是基于词表匹配的分词方法，因而词库是分词系统的基础。整个分词过程实际上就是在词词上的查找匹配过程，因而词库的组织相当重要。

对于词表存放在一个文本文件里，每一个词条由两项组成，一个是词的 ID（WordId）、另一个就是词本身。同时在词表上加上静态索引，本人对词表进行分组管理并在上面添加三级索引。首先对词条按字数分组，字数相同的词条放在同一组里，并对词表按首汉字的内码从小到大排序。一级索引是加在各个分组上，一级索引记录了各分组的开始位置，再根据下一分组的起始位置可以确定当前分组的终止位置。二级索引是加在一级索引内部的，在同一组内部由于有很多的词条，二级索引是按词的首汉字内码建立的，它加在以不同汉字开头的词条组中，这样通过三级索引可以进一步缩小查找范围。另外在汉字中以有些字开头的词条过多，这样进行匹配的次数过多，不利于提高匹配速度。因而在二级索引的基础之上添加一个三级索引，它是按照一定的密度间隔添加，我设定了一个默认的合理的值就是每隔 50 个词条添加一个三级索引，同样三级索引也是根据汉字内码添加的（三级索引和二级索引的定义相同）。词条及索引的定义如下：

//根据汉字内码建立的索引结点（二级和三级索引）

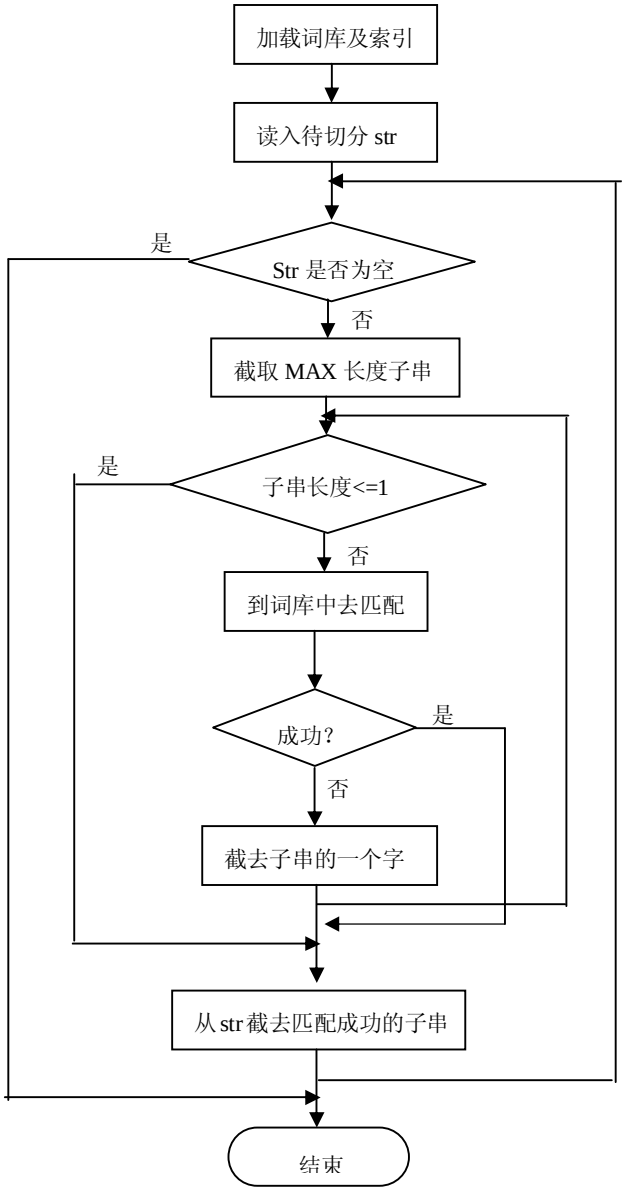
```
typedef struct CodeIndexNode{
 char KeyValue[17];
 int nIndex;
}CodeIndex;

//根据词语字数建立的一级索引结点
typedef struct WordsIndexNode{
 int nKeyCount;
 int nKeyBeginIndex;
 int nKeyEndIndex;
 int CodeIndexCount;
 CArray<CodeIndexNode*,CodeIndexNode*>HexIndex;
}WordsIndex;

//关键字结点
typedef struct KeyWordNode{
 CString strKeyWord; //关键字
 int          nID;              //关键字 ID
};
```

