**摘要**

这里记录了我学习搜索引擎的一点以及，与大家共享！

# 第1章 引言

# 第2章 搜索引擎概貌

## 2.1 搜索引擎的发展历史

Excite

yahoo

Webcrawler

lycos

infoseek

altavista

hotbot

northernlight

sohu

google

fast(alltheweb)

teoma

wisenut

gigablast

openfind

北大天网

baidu

有道

## 2.2 搜索引擎的主要需求

1. 查得快

商用搜索引擎的查询速度达到毫秒级。

影响速度的原因有很多：索引库的效率，分布查询的处理能力和查询缓存的命中率

1. 查得全

索引的网页越多，查全率越高。

1. 查得准

对于搜索引擎，查全率往往不是最重要的，衡量的意义也不大，因为没有一个用户会把所有查询到的网页都浏览一遍。一般情况下，用户最为关注的仅仅为搜索结果中的前几条。因而查准率很大程度上决定了搜索的质量，在前10条搜索结果中满足用户的查询目的的数目是查准率的主要体现。

是否查得准，主要取决于网页排序。

1. 查得稳

商用搜索引擎，对稳定性有很高的要求，必须能够提供持续的信息检索服务。

## 2.3 搜索引擎的4大系统

无论是工业界还是学术界，搜索引擎一致地被认为分为下载，分析，索引，查询四大系统。

下载系统负责从万维网上下载网页，并且保持对万维网变化的同步。

分析系统负责抽取下载系统得到的**网页数据**，并进行PageRank和分词计算。

索引系统负责将分析系统处理后的**网页对象**索引入库**。**

查询系统负责分析用户提交的查询请求，然后从索引库中检索出相关网页并将网页排序后，将查询结果返回给用户。

搜索引擎的简要结构图如图2-1所示。



图2-1 搜索引擎的简要结构图

从整体上，下载、分析和索引系统组成了搜索引擎的数据制作部分，被称为是“离线部分”（offline part）；查询系统为搜苏偶引擎的数据服务部分，要求快速响应，因此被称为“在线系统”（online part）。按照离线和在线划分，搜索引擎又可以分为在线系统和离线系统。其中在线系统需要毫秒级的访问速度，而离线系统中有些需要长达几周的时间才能计算完毕。

# 第3章 搜索引擎的下载系统

## 3.1 爬虫的发展历史

世界上第一个Spider程序是MIT Matthew Graw的World Wide Web Wanderer，用于追踪万维网的发展规模。

1994年7月，Michael Mauldin将John Leavitt的蜘蛛程序接入到其索引程序中，创建了目前为人熟知的Lycos。

## 3.2 万维网及其网页分析

### 3.2.1 蝴蝶结型的万维网

如果将万维网定义为一个相互连通连通图，网页为结点，链接（link）为边，那么任意一个网页可能被其他网页链接，这种链接称为“反向链接”（backlink）。这个网页也可能链接到其他网页，这种链接成为“正向链接”（简称“链接”）。

Border通过一个有趣的Rrandom-start BFS（Breadth-First Searsh）实验随机取得570个网页作为开始结点，依次逐个地实验这570个网页。首先采用正向（forward direction）宽度优先遍历，即按照网页到网页的链接进行遍历。然后采用反向（backward）宽度有限遍历，即按照网页到网页反向链接进行遍历。例如网页A有一个链接指向网页B，则从A开始，正向遍历可以到网页B；从网页B开始，反向遍历可以到网页A。

通过实验发现，无论是正向遍历还是反向遍历，表现出来的是截然不同的遍历效果。要么是遍历到很小的结果集（90%的情况下探测到的集合不超过90个节点，极端情况下也只有几万个网页而已）后遍历终止；要么是爆炸性地遍历到了大约1亿个网页，但是没有探测到全部的18600万个结点。此外，对于一部分结点，无论是正向，还是反向遍历都能够探测出大约1亿个网页，这部分属于后面提到的SCC部分。通过实验数据，Border得出这样一个结论，即万维网具有蝴蝶结型（bow tie）结构，如图3-1所示。

图3-1 万维网的蝴蝶结型结构

其中网页分为如下4中类型，各约占1/4。

（1）蝴蝶结的中部（SCC, Strongly Connected Component）。

这种类型的网页彼此相连，任意去掉有限个网页，不会影响其连通度。Random-start BFS 遍历实验的起始点选择该部分网页，采用正向遍历的方法，从统计的角度上看可以遍历占全部网页的3/4的网页数；采用反向遍历的方法也可以遍历大致同样数量的网页。

（2）蝴蝶结的左部（IN）。

这种类型的网页指向中心部分（SCC），称为“目录行网页”（hub page），即通常说的导航网页。Random-start BFS遍历实验的起始点选择该部分网页，采用正向遍历的方法可以遍历占全部网页3/4的网页数；采用反向遍历的方法只能遍历很有限的一些网页，所占比例可以忽略不计。

（3）蝴蝶结的右部（OUT）。

这种类型的网页被中心部分指向，称为“权威性网页”（authority page）。这些网页被引用次数多，表示为大多数网页对其“认可度”高。Random-start BFS 遍历实验的起始点选择该部分网页，采用正向遍历的方法只能遍历有限的网页；采用反向遍历的方法可以遍历占全部网页约3/4的网页。

（4）蝴蝶结的须脚（Tendrils）。

这种类型的网页为从左部链出到其他网页，或者从其他网页链入右部或从左部直接进入右部，以及少数部分与中部、左部或右部都没有链接（非连通量DISC）。Random-start BFS遍历实验的起始点选择这部分网页，无论采用何种遍历方法都只能遍历有限的网页。

通过万维网的结构特征得出如下两个结论。

（1）爬虫尽可能选择蝴蝶结的左部，或者中部的网页为起始访问节点集合（starting set of URLs）进行遍历，这样可以得到尽可能完整的遍历效果。如果选择右部或者须脚部分的网页为起始节点，则只能抓取很有限的网页。

（2）网页分为目录性和权威性网页，目录型网页为普通网民服务，便于网民点击从而继续浏览更多的网页。该部分网页对于深入抓取权威性网页有重大意义；权威性网页是那些处于蝴蝶结中部或右部的网页，这类网页的反向链接数很多，而正向链接数相对较少，通常认为这类网页比较重要。

### 3.2.2 万维网的直径

万维网直径（Diameter of the World Wide Web）又称为“Web直径”，其定义为如果用d表示存在一条从网页u到网页v的路径，那么这些万维网上所有不同的连通网页对（pair of connected page）所构成的最短路径的平均长度即Web直径。按照这样的定义和大规模网页的计算得到Web直径约为17左右。有兴趣的读者可以参考文献[Reka Albert et al. 1999]。万维网直径计算公式d=0.35+2.06log(N)（其中N为网页的数量）。有研究表明中国的万维网直径为16.26，即如果任何两个网页之间存在一条路径可达的话，平均点击不超过17次即可从一个网页到达另一个网页。

事实上，网页的平均出度为25.7，即平均一个网页含有25.7个指向其他网页的链接。这些链接引导网民继续在网上冲浪。直到对某个主题厌倦，开始下一个主题。下一章还会讲到这个冲浪模型。

同样得到如下结论。

（1）遍历的方法很大程度上影响了爬虫的效率，万维网的网页结构并没有我们想象的深，却出乎我们意料的宽，因此爬虫的遍历方式多采用宽度优先的遍历方式。当然这里还有一个网页重要性的原因，采用这种方式可以较好地抓取重要性高的网页。

（2）万维网错综复杂，任选一个抓取路线不能保证总是最优。为了防止爬虫一路走到黑，充分考虑万维网的万维网直径后，采用“深度策略”控制抓取深度，从而完美地解决了这个问题。

### 3.2.3 万维网的规模及变化特征

在了解了万维网宏观静态的一面之后，再来看看其宏观动态变化的一面。

毫无疑问，互联网的规模是巨大的。早在1998年，各种研究就在试图估计万维网的规模。尽管方法各异得出的具体数值不尽相同，但都认为万维网的规模（有价值的网页数量）在十亿数量级上。根据Lawrence和Giles在1999年研究显示，世界万维网的数目每两年增加一倍，因此截至目前万维网的规模已达百亿数量级。

2000年斯坦福大学的Cho和Garcia-Molina随机选用50万个网页做样本，发现23%的网页时每天更新的，其中40%域名后缀为.com的网页时每天更新的，网页的半衰期（half-life）为10天。也就是说50万的网页在10天后只剩下25万，再过10天只剩下12.5万。此外，网页的变化可以归纳为泊松过程（Poisson process）模型（在网页重访策略部分会详细提到这个模型）。可以说万维网的变化日新月异，一日千里，如何与万维网的变化保持同步成为爬虫的又一个主要难点。

### 3.2.4 网页的特征

让我们从宏观到微观，考察一下网页的特性。归纳起来，对于搜索引擎来说，网页具有3大特征部分别是挥发性（volatile）、半结构性（semi-structed）、和隐蔽性（hidden）。网页诞生到网页消亡（挥发性）总是不可避免的，HTML语言描述的网页是一种半结构化的数据。除了静态网页以外，还有很多隐蔽的动态网页，例如需要登录才可以看到的某些动态网页。

接下来从有关爬虫的基本概念开始，由点到面，层层深入地理解爬虫是如何针对万维网的这些特点，采取相应的策略实现了抓得全、抓得快且代价低的基本目标。

## 3.3 有关爬虫的基本概念

**爬虫**

爬虫也称为”Wanderers”（漫步者），”Robots”（机器人）或者”Spider”（蜘蛛），它是一个程序，在搜索引擎系统中负责抓取时新的且公共可访问的Web网页、图片和文档等资源。这种抓取的过程为下载一个网页，分析其中的链接，继而漫游到其他网页，循环往复。

**种子站点**

种子站点是爬虫开始抓取的站点，通常为各大门户网站和观望网站的首页等。

**URL**

URL是”Uniform Resource Locator”（统一资源定位器）的缩写，可以理解为网址，用于表示万维网上一个资源的地址。

**Backlinks**

一个网页的Backlinks是那些指向本网页的链接的集合，B acklinks的数目是衡量网页“受欢迎”程度的重要度量。

## 3.4 网页抓取原理

爬虫的工作原理包括抓取、策略和存储，抓取是爬虫的基本劳动过程；策略是爬虫的智慧中枢；存储是爬虫的劳动成果。我们按照逐个击破，由浅入深的方式逐步了解整个爬虫的工作原理。

### 3.4.1 从种子站点开始抓取

基于万维网的蝴蝶型结构，一般来说，爬虫选择蝴蝶结左部的网页。即目录型网页作为种子站点（抓取出发点），典型的如sina.com和sohu.com这样的门户网站的主页。每次下载一个网页后提取其中的链接，它们指引爬虫更加深入的抓取其他网页。如何继续抓取其他网页，爬虫有两种选择，深度优先和宽度优先。一般选择宽度优先策略，有如下3点原因：

首先，重要的网页往往离种子站点的距离较近，这符合直觉。

其次，万维网的深度没有我们想象的深，达到某一个网页的路径通常很多，总会存在一条很短的路径达到。研究表明，中文万维网直径长度只有17.

最后，宽度优先有利于多爬虫合作（这种策略后面还会提到）。这是因为该规则开始抓取的网页通常都是站内网页，逐渐才会遇到站外链接，因此抓取的封闭性较强。

### 3.4.2 不重复抓取策略

万维网存在大量环路，如果没有任何判断，则爬虫永远抓取不完，出现死循环。死循环会带来两个不利后果，不该抓取的重复抓，该抓的没机会抓（大量的资源被用在做无用功）。为了解决死循环问题，通常需要一个称为“不重复抓取策略”和一个“深度控制策略”的方法来解决。

不重复的关键在于记住历史，只有记住过去才可能不重复。爬虫记录历史的方式是哈希表（也称为“杂凑表”），每一条记录是否被抓取的信息存放在哈希表的某一个槽位上。如果网页在过去的某个时刻已经被抓取，则将其对应的槽位的值置，反之置0。而具体映射到那一个槽位，则由哈希函数决定。

1992年8月，Ronald L.Rivest 在向IEFT提交的一份重要文件中描述了这种MD5签名算法的原理。由于这种算法的公开性和去安全性，所以在90年代被广泛使用。MD5签名是一个哈希函数，可以将任意长度数据流转换为一个固定长度数字（通常为4个整形，即128位）。这个数字称为“数据流的签名”或者“指纹”（Digital Finger Print），并且数据流中的任意一个微小变化都会导致签名值发生变化。

在一个URL哈希函数映射下，任意一个网址都唯一的对应一个整数。一个64位整数可以表达1800万TB，而字符串空间的大小远远大于64位整数所表达的整数空间大小，因此碰撞时不可避免的。碰撞时指那些字符串不同，而计算出相同的签名值的情况。然而如果哈希函数设计得足够好，则发生碰撞的几率可以小到忽略不计。正是由于MD5函数符合这种特性，因此被广泛地用作哈希函数。

标准MD5签名的整数空间很大，128位整数能表达2的128次方个不同的数，这是十分巨大的，而实际分配的哈希表空间却是十分有限，一个普通32位处理器，理论上最多分配2的32次方大小的内存，即4G大小的内存（Linux系统中操作系统内核还需占用1G内存，实际上可供搜索引擎使用的极限内存只有3G）。

因此，在实际处理中将签名值进行模运算映射到实际的哈希表中（可以理解为哈希表存放在内存中）。

一般用一个比特位来记录历史信息，1表示已访问，0表示未访问，通常采用Bitmap数据结构来压缩内存使用量。Bitmap结构使用整型作为基本单元，一个整形为32位，一个int Hash[8] 的数组仅通过8个单元来记录历史，可以记录256条历史。

定义这样一个哈希表，即int Hash[8]。将[www.sohu.com作为字符串执行一次MD5](http://www.sohu.com作为字符串执行一次MD5)运算，假定得到的签名值为34。用34除以32，得到商为1，继续用1对8取模得到1，表示槽位在Hash[1]中。34除以32余数为2，表示34映射到Hash[1]这个整数的第3个比特位上，将这个比特位置1。

通常使用以为运算来快速计算除以或者乘以一个2的n次幂；将某个比特位置1采用按位或；查询某个比特位是否为1采用按位与。

对前面的计算过程用C代码的方式表示如下：

int MD5 = 34;

int index\_int = MD5 & 31;// 相当于%32

int index\_Hash = (MD5 >> 5) & 7;// 相当于(34/32)%8

if(Hash[index\_Hash] & (1 << index\_int)) // 探测该槽位是否已置 1

{

// 已经访问过

}

else

{

// 此网页没有访问过

// 下载此网页

// 最后记录下历史，表示已访问

Hash[index\_Hash] |= 1 << index\_int;

}

此外，Bloom filter 算法还可以更加经济地理用好哈希表中的比特位，使得更加经济地灵活地使用内存。简单地说，对于一个字符串独立地使用m个哈希函数计算，然后将其值映射到哈希表的m个槽位上。如果这m个槽位都是1，说明该字符串已经出现过；否则说明该字符串是第一次出现，将m个槽位置1。通过图3-12 来进一步理解。



图3-12 Bloom filter 原理

在抓取到 [www.sohu.com](http://www.sohu.com) 时，执行3次独立的不同的哈希函数（如图3-12中的Hash1、Hash2和Hash3）运算。映射到哈希表的3个槽上，假定是1、3和7。接下来如果抓取到[www.sina.com](http://www.sina.com)，经过3次哈希函数运算，假定计算结果是1、3和6，很显然存在一个槽6是之前没有置位的，因此[www.sina.com](http://www.sina.com)没有抓取过。成功抓取[www.sina.com](http://www.sina.com) 后，将槽位6置1.Bloom算法一方面提高了哈希数组的额利用效率，并被证明适宜于应用在大规模集合查找计算中。同时在网络应用中也被大量采用，有兴趣的读者可以进一步阅读文献[Bloom, B.H. 1970]深入理解该算法。

在抓取提速部分讲到过通过多爬虫的合作抓取，这样可以进一步降低每个爬虫的用于记录历史抓取情况的哈希表大小。即如果有n个爬虫，则可将哈希表继续压缩到原有大小的n分之一。如果n个爬虫分别运行在不同的机器上，那么每个机器被哈希表占用的内存量将非常少，通常保持抓取历史记录所需要的内存在百兆字节左右是恰当的。

通过记录历史的方法初步解决了死循环问题，即抓过的不再抓，循环条件被破坏。然而实际操作中还有这样一个隐含的问题，假如从种子站点A、种子站点B和种子站点C开始均存在一条到达网页P的路径，路径长度分别是3、27和143。很明显，需要浪费两次检查P是否被抓取的开销。