词表及一级、二级、三级索引之间的关系如下图所示：



2、 切分方法

由于采用的是基于词库匹配的正向最大匹配算法（通常简称为 MM 法），其基本思想为：设 D 为词典，MAX 表示 D 中的最大词长，str 为待切分的字串。MM 法是每次从 str 中取长度为 MAX 的子串与 D 中的词进行匹配。若成功，则该子串为词，指针后移 MAX 个汉字后继续匹配，否则子串逐次减一进行匹配。所以主要切词过程如下：(流程图如下图所示)



1)、读取词库，并读取相应的静态索引，建立词库上的索引；

2)、读取待切分的字串 str；

3)、从待切分字串 str 中取出一个长度为 MAX 的子串 sstr，到词典中去匹配，若匹配成功则取下一个长度为 MAX 的子串进行匹配，否则将子串 sstr 从后面截去一个字后继续匹配，直到匹配成功或者子串 sstr 中只有一个字为止。若匹配成功则从匹配成功的词的位置开始再截取下一长度为 MAX 的子串 sstr 进行匹配，依次循环直到将 str 串匹配完为止。

4）匹配过程：首先根据子串 sstr 的长度（这里指的是字数）确定一级索引也就是确定分组，这个过程可以二分查找法，也可以采用 Hash 函数直接定位，但是由于分组数很少（不会超过 20）因而两种方法没有多大的区别。在确定了分组后再根据首汉字的内码确定二级索引，因为二级索引是按内码从小到大的顺序因而可采用拆半查找方法，找到以后再确定三级索引，这样将进行匹配的过程缩小到一个很小的范围，在这个范围内匹配不成功则进行下一个串的匹配。通过确定三级索引确定了进行匹配的最小词条集。

3、 切分结果的保存（也就是顺排档数据的保存）

由于数据量很大，不能全存放在内存中，所以每处理完一文档就将其切分结果存放到外部文件中，这里可以借助其它关系型数据库，这有利于于索引器导出数据将其导成倒排档索引文件。主要用到的结构体定义如下：

```
//Hit 结点
typedef struct HitNode{
 int          nPos;//位置
 HitNode* pNext;//下一 hit 指针
};
//文档列表结点
typedef struct DocListNode{
 _int64       nDocID;//DOC ID
 int          nHits;//词出现的次数
 float        fWeight;//词在文中的要重
 HitNode*    pHitHead;//Hit 链表头指针
 DocListNode* pNext;
};
//一级索引结点
typedef struct LexIconNode{
 int                nKeyWordID;//关键字 ID
 int                nDocs;//文档数
 DocListNode* pDocListHead;//文档链表头指针
 LexIconNode* pNext;
};
```

在数据库中存放的字段主要有：DocID、WordID、Hit（位置）、Weight（权值）。这样索引器导出时将会按 WordID 和权值进行排序。

经验总结

1、 存在的问题

1）、在词库组织方面采用静态的索引不利于于词库的变化，若有一新词出现，则需要重建整个词库的索引，因为下标都发生了变量。这不利于词库的扩充。

2）、词的 ID 分配机制也有些不足，这里的 ID 都是跟词的内码相关，也就是跟词在词库中的排序顺序有关，这样若有新词添加进来后，就会改变其后面所有词的 ID。

3）、切词的速度不够快，虽然每秒能达到 400 多字，但是词库比较小，只有 5 万多的词条。若司库很大时速度会有所下降。

4）、因为汉字是双字节表示的，所以在切分之前转换成 Unicode，转换成多字节表示，经过测试发现，多字节转换占用了很大一块 CPU 时间，将近占去了 40%的时间。