为了防止爬虫无限制的宽度优先抓取，必须在某个深度上进行限制，到达这个深度后就应该停止抓取。这个深度就是万维网直径。当在最大深度上停止时，那些深度过大的未抓取的网页，总是期望可以从其他种子站点更加经济地到达。不难看出，限制抓取深度也破坏了死循环条件，即使出现循环也会在有限次后停止。此外，深度策略和宽度优先遍历策略的组合可以有效地保证抓取过程中的封闭性。即在抓取过程中总是在抓取相同域名下的网页，而很少出现其他域名下的网页。

爬虫总在不停地抓取网页，其工作负荷能力总是有限的。同时下载带宽和时间也是有限的，因此必须能够优先下载重要性高的网页。

### 3.4.3 网页抓取优先策略

网页抓取优先策略也称为“页面选择问题”（Page Selection），通常是尽可能地首先抓取重要性的网页，这样保证有限的资源内尽可能地照顾到那些重要性高的网页。那么哪些网页才是重要性高的呢？如果量化重要性呢？

重要性度量由链接欢迎度、链接重要度和平均链接深度这3个方面决定[Arvind Arasu et al.2001]。

定义链接欢迎度为IB(P)，它主要由反向链接（Backlinks）的数目和质量决定。首先考察数目，直观地讲，一个网页有越多的链接指向它（反向链接数）越多，那么表示其他网页对其的认可。同时这个网页被网民访问的机会就打，推测出其重要性也就越高；其次考察质量，如果被越多重要性高的网页指向，那么其重要性也就越高。如果不考虑质量，会出现局部最优，而不是全局最优的问题。最典型的就是作弊网页，认为地在一些网页中设置了大量反向链接指向其自身的网页，以提高该网页的重要性。如果不考虑链接质量，就会被这些作弊者所利用。

定义链接重要度为IL(P)，它是一个关于URL字符串的函数，仅仅考察字符串本身。链接重要度主要通过一些模式，比如认为包含 “.com” 或者 “home” 的URL重要度高，以及具有较少斜杠的URL重要度高等。

定义平均链接深度为ID(P)，此为笔者所创。ID(P)表示在一个种子站点集合中，每个种子站点如果存在一条链路到达该网页，那么平均链接深度就是这个网页的又一个重要性指标。因为距离种子站点越近，说明被访问的机会越多。事实上，宽度优先遍历已经符合了这种重要性高的网页被优先抓取的需要。

最后，定义网页重要性的度量为I(P)，它由以上两个量化值线性决定，即：

**I(P)=α\*IB(P)+β\*IL(P)**

平均链接深度由宽度优先遍历的遍历规则保证，因此不作为重要性评价的指标。

尽管这样看起来已经足够完美，事实上，还有一个重要的要素——时间。如何抓取新增的页面呢？如果重访（revisit）那些被修改了的网页呢？如果发现那些被删除了的网页呢？为了保持和万维网网页的同步变化，就必须有网页重访策略。通过该策略可以识别增加、修改及删除网页这3种网页变化的情况。

### 3.4.4 网页重访策略

文献[CHO, J. AND GARCIA-MOLINA, H. 2000a]表明网页变化可以归结为泊松过程（Poisson prcess）模型。

在泊松模型的理论基础上，结合人们的直观感知，目前网页重访策略大致可以归为以下两类。

（1）统一的重访策略：爬虫以同样的频率重访已经抓取的全部网页，以获得统一的更新机会，所有的网页不加区别地按照同样的频率被爬虫重访。

（2）个体的重访策略：不同网页的改变频率不同，爬虫根据其更新频率来聚顶重访该个体页面的频率。即对每一个页面都量身定做一个爬虫重访频率，并且网页的变化频率与重访频率对任何个体网页来说都是相等的。

两种重访策略各有利弊，深入的研究表明对于那些更新频繁的网页采用更加频繁的重访策略反而有害[CHO, J. AND GARCIA-MOLINA, H. 2000b]。因此策略（2）还需要增加一些其他子策略，例如限定最大重访频率。策略（1）的效果在数学上可以证明总是不低于个体重访策略[CHO, J. AND GARCIA-MOLINA, H. 2000b]，并且具有计算量小的优势。但是策略（1）不具有优化的空间，难以满足质量不断改善的要求。

此外，不同类型的网站更新频率也不同。通过对不同类型的网页寿命的半衰期实验证明了这一点[CHO et al.2000c]。.com 类型的网页变化最聚类，.gov类型的网页变化最少，因此给予不同类型的网页不同的重访频率是科学的。

回顾网页更新策略既能保证抓取历史的网页，也能够抓取随时出现的新网页，几个重要的结论如下。

（1）网页更新过程符合泊松过程。

（2）网页更新时间间隔符合指数分布。

（3）对于不同类型的网页采用不同的更新策略。

粗放的工作做完了，然而还远远没有达到完美。爬虫的工作模式和方法存在一些问题，甚至可以说是社会问题[BRIN, S.AND PAGE, L. 1998]。

有些网站不希望爬虫抓取敏感信息；有些网站希望爬虫避免在白天对其网页进行抓取，从而不影响白天正常的对外公众服务；DNS服务提供商也不希望大量的域名解析工作量被搜索爬虫的域名请求所占用。

### 3.4.5 Robots协议

Robots协议是Web站点和搜索引擎交互的一种方式，即将一个robts.txt 的文件放在网站的根目录上，例如 <http://www.w3.org/robots.txt>。

该文件内容如下：

#

# robots.txt for http://www.w3.org/

#

# $Id: robots.txt,v 1.54 2008/11/13 23:37:54 ted Exp $

#

# For use by search.w3.org

User-agent: W3C-gsa

Disallow: /Out-Of-Date

User-agent: W3T\_SE

Disallow: /Out-Of-Date

User-agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT; MS Search 4.0 Robot)

Disallow: /

# W3C Link checker

User-agent: W3C-checklink

Disallow:

# exclude some access-controlled areas

User-agent: \*

Disallow: /2004/ontaria/basic

Disallow: /Team

Disallow: /Project

Disallow: /Web

Disallow: /Systems

Disallow: /History

Disallow: /Out-Of-Date

Disallow: /2002/02/mid

Disallow: /mid/

Disallow: /2005/06/blog/

Disallow: /2004/08/W3CTalks

Disallow: /2007/11/Talks/search

Disallow: /People/all/

Disallow: /RDF/Validator/ARPServlet

Disallow: /2003/03/Translations/byLanguage

Disallow: /2003/03/Translations/byTechnology

Disallow: /2005/11/Translations/Query

Disallow: /2003/glossary/subglossary/

#Disallow: /2001/07/pubrules-checker

#shouldnt get transparent proxies but will ml links of things like pubrules

Disallow: /2000/06/webdata/xslt

Disallow: /2000/09/webdata/xslt

Disallow: /2005/08/online\_xslt/xslt

Disallow: /Bugs/

Disallow: /Search/Mail/Public/

Disallow: /2006/02/chartergen

Disallow: /2004/01/pp-impl

在上述文件中，Robots协议通过User-agent和Disallow告知搜索引擎非公开目录和非公开网页，说明如下。

（1）User-agent:\*：表示对一切搜索引擎有效，如果特别针对某个爬虫，则可以写明。

（2）Disallow: /2004/ontaria/basic：表示禁止抓取这个目录。

通过遵守Robots 协议，表示出爬虫尊重和执行Web站点的要求。因此爬虫需要有一个分析Robots协议的模块，并严格按照Robots协议的规定只抓取Web主机允许访问的目录和网页。

### 3.4.6 其他应该注意的礼貌性问题

除了遵守Robots协议以外，通常搜索引擎需要自身维护一个DNS域名解析的服务模块，以便减少对公开域名解析服务器的频繁请求。

此外，应该尽可能合理地规划抓取强度。在百联尽可能地减弱抓取强度，例如每10秒抓取一次；在夜间Web主机访问量低，可以适当增加抓取强度，例如每秒一次。由于时差的原因，东半球的白天恰好是西半球的夜晚，因此白天可以加强抓取美国及欧洲站点的强度，雅间增加对中国站点的抓取强度。

即便如此，爬虫总不可避免会给其他万维网上的Web主机带来麻烦。因此站点抓取监控程序也是必不可少的，该程序记录每个站点的抓取流量，避免在偶然情况下因抓取强度太大而导致各种问题。

这样，最后的问题就只剩下规模问题。如此巨大规模的万维网，每天新增的大量网页需要及时地被抓取到。对于爬虫来说，一方面对历史的网页需要重抓；另一方面要及时抓到新增的网页，在如此沉重的工作负荷下，必须提高抓取速度才能满足这种需要。

### 3.4.7 抓取提速策略（合作抓取策略）

提速基本可以采用下面几种方法。

（1）提高抓取当个网页的速度。

（2）尽可能减少比必要的抓取任务。

（3）增加同时工作的爬虫数量。

事实证明，受到万维网发展水平的限制，第（1）种方法基本不可行，单个网页的抓取速度收到下载带宽的限制，在现有技术条件下很难任意提高；第（2）种方法难度很大，由于需要和万维网的变化保持紧密同步，所以冗余的抓取总是不可避免的。减少不必要的抓取会导致网页重访不及时，这样就不能快速同步目标网页的变化；第（3）种方法通过增加爬虫数量提高总体抓取速度是可行的。

在多个爬虫抓取的情况下，如何将工作量分解成为主要的问题。纪要解决一个网页交给哪个爬虫抓取？如果分工不明，很可能多个爬虫抓取了相同的网页，从而引入额外iade开销。通常采用以下两种方法来进行抓取任务的分解。

（1）通过Web主机的IP地址来分解，使某个爬虫仅抓取某个地址段的网页。

（2）通过网页的域名来分解，使某个爬虫抓取某个域名段的网页。

如何选择这两种方案呢？

存在多个域名对应同样IP的情况，对于中小网站来说，通常采用这种方法提供不同的Web服务。这主要出于经济的考虑，因为可以只配置一台服务器；而对于大型网站，如新浪和搜狐这些门户网站通常采用负载均衡的IP组技术，同样的域名对应于多个IP地址和，一方面提高系统健壮性，一方面做到了负载均衡。

通常的做法是按照域名分解任务。即只要保证不重复抓取大型网站的网页，小型网站即使重复抓取也是可以接受的。

这两种方案的主要区别可以通过下面的这两个例子进一步理解。

假定zoujinsousuoyinqing.cn和zoujinsousuoyinqingbook.cn 是两个域名不同，但IP相同的网站，假定为10.10.67.208。有这样两个网页，即http://www. zoujinsousuoyinqing.cn/index.html 和 http://www. zoujinsousuoyinqingbook.cn/index.html 。在经过域名解析后实际上它们同为http:// 10.10.67.208/index.html。采用域名分解的方案，spider1和spider2就会重复抓到这个网页。但是由于这两个站点信息较小，所以可以容忍因抓重而带来的损失。

反之，如果我们通过IP分配抓取任务的方案，例如sina.com.cn （71.5.7.138）被“指定”交给spider1，sina.com.cn（71.5.6.136）被“指定”交给spider2，这样两个爬虫的抓取任务就互相重复。很显然，新浪属于大型网站，这种因为抓重带来的损失往往是巨大的。

通过比较，为了照顾大站按照域名分解的策略更加合理。在下载系统中，按照域名分解抓取任务（网页集合）的工作由一个成为“调度员”的模块来处理，通过域名分解将不同的网页调度给不同的爬虫进行抓取。因此，，下载系统主要由爬虫和调度员组成。

形式化的调度分配方式如下：

首先假定有n个爬虫可以并行工作，并且顶一个可以提取URL域名的函数domain()，例如：

URL= <http://hi.baidu.com/soulmachine>

domain(URL) = hi.baidu.com

说明如下：

（1）对于任意的URL，利用domain函数提取URL的域名。

（2）计算出爬虫编号，int spider\_no = MD5(domain(URL)) % n。

（3）将该URL分配给编号为spider\_no的爬虫进行抓取。

在介绍了爬虫和调度员之后，已经能够完整地理解搜索引擎下载系统的体系结构，入股3-15所示。



图3-15 搜索引擎下载系统的体系结构

这里我们引入了URL库和Page库这两个概念，URL库存放全部历史上曾经抓取过的URL和新增的URL，Page库存放爬虫实际抓取下来的原始网页。这样完整的合作抓取过程如下，注意下面的标号对应于上图中的数字。

（1）调度员通过更新规则向URL请求一个URL抓取任务。

（2）调度员计算出该URL的爬虫编号，然后分配给对应的爬虫，例如爬虫0.

（3）爬虫0将实际抓取的网页存放在Page库中。

（4）爬虫0在抓取的网页中提取其他链接后反馈给调度员。

（5）调度员判断网页类型，并设定初始更新时间等后存放在URL库中。继续转（1）。

在实际应用，多采用多爬虫多调度员的体系结构，如图3-16所示。



图3-16 多爬虫多调度员的体系结构

值得注意的是，抓取的封闭性越强，对外的通信开销越小。如上图所示，加入爬虫1从sina.com 这个种子站点开始抓取。由于总是抓取sina.com 的网页，而这些网页总是归属爬虫1花去，因此不需要和其他爬虫通信（不需要经过总调度）；反过来，如果抓取的封闭性差，表示可能抓取到各种各样域名下的网页，并且可能需要交给其他爬虫抓取。总调度不得不把这些信息相互转发，这样就会增加额外的通信代价。因此提高抓取的封闭性可以减少这种合作抓取带来的通信开销。前面提到过的宽度优先遍历及深度控制策略能有有效地保证这种抓取的封闭性。

最后，简单了解一下大规模网页存储的有关知识。

## 3.5 网页库

很显然，爬虫抓取的网页必须及时保存在硬盘上，因此网页库的首要挑战来自于能够快速存储大规模网页；其次这些网页必须能够被其他模块快速地读取，因此读写问题是围绕网页库的主要难点。

1. 可伸缩性（Scalability）

网页库的存储必须具有可伸缩性，可以将大规模网页平滑地分布在一组计算机的硬盘中。

1. 双访问模式（Dual access modes）

网页库必须能够有效地支持两种不同的访问模式，即随机访问模式和顺序访问模式。使用随机访问模式，任意给出一个网页的标识即可读取相应的网页；使用顺序访问模式，顺序读取全部网页或者一部分网页。随机访问模式主要为搜索系统的查询系统中网页缓存部分提供随机的读取需要（在第6章还会提到）；顺序访问模式主要用在索引系统中顺序读取网页按序索引的过过程中。

1. 大规模更新（Large bulk updates）

由于万维网瞬息万变，所以网页库必须能够在网页删除后删除老版本的网页，如此处理可能会留下存储空洞。更新可以理解为删除后添加，而添加使用顺序添加到网页库的方法。这样不得不采用一些磁盘空间紧缩（space comaction）技术回收那些存储空洞。此外更新和访问还要保证互斥，避免同步出现的错误。

为了照顾这些特性，网页的存储方式大致分为以下3种类型。

1. 日志结构（Log-structured）；
2. 基于哈希的结构（Hash-structured）；
3. 哈希日志（Hashed-log）。

日志结构将磁盘看作一个连续存储的戒指（可以想象为磁带机），这种存储方式只能顺序读写，因此对网页添加的操作非常有利。然而对于随机访问，则必须通过B+树的索引方式，因此每次访问需要执行两次读操作，分别读取索引和目标网页。此外还需要维护索引的代价，因此日志结构不能理想地支持随机访问的需要。

基于哈希的机构将磁盘看作一个哈系桶（Hash bucket）的集合，每个集合都足以存放在内存中。由于不需要构建而外索引，因此对于随机访问非常有利。然而对于顺序读写将十分不利，因为由于哈希函数的不确定性，对于一次网页增加，往往读入内存一个哈希桶，然后写入新增的部分，在写入磁盘。对于那些哈系统中不变的数据，这样的一次读写操作无疑是浪费的，不断新增的网页被哈希映射到同一个桶的概率极低。而且由于哈希函数的不确定性，这种频繁读入哈希桶，更新并写入的草组被不断重复，因此对于网页新增是极其不利的。

哈希日志的结构是将哈希和日志的优势相结合，哈希作为索引找到块后在顺序写入，如图3-17所示。



图3-17 哈希日志的网页存储结构

对于新增的网页，通过URL计算出该网页的签名。进而通过模运算之后，将一个网页映射到哈希表的一个单元上，每个哈希表的单元对应于一个日志文件的位置。这个新增网页通过哈希函数的计算映射到Hash[1]上，继而顺序写入Log1这个文件中。如果要随机读取某个URL的已存储的网页，仍然通过类似的哈希函数映射到具体的日志文件上，然后就通过读取该日志文件上的B-树索引，进而读取相应的网页文件。

值得一提的是，哈希日志结构可以支持批处理写入，从而大大改进了纯哈希结构。在每个日志文件中增加一个写入队列，只有积累到一定数量的文件后，才批量写入，如图3-18所示。



图3-18 哈希日志批量写入记录

如上图所示，通过一个哈希表可以将那些新增网页插入的不确定性（与哈希结构相比）变为确定性插入，因此增加一个插入队列可以批处理方式插入到目标日志文件中。由于通过哈希函数的分解，所以在基于哈希结构的日志中每个日志的大小远小于日志结构中的日志，同时远大于哈希结构中的哈希桶。另外还必须保证每个日志也能够存放在内存中，因此确定哈希日志中哈希表的大小在很大成都上还需要要考虑实际物理内存大小和需要存储的网页规模。

文献[Arvind Arasu et al.2001]中给出了对于这3中网页存储方式的一个定性评价，如表3-1所示。

表3-1 3中网页存储方式的一个定性评价

|  |  |  |  |
| --- | --- | --- | --- |
|  | 日志结构 | 哈希结构 | 哈希日志 |
| 顺序访问 | ++ | - | + |
| 随机访问 | +- | ++ | +- |
| 增加网页 | ++ | - | + |

综上所述，网页库是搜索引擎中重要的一环，网页库具有不同的访问需求（顺序访问和随机访问），具体问题具体分析时需要结合这些需求选择相应的存储方式。在实践中，日志结构和哈希日志是较为常见常见的选择。

## 3.6 下载系统回顾及未来发展

通过系统地学习，总结下载系统的主要需求有以下3点。

1. 抓得全：通过网页更新策略（更新目录型网页）。
2. 抓得快：通过合作抓取策略。
3. 代价地：通过宽度优先策略、深度控制策略和合理的网页更新策略。

当然这与一个世纪的商用大型搜索引擎下载系统还有很大的距离，例如大规模网页存储、DNS服务模块、反垃圾、反病毒、多爬虫的卸掉工作，以及爬虫监控等。读者如果亲自实现一个爬虫还会遇到很多困难。目前比较著名的关于爬虫的开源代码很多，例如Web sphinx 和 Heritrix等。

虽然前人做出了巨大的努力，取得的了大量的成果，然而爬虫的发展还远远没有停止。未来还有很多工作等待我们去探索，那么还有哪些工作可以做呢?