5）、在进行多字节转换时开设的缓冲区为 1000 个汉字，若需要转换的汉字多于 1000 则会出错，但若开设的缓冲区过大是对系统资源的浪费。

6）、是一种机械的切词方法，没有对歧义词进行排除和分析。

7）、没有新词的识别功能，也就是不通过词典不能识别切分出新的词。

2、 改进的方向

1）、词表组织

在词表上添加三级索引是一个较好的方法，但是采用静态的索引不利于词库的扩充，因而词库的索引可动态生成，在读取词库的同时构建索引，但是这种方法对于查询器会产生一个不利影响，因为查询器也需要读取词库而构建索引的过程比较费时间，这样用户查询时会有一定的延时，所以应根据需要采用一种折衷的办法。

整个切词过程实际就是在词表上的查找过程，而词表相当于就是一个查找表，所以这个查找表的组织相当关键，它决定切分的速度。所以在这个查找表上必须添加索引来加快速度，本人觉得可以根据各词的前两个汉字的内码建立一个 Hash 函数索引，这样即不需要分组也不需要用二分法来查找，直接定位肯定会减少进行匹配的次数。但需要解决冲突问题，因为有的词属于其他词的前向子串。

2）、词条 ID 分配

词条 ID(WordId)可按入库的先后顺序递增，不管其内码。但是入库后应该按其内码插入到适当的位置。但是在建立倒排档索引数据时应该按 WordID 从小到大排序，这样对于查询器可以根据 WordID 用 Hash 函数直接定位到相应的读取位置。

3）、多字节转换问题

多字节转换需要将所有的待切分串都转换成多字节表示，这个过程相当费时，若不需要转换直接切分则会大大提高速度，对于纯汉字的字串可以这样做，但是有中英文混合的字符串就不太适合，因为英文字符只占一个字节表示，这样会出现将一个当字从中间切开。这样字符串移位时先判断其是否是 Ansi 码，若是则移一个字节，若不是则移两个字节。

4）、歧义识别

5）、新词识别

新词的识别，可以按词频统计的方法，若某一字串出现的次数高于某一频率时可以认为一个词，将其切分出来。

四、索引器

概述

索引器是搜索引擎系统心须也是很关键的一个环节，它主要完成将切词形成的顺排档文档组织成倒排档索引数据。(索引的合并用拉链)

实现原理

1、 索引文件结构

倒排档索引文件分三个文件保存，一个是存放各词条索引文件，另一个是各文档索引文件，再一个就是各词在文档中出现的位置信息文件。

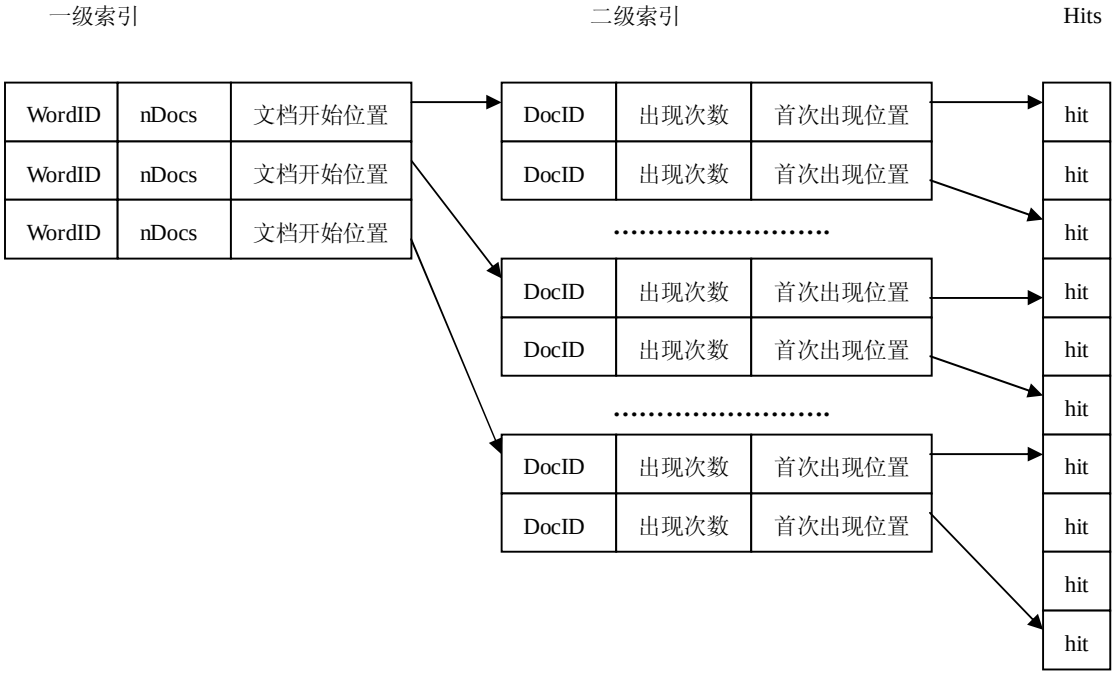
1)、顺排档结构

顺排档文档是以 DocID 为主序的，每一文档下存放各自出现的词的 ID 及各词所出现的次数和具体位置信息，各数据项的存储长度固定。

Hits(位置)占 16 位

DocID	WordID	出现次数	hit	…….	hit
	……..	…….	…	…….	…
	WordID	出现次数	hit	…….	hit
DocID	WordID	出现次数	hit	…….	hit
	……..	…….	…	…….	…
	WordID	出现次数	hit	…….	hit

2）、倒排档结构



2、 索引数据组织

1）、一级索引：一级索引文件属于记录式文件，每一记录大小固定，共有三个数据项构成，WordID、文档数、第一个文档开始位置。其中 WordID 是词典中词条的 ID，文档数是指这个词总共在多少个文档中出现，文档开始位置是一个文件指针指向二级索引中出现当前词的文档集中的第一个文档存储位置，这个指针是一个长整形值相当于指明了是二级索引文件中的第几条记录，因为各记录长度也是固定大小。通过这个指向可以直接定位到二级索引文件读取位置，然后读取 nDocs 个记录即可，因为它们是存放在连续的地址空间上。

2）、二级索引：二级索引也是一种记录式文件，每一记录有三个数据项组成，DocID、出现次数、第一个 Hit 位置。其中 DocID 是文档的 ID，出现次数指的是当前文档中某一个词出现的次数，第一个 Hit 位置也是一个指针，指向 Hits 文件中的某一位置。通过这

个指针就可以直接定位到 Hits 位置中的读取位置，这样连续读取 nHits 个记录就可以将所有当前词在当前文档中的出现的位置信息都读入。些文件将属于同一 WordID 下的所有文档记录按其词在整个文档的权值从大到小排列。

3)、Hits 位置信息文件：些文件每一记录只有一个数据项，即 Hit 位置信息，只记录了各词在文档中出现的位置。将同一词在同一文档中的出现位置按出现的先后排列。这样在读取文档并提取摘要时只需对字符串从头到尾扫描一边即可，不需要来回扫描。

3、 索引文件导出过程

1)、以文档为单位处理先将切分结果处理为顺排档并存入到外部数据库。在此过程中计算各词的权值，主要考虑了出现的次数和出现的位置，若出现在网页的链接文字和 title 上则其权值比普通位置高一个数量级将其设为 0.1，若在其它位置上出现，则每出现一次将其权值加 0.01。