1. 动态网页支持

Web 上的动态网页是静态网页的400~300倍，光明星球[brightP]公司宣称，存在的网页总数至少为5500亿个，这个数量是相当今人的。目前几乎所有的搜索引擎都不能完全解决抓取动态网页的难题，因为这些动态网页通常都受到了帐号和密码的保护，这也成为“深度挖掘问题”。

1. 定向抓取

定向抓取也称为“聚焦爬虫”，目的是使爬虫的工作方式不再是漫无目的，而是在某种意图下抓取有价值且特定的网页。

1. 抓取XML网页，以及DOC和PDF文件

抓取的内容突破HTML网页，也要抓取分析那些XML网页，以及DOC和PDF等文件。

1. 智能分布抓取

随之Web的迅猛发展，可以考虑物理上分布多个可扩展Web搜集系统，在更广泛意义下并行分布式工作。

## 参考文献

# 第4章 搜索引擎的分析系统

搜索引擎的4大系统中的第2个系统是分析系统，分析系统主要完成的工作包括信息抽取、网页消重、中文分词和PageRank计算等。

## 4.1 知识储备

**锚文本（anchor text）**

锚文本是网页中关于某个链接的一段描述。例如HTML代码

<a href=<http://zoujinsousuoyinq.com>>走进搜索引擎</a>

中的“走进搜索引擎”就是一个“锚文本”，它用来描述一个超链接。

**半结构化数据（Semi-structured data）**

跟普通的纯文本相比，万维网上的网页数据具有一定的结构性，但是这种HTML标签带来的结构性不能满足网页结构化的需要，例如<TITLE>标签标识网页注意，而<TD>标签有些表示文章主题，有些表示文章段落，或者其他广告信息等。因此人们称网页的原始数据为“半结构化数据”，这是HTML语言的基本特点。

## 4.2 信息抽取及网页信息结构化

对于分析系统来说，基础和兽药的工作是分门别类地从半结构化网页（semistructured page）中抽取（extract）出有价值的能够岱庙网页的属性，例如锚文本、标题和正文等。并将这些属性结合起来组成一个网页对象，这种处理称为“网页结构化”。

### 4.2.1 网页结构化的目标

网页结构化的目标是针对搜索的需要，将半结构化的HTML网页中的数据按照如下几个基本属性依次抽取，最后打包（wrap）出一个网页对象。

1. 锚文本（anchor text）；
2. 标题（title）；
3. 正文标题（content title）；
4. 正文（content）；
5. 正向链接（link）。

网页的这5个基本属性对于检索来说至关重要，下面一一介绍。

（1）**锚文本**：除了网页标题可以描述网页意外，还会有一些锚文本来描述它。例如清华大学主页可能被另外一些网页中存在的链接所指向，其锚文本就是该网页的最佳描述。特别是对于某些没有标题的网页，锚文本是有益的补充。

（2）**标题**：这里的标题特质HTML代码中<title></title>中间的文字，这部分文字表达了网页的基本含义。和锚文本相同的是，都是用来描述网页的内容的属性；和锚文本不同的是这个标题是由该网页制作者本人编写的。

（3）**正文标题**：很多网页没有<title>标题，或者<title>标签中的文字没有任何描述能力，并不是合格的标题，为此需要抽取正文中的适当文字作为正文标题。

（4）**正文**：锚文本、标题和正文标题都市网页的简短描述，而正文是一个网页的主题内容，它完整地表达了网页的主题内容，一边出现在<DIV>、<TABLE>和<P>等HTML标签中。

（5）**正向链接**：一个网页中往往有很多指向其他网页的超链接，这些链接也有其他网页的锚文本。

理解结构化网页的目标后，加上对HTML语法的特点，以及搜索引擎的实际需要，为这种分析原始网页的过程指定了如下“两步走”的方法。

1. 建立HTML标签书（tag-tree）。
2. 通过投票方法识别正文中的文本块，并按照深度优先遍历的规则组织为正文。

### 4.2.2 建立HTML标签树

HTML是一种标识语言（Markup Language），它把其描述的全部内容都按照HTML语法存放在标签之中为了更清楚地描述网页内容的组织结构，将网页中的标签按照出现顺序依次整理出来并用适当的结构记录。由于标签之间的嵌套关系，因此整理结果自然是一个树状结构，我们把整理一个网页中的标签得到的树状结构称为该网页的“标签树”。

由于HTML语言中标签似乎嵌套，并称对出现的，符合先进后出的特性，因此可以用栈来分析。

建立的标签树如图4-8所示。



图4-8 从HTML网页到结构化网页

### 4.2.3 通过投票方法得到正文

由于网页变结构化的特性，一方面，TITLE很容易通过第1步标签分析得到；另一方面得到完整的正文却非常复杂。首先，网页中没有明显的标签标识出正文；其次正文可能是分散在多个HTML标签中，问题在于如何组合出完整的正文。首先来解决第1个难点，即使别正文。

正文具有分块保存的特性，因此我们引入文本块的概念，对于那些诸如<P></P>等标签间的文本认为是一个文本块。例如：“<TD>走进搜索引擎：第一章</TD>”是一个文本块。

一般来说，网页会出现3种类型的文本块。

1. 主题型文本块（topic）。
2. 目录型文本块（hub）。
3. 图片型文本块（pic）。

主题型文本块是大段文字的文本块，例如：“<TD>走进搜索引擎：第一章</TD>”是一个文本块。

目录型文本块就是锚文本。

图片型文本块是描述图片的文本块，例如：“<img src=./bg.gif>走进搜索引擎</img>”。

目录型文本块和图片型文本块容易被区分，而主题型文本块可能包含广告等其他内容，必须与正文相区别。

判断哪个文本块是正文采用称为“投票算法”的计算方法，这种方法在搜索引擎中特别常用。

首先我们会顶一个规则，然后通过这些规则为每个文本块打分。得分最高的被认为是正文的可能性比较大。

举个例子，假定一个规则集合中包含以下3条规则。

（1）如果文本块文本的额长度少于10个字，得分为0；介于10个～50个字得分为5；介于50～250个字的氛围8分；超过250个字得分为10分。

（2）如果文本块位置在右侧，得分为0分；在顶部得分为3分；在左侧，得分为5分；在中间，得分为10分。

实际的规则更加复杂。投票算法的过程是依据不同的规则依次打分，文本块得分高的是正文的一部分。

在实际的程序中，规则其的打分是不断调整的，其个数也可以支持动态添加。如果打分计算比较耗时，则可以采用并行打分的方法。

除此之外，规则的定义还需要通过足够多的网页进行反馈之后才能得到一个公正客观的额打分。这类似裁判员在打分前的业务学习，只有在比赛前透彻地理解评分标准，并且很好地观察运动员完成动作的情况，通过不断地训练才能区别动作的难易程度，达到比赛裁判的水平。此外，有些动作在若干年后由于掌握的运动员很多，因此评分标准就会发生变化。新出现的高难动作将会得到评分上的青睐。规则其也需要类似这样的学习过程。如果经常发现某些网页的正文段抽取错误，则要找出是哪一个规则器打分不合理才会导致这个结果。反复这样的过程，最后的打分将会也趋于合理。

最后，还需要注意这样的问题，如果规则A打分为0！1000分，规则B为0～10分，很显然，规则B对最后打分结果的影响力几乎可以不考虑。因此如果平衡各个规则对最后结果的影响也需要充分考虑。

即便做出了如此多的工作，投票的结果也并不总是正确的。真理也常常会站在少数人的一边。因此这种方法不可能百分之百正确，只能是达到某种概率下的正确。

剩下的工作就是如何将一个个文本块组织称一个正文。深度优先遍历标签树即可。

对于其他网页属性抽取，例如正文标题等也大多采用类似的方法。

锚文本的提取有一些复杂，由于锚文本对网页的描述存在于其他网页中，因此锚文本提取、帅选工作需要在获得足够多的网页后才能进行，采用分块计算的方法，比较复杂，这里不再展开叙述。

## 4.3 网页查重

网上经常有一些重要的文章被转载，对于网民来说，这种重复是有利的，便于大多数人看到重要的信息。然而对于搜索引擎来说，重复网页的存在以为这这些网页至少被多处理一次。更糟糕的是在接下来的索引制作中可能会在索引库中索引两份相同的网页。当有用户查询是，在有限的查询结果页中（一个查询结果也显示20条网页链接）就会出现重复的网页链接。因此无论从系统效率还是检索质量来说，重复网页都是有害的。

### 4.3.1 网页查重技术发展历史

网页查重技术起源于复制检测技术，复制检测即判断一个文件的内容是否抄袭、剽窃或者复制于另外一个或者多个文件。

1993年，Arizona大学的Manber推出一个sif工具，用于在大规模文件系统中寻找内容相似的文件。1995年，Stanford大学的Brin和Garcia-Molina等人在“数字图书馆”工程中首次提出了文本复制检测机制COPS（Copy Protection System）系统与相应算法[Sergey Brin et al 1995]。之后这种检测重复技术被应用到搜索引擎中，基本的核心技术即比较相似。

和简单的文档不同，对于网页来说，看这种相似包含了特别丰富的含义。网页具有内容和格式，因此在内容和格式上的相同相似构成了如下4种网页相似的类型（如果把相同看做相似的一种特殊情况）。

1. 两个网页在内容和格式上完全相同（full-layout duplicates）；
2. 两个网页的内容完全相同，但是格式不同（full-content duplicates）；
3. 两个网页有部分重要的内容相同并且格式相同（partial-layout duplicates）；
4. 两个网页有部分重要的内容相同，但是格式不同（partial-content duplicates）。

检测大规模的相似网页需要考虑准确性和效率，文献[Abdur chowdhury et al 2002]，[Broder 1997]分别提出了两种经典的处理大规模网页相似检测的方法，下一节我们将看到这一有趣的方法。

### 4.3.2 网页查重实现方法

由于不考虑格式的异同，网页查重首先需要提取结构化网页的正文和标题，将复杂的网页转化为具有标题和正文的文档，因此下面称网页查重为“文档查重”。

日常生活中判断两个相似事物一般采用比对特征的方法，特征是那些有代表性、不一变化、能够反映主题本质的信息。文档查重的第1步就是特征抽取，下面介绍两种特征抽取的方法。

第1种特征抽取方法着眼于尽可能抽取一个特征，这样比较两个文档是否相似只要比较一次即可。文献[Abdur chowdhury et al 2002]首先提出了称为“I-Match”的算法，这个算法基于一个假设，及一边文档中特别高频和特别低频的词汇往往不能反映这篇文档的本质。因此通过将文档中去掉高频和低频词汇后的词汇通过排序得到一个字符串，使用签名算法获得该字符串的签名，如果有其他文档和这个签名值相同，则判定为相似。

I-Match 算法的实例如图4-13所示。



图4-13 I-Match 算法抽取特征

第二种特征抽取方法是抽取多个特征词汇，通过比较两个特征集合的相似度实现文档查重。抽取多个特征的方法很多，这里介绍其中的一个经典算法，称为 “Shinger算法”。Shinger在英文中表示相互覆盖的瓦片。接下通过上面的例子来说明Shingle算法为什么命名为 “Shingle”。例中使用没无个汉字为一个Shinger的方法，如图4-14所示。



图4-14 Shingle算法抽取特征

如图所示，每5个汉字组成的Shingle恰如其名，好似相互覆盖的瓦片一样。由于把一个文档转化为一组字符串集合，因此判断两个文档的相似性就转化为字符串的相似性。

在特征抽取完毕后，就需要进行特征比对，因此网页查重的第2步就是相似度计算和评价。

I-Match 算法抽取的特征只有一个，因此只需要为文档计算一个签名值，如果两个文档的签名之相同，则文档相似。大规模文档做查重只需要维护一个哈希表即可，每来一个文档查找一次哈希表。如果哈希表的槽位（bucket）被置位，说明已经存在相似文档。

Shingle 算法抽取的特征有多个，因此处理起来稍微麻烦一些，比较的方法是记录完全一致的Shingle个数。然看后出一两个文档的Shingle总个数减去一致的Shingle个数，这种法计算出来的数值成为“Jaccard 系数”，它可以用来判断集合的相似度。Jaccard系数的计算方法为集合的交际处以及和的病急，即令集合A与集合B的Jaccard 系数为J，则有：



对比一下这两种方法，I-Match 算法提取特征时需要文本分词和词频比较的代价，因此提取特征比较复杂，单文档是否相似的计算简单；Shingle 算法提取特征简单，单文档是否相似的计算复杂，因此各有利弊。由于Shingle算法在性能上的优异表现，因而被广泛采用。

这里介绍的是最经典的两种方法，其他网页查重的方法还有很多。文献[李晓明 2004]中介绍了多种网页查重的方法，有兴趣的读者可以深入阅读。

网页查重的目标是消重，因此网页查重的第3步就是网页消重。

消重就不可避免遇到这样一个问题，记载相同或者相似的网页集合中保留哪一个，而消除哪些呢？从版权的角度考虑，应该尊重原创，过滤转载或者复制的网页；从网页寿命的角度考虑，过滤到那些网页质量不高的网页，保留大型网站的网页；从容易实现的角度考虑，保留首先被爬虫抓取的网页，然后丢弃被抓取的相同或相似网页。最后一种方法最为简单实用，由于保留先辈爬虫抓取的网页同时在很大程度上优先保留原创网页的原则，因此被广发采用。

综上所述，一般说来，网页查重至少需要如下3个主要步骤：

1. 特征抽取；
2. 相似度计算，评估是否相似；
3. 消重。

网页查重工作室分析系统不可缺少的重要一环，由于去处了重复网页，所以在接下来的其他模块的处理环节中，节约PageRank 的计算代价、节省了索引存储空间、减少了查询成本、提高了查询结果的多样化效果。

## 4.4 中文分词

网页查重的工作完成后，分析系统在将分析的结果发往索引系统前还需对正文进行分词，也成为“切词”（word segment）。分词的方法繁多，本介绍一些基本方法。

### 4.4.1 什么是中文分词

任何文档都可以看作是一些连续的词的集合，然而中文并没有明显的词间分割，这一点和英文不同。当然英文也有难点，例如时态和词性的变化等。在中文语法中，词汇是由两个或两个以上汉字组成的，并且句子是连续书写的，句子间有标点分割。这就要求在自动分析中文本钱，首先将整句切割成小的词汇单元，即中文分词。

举个简单的例子来说明分词的难度，对于“学历史学好”这个句子，作为人来说，能够很容易得到正确的切分方法，即“学/历史学/好”。然而计算机要具要这样的智慧还有很多工作要做，否则可能分为“学历/史学/好”。

目前中文分词手段主要依靠了字典和统计学的方法。由于索引是按照关键词间索引的，所以分词的效果直接决定了索引词及检索的效果。因此例如将文档“学历史学好”错分成“学/历史学/好”，所以索引时，只会对“学历”、“史学”及“好”这3个索引词建立它们与该文档的关联关系。这样用户查询“历史学”这个关键词时，就无法检索出这个文档，可见分词质量在很大程度上影响了搜索的结果。

### 4.4.2 通过字典实现分词

中文分词中有以下3种歧义给我们分词带来了困难。

1. 交集型歧义

在字符串AJB中，若AJ∈D、JB∈D、A∈D且B∈D，则AJB为交集型歧义字段。此时AJB有AJ/B和A/JB两种切分形式，其中J为交集字段。例如，“从小学”有两种分法。

1. 组合型歧义

在字符串AB，若AB∈D、A∈D且B∈D，则AJB为组合型歧义字段，此时，AB有AB和A/B两种切分形式。

“中将”这个词可能有多种切分方法。

对于“美军中将竟公然说”，征得切分为“美军/中将/竟/公然/说”。

对于“新建地铁中将禁止商业摊点”，正确的切分为“新建地铁/中/将/禁止商业/摊点”。

1. 混合型歧义

同时包含交集型歧义和组合型歧义，则为混合型歧义。

对于“人才能”，可能切分为“人才/能”、“人/才能”和“人/才/能”。

目前解决这些问题的技术手段主要借助字典和统计学的方法共同完成。我们首先从字典开始讲起，梁南元教授[梁南元，1987]最早提出利用字典的方法分词。但如何让计算机读字典，如何让计算机在各种字典的词义种选择合理的分词却并不容易。

字典通常采用前缀书或者后缀树的数据结构存储，什么是前缀树？我们来举个例子。

将字典做成一个前缀树结构的数据结构，如图4-15所示。



图4-15 前缀树结构的词典组织形式

选择若干词建立一个前缀树数据结构作为实例，整个字典将是一个有多个树构成的森林。其中从树根开始，每一个字及其全部祖先构成一个词汇的一部分。对于途中双线圈的词，认为是一次终止，表示它及其全部祖先可以构成字典中的一个词汇，这里“走进”是一个词汇，而“搜索引”不是一个词，因为“引”是单线圈。“搜索引擎”是一个词汇，因为“擎”这个字是双线圈。这种结构十分类似于日常生活中字典的组织方式，即均以一个字开头，其后为以该字开头的全部词汇。日常生活中查字典也是如此，在字典中查一个词，首次按查出该词汇的首个汉字在字典中的页数（这里可以理解为在字典森林中找到一某个字开头的树），继而在其中查找相应的词汇（可以理解为在该字的树的后续中查到指定的词）。

实际的分词并不总是如此简单，这种前缀优先的最大长度匹配的方法对于“学历史学好”这样的句子会分成“学历/史学/好”这样的错误结果，那么如何避免呢？

使用后缀树的方法可以有效地降低这种错误，不过在从一个字典转到一个后缀树多少有些不自然。后缀树的结构如图4-16所示。



图4-16后缀树结构的词典组织形式

与前缀树的构造方法不同，树根表示一个词的结尾字，双线圈表示某个词的开头。

将句子从右往左读，首先找到“好”这个分支。然后读入“学”，恰好存在这样一个词，因此抽取出它。继续读入“史”，依次读入全部句子，直到切分结束，最后切分的结果为“学/历史/学好”。

虽然没有错分出“学历”，但是因为“好”这个字的特殊性，这样的逆向切分仍然不能达到理想的结果。

第1种方法称为“最大正向匹配法”（Maximum Matching Method），通常简称为“MM法”；第2种方法称为“逆向最大正向匹配法”（Reverse Maximum Matching Method），通常简称为“RMM法”。RMM法的基本原理与MM法相同，不同的是分词的扫描方向。

这类分词的方法属于“贪婪算法”，“贪婪”在于总是选择最长匹配的词最后。或者说最大匹配可以使得切分出的词汇数量最少。

基于字典的分词方法能够解决大部分的分词问题，但是字典总是滞后于语言的发展。很多新兴词汇需要每隔一段时间才能成批地收录进词典。因此能够及时地、自动地、准确地发现新词才能把字典分词的威力发挥到最大。新词发现主要通过统计推断的方法来实现。

4.4.3 通过统计学方法发现新词

事实上，发现新出现的词（学术上称为“新词发现”）相当复杂，需要很多基于概率及信息学方面的知识，在这里不展开深入研究。基本的原理就是探索那些经常一同出现的字，总是相互出现的字很有可能构成一个词。为此需要分析和探索大量网页内容，这种分析和探索过程均离线完成，最后还会有一个人工确认的过程。

即使到现在，中文分词技术还在不断地向前发展，还没有哪一个分词算法能够解决一切问题，语言学多样化的重重难关激励着更多的科学家展开更加深入的研究。中文分词的方法也被应用到英文处理，主要是手写体识别中。因为在识别手写体时，单词之间的空格变得比较模糊。中文分词方法可以帮助判别英语单词的边界。因此其研究工作越来越引起广泛的关注。

## 4.5 PageRank

网页搜索的本质是网页信息的聚合，把本来很难聚合在一起的网页通过包含共同的关键词聚合起来。网页被聚合后就自然会产生排序问题。归纳起来就是既不能不排，也不能乱排，需要通过科学有效的方法将“好”的搜索结果按照“好”的程度依次排列。当然“好”是一个相当主观的且难以量化的概念，因此如何评价这个“好”成为搜索质量的关键。在搜索引擎的发展史中，Google发明的PageRank无疑是浓墨重彩的一笔。

### 4.5.1 PageRank 的来由

Google 公司的两位创始人L. Page及S. Brin 在1998年提出了PageRank的概念[Page, et al., 1998]。在他们提出这个概念时，一方面，万维网的发展正处于信息大爆炸的时期。当时估计大约有1亿5千万网页，而且不到一年的时间网页的数目就会翻倍；另一方面，网页质量参差不齐，例如文献Page, et al., 1998]中提到网页信息“从Joe 在哪里吃饭到信息检索的期刊”无所不包，但是实际有意义的、有价值的，以及经常被用户检索的网页规模并没有想象的大，一个基本的方法是通过为网页排名使得重要性高且有价值的网页能够被优先检索。PageRank（网页排名）就是在这样的应用背景下诞生的。

### 4.5.2 PageRank 的基本想法

直觉上说，如下3点可以认为是网页重要性的一种评价。

1. 反向链接（backlink）越多的网页越重要；
2. 反向链接的源网页质量越高，被这些高质量网页的链接指向的网页越重要；
3. 链接数越少的网页越重要。

在描述下载系统时曾提到过“随机冲浪”模型[Bernardo A. Huberman, et al., 1998]，它是PageRank的理论基础，该模型描述网民访问网页的如下行为：

1. 用户随机选择一个网页作为上网的其实网页。
2. 看完这个网页后从该网页内所含的超链接中随机选择一个继续浏览；
3. 沿着超链浏览，直到对某个主题厌倦而重新随机选择另一个网页浏览。如此反复，直到结束。

### 4.5.3 PageRank 的计算公式

图4-17 为文献[Page, et al.,1998] 的原图。图中我们看到一个直观并简单的计算方法。以最左上角的网页为例，这个网页被3个箭头指向，假定共计得到了100分。然后通过两个链接（小横线表示网页内部的链接，方块表示一个网页）平均将这10分均分给了它所指向的两个网页。平均的意义在于假定用户是随机冲浪的，也就是点击第1个链接和第2个链接的可能性相等。这种由于指向关系而得到一定的分数，最后通过计算每个网页得到分数的多寡评价网页重要性的方法，就是PageRank的基本设计想法，也称作链接分析（link analyze）。



图4-17 文献[Page, et al.,1998] 的原图

如果分数被这样不断地传递下去，因为万维网彼此相连，这样的计算会没完没了吗？或者说PageRank的计算会收敛吗？

图4-18 所示为对三个网页分别计算他们PageRank的一个迭代收敛的例子。



图4-18 PageRank的迭代收敛的例子

图4-18中，无论怎样迭代的计算，不难发现，每个网页的得分是固定不变的。网页A的分数平均分给了网页B和网页C，网页B有完全给了C，最后网页C把得到的分数又还给了网页A。可以看出这三个网页得到的分数值和总是1，而且无论如何循环计算，这种分数分布不再发生变化。这种特性在数学上称为不变分布，这也是PageRank计算收敛的最终奥秘。当然PageRank实际的计算公式考虑了各种因素，使得这种不变分布必然存在，这些数学上的原理，请读者参阅相关随机过程的书籍货轮文娱一头车辆，下面我们将介绍PageRank的计算公式。

PageRank采用以下公式计算：[Sergey Brin 1998]



说明如下：

（1）：网页A的PageRank值。

（2）：网页存在指向A的链接，网页在上一次迭代时的PageRank值。

（3）：网页的外链数量。

（4）d：阻尼系数，0<d<1，表示在随机冲浪模型中网页将自身的d的份额的PageRank值平均分给每个外联。由于网页指向网页A，因此网页A获得来自网页的分之一的PageRank值。

从整体上来看这个公式，一个网页A的PageRank值，由两方面得分决定，其比例为1-d和d。

一方面，任何网页都有可能跳转到网页A来（不通过链接方式）。假定全集为m个网页，那么其中任意一个网页就有m个可能的无条件跳转的去处。每种跳转的概率相等，均为1/m。因此网页A得到了每个网页的1/m的(1-d)的分数，每个网页的分数初始值都为1。每次分数不论如何流动，其总值都为n。因此=1-d的分数被分配到了网页A上，PageRank中1-d的分数也可以理解为一个网页的基本分数，这是在整个计算中固定不变的常数。

另一方面，又有有些网页存在明确的指向网页A的链接，因此的分数被分配给了网页A。

因此对于公式



第1个部分可以认为是固有得分；第2个部分可以认为是因为被其他网页指向而得到其他网页的一部得分。

在数学上可以证明，PageRank由两方面因素共同决定的计算方法可以避免Dangling page 和 Page sink 的问题[Langville et al 2003]。

PageRank 的计算涉及随机过程方面的数学原理，接下来通过一系列的例子来深入理解。假定存在如图4-19所示的简单的网页链接关系。



图4-19 简单的网页链接关系

由PageRank的计算方法得到下列方程组，假定d=0.5：

PR(A) = 0.5 + 0.5 × PR(C)

PR(B) = 0.5 + 0.5 × ( PR(A) /2 )

PR(C) = 0.5 + 0.5 × (PR(A) /2 + PR(B) )

解这个三元方程组，得到：

PR(A) = 14/13 = 1.0769

PR(B) = 10/13 = 0.76923

PR(C) = 15/13 = 1.1538

从最后的PageRank 值可以清楚地看出，在这3个网页中，网页C的重要性最高，其次是网页A，最后是网页 B。这也是合理的，网页C被网页A和网页B“认可”，网页A被更高级别的网页C“认可”，而网页B被网页A“认可”。虽然网页A和网页B都只有一个Backlink，由于A的Backlink来自重要性高的C，因此最终网页A的重要性大于网页B。

此外，PageRank 还可以通过迭代计算的方法得到，计算过程如表4-2所示。

表4-2 PageRank的迭代计算过程

|  |  |  |  |
| --- | --- | --- | --- |
| 迭代次数 | PR(A) | PR(B) | PR(C) |
| 0 | 1 | 2 | 3 |
| 1 | 1 | 0.75 | 1.25 |
| 2 | 1.125 | 0.75 | 1.125 |
| 3 | 1.0625 | 0.78125 | 1.15625 |
| 4 | 1.07812 | 0.765625 | 1.15625 |
| 5 | 1.07812 | 0.769531 | 1.15234 |
| 6 | 1.07617 | 0.769531 | 1.1543 |
| 7 | 1.07715 | 0.769043 | 1.15381 |
| 8 | 1.0769 | 0.769287 | 1.15381 |
| 9 | 1.0769 | 0.769226 | 1.15387 |
| 10 | 1.07693 | 0.769226 | 1.15384 |

这里初始系向量为，表示每个网页的初始PageRank值均为1。到达第9此迭代时，迭代结果基本开始稳定。

当网页数量很大，或链接关系复杂是，这种迭代的方法是低效的。

### 4.5.4 PageRank 的计算方法

在实践中，采用幂法（Power method）来计算PageRank。

PageRank的这个经典公式：



可以转化为求的值，其中矩阵为（A也被称为“Google matrix”）。D为阻尼系数，P为概率转移矩阵，为n维的全1行，m为网页个数。X是各个网页的初始PageRank值，初始每个网页的PageRank值均为1，因此e/m矩阵在每次迭代时与全部网页的PageRank向量乘积总是保持一个n维的全1-d的向量。这可以理解为总能够从其他网页得到1-d的分数，幂法计算过程的伪码如下。

（1）x ← 任意一个初始向量；

（2）r ← Ax；

（3）if(||x-r||) < ε，返回r；

（4）else x ε← r, goto （2）。

使用图4-19的例子来理解上面的计算过程。

（1）P概率转移矩阵的计算过程。

初始化一个矩阵，若一个网页i存在一个指向网页j的链接，则 = 1，否则 =0，在图4-19的例子中，，然后将每一行除以该行非零数字之和。例如矩阵的第一行除以该行非零数字值和等于，每一行均按照这样的方法处理得到如下矩阵：



这样的矩阵也称为“马尔可夫转移矩阵”（markov transmit matrix），它记录了每个网页跳转到其他网页的概率。对于第一行，网页A跳转到网页B和网页C的概率各为1/2。通常使用其转置矩阵进行计算，即：



这时即



公式中的概率转移矩阵P。

（2）A矩阵的计算过程。

由上面的计算结果，，且易知，不妨设d = 0.5，这样，初始每个网页的PageRank值均为1，即。

（3）循环迭代计算PageRank过程。

第1步

，

由于x和r的差别较大，所以r的值赋给x，继续计算。

第2步

，反复迭代，最终的结果依次为，，，…，和。

在最后两次的结果近似或相同的情况下迭代结束，这个结果和解方程、简单迭代的方法求出的结果是一致的。用幂法计算PageRank总是收敛的，即总会在有限的迭代计算轮次内结束。

参考文献[Langville et al 2003] [HAVELIWALA ET AL 2003] [Lars Elden 2003] 可以进一步深入理解PageRank计算过程收敛的奥秘，以及参数d是如何影响收敛速度的基本原理。

在实际应用中，计算PageRank的过程要比想象得更为复杂。例如从提高收敛效率角度，加快计算速度；采用将Google matrix（PageRank 这个概率转移矩阵学术上也称作Google matrix）分块的思想执行并行计算减少磁盘I/O，从而发挥多CPU的作用，有兴趣的读者可以进一步探索。

至此，我们结束了分析系统的全部旅程，为了更好地从全局理解分析系统各个模块的工作关系，下一节中将和大家一起从宏观全局的视角重新回顾分析系统。

## 4.6 分析系统结构图

分析系统在搜索引擎的架构中承担了网页结构化、网页消重、文本分词及PageRRank计算4项基本任务。通过前面的学习最后通过一个分析系统结构图来全面了解分析系统的运作方式。分析系统结构图如图4-20所示。



图4-20 分析系统结构图

其中Page库是下载系统通过爬虫下载到的原始网页，分析系统通过以下步骤完成对这些网页的分析工作。

（1）经过一个网页结构化的过程，包括建立标签书并从网页中抽取有价值的信息，完成从原始网页打包成一个网页对象的过程。

（2）网页消重模块去除冗余的页面，相似或相同的网页仅保留一个传给分词模块。

（3）文本分词模块将正文切分成以词汇为单位的集合。

（4）将分析的结构发往索引模块，进行索引入库。

以上4个步骤中网页结构化、消重、分词这三项工作同步计算，因此速度非常快。仅PageRank的计算非常耗时，而且必须积累一定数据后才能生成一次数据（这里信息抽取过程中得到的网页链接信息发往PageRank计算服务器 ）。由于执行一次PageRank的计算代价极大，因此采用离线计算方法。离线计算的结果是一个PageRank列表，其中包含每个网页的一个PageRank值。该值越高，网页的重要性越高，在检索时就越容易被检索到。在索引系统中会利用该PageRank值对文档列表索引项的先后顺序产生影响，也就是越重要的网页在索引中越能够占据有利的位置。