2)、将顺排档文件按多种关键字排序，首先按 WordID 从小到大排序，再按词的权值从大到小排序，最后按各词的出现的先后顺序排序。这样基本形成了倒排档文件结构，再分组统计各词出现的文档数及各文档中同一词出现的次数，最后写到索引文件里即可。（注：这里的权值是同一词在同一文档中所有出现位置的权值之和）

经验总结

1、存在的问题

1)、需要借助其它数据库系统来存放顺排档数据，若是用文件系统，则按多关键字进行排序时需要反复进行文件操作，而且需要文件的归并。这样效率不高，且容易出错。

2)、每处理完一文档就需要导出顺排数据。

3)、这里考虑的权值只是对 title 和链接文字中出现的词进行了考虑，对于其他一些特殊位置没有考虑，比如加粗文字、内容标题等没有考虑。

4)、数据的更新比较麻烦。若是全新更新则简单，只需要用最新的数据替换旧数据即可，但是若是增量更新则需要将前后两次的索引文件进行合并，在全并过程中需要考虑排序问题，且三个文件都要考虑，而且它们是相互关联的，需要修改插入点以后所有的数据。若进行数据删除同样存在这样的问题，也需要修改插入点以后所有的数据。因而不利于数据的维护。

2、 改进的地方

1)、若要使用户的查询结果更准确应该在切词和建索引时考虑其它网页标志上的权值。

2)、若能将倒排档索引文件组织成数据库系统，由数据库系统来管理会大提高系统的效率并在数据维护方面有一定的优越性。

3)、对于索引的合并用拉链的方法有利于合并的速度

4)、对倒排文档文件采用数据库的管理方法

五、查询器

概述

查询器是搜索引擎系统中最后一个环节，是最终和用户打交道的用户搜索界面。查询器是通过 Web 页接受用户输入的搜索参数并切分用户输入的字串，访问倒排档索引文件检索出所有符合检索条件的文档，并对其进行并集运算和排序运算，最后得到最终的结果文档，再从各文档中提取摘要信息写入用户反馈网页中。由于在检索过程中需要读取索引文件并进行一系列的运算，因而查询器很难用 ASP、PHP、JSP 等一些服务器脚本来实现，必须通过 CGI 程序来完成。采用 ISAPI 来实现是一种很好的选择，它是运行在 Windows 平台上并配合 IIS 服务器，是以 DLL 的形式发布，用户的查询只需要提交给此 DLL 处理，处理完后会自动以 HTML 的形式反馈给用户。

实现原理

1、 所用的数据结构定义

```
//根据汉字内码建立的索引结点
typedef struct CodeIndexNode{
    char KeyValue[17];
    int nIndex;
}CodeIndex;
```

以上结构体是定义的内码索引结点，主要用于在词表上的二级索引和三级索引。是用在词库上的结构体。

```
//根据词语字数建立的一级索引结点
typedef struct WordsIndexNode{
    int nKeyCount;
    int nKeyBeginIndex;
    int nKeyEndIndex;
    int CodeIndexCount;
    CArray<CodeIndexNode*,CodeIndexNode*>HexIndex;
}WordsIndex;
```

这一结构体是用在词库上的一级索引结点结构体。

```
//关键字结点
typedef struct KeyWordNode{
    CString strKeyWord; //关键字
    int         nID;            //关键字 ID
};
```