回顾本章，分析系统中，除了第1步分析工作是为本模块服务，其余的分析工作包括网页消重、分词及PageRank的计算都是为索引系统和查询系统服务，因此只有在深入了解索引和查询系统的原理后我们才能深刻理解这些分析系统的工作的重大意义。接下来我们将继续走进搜索引擎的索引系统中，领略索引系统的魅力。

## 参考文献

# 第5章 搜索引擎的索引系统

## 5.1 知识准备

索引系统就好像是搜索引擎的数据大本营，在这里存储了并索引了数以亿计的网页。由于需要支持多用户同时检索，索引系统还必须提供低于秒级的检索时间，因此“存得下”和“查得快”是本节的重要话题。

先了解一些基本概念。

**信息**

信息是能够被传达和理解的消息，是将通过学习和经历获得的知识，是用来做出判断的一组事实[WordNet]，不同的角度上对信息具有不同的解释。这里我们认为信息就是结构化的网页数据，及一组有价值的数据的集合。

**索引**

索引可以说是信息的信息，或者说是描述信息的信息。例如，树中包含的目录。其中每一条目就是一个索引，用来标识某个章节的位置和主题，帮助读者快速浏览。索引就是这样一种短小精炼的检索信息的信息。

**倒排索引、倒排表、临时倒排文件、最终倒排文件**

倒排表是指存放在内存中的能够追加倒排记录的倒排索引。倒排表是迷你的倒排索引。

临时倒排文件是指存放在磁盘中，以文件的形式存储的不能够追加倒排记录的倒排索引。临时倒排文件是中等规模的倒排索引。

最终倒排文件是指由存放在磁盘中，以文件的形式存储的临时文件归并得到的倒排索引。最终倒排文件试驾大规模的倒排索引。

倒排索引作为抽象概念，而倒排表、临时倒排文件、最终倒排文件是倒排索引的三种不同的表现形式。

## 5.2 全文检索

全文检索（full-text retrieval）技术的出现是信息检索领域的一场革命，它细化了信息检索的力度，提供了实现多角度，多侧面且全新的信息检索体验。因此搜索引擎全面采用了这种崭新的技术，并使之成为主流的检索方法。

### 5.2.1 全文检索

早期的信息检索主要通过检索数据信息的外部特征，例如标题、作者、照耀、附录及资料的编号等。这样的检索系统常见于图书馆的馆藏图书检索，它主要存在如下两个大问题。

（1）检索结果排序不理想。

（2）只能对标题进行检索。

出现这些问题是因为没有考虑到文档内容（本章使用文档笼统地代表书籍或者网页）。全文检索顾名思义，是对文档的全部信息进行检索，这些信息包括标题和正文等。简单地说，全文检索的内在本质归纳起来就是如下两条。

（1）文档的全部文字参与索引。

（2）检索结果能够提供检索词出现的实际位置。

在全文检索的过程中，只需要用户提供一个或多个检索关键词（以下简称“关键词”）。不仅能够检索出命中的文本，还能够提供这些关键词在文本中出现的位置。受到搜索引擎检索结果的展示窗口的限制，不可能把全部关键词出现的位置一一列出，最终的展示效果如图5-1所示。



图5-1关键词位置的展示方式

其中椭圆矩形表示网民向搜索引擎提交的关键词，矩形框表示关键词在文档中的位置。这里搜索引擎采用一种称为“标红”的技术用红色标出关键词，其余文字为黑色，这样的方法比用数字标出位置信息更加直观。

## 5.3 文档编号

一个唯一的编号是描述一个复杂事物最好的方法，每一个文档赋予一个唯一的编号后，这个编号就能代表这个文档。

### 5.3.1 编号的本质

搜索引擎需要处理的信息是海量的，为了能够快速检索信息，通常的做法是会先为信息创建索引，索引也是信息，是用来查找信息的信息。对于搜索引擎来说，网页是信息，关键词就是索引。文档千差万别，而关键词总是有限的，因此索引相对于信息来说是轻量级的。检索在索引上完成后，得到一组匹配关键词的文档编号。继而将这些文档编号在文档库中取出来，通过一系列计算展示出来，编号就是这样在检索中发挥了重要的作用。

### 5.3.2 文档编号的方法

文档编号需要满足下面3个条件。

（1）任何一个文档在其生命周期中仅有一个编号。

（2）任何两个不同的文档的编号不同。

（2）编号在计算中尽可能高校，并且为了便于存储，要越短越好。

第一个条件表明文档编号和文档内容无关，有些文档内容常常发生变化。如果编号与内容有关，编号的稳定性将不能够被满足。

由于只有URL才能够唯一的表示一个文档。通过浏览器输入一个URL只能到达唯一的网页，这首先保证了第2个条件；其次也满足了第1个条件，网页的内容变化不会影响其URL。因此URL是天然的计算文档编号的来源。

显然直接将URL直接作为编号是不合适的，一个字符就需要一个字节，不论是计算还是存储都是不经济的。一般可以采用将字符串映射为64位整数的方法。8个字节的整数，目前可以表达当前时代万维网的网页规模。对于大规模网页编号来说，8字节还是太大，通过编码的技巧可以让其变得更段。

### 5.3.3 游程编码

假定一个文档编号序列为1，17，34，69，489，512，3456，这些都是文档的内部编号。编号为升序序列。不妨假定编号为一个32为整数，上面这个文档编号序列包含7个整数，因此需要7个整形（28字节）的数组来存储。

对于这种单调递增的序列，可以将其变为差分序列，这种编码方式也称为“差分编码”（gap encoding）。通过转换得到1，17-1，34-17，69-34，489-69，512-489，3456-513，即1，16，17，35，420，23，2944.由于差分序列除第一个编号外其余都保存的是间隔距离，因此带来一个好处和一个坏处。

好处是因为存放的是相对数，所以出第1个数以外，序列中的数都变得比较小。

坏处是为了取出序列中某个特定值，需要由头至尾取出所有数据，并进行多次加法运算。一旦确实一块数据，则序列中的后续部分无法恢复。

由于序列中的数都变得很小，所以可以采用一种变长编码（Variable Byte Coding）[Wiliams 1999]的技巧。这是一种字节对齐的编码方式，即将整数转化成二进制后，以7为单位对其分段。每段段尾加一位成为8位，恰好用一个字节表示。末尾为0表示该段是最后一段，为1表示还有后续段。例如，135的二进制编码表示为1000111通过编码后为00000011 00001110。第1段（00000011）末尾的1表示该字节还有后续段，倒数第2个1表示135 二进制数据中的第1个1。第2段（00001110）末尾的0表示该段不是段首，其余7位表示135 二进制数据中的后7位（0000111）。编码过程很容易，解码的过程也相当快。

在实践中，Variable Byte Coding 不是压缩率最高的编码方式，例如其压缩率不如γcode(Elias)，但由于Variable Byte Coding具有字节对齐的优秀特性，所以解码非常快速。具体的解码过程参考下述伪码。

VariableByteCode(A){

int i = 0x01;

int v = 0;

while( (i & 0x01) == 0x1){

i ← HEAD(A);

v ← (v << 7) + ( (i>>1) & 0x7F)

}

return v;

}

在上述伪码中，HEAD(A)表示从A中读取头部一个字节，并去掉一个字节。例如，135表示为00000011 00001110，执行一次HEAD操作后，i等于00000011，00000011 00001110变成了00001110。

对于32位整数，采用Variable Byte Coding 编码方式，西哦啊与128的数可用1个字节来表示，节省了3/4的空间；而大于268 435 455（二进制1111111-1111111-1111111-1111111共24位，每位存储7位，因此需要4个字节存储268 435 455）的数则需要5个字节。这样大于268 435 455 的数用Variable Byte Coding 编码方式需要的存储空间超过了一个整型的大小，不但没有达到压缩的目的，反而增大了空间。但是由于经过排序，序列之间的差值都会保持相对较小，因此差值达到如此大的距离的概率极小。可以忽略这种情况带来的代价，整体的存储是经济的。此外，由于计算机以字节位操作单位，字节对齐的编码往往更能发挥硬件的优势，所以可变字节编码的压缩和解压缩的速度较快。

这个差分序列中的1才6，17，35，23都小于128，因此只需要1个字节。420需要两个字节，2944也需要两个字节。这样全部序列需要9个字节。和原序列需要28字节相比压缩了60%。由于这些表示文档编号的序列存放在磁盘中，因此节约存储相当于减少了磁盘I/O的次数。压缩存储是典型的以时间换空间。一般来说，I/O是索引系统的瓶颈。而且CPU和I/O之间的性能差距还在不断扩大，因此采用这种技巧带来的减少磁盘I/O次数的好处完全可以抵偿因此带来的CPU计算量增大的缺点。

文献[Witten 1994]堪称信息检索的经典书籍，其中介绍大量关于大规模数据压缩的其他方法和技巧，它们各有优劣。

解决了文档编号存储的问题后，接下来就是如何查找到这些文档。用什么方式来索引这些文档才能符合全文检索的需求呢？下一节我们将着重介绍有关倒排索引的概念。

## 4.5 倒排索引

全文检索的检索是通过关键词来进行检索，因此为关键词建立索引是很自然的。我们把按关键词创建的索引称为“倒排索引”，在这里关键词称为“索引词”，因为并不是所有的关键词都会创建索引。

### 4.5.1 经典的倒排索引

笼统地说，索引系统的索引如图5-2所示。



图5-2 索引系统的索引

索引系统中的索引[S. Brin 1998]包含了3个概念，分别是命中（Hit）、正排索引（Forward Index）和倒排索引（Inverted Index）。接下来就围绕这3个概念展开，首先理解这3个概念.Hit表示索引词在文章中的位置（position）和字体信息，这里为了突出重点，忽略Hit的影响，集中精力了解正排索引和倒排索引。

### 4.5.2 正排索引（前向索引）

正排索引也称为“前向索引”。它是创建倒排索引的基础，具有以下字段。

（1）LocalId 字段（表中简称 “Lid”）：表示一个文档的局部编号。

（2）WordId 字段：表示文档分词后的编号，也可以称为“索引词编号”。

（3）NHits 字段：表示某个索引词在文档中出现的次数。

（4）HitList 变长字段：表示某个索引词在文档中出现的位置，即相对于正文的偏移量。

正排索引如图5-3所示。



图5-3 正排索引

由于一篇文章中的某些词可能出现多次，而且位置不同，而全文检索的本质要求是把这些位置标志出来，因此HitList中的每个命中都表示索引词在文档的某个位置出现了一次，这个序列为单调递增序列。基于游程编码的方法，变升序序列为差分序列，采用前文提到的Variable Byte Coding 方法编码可以大大压缩正排索引的HitList 字段。

事实上，文档编号在不同的计算过程里，分别称为局部编号和全局编号，它们在编号长度上稍有不同，为了便于理解，本书不区别这些区别，认为LocalId 和 DocId 完全一致，都表示一个文档的唯一编号。在正排索引中LocalId 采用升序序列编号（假定编号采用自增1的方式递增），这为下面的计算创造条件。进行倒排索引的转化时，由于正排索引中Lid 天然的有序性，因此在正排索引转化为倒排索引的创建过程中，自然可以保证倒排索引中每个词汇对应的文档编号也是有序的，倒排索引将在下一节中介绍。

本质上说，正排索引以文档编号为视角看待索引词，也就是通过文档编号去找索引词。任给一个文档编号，能够知道它包含了哪些些索引词、这些索引词分别出现的次数，以及索引词出现的位置。然而全文索引是通过关键词来检索，而不是通过文档编号来检索，因此正排索引不能满足全文检索的要求。

虽然正排索引不能满足全文检索的需要，但是正排索引为创建倒排索引创造了有利条件，是计算倒排索引的不可缺少的一环。

### 4.5.3 倒排索引

倒排索引是一种以关键字和文档编号结构，并以关键字作为主键的索引结构。倒排索引分为两个部分。

（1）第1个部分：有不同索引词（index term）组成的索引表，称为“词典”（lexicon）。其中保存了各种词汇，以及这些词汇的一些统计信息（例如出现过的文档总数nDocs），这些统计信息用于各种排名算法（Ranking Algorithm）[Salton 1989;Witten 1994]。

（2）第2个部分：由每个索引词出现过的文档的集合，以及命中位置等信息构成，也称为“记录表”（posting file）或“记录列表”（posting list）。如图5-6所示。



图5-6 倒排索引

左边的表结构（词典）记录索引词Id号、匹配该索引词的文档数量，并匹配文档在记录文件内的偏移量，通过这个偏移量就可以读取记录文件对应区域的信息。例如在图5-6中，通过读取T1的偏移量x，读取在记录文件中T1命中的相关文档Doc1、Doc2和Doc3的相关信息。

右边的记录表记录了文档编号（DocId）、索引词在该文档的命中个数（NHits），以及命中位置的列表（HitList）。例如在图5-6中，记录表显示了T2关键字在Doc1中出现了3次。

在图5-6所示的倒排索引示例中，以索引词T2为例，T2在两个文档中出现。通过偏移量在记录表中找到了存放T2有关的信息，即T2匹配的Doc1和Doc2两个文档，并且在Doc1中出现了3次，位置分别为3，5，和7（途中3，2，2表示为查分序列后的实际值）。

注意到这种按照索引词组织文档的方式，在索引词WordId，其Id不会重复；而在DocId一边，由于每个文档都可能包含多个索引词，DocId的重复非常普遍，因此对DocId就需要进行大规模的压缩。

压缩编码也采用Variable Byte Coding 编码方式。首先对DocId排序，接下来将DocId的递增序列变为差分序列，最后用Variable Byte Coding编码方式进行压缩编码。例如这样的DocId序列（22，5，9，1），通过排序得到（1，5，9，22），变差分序列得到（1，4，4，13），压缩编码后得到（2，8，8，26）（对于小于128的数进行压缩编码，相当于乘以2）。

最后，在倒排索引中，按照何种顺序存放DocId更加有利于检索呢？在图5-6中，T1这个词有Doc1，Doc2，Doc3与之相匹配，也就是在这些文档中出现了T1，那么在倒排索引的记录表中，那个文档编号先存放，那个文档编号后存放呢？这种存放顺序的策略大致有如下3种.

（1）按照DocId升序存放。

（2）按照索引词在文档中出现次数降序存放。

（3）记录表分块存放，块内按DocId升序存放，块间按PageRank值降序存放。

对于方案1来说，它有助于在多关键词查询中，得到相同的DocId（在第六章中介绍查询系统时，会提到因为DocId有序而为查询带来的好处），并且能够对DocId进行压缩编码，同时降低磁盘I/O的开销。

对于方案2来说，因为索引词出现次数越多的文档与查询关键词的相关性越好，检索就是检索那些与检索词相关性高的文档，所以这种排序是很自然的。文献[S. Brin 1998]阐述了一种折中的设计方案，即不同的索引词命中区分对待，对于索引词在标题或者锚文本命中的文档，以及命中信息存放在记录表中，不妨称为“表A”，表内数据按DocId排序；命中其他位置（如正文）的文档及其命中信息存放在另一个记录表中，称为“表B”，同样表内数据也是按照DocId排序。这样对于每个索引词的查询，优先在表A查询，只有当结果数不够是，再到表B查询，这样既照顾到了压缩存储的需要，也照顾到了相关性，通常标题中的词大多在正文中会多次出现。当然这种这种的方法有时也不够合理，它过分倚重锚文本的关键词，常常被作弊者利用。

对于方案3来说，一方面照顾到了方案1的压缩功能（DocId按升序存储），另一方面照顾了重要的文档在检索过程中优先被检索的需要，而且防止了方案2中有网页设计者作弊可能带来的麻烦。有些网页设计者在网页中标题，锚文本中堆砌大量经常被查询的重要关键词，因此方案2在设计上不可避免的缺陷容易被利用。当然PageRank算法也会被作弊者利用，然而PageRank的作弊较为困难，所以方案3是较为理想的解决方案。

最后总结一下正排索引和倒排索引的关系。本质上说，存在这样两个空间，一个称为“索引词空间”，一个称为“文档空间”，正排索引可以理解成一个从文档空间到索引词空间的映射，任意一个文档对应唯一的一组索引词；而倒排索引可以理解成一个从索引词空间到文档空间的映射，任意一个索引词对应唯一的一组该索引词命中的文档。因此从文档到正排索引，进而从正排索引到倒排索引就是理顺这种关系的过程，使得给出一个索引词，就能通过倒排索引找到其命中的文档，以及位置信息。

## 5.5 数据规模的估计

在介绍索引系统主要概念后，通过索引数据规模估计的计算方法来体验索引系统设计中必须考虑的数据规模问题，我们首先从齐普夫（zipf law）法则开始说起。

### 5.5.1 齐普夫法则

齐普夫（Zipf）于1902年1月出生于一个德裔家庭（其祖父在19世纪中叶移居美国），1924年已优异成绩毕业于美国哈弗学院。1925年，齐普夫在德国波恩及柏林学习比较与文学专业。但是以其名字命名的定律却早已走出语言学，进入了信息学、计算机科学、经济学和社会学等众多研究领域，在学术界享有极高的声誉。

简单说来，齐普夫法则可以描述为第k个最经常出现的词，其词频与1/k成正比。

齐普夫是这样得到这个定律的，首先选用一个有关James Joyce Ulysses（不妨理解为一本中等篇幅的书）的用词数据，统计不同词汇的出现次数，并按照出现次数排序。他发现了一个惊人的现象，即一个词汇在词频上的排名乘以其出现的次数惊人地接近于一个常数，而这个常数即书的实际总词数260 430。即有如下公式：



其中occurence(T)表示词T的出现次数，N是总词数，rank(T)表示此T在全部出现的词汇中词频排序，C为一个常数。齐普夫在[zipf 1949]中提出这是由于语言上的省力原则起作用的结果。有兴趣的读者可以进一步阅读文献[姜望琪 2005]了解省力原则的社会学原理。

### 5.5.2 检索模型下的索引规模估计

通过学习齐普夫法则，接下来利用这个法则粗略地估计布尔检索模型下的索引规模，从实践的角度体验一下索引的规模[Stanford IR]。

首先，由齐普夫负责，第i个最经常使用词汇的频率和1/I 成正比，因此第i个最经常使用的词的频率为c/i。

接着，假定全部汉语词汇数目为50万，因此有。这不难理解，在全世界所有中文书籍中取出50万个互不相同的词汇，并统计全部中文书籍的实际总词数，让你后分别计算着50万的词汇各自出现的数目，用数目除以总词数及得到词频。这50万词汇的词频总和可以看作（总次数）分母，分子（各自出现的次数）相加。分子之和恰好为总词数，因此除的结果为1。

数学上，称为“调和级数”，因此可以简化为：



调和级数的计算公式为：



因此常数C通过计算得到c=1/13。

综上，一个词频排序为i的词汇，它的实际词频大约是1/13i。这里的词频排序可以认为是海量数据中的排序，而不是局限于一本书。对于搜索引擎来说，一个词汇的词频就是它在全部网页中出现的实际次数除以全部网页实际出现的词汇数。例如一个词汇只在一个网页中出现，且出现了K次，所有网页为N个，平均每个网页长度为L个词。这样词频为K/(N×L)，文档频率为1/N，注意文档频率和词频在计算上的区别。

布尔检索模型在下一节详细介绍，这里认为关键词只记录是否出现在文档中，而不记录其出现的具体位置。即在图5-6所示的倒排索引中，不考虑NHits和HitList字段的信息。接下来对倒排索引长度的估计中，默认倒排索引不包含NHits和HitList字段。

在仅符合布尔检索的条件下，对倒排索引的长度进行粗略的估计。假定每个文档的平均词汇书为1000，则第i个最经常使用的词汇在文档中期望出现的次数为1000×1/13i=76/i。

第1个最经常使用的词汇在文档中出现了76/1=76次，从期望值的角度看，必定出现在所有的文档中，第2个最经常使用的词汇，在文档中平均出现76/2=37次，同理也必然出现在所有的文档中。因此得出这样的结论，即在字典中词频排名1～76名的词汇几乎出现在所有的文档中；词频排名77～152的词汇几乎出现在一半的文档中。这样我们得到下面的一个关系，如图5-8所示。



图5-8 不同词频词汇的文档频率关系

在图5-8中用6个文档代表全部文档，标“1”表示词汇在文档中出现。可以看出词频排名1～76的词汇出现在全部文档中，词频排名77～152的词汇出现在1/2的文档中，词频排名153～304的词汇出现在1/3的文档中。

对于50万个词汇，则有500 000/76=6500个这样的块。对于第i块，其出现在n/i个文档中。假定记录每个DocId需要k个字节，第i个会（含76个词汇）需要的空间为76kn/i，这样的块共有6500个。不妨假设有有100亿（10G）的网页，每个DocId的平均编码长度为1个字节（这里采用查分序列对DocId压缩编码，实际平均字节数略大于1）。例如，在倒排索引中对词频排名为1～76的高频索引词，其倒排索引简化为如图5-9所示。



图5-9 高频索引词在倒排索引中每个DocId的差值均为1

在图5-9中由于高频索引词（例如T1）几乎出现在所有的文档中，所以其指向的文档列表几乎为1，2，3，…，n，变为差分序列得到1，1，…，1.由于序列间距足够小，因此对于大部分高频索引词，其所对应的每个DocId在压缩后所占用的空间为1个字节。粗略地计算一下，采用Variable Byte Coding编码方式，对于小于128的数可用1个字节来表示。对于第1档高频词（共76个），文档间距均为1，因此需要1个字节存储文档间距；第2档高频词，文档间距为2。由于小于128，所以只需要1个字节存储文档间距。依次直到第127档高频词，均采用1个字节存储文档间距。这样共计76×127=9 652个高频词，均用1个字节表示。剩余的低频词出现在极少的文档中，因此可以认为平均每个文档间距的存储接近1个字节。

综上，在布尔检索模型下的倒排索引规模为：



一般情况下，我们可以得出这样的经验公式，即1MB的文档大约需要1GB的索引。为100G（10G）网页创建倒排索引（不记录关键词出现的位置信息HitList，只记录是否出现），大约需要10TB的索引存储空间。如果还需要存储位置（HitList）信息，则需要的存储代价更加巨大。一般认为包含HitList信息后，索引的大小依然是10TB这个数量级。而且为了一些安全稳定的因素，实际需要的空间会更多。通过以上计算可以得出，索引主要面临的首要问题是“存得下”，下面我们将通过一些技巧来介绍如何存下如此多的索引。

## 5.6 涉及存储规模的一些计算

前面提到对于索引目前100亿中文网页进行倒排索引需要TB级的存储空间，这么大规模的数据如何生成？在生成后如何存储？存储后如何支持高效的检索？这一连串的问题将在本节解答。

### 5.6.1 正排表和倒排表的合并

在索引系统中，通过计算不难得到如图5-10所示的正排表。



图5-10 正排表

在实际操作中，正排表不写入文件，而是保存在内存中。可以理解为内存中存储的正排索引，是正排索引的一种具体表现形式，同理，倒排表也是内存中中存储的倒排索引，也是倒排索引的一种具体表现形式。

图5-11是一个倒排表，它们如何合并呢？



图5-11 倒排表

正排表和倒排表合并的结果如图5-12所示。



图5-12 正排表和倒排表合并的结果

如图5-12所示，T1是文档Docx的一个词汇。在左边的词典中找到T1的位置，然后通过其DocList存放的信息找到正排表中对应的记录，并将这些记录的文档编号（DocId）、在文档中的命中次数（NHits），以及命中的位置列表（HitList）作为倒排表中的记录表中的一个记录。可以认为正排表和倒排表合并的过程，就是正排表中的数据追加到倒排表的过程。追加后，正排表并不保留，而倒排表在内存中存储一定的记录后，成批顺序地写入磁盘，成为临时倒排文件（本章约定，在提到正排表时，表示其存放在内存中，而特指倒排文件时，表示其存放在磁盘中），如图5-13所示。



图5-13 临时倒排文件

众所周知，由于磁盘的存储特性，所以很难在较大文件的中间追加数据。追加数据就不得不进行大量的数据移动，这种开销是极大的。回到这个例子中来，如图5-13所示的临时倒排文件中无法再追加有关T1的记录。

综上所述，生成倒排文件之前，倒排表由于存放在内存中，因此可以任意追加数据。在顺序写入磁盘成为倒排文件后，倒排文件不再变化。但由于不断有新的数据需要进行索引，所以这样的临时倒排文件的数量不可避免地还会不断地增加。再倒入全部需要做索引的数据后，索引系统会有多个这样的临时倒排文件，不妨假设共有64个这样的临时倒排文件。

出于性能上的考虑，在一次检索时只需要读取一个倒排文件即可得到全部与检索词相关的索引信息，而不是一次读取全部64个临时倒排文件，然后进行结果组合。因此必须将这64个临时倒排文件归并成一个大的倒排文件，称为“最终倒排文件”，其大小略小于64个临时倒排文件之和。因为在这64个临时倒排文件中分别保留了词典信息，所以这部分冗余的数据在归并成最终倒排文件后只需要保留一份即可。

在具体实现上，有一种简单的方法，即将一个正排表对索引词排序，然后一次性或分批次写入倒排表中。在第七节会提到这个方法，我们这里先通过一个例子理解这个方法。首先正排表的结构需要进行调整，如图5-14所示。



图5-14 支持排序的正排表

这里的正排表和前面介绍的稍有不同，可以看到Lid被冗余地存放。

在内存中保留一块这样的区域存放正排表，每增加一个需要索引的文档，就在正排表中追加一条记录。当追加足够多的记录后，正排表足够大，并符合批量做倒排表的条件后，按照关键词编号（WordId）进行一次稳定排序，得到如图5-15所示的排序后的正排表，这里认为T1<T2<T3。



图5-15 排序后的正排表

在图5-15中，全部正排表记录按照WordId排序，由于采用稳定排序，所以Lid 的顺序没有被破坏，图中T2关键词对应的Lid依然是1，2，3。由于在倒排索引中WordId是顺序存放的，因此排序后的正排表可以依序逐个写入到倒排表中，如图5-16所示。



图5-16 排序后的正排表逐个写入倒排表中

通过估计，每个临时倒排文件的大小约为100M，因此甚至可以不需要写入倒排表，而直接顺序写入磁盘中的临时倒排文件，这样就形成了内存中存放正排表，磁盘中放临时倒排文件的结果。

### 5.6.2 多个临时倒排文件的合并

首先考虑两个临时倒排文件的归并，通过前面数据规模估计，我们知道全部索引大小为TB量级的数据，后面将介绍倒排文件如何进行分布式存储。存储的节点数目控制在百这个数量级上，因此全部索引大小分布在100个索引节点上，平均每个索引节点大约需要存放10GB的倒排文件（降两个数量级）。如果每个索引节点存放64个临时倒排文件，这样每个临时倒排文件约100MB（降两个数量级），64个100MB大小的临时倒排文件最终归并成一个10GB大小的最终倒排文件是十分有挑战性的。每个临时倒排文件内部字典中词汇的编号是有序的，每个词汇对应的记录表（posting list）中的DocId是有序的，每个HitList中也是有序的，因此归并后依然要保持这种有序性。我们将这种归并过程笼统地成为“归并排序”，很显然这种规模的归并排序在内存中无法完成，下面介绍两种解决这个问题的方法。

（1）拉链法和二路归并。

（2）拉链法和多路归并。

无论是两路归并和多路归并，拉链法都是必须的。我们首先考察拉链法和二路归并的组合，对于两个较大的文件进行归并排序，不可能将两个文件同时读入到内存中后进行排序。因此可以读取一部分归并，将结果写入磁盘，然后继续读取两个待归并文件的一部分，周而复始，直到读完某一个文件，另一个文件顺序写入结果文件。因此归并的方法如下：

（1）从头开始读取两个临时倒排文件的一小部分（例如每次读取10MB）；

（2）分别对DocId进行解压，将压缩后的差分序列还原成原始的差分序列；

（3）两个按照DocId有序的列表进行归并；

（4）归并的结果进行压缩；

（5）写入归并后的临时倒排文件1&2中。

以上方法如图5-17所示。



图5-17 Zipper方法

这种拉链法是处理大文件（Big File）的一种通用思路，即每次仅去除大文件的一部分在内存中进行计算。计算结束后存储计算结果，并释放内存。继续读取文件的下一部分到内存、计算、存储计算结果并释放内存，周而复始直到处理全部数据。

64个这样的临时倒排文件进行归并需要多少趟这样的两两归并呢？第1趟，64个100MB的临时倒排文件两两归并得到32个200MB的临时倒排文件；第2趟，32个200MB的倒排文件两两归并，得到16个400MB的临时倒排文件，直到最后剩下两个3200MB大小的临时倒排文件两两归并得到1个6.4GB大小的最终倒排文件，这样整个归并过程结束。

一般情况下，m个初始归并段采用k路归并，则归并趟数位。如果64个初始归并段采用2路归并，则需要6趟（）。由于每趟不可避免地将全部文件读入内存和写入磁盘，生成一个最终倒排文件需要大量的内存和磁盘数据的换入换出，所以整个索引制作的瓶颈主要在这里。通过下面的例子来直观地认识归并的趟数对索引的巨大影响，如图5-18所示。



图5-18 二路归并

图中假定有4个临时倒排文件，并且为100MB，在第1块和第2块进行归并时，需要读取的磁盘量为200MB，写入量也为200MB，因此第1趟的归并（第1块和第2块归并，第3块和第4块归并）共计需要的磁盘I/O数量（读写量）为800MB。不难计算，第2趟归并的磁盘读写量也必然是800MB。实际上从宏观上看，每次都要读出全部的内容，计算，然后写入全部的内容，因此每趟的读写量是全部归并段的两倍。这样62个100MB大小的临时倒排文件进行6趟归并，需要的读写数量为64×100MB×2×6，约77GB。

基于前面的分析，对于m个初始化归并段，外排时采用k路归并，则归并趟数为。显然，随着归并路数k的增大，归并的趟数将减少，趟数的降低将大大减少磁盘I/O的数量，这样很自然地想到使用多路归并的方法。对于64个临时倒排文件的归并，如果能同时归并64路（k=64），则只需要一趟归并即可。

多路归并的方法在与数据结构相关的书籍中都有介绍，通常采用“败者树”这样的数据结构，当然同时也需要采用拉链的思想。每次读取这64个临时倒排文件的一部分进行多路归并，直到处理完全部的临时倒排文件，结束归并过程，这里不再展开实现细节。

二路归并在实际处理中还有很多细节，例如每次解压出来的块中并不是全部都能够被归并写入结果倒排文件中，会有一些块尾部分，例如解压后的块1的尾部包含了编号为Tx词汇及其对应的记录表（posting list），而在解压后的块2中没有包含编号为Tx的这个词，那么这部分需要等待下一次归并才能写入磁盘。问题是一次解压多少大小才最合适，以及如何进行并发的归并操作等。

针对这些实现细节，读者可以写一些模拟程序来体验大规模数据归并的全过程。另外可以参考[Managing Gigabytes]书籍官方主页中的一些极有价值的源代码。

### 5.6.3 倒排索引分布式存储

通过前面的学习，我们知道索引数据的规模为TB级。因此将全部索引文件存放在一台主机上，不仅是不合适的，而且是不安全的。这样一旦这个倒排索引文件损坏，全部服务就会受到很大影响，因此倒排索引的分布式存储技术应运而生了。

目前倒排索引分布式主要有两种方案，这里假定分布式的方案采用多索引节点（可以理解为多主机）的方法。即把一个巨大的倒排索引文件通过一些划分方法进行切分，使得每一个索引节点上只保留倒排文件的一部分。这样一方面加快了倒排文件的创建速度，降低了倒排文件损坏带来的损失；另一方面也提高了检索的效果。

多机分布式索引一般按照文档编号（DocId）或者按照索引词编号（WordId）进行划分。按照DocId划分的结果成为“局部倒排文件爱你”（Local inverted file）；按照WordId划分的结果称为“全局倒排文件”（Global inverted file），如图5-19和图5-20所示。



图5-19 局部倒排文件



图5-20 全局倒排文件

图中每个索引节点可以理解为一台独立主机。由于索引被分布式地存储到不同的索引节点上，所以全局还是局部是相对于索引词来说的。换句话说，在图5-19中索引节点A中的倒排文件中存放了某个关键词的一部分匹配的DocId；而全局倒排文件则存放了一个关键词全部匹配的DocId。为了便于表述，以下称这两种方案分别为“局部方案”和“全局方案”。

对于局部方案，索引节点按DocId的不同将倒排文件分布式地存储在不同的索引节点上。每个索引节点负责对一个DocId区间的文档进行索引，因此每个索引节点的倒排文件中的DocId互不相同。在检索时，将检索请求广播到每个节点。每个节点分别查询，最后由检索代理合并查询结果。

对于全局方案，索引节点按索引词ID的不同将倒排文件分布式地存储在不同的索引节点上。每个索引节点负责对一个索引词ID区间的文档进行索引。因此每个索引节点的倒排文件的索引词ID互不相同，然而每个索引节点的文档可能重复。如果某一个文档的索引词足够多，以至于能够覆盖两个以上的索引词ID区间，则可能会被存放在多个索引节点上，这种重复存储不可避免。