这一结构体是词表的组织结点。

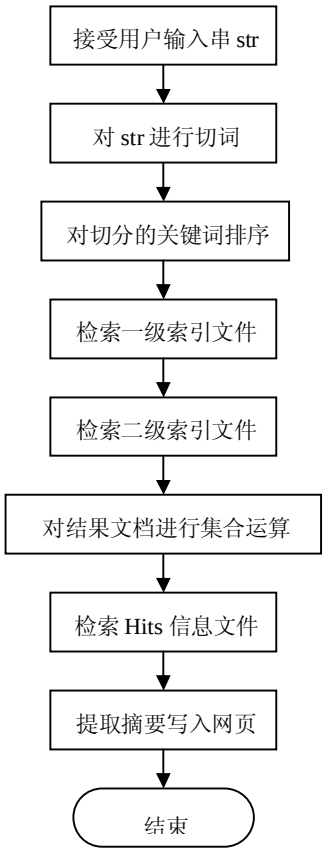
```
/*********************************************/
////建立索引的数据结构体////
//Hit 结点
typedef struct HitNode{
    int         nPos;//位置
};
//文档列表结点
typedef struct DocListNode{
    _int64      nDocID;//DOC ID
    int         nHits;//词出现的次数
```

```
    float       fWeight;//词在文中的要重
    long    HitHead;//Hit 链表头
    int         nLen;
//  DocListNode* pNext;
};
//倒排文件一级索引结点
typedef struct LexIconNode{
    int             nKeyWordID;//关键字 ID
    int             nDocs;//文档数
    long        DocListHead;//文档链表头
    int             nWLen;//关键字的长度
};
//在查找字符串中存在的关键字列表
typedef struct ResultKeyNode{
    int         nLen;           //关键字的长度（是宽字符型的）
    char KeyWord[20];//关键字
    int         nDocs;          //此关键字出现的文档数
};
/*******************************************/

/*******************************************/
//结果集合运算时的数据结构结点//
typedef struct PosNode{
    int nLen;//长度
    int nBegin;//开始位置
    PosNode* pNext;//下一指针
};
typedef struct FinalDocNode{
    _int64 nDocID;
    PosNode* pPosHead;
    FinalDocNode* pNext;//下一文档的位置
};
/*******************************************/
```

2、 检索过程

查询器的检索过各主要如下图所示：

查询器接受到用户输入的查询串后，先对些串进行切词，切出一个个的词，然后对各词按其词的 WordID 从小到大排序，这样是为了在读取一级索引文件时只需要扫描一次文件，读写磁头不需要来回移动。然后读取一级索引文件和二级索引，检索出各词所出现的所有文档 ID 及各文档所对应的 Hits 个数及开始位置。读取之后对所有切分出的关键词所检索出的文档结果进行并集和交集运算，得到最终的结果集，再根据各文档的 Hits 指针信息读取得到各个 Hit，最后根据文档 ID 读取文档内容，将根据 Hits 信息提取简单摘要信息写入网页反馈给用户。



经验总结

1、 存在的问题

1)、词库的加载和切词器一样，所以还需要解决动态构建词库索引，但是这样会浪费一定的 CPU 时间，对用户的查询不利。

2)、每个用户查询一次就需要加载一次词库这对多用户查询是不利的。

3)、查询结果的分页显示问题，分布显示基本可以，但是若查询出的结果过多，会导致内存不足及响应慢问题，因而需要解决缓冲区和分页调入策略。

4)、摘要的提取不够正确，有时将一个词从中间截断。

5)、进行并、交集运算的算法不够好，有待改进。另外进行并交运算是对所有的结果文档都进行了并交运算，若查得的结果文档过多时会导致内存不足和响应时间过长问题。并没有考虑将关键字出现比较全的文档的提前。