全局方案带来的好处是如果只检索一个单词，那么只需要在一个索引节点中检索即可。在图5-20中，对于一个检索请求，发现这个检索词在索引节点B中，因此整个检索只在索引节点B中完成，这就大大节约磁盘I/O。此外，由于查询可能是不同的查询词，因而被分布在不同的索引节点上。这样并发的用户查询不需要在检索代理中排队，可以并发地查询从而提高效率。例如查询“XML”时，检索代理检测发现这个词存放在索引节点A上，查询“NJU”，检索代理发现这个词存放在索引节点B上，因此这两个检索词的查询可以做到并发查询。

文献[Melnik et al. 2000]阐明了采用局部倒排文件的方案是有利的，主要是以下两点：

（1）可靠性高；

（2）降低网络负载，提高查询效率。

首先，对于全局方案，如果某个索引节点出现故障，可能导致某些关键词无法查询；而局部方案在这种情况下（某个索引节点出现故障），最多是损失来自于一个DocId区间内的文档。对于查询效果的影响不大，因此该方案提供了足够的可靠性。

其次，对于全局方案，由于所有的查询结果来自于一个索引节点，因此检索代理要等待这个索引节点传输全部的查询结果，这是低效的。而局部方案中多个索引节点同时将查询结果并发，从而充分利用了网络带宽。有意思的是局部方案有利于并发地获取检索结果，而全局方案有利于查询并发，这一点请读者细细体会这种平衡的奥妙，因此局部方案在获取查询结果方面是有优势的。

到底哪一种方案最佳呢？在业界，一般认为局部方案的可靠性是必需的，因此主要应用了该方案；而在研究界，有研究表明[B2S Jeong et al 1995]，在多处理器多磁盘系统下，如果检索词均匀地被请求或者索引词分布偏差不大的情况下，由于避免的局部方案中检索请求必须排队的弊端，因此全局方案在性能上是最佳的。

### 5.6.4 倒排文件缓存

一般认为一个词被查询的频率与其被使用的频率相当，即频率高的词往往也是查询的热词，查询的频率依然符合齐普夫法则，即查询频率排名为i的关键词，其查询的实际频率与1/i成正比。大量的实验科学证明，在一段时间内那些有机会被检索到的检索词总是少数的，将这些少数的检索词存放在内存中可以大大降低读取磁盘中倒排文件的机会。关于倒排文件的缓存，可以参考文献[李晓明 2004]。这里只给出一个结论，如果一个索引节点需要10GB的倒排文件，那么在这个10GB的倒排文件中，只有不到20%的索引词机器记录表应该进入缓存。然而这20%的索引词占用的空间几乎是80%，即需要8GB的内存，这显然是难以实现的。因此业界采用了很多特有技术来完成这个工作，由于超出本书的范畴，所以不再深入下去。

倒排文件的缓存及第6章中提到的搜索结果页缓存的基本原理大致相当，读者可以参考第6章中的相关内容，深刻地理解在查询系统和索引系统这种缓存机制的重要性。

### 5.6.5 倒排索引词典统计信息的计算

倒排文件中的词典还需要有关每个索引词的统计系，主要是词汇出现的文档树。这些信息用在查询系统中，在下一章中会详细介绍这些统计数据是如何应用。

在索引系统中，这些关于索引词出现的文档树的统计是在查询请求发生之前预先计算好的，是倒排表的词典部分中不可分割的一部分。把做统计工作的这个模块称为“统计员”。关键词的文档频率的统计信息是全局的，因此在整个系统中仅有一个服务模块来完成这项工作。在系统结构图中，统计员的位置如图5-21所示。



图5-21 统计员的位置

如图所示，统计员把各个索引节点的词典信息综合起来做全局统计。然后将统计结果传回各自的词典中，继而保存这些全局统计信息。在下一节说明倒排索引创建过程时会详细提到关于统计信息的计算过程。

## 5.7 倒排索引文件的创建过程

倒排索引文件的创建过程更像是一个工程建设，其中大量应用了批量计算及流水计算的技巧，完成如此大规模的倒排索引文件的创建一直是搜索引擎的核心难点。

### 5.7.1 创建倒排表

首先通过一个例子完整地体会单个索引节点倒排我饿，如图5-22所示。



图5-22 单个索引节点倒排文件的创建过程

这里的网页库支持顺序访问模式（参见第3张第5节），能够顺序读取存放在其中的文档。为了简化，我们假定读取的文档按照顺序分别为文档1、文档2和文档3，其正文内容分别为 “cat dog”、”dog cat” 和 ”rat dog”。通过在内存中完成正排得到索引词出现的文档和位置信息，例如，cat(1,1) 表示在文档1的第1个位置出现 “cat” 这个索引词。接下来通过对字母排序（汉字可以按照汉字词汇编号排序），得到一个临时的按照索引词有序的结构，这有助于顺序写入各个索引词对应的记录表。在图5-22所示的倒排表达到一定大小（例如 100MB）时，将倒排表顺写入到临时倒排文件中。完成全部网页库的索引工作后，将产生的多个临时倒排文件归并为一个最终倒排文件（图5-22中忽略临时倒排文件的归并的过程）。

从计算的角度上讲，临时倒排文件创建的全过程包含了磁盘读取（loading）、计算（processing）和写入磁盘（flushing）这3个过程，磁盘读取从网页库中读取一个个的文档；计算过程包括了正排计算、排序及归并等；写入磁盘主要是写入临时倒排文件。如果采用多道并行处理，则可以大大提高索引创建的效率。

如果将整个临时倒排文件创建过程抽象为L（loading）、P（processing）和F（flushing）3个顺序的操作，即到写入临时倒排文件为止，则L操作和F操作时磁盘密集型操作；而P操作是CPU密集型操作。为了简化，采用两个线程并发操作的处理方法，如图5-23所示。



图5-23 两个线程并发操作

这里线程1首先开始执行L操作，在执行P操作时磁盘空闲，这时线程2使用磁盘执行L操作。理论上最佳的效果是无论在什么时间点上，总是一个线程占用CPU，另一个线程占用磁盘，这样相互配合可以高效地完成索引创建过程。这里还有一个问题就是一次L操作读取多少文档才是最佳的，可以使得这种合作能够最少出现相互等待的情况。对于两道线程并发，一个L操作读取的文档规模应该满足，使得P操作执行的时间与执行一次L操作的时间相当，即线程1的P操作能够和线程2的L操作重叠。当然实际处理中还有很多技巧，读者可以参阅文献[Arvind rasu et al. 2001]获取更加详细的技术细节。

### 5.7.2 计算统计信息

计算统计信息在上一节中提到，这里给出两种计算方法，两种方法各有优劣。第1种方法从排序后的正排表开始统计；第2种方法从临时倒排文件统计。分别来看这两种方法的区别。首先通过图5-24来理解地1种方法。



图5-24 基于全局索引词出现文档数统计方法

内存中经过排序的正排结果在转换为倒排表之前，发给统计员一份拷贝。统计员为每个索引节点建立一个哈希表，这个哈希表用来计数。在全部网页库中的文档被处理完后，统计员将各个哈希表中的词进行综合统计，把相应的结果发给各个索引节点。注意这里发给索引节点A的统计结果和发给索引节点B的统计结果是不同的，因为索引节点B不包含 “rat” 这个索引词，因此没有必要把 “rat” 的信息发给他。这种方法由于需要维护哈希表的代价，因此需要耗费一定的内存空间，这是其主要缺点。

第2种统计方法主要采用基于已经计算好的倒排表数据来，整个过程相对简单。相当于对各个索引节点自身的统计结果进行综合，然后回传给各个索引节点。这种方法的主要缺点是需要等待最慢的索引结点做完索引后才能开始进行计算。

在完成了创建最终倒排文件和词典后，全部倒排索引文件创建工作完毕。从某种角度上看，这些都是一种预先计算（precomputation）。这种预先计算都是在为查询时节省时间，海量数据完成一次最终倒排索引文件的制作是非常耗时的，这些尽可能预先完成的计算为查询争取了宝贵时间。

现在距离搜索越来越近了，下一章我们来到搜索引擎最直接面对用户的查询，继续了解有关搜索及查询的知识。

## 参考文献

[1] [Arvind Arasu et al. 2001] Arasu, A. and Cho, J. and Garcia-Molina, H. and Paepcke, A. and Raghavan, S. Searching the Web, ACM Transactions on Internet Technology, 42 pages.

[2] [B2S Jeong et al 1995] B2S Jeong, E Omiecinski. Inverted file partitioning schemes in multiple disk systems. IEEE Transactions on Parallel and Distributed Systems

[3] [S. Brin 1998] S. Brin and L. Page. The anatomy of a large-scale hyper textual Web search engine. Computer Networks and ISDN Systems, 30 ( 1-7 ) :107-117, 1998

[4] [Melnik et al. 2000] Melnik, S., Raghavan, S., Yang, B., and Garcia-Molina, H. 2001. Building a distributed full-text index for the web. In Proceedings of the Tenth International Conference on The World-Wide Web.

[5] [Witten 1994] I. H. Witten, A. Moffat, and T. C. Bell, Managing Gigabytes: Compressing and Indexing Documents and Images. New York, NY: Van Nostrand Reinhold, 1994

[6] [Managing Gigabytes] <http://www.cs.mu.oz.au/mg/>

[7] [Williams 1999] H. E. Williams, J. Zobel. Compressing Integers for Fast File Access. The Computer Journal, 1999, 42 ( 3 )

[8] [Zipf 1949] Zipf G K. Human Behavior and the Principle of Least Effort. New York: Addion2Wesley, 1949

[9] [Stanford IR] <http://www.stanford.edu/class/cs276a/>

[10] [李晓明 2004] 李晓明, 闫宏飞, 王继民, “搜索引擎——原理、技术与系统”, 科学出版社

[11] [姜望琪 2005] 姜望琪 “Zipf 与省力原则[J]”, 同济大学学报:社科版, 2005,16 ( 1 ) :87-94

# 第6章 搜索引擎的查询系统

## 6.1 知识准备

在搜索引擎的4大系统中，第4个系统称为“查询系统”。查询系统直接面对用户，在接受用户的查询请求后，通过检索、排序及摘要提取等计算，将计算结果组织成搜索结果也返回给用户。整个查询过程不仅要快，而且必须能够提供用户满意的查询结果。文献[S. Brin 1998] 提到 “The goal of searching is to provide quality search results efficiently.” 因此，本章从效率和效果这两个角度进行探讨，首先一起来了解一些关于查询系统的常用术语。

### 6.1.1 什么是信息熵

信息是个很抽象的概念，直到1948年，香农提出了“信息熵”的概念才解决了信息的量化问题。

回忆一下数据结构中介绍过的哈夫曼编码，该编码通过计算不同词汇的词频并依据大小关系构建哈夫曼殊，通过哈夫曼殊为不同词频的词汇创建不同长度的前缀编码。从直观上看，高词频词汇编码较短，低词频词汇编码较长，这些编码都是0和1组成的比特传。直观的感觉还是难以解释事实的真相，下面通过一个例子来揭开编码长度和概率之间的联系。

假定红军和蓝军进行战术演习，红军打算左右夹攻蓝军，那么为了使得红军的兄弟部队能够相互通信，需要事先商量进攻的口令，并且由此确定是否同时发起攻击，取得更大的战果。假定有如下3类通信口令：

（1）如果太阳围绕地球转，就发动攻击。

（2）要么在白天攻击，要么在黑夜攻击。

（3）在0点～6点或者6点～12点或者12点～18点或者18点～24点攻击。

不考虑信息安全的情况下，红军的两支部队需要如何准备这些消息编码呢？显然编码越短，越有利于战场恶劣的条件，并节约通信成本。下面就对这3种情况分别进行分析。

对于第1种情况，红军两支部队不需要任何通信。因为太阳围绕地球转，这是肯定的。通信代价为0，这样无论两支部队选择何时攻击都是合理的。

对于第2种情况，红军两支部队需要约定通信方式。假定为红A部通过网络传送一个比特0，表示在白天发动攻击；传送一个比特1，表示在黑夜发动攻击。显然，此时的通信代价为1个比特，1个比特能够表达两种可能性。由于红军两支部队保证在白天或者黑夜攻击，而不会出现一支部队选择白天，一支部队选择黑夜进行攻击的可能性。因此这种信息更加有价值，然而也付出了1个比特的通信开销，不妨认为这个信息“值”1个比特。

对于第3种情况，虽然相对复杂，但也可以以如下4种编码方式约定口令：

（1）00：在0点～6点攻击；

（2）01：在6点～12点攻击；

（3）10：在12点～18点攻击；

（4）11：在18点～24点攻击。

此时通信成本为两个比特，两个比特能够表达4种可能。这样红军两支部队的进攻同步性更强，其先后进攻的时间最多差6个小时，很显然这种信息比第二种更有价值。同样道理，认为这个信息“值”两个比特。

综合上述3种情况的分析，从直观上，信息包含的情况越多，信息越多，需要的通信代价就越大。信息是否有价值隐约地和概率有着密不可分的关系，那么如何衡量发起攻击这个信息，如何从直观跨越到客观呢？

1948年，香农长达数十页的论文“通信的数学理论”成为信息论正式诞生的里程碑，在其通信数学模型中清楚地提出信息的质量问题，得到了如下著名的计算信息熵（Entropy）公式：



对于上例中的第3种情况，假定每一种攻击可能均等，均为四分之一，那么何时发动攻击这样一个随机变量X包含4个随机事件，概率分别表示为：

P（X=“在0点～6点攻击”）=

P（X=“在6点～12点攻击”）=

P（X=“在12点～18点攻击”）=

P（X=“在18点～24点攻击”）=

那么



与前面计算的结果2个比特一致。

信息熵H（X）在信息论中称为消息X的“熵”，其含义是信息集X发出任意一个随机事件的平均信息量。“熵”值H（X）说明了消息集X的每个事件的平均存储的位数，即用多少个二进制表示一个消息。在约定进攻口令的这个例子中，4个口令中发生一个口令平均需要的通信或者存储代价（平均信息量）为两个比特。

香农通过“熵”阐明了概率与信息的关系，即变量的不确定性越大，熵也就越大，将其搞清楚所需要的信息量也就越大。信息熵是一个十分重要的概念，下面在介绍经典的TF/IDF方法时还将就此问题继续展开。

### 6.1.2 检索和查询的区别

本章约定对于查询来说，适用于真实用户进行的一次查询是相对于搜索引擎而言的；对于检索来说，适用于检索代理对索引库进行的一次检索是相对于搜索引擎索引系统而言的。查询的结果是搜索结果网页，检索的结果是与查询词相关的文档列表（doclist）。

### 6.1.3 检索词和查询词的区别

严格意义上，普通用户提交给查询系统的关键词称为“查询词”，经过查询系统分词，提交检索代理的称为“检索词”。例如用户提交查询词为“清华大学图书馆”，通过分词，提交给检索代理变成“清华大学”和“图书馆”两个检索词。为了简化，本章并不区分查询词和检索词，而统一使用查询词这个术语。

### 6.1.4 自动文本摘要（Automatic Text Summarization）

自动文本摘要简称“自动摘要”，是从文档中自动提取出一个正文片段。用户仅仅需要浏览整个正文片段就能够了解文档中与查询词相关的部分，进而判断是否值得详细阅读整篇文档。

## 6.2 网页信息检索

网页信息检索的数据源来自于网页索引库（在前一章中介绍了网页对象被索引入库的全过程），网页信息检索输出的是一组文档编号，这些被编号的文档都是索引库中包含查询词的文档。

### 6.2.1 早期的检索模型

早期的检索模型是一种称为“布尔模型”（Boolean Models）的检索模型。布尔模型也称为“集合模型”，是一种采用 AND、OR 及 NOT 等逻辑运算符将多个查询词连成一个逻辑表达式，继而通过布尔运算进行检索的简单匹配模型。例如查询词为“走进搜索引擎 检索模型 – 搜索”，将会被翻译成“走进搜索引擎 AND 检索模型 NOT 搜索”这样的逻辑语言。按照自然语言的翻译，这个逻辑语言表示包含“走进搜索引擎”且包含“检索模型”却不包含“搜索”的文档集合。对于查询系统来说，这样的查询词表示用户请求检索包含“走进搜索引擎”且包含“检索模型”，却不包含“搜索”的文档集合。

布尔模型的这种检索易于实现，检索速度快。但是由于没有考虑文档和查询词的相关性问题，没有区分查询词的权重问题。因此在“效率”和“效果”的两难选择上放弃了“效果”，而仅仅考虑了“效率”。

此外，如果查询词中有一个关键词没有包含，这个文档就不会出现在结果中，而实际上这个文档可能正是用户需要的。在布尔模型中很难进行相关性强弱的度量，它只解决“有”还是“没有”的问题，不解决“好”还是“不好”的问题。