2、 改进

1)、若能解决词表的 Hash 函数索引，则能大加快访问速度。

2)、若多用户共享一个词表则也会节省一些时间。

3)、检索文档时若满足条件的文档过多，可分块读入，只有用户移动页数到达所需要调入的文档时才去读取文档信息，否则不进行读取。

4)、应该对查询结果按其关键字出现的个数进行排序，这样多个关键字同时出现的页面将会先显示。

六、系统关键分析

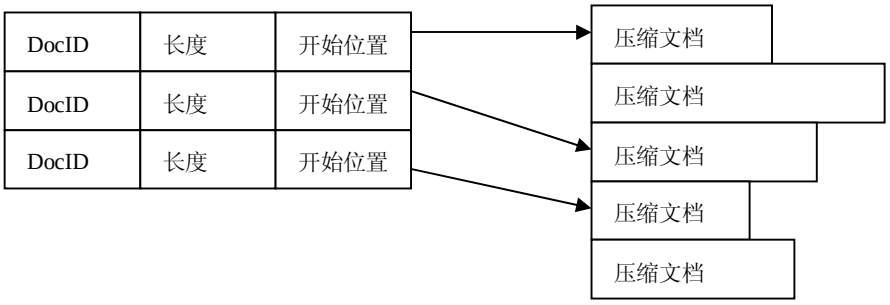
搜索引擎是一个复杂的综合系统，各个子系统之间是相互协调，紧密相关。所以在设计时需要全面考虑，任何一个环节的效率都会影响到整个系统的效率。本人认为主要有如下几个关键的方面：

1、 蜘蛛的抓取速度

搜索引擎系统需要抓取上亿的网页，因而速度很关键，它影响到整个数据的更新周期。在此过程中应该设计分布式的蜘蛛程序，并将抓取工作、超链分析及内容分析几个串行工作任务设计成并行任务。

2、 文档的压缩

目前本人采用了文本文件来保存各文档，一个文档一个文本文件。这种方法在数据量很大的情况下会有一些不足，因为在得到文档的文件名后，在文件下查找文件的过程就会耗去一定的时间，并且不利于管理。因而可以采用一种比较简单且解压过程比较快速的压缩算法，将所有的文档压缩到一个文档数据文件里，并在这个压缩文件上建立一个索引文件，这个索引文件的各个记录大小固定（其结构可见下图）。其有三个数据项组成，即 DocID、开始位置、长度。而索引文件是按文档 ID 从小到大排序，这样给定一个 DocID 后可以用 Hash 函数直接定位到索引文件中具体文档的信息，得到其开始位置和长度后就可以从压缩文档中读取相应的文档。然后再解压后进行摘要的摘取。这样通过这个索引文件可以方便地实现对文档的维护和管理。



3、 切词的速度

切词器的切词速度也关键，当蜘蛛抓取完数据后，就由分词器进行切词。速度不够快同样影响数据的更新周期，可将切词器与蜘蛛一起并行运行，蜘蛛每抓取完一个页后将其放入切词缓冲区中，切词器循环检查缓冲区，若有数据则邓出一个进行切词处理直到蜘蛛抓取完且缓冲中没有数据为止。这个缓冲中直放入待切分的 DocID，这样切词器得 DocID 后去访问文档索引，并读取文档进行解压后再进行切词。

4、 网页的权值运算问题

切词器在切词过程中就应该考虑权值问题，应该定义一套权值规范，定义各种超标志中间出现的词应该采用什么样的权值函数。这样切分结果本身就有了权值，索引器就可以根据权值建立更加合理的倒排索引文件。这直接影响到用户的查询结果。

5、 查询器的分页机制

一个实用的搜索引擎有上亿的文档，因而同时出现一个词的文档数也会多达上万，这样查询器在查询时不可能一次性将所有的文档都处理一次，只能采用分页调入或其他策略

来解决这一问题。因为有时的查询结果有上千万条结果，这样不可能对所的结果文档都进行并、交运算。

6、 系统之间的自动协调

搜索引擎系统的各个部分是相互协调来工作的，因而若有实现自动工作，需要有一个调度控制程序来调度。另外各个部分也需要提供一个供调度器调用的接口，这样调度器就可以调用接口控制其工作。但是蜘蛛程序、切词器等属于长任务作业，我们并不能保证在其运行周期内不会出错，所以各系统还需要有一个故障检测、任务重启动功能，这样才能保证各系统在无人监管的情况下自动运行。

七、参考文献

[1] Google 搜索引擎技术实现探究　化柏林　中国科学技术信息研究所

[2] 汉语分词在中文软件中的广泛应用　李东 张湘辉　微软中国研究开发中心