布尔模型归纳起来有如下两大优点：

（1）表达简单且易于实现。在关键词检索的过程中，把检索计算转变为集合运算，特别是集合间的求交集运算和集合间的差运算。

（2）检索速度快。布尔模型的计算主要是集合求交运算，这将在下一节中介绍。

正是由于布尔模型的两个优点造成了布尔模型的如下两大不足：

（1）如果有一个查询词没有被包含，则检索失败。

（2）检索出来的结果很难进行相关性排序。由于布尔模型计算简单，网页库中的文档与查询词的相关性没有计算，从而无法了解哪个文档更加符合用户的查询意图（通常认为符合用户查询意图的文档在搜索结果中应排名靠前）。

布尔模型的不足主要由于没有考虑到关键词在查询中哦你，这一点不足在向量空间模型（Vector Space Models）中得到部分解决。

### 6.2.2 向量空间模型（Vector Space Models）

和布尔模型不同，向量空间模型主要关心的是“效果”，而非“效率”。向量空间模型提出了将查询词和文档按照关键词的维度分别向量化，然后通过计算这两个向量间夹角余弦的方法得到文档与查询词的相似度。从而优先检索那些和查询词相似度大的文档，并且能够对检索出的文档按照与查询词的相似度进行排序。向量空间检索模型的计算方法如图6-1所示。



图6-1 向量空间检索模型的计算方法

在向量空间检索模型中，通过下面三个步骤进行检索：

（1）把原始查询和文档都看作是文本，使用同样的向量化过程分别得到查询向量和文档向量。

（2）通过计算向量相似度的方法计算原始查询和文档的相似度。

（3）按照与查询词的相似度从大到小排序文档，返回给用户。

通过下面的例子来理解向量化的过程。假定汉语的词汇表只有“走进”、“搜索引擎”和“学习”这3个词（实际上，常用的汉语词汇过万），那么这3个词组成的向量空间就是我们熟悉的三维空间，如图6-2所示。



图6-2 三维空间

在图中，对于“走进搜索引擎，学习搜苏引擎”这个句子，通过计算每个词汇的出现的次数，得到这样的统计信息，即“搜索引擎”出现2次，“走进”出现1次，“学习”出现1次。将3个词的维度理解为三维空间的XYZ轴，这样“走进搜索引擎，学习搜索引擎”在词汇表构成的向量空间内表示为向量（2，1，1）。这个向量的3个分量的意义可以理解为对3个轴的投影分别是2，1，1，物理含义为这些关键词在查询句子中分别出现的次数，同时注意这里的向量的方向性用箭头表示。

现在我们扩展到四维空间上理解，假定词汇表中还包括了“检索模型”一词，这样对于“走进搜索引擎，学习搜索引擎”这个句子进行向量化的结果可能是（2，1，1，0），其中四维空间的第四维表示“检索模型”。由于这个句子中没有出现“检索模型”，因此它在这个关键词维度上的投影为0。

据统计常用汉语词汇大约5 000条，如果用这 5 000 维的词向量空间表示这个句子，可能是这样的形式，即(0, 0, …, 0, 2, 0, ...1, …1, …）。其中“0”的分量表示句子在这个词汇上的投影是0，或者说句子中没有出现这个关键词。由于句子中只出现3个词，因此在向量中只有3个分量为有效的非0值。可见在实际的计算中，向量通常都是十分稀疏的。

向量化的过程就是对一个文档按照关键词的维度，每个向量的分量可以理解为包含这个词的权重（weight）。出现次数多的词权重就较大，对向量方向的影响力也较大。为了使不同文档和查询词相关性具有比较性（相关性排序的需要），保证对打文档和小文档做到公平，还需要对关键词的出现次数做归一化的工作，即转化为词频（词数/总词数）作为向量的分量。因此在上面例子中，“走进搜索引擎，学习搜索引擎”中的“走进”的词频为1/4，“搜索引擎”的词频为2/4，“学习”的词频为1/4。因此在如上图所示的关键词向量空间下，这个查询词被向量化为 (2/4, 1/4, 1/4)。

事实上，向量中每个分量同除以相同的数不会改变向量的方向，但是会改变向量的距离。因此在只考虑向量方向，而不考虑向量长度的情况下，没有必要使用词频作为向量的分量，这样反而引入浮点计算的麻烦。考虑到其他可能须需要进行向量距离运算的场合，以及为下一小节中的TF/IDF 的权重量化计算做准备，提前了解词频的有关概念，并使用关键词词频作为向量的分量表示。

在向量化的工作完成后（下一节将提到实际上采用经典的TF/IDF方法进行向量化的工作），就需要解决计算文档和查询词相似度的问题。向量空间模型中一般采用向量之间的夹角余弦值作为向量是否相似的度量依据。

向量间的夹角余弦的计算公式：



其中a,b表示向量，表示向量的点成，|a|表示向量的模。

通过一个具体的例子理解这个计算过程。

假定在一个7个关键词的向量空间下，一个查询词向量化为 a(0, 0, 2, 0, 1, 0, 1)，一个文档向量化为 b(0, 1, 3, 5, 2, 4, 0)，夹角预先计算如下：



= 0×0 + 0×1 + 2×3 + 0×5 + 1×2 + 0×4 + 1×0

= 8







这样查询词a和文档b的相关性就转化为0.44这个具体的数值上，使得相似性成为可以量化的概念，因此相似性量化的结果成为“相似度”。

实际计算中，如果向量a表示查询向量，向量b表示文档向量，在计算查询向量和一组文档向量的相似度时，查询向量总是不变的，或者说对每个文档向量来说查询向量都是相同的。因此相似度计算中是否除以|a|，对将来进行的相似度排序没有影响，可以作为公共因子消失。计算相似度实际只需要计算 a×cosθ 即可，方法如下：



其中每个文档向量的模可以预先计算并保存，而不需要每次查询都执行一次文档向量的模运算。这样，每次求相似度只需要一次向量点乘和除法计算即可。

对于两个高维稀疏向量（由于汉语词汇众多，实际向量化后的向量维数高，非0值少），向量的表示和向量点乘的计算也是需要一定的技巧。可以采用哈希表的方法快速找到两个向量相同分量的非0值进行计算，这里不再详细展开。

为了简化表述，在此前提到的关键词量化过程中采用词频作为向量化中每个向量的分量，而事实上却采用了经典的TF/IDF方法为每个关键词进行更加合理的量化。下面我们将走进经典的TF/IDF方法，领略信息检索的精髓。

### 6.2.3 关键词权重的量化方法 TF/IDF

在实际的查询词及文档向量化应用中仅仅使用词频作为分量是不够的，接下来将介绍搜索引擎中经典的TF/IDF 权重计算方法。

首先，我们来继续本章开头提到的信息熵这个概念深入理解。下面我们科学地给出“自信息”和“熵”的概念。

**自信息**：任意随机事件的自信息量定义为该事件发生概率的对数的负值，设事件 x 的概率为 p(x)， 那么其自信息定义为：



自信息也可以理解为某个概率的事件进行编码需要的最小编码长度。

**熵**：在信息论中自信息量是一个随机变量，它不能用来作为整个信源的信息测度。因此我们引入平均自信息量，即熵，定义为：



Claude Shannon（香农）的源代码[Shannon, 1948] 理论指出，最理想的编码方法是词汇表中第i个词汇预期出现的概率为，那么该词汇需要分配个比特长度的编码。最佳编码符号中的比特数目表示符号的信息内容（information content），整个词汇表中的全部词汇的信息量的平均大小称为“概率分布的熵”，即：



E用比特/符号为单位表示，表示词汇表平均每个词汇需要的加权平均编码长度 [Baeza-Yates et al, 1999]。

在编码中用“熵”值衡量是否是最佳编码。若以W表示采用一种编码方式后词汇表平均编码长度，则可能情况如下：

（1）W>H(X)：有冗余，不是最佳编码；

（2）W<H(X)：不可能；

（3）W==H(X)：最佳编码（一般W稍大于H(X)）。

“熵”值是平均编码长度的下限值。数据结构中提到的哈夫曼编码就是对“熵”值的极限逼近，由于实际编码长度不能为小数，因此哈夫曼编码所能达到的压缩比距离压缩极限还有一定距离。

熵最大极限地压缩冗余的信息对于衡量关键词权重具有特殊意义，我们通过一个例子循序渐进地来理解为什么要把熵这个概念引入到关键词权重的量化计算中来。

假定一个文档就是一个信息源，这个文档包含了T1, T2, T3, …, Tn, 共n个词汇。每个词汇各出现了N1, N2, N3, …, Nn 次，其中在海量文档中出现的文档频率（词汇的发生概率）分别为D1, D2, D3, …, Dn ，那么传输一个这样的文档需要多少编码长度呢？

假定每个词汇的出现相互独立，并且不考虑出现的先后顺序，因此由这些词汇组成这篇文档的概率为：X=D1N1×D2N2 ×…DnNn（概率的乘法公式）。参考自信息的共识，对具有这种概率的事件进行编码，需要的编码长度为：

-log(X)=-log(X=D1N1×D2N2 ×…DnNn)

=-N1×logD1-N2×logD2-…Nn×logDn

因此这篇文档理论上能够最大极限地被压缩到-log(X) 个比特。数学上可以证明，这样的压缩形式是极限压缩。所有的压缩算法都参考对熵压缩的逼近程度来量化压缩能力的大小，在数据结构中学过的哈夫曼编码就是对信息熵的一种逼近编码。另外上面的公式也可以这样理解，对于关键词Ti，其文档频率为Di，这个关键词的编码长度为-logDi，在文档中该词出现了Ni次，因此总共需要的编码长度为-Ni×logDi。而全部文档需要的编码长度为，这与前面的计算结果相同。接下来我们考察该文档中每个词的平均编码长度。

很明显，平均编码长度为全部文档需要的编码长度除以总词数，即：



展开这个计算式得到：



在这个平均编码长度中，各个关键词度都做出了不同的贡献。我们将关键词在文章中的重要性量化为对平均编码长度的贡献上，不难得出这样的结论，即越是出现次数多（词频高）且罕见的词汇（文档频率低）对平均编码长度大小的贡献越大。

假定一个文档总词数为K（），对于关键词Ti来说，它对平均编码长度做出的贡献为：



进一步转化这个表达式得到：



其中，为关键词Ti在文档中的词频（TF, term frequency），为文档中关键词Ti的文档频率倒数的对数，称为“文档频率”（IDF, inverse document frequency）。上面这个平均编码长度的公式可以改写为：



显然不同关键词的TF×IDF值对编码长度做出了不同的贡献，越是出现次数多（词频高）且罕见的词汇（文档频率低）对平均编码长度的贡献大，这就是经典的TF/IDF方法。

文档频率从哪里来呢？回顾前面搜索引擎索引层的倒排索引词典统计信息中提到的文档频率的计算方法，在索引阶段这些数据已经计算完毕，因此这里可以直接使用文档频率来进行权重计算。对于“走进搜索引擎，学习搜索引擎”这个句子，假定“走进”的文档频率是1/256，“搜索引擎的文档频率为1/4096，“学习”的文档频率为1/256。那么它们各自的权重分别如下：

（1）搜索引擎：2/4 × log(4096) = 6；

（2）走进：1/4 × log(256) = 3；

（3）学习：1/4 × log(256) = 3。

这样一个句子在仅由这3个词构成的向量空间中被量化为（6, 2, 2），和前面仅使用词频作为量化标准的（2, 1, 1）相比，突出了“搜索引擎”这个关键词在句子中的重要性，使得特别是一些诸如“我们”及“他们”这样的高频词能够很好地降低权重，而提高了低频词的权重。并且使得具有不同权重的词汇在进行向量相似度计算过程中发挥不同的作用，使得量化的结果更加科学。

### 6.2.4 搜索引擎采用的检索模型

搜索引擎采用了布尔模型和向量空间模型结合的方法来进行信息检索，布尔模型的检索效率高且易于实现，向量空间模型能够提高检索的相似度，通过相似度排序的手段能够大大改善查询结果。因此搜索引擎将二者的优势相结合，完整的检索过程如图6-3所示。



图6-3 完整的检索过程

如图6-3 所示，图中方块为计算部分，斜方块为数据部分，详细的检索过程如下：

（1）对查询词进行分词，得到一个逻辑表达式。例如查询“走进搜索引擎”，将会被切分成“走进”，“搜索引擎”这两个词。并且转换为用 AND 逻辑表示的表达式，即“走进” AND “搜索引擎”。

（2）采用布尔模型的方法得到结果文档列表，例如从倒排索引中提取包含“走进”的文档列表和包含“搜索引擎”的文档列表，并将检索出的文档列表求交集，得到既包含“走进”，也包含“搜索引擎”的文档列表。

（3）将步骤（2）得到的文档列表中的全部文档和查询词分别向量话，并求向量间的相似度。

（4）按照相似度排序输出检索结果。

综上所述，全部过程包括分词、doclist求交、向量化并求向量夹角及排序这4种计算，并且这4项计算依次完成。

接下来我们介绍文档列表求交的计算方法和结果排序的技巧。

### 6.2.5 多文档列表求交计算

假设在倒排索引表中检索出包含“走进”一词的文档列表为 doclist1(1, 5, 9, 12)，表示这4个文档编号的文档含有“走进”这个词汇。同理假定包含“搜索引擎”的文档列表为 doclist2(5, 9, 7, 11)， 这样同时包含“走进”和“搜索引擎”这两个关键词的文档为doclist1∩doclist2=(5, 9)。”∩” 表示求两个集合的公共元素。

这里介绍一种多文档列表求交的方法，这种方法称为“最佳归并树算法”。基本思想是越短的文档列表越早开始参与文档列表的求交过程，越长的文档列表越要推迟计算求交的过程。每次对现存的两个最短的文档列表进行归并，之道归并全部的文档列表，每次归并后减少一个文档列表。因此如果是n个文档列表，则需要归并n-1趟。下面通过一个例子来说明这个归并过程，最后得到的一个最佳归并树能够让我们完整地理解这个方法。

假定有多个查询词检索如下的文档列表组，文档列表按照长度进行升序排序。

doclist1(2, 5, 7, 9, 10, 15, 18)

doclist2(1, 2, 3, 7, 8, 9, 10, 12, 15, 18, 22)

doclist3(1, 5, 7, 8, 9, 10, 15, 16, 18, 19, 22, 23)

doclist4(4, 5, 6, 7, 8, 9, 10, 11, 15, 16, 17, 18, 19, 20)

第1步归并最短的两个文档列表，即doclist1和doclist2。由于文档列表从倒排索引中去除，因此文档列表中包含的文档自然地按照文档编号升序排序。这样归并两个有序的文档列表非常简单且高效，计算方法如下：

（1）为doclist1和doclist2分别设置一个当前下表point1和point2并初始化为0，转（2）；

（2）如果point1>len(doclist1)或者point2>len(doclist2)，则求交结束，否则转（3）；

（3）如果doclist1[point1] == doclist2[point2]，说明这是一个公共元素，将其存放在结果中，point1++，point2++，转（2）；否则转（4）；

（4）如果doclist1[point1] > doclist2[point2]，point2++；否则point1++，转（2）。

读者不妨拿出纸笔用这个计算方法一步一步将求doclist1和doclist2的交集过程画出来，实际的文档列表求交集实例第1步如图所示。



图6-4 文档列表求交集实例第1步

首先初始化下标同时指向两个doclist的首个元素，通过比较得到doclist2[0]小于doclist1[0]，于是将point2右移动一位，如图6-5所示。



图6-5 文档列表求交集实例第2步

当前doclist1和doclist2的下标指向的值相等，均为2。因此2作为一个交集元素输出，并且将doclist1和doclist2的下标同时向有移动一位，如图6-6所示。



图6-6 文档列表求交集实例第3步

此时doclist1的当前位置的值为5，大于doclist2当前位置的值3，因此将point2右移一位，如图6-7所示。



图6-7 文档列表求交集实例第4步

接下来的计算将找到7作为一个交际元素输出，完成这个过程可以得到doclist1和doclist2的交集为( 2, 7, 9, 10, 15) 。

这种计算过程是线性的，因为每次无论何种情况，至少有一个文档列表的指针会向右移动一位，只要有一个文档列表的下标移动到文档列表尾部，整个计算结束。

此外，很容易证明计算结果的交际长度必然比doclist1和doclist2都要短，至多和最短的文档列表长度相同（当doclist2包含doclist1时，或者说当doclist1是doclist2的子集时）。由于文档列表组是按照文档列表长度升序排列，doclist1和doclist2的交集（不妨称为 “doclist1\_2”）下一次必定和doclist3求交，因为此时文档列表组中最短的两个文档列表必然是doclist1\_2和doclist3。

不难得出，最佳归并的顺序如图6-8所示。