1- 搜索引擎基础倒排索引

作者：Gao-Eason
原文链接：https://ld246.com/article/1649663246292
来源网站：链滴
许可协议：署名-相同方式共享 4.0 国际 (CC BY-SA 4.0)

Table of Contents

<!-- more -->

什么是倒排索引?

** 见其名知其意，有倒排索引，对应肯定，有正向索引。**

** 正向索引（forward index），反向索引（inverted index）更熟悉的名字是倒排索引。**

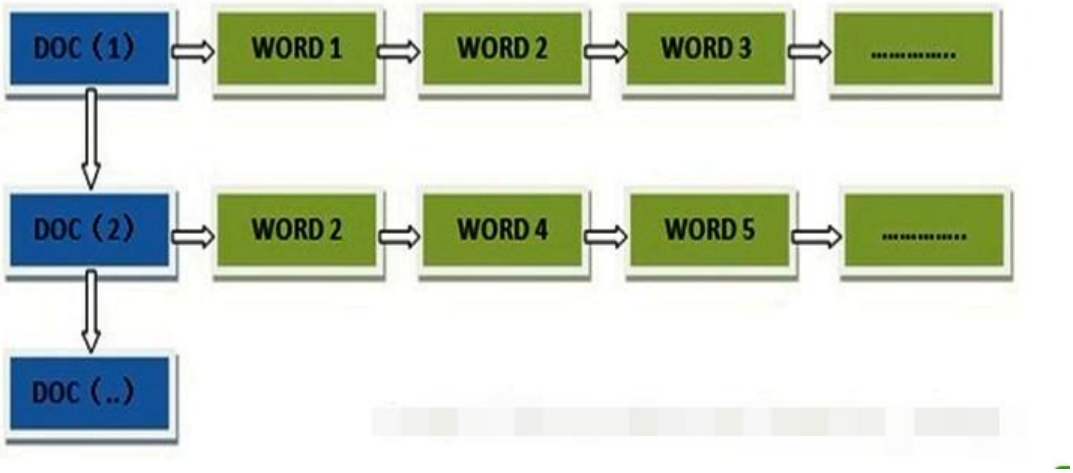
** 在搜索引擎中每个文件都对应一个文件ID，文件内容被表示为一系列关键词的集合（实际上在索引擎索引库中，关键词也已经转换为关键词ID）。例如“文档1”经过分词，提取了20个关键词，个关键词都会记录它在文档中的出现次数和出现位置。**

** 得到正向索引的结构如下：**

** “文档1”的ID > 单词1：出现次数，出现位置列表；单词2：出现次数，出现位置列表；……。**

** “文档2”的ID > 此文档出现的关键词列表。**

** **



** 一般是通过key，去找value。**

** 当用户在主页上搜索关键词“华为手机”时，假设只存在正向索引（forward index），那么就要扫描索引库中的所有文档，找出所有包含关键词“华为手机”的文档，再根据打分模型进行打分，出名次后呈现给用户。因为互联网上收录在搜索引擎中的文档的数目是个天文数字，这样的索引结构本无法满足实时返回排名结果的要求。**

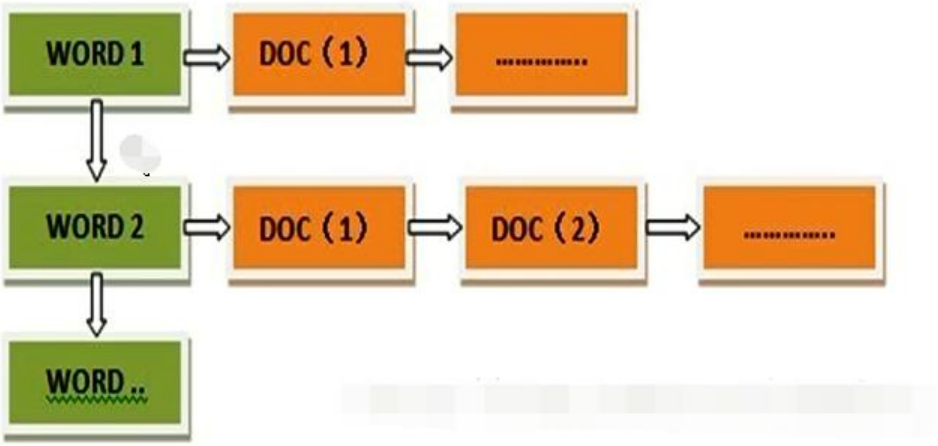
** 所以，搜索引擎会将正向索引重新构建为倒排索引，即把文件ID对应到关键词的映射转换为关词到文件ID的映射，每个关键词都对应着一系列的文件，这些文件中都出现这个关键词。**

** 得到倒排索引的结构如下：**

** “关键词1”：“文档1”的ID，“文档2”的ID，……。**

** “关键词2”：带有此关键词的文档ID列表。**

** **



** 从词的关键字，去找文档。**

1.单词——文档矩阵

** 单词-文档矩阵是表达两者之间所具有的一种包含关系的概念模型，图1展示了其含义。图3-1的列代表一个文档，每行代表一个单词，打对勾的位置代表包含关系。**

**

** 图1 单词-文档矩阵**

** 从纵向即文档这个维度来看，每列代表文档包含了哪些单词，比如文档1包含了词汇1和词汇4，不包含其它单词。从横向即单词这个维度来看，每行代表了哪些文档包含了某个单词。比如对于词汇来说，文档1和文档4中出现过单词1，而其它文档不包含词汇1。矩阵中其它的行列也可作此种解读。**

** 搜索引擎的索引其实就是实现“单词-文档矩阵”的具体**数据结构**。**可以有不同的方式来实现述概念模型，比如“倒排索引”、“签名文件”、“后缀树”等方式。但是各项实验数据表明，“倒索引”是实现单词到文档映射关系的最佳实现方式，所以本博文主要介绍“倒排索引”的技术细节。**

2.倒排索引基本概念

** 文档(Document)：一般搜索引擎的处理对象是互联网网页，而文档这个概念要更宽泛些，代以文本形式存在的存储对象，相比网页来说，涵盖更多种形式，比如Word，PDF，html，XML等不格式的文件都可以称之为文档。再比如一封邮件，一条短信，一条微博也可以称之为文档。在本书后内容，很多情况下会使用文档来表征文本信息。**

** 文档集合(Document Collection)：由若干文档构成的集合称之为文档集合。比如海量的互联网页或者说大量的电子邮件都是文档集合的具体例子。**

** 文档编号(Document ID)：在搜索引擎内部，会将文档集合内每个文档赋予一个唯一的内部编，以此编号来作为这个文档的唯一标识，这样方便内部处理，每个文档的内部编号即称之为“文档编”，后文有时会用DocID来便捷地代表文档编号。**

** 单词编号(Word ID)：与文档编号类似，搜索引擎内部以唯一的编号来表征某个单词，单词编可以作为某个单词的唯一表征。**

** 倒排索引(Inverted Index)：倒排索引是实现“单词-文档矩阵”的一种具体存储形式，通过倒索引，可以根据单词快速获取包含这个单词的文档列表。倒排索引主要由两个部分组成：“单词词典和“倒排文件”。**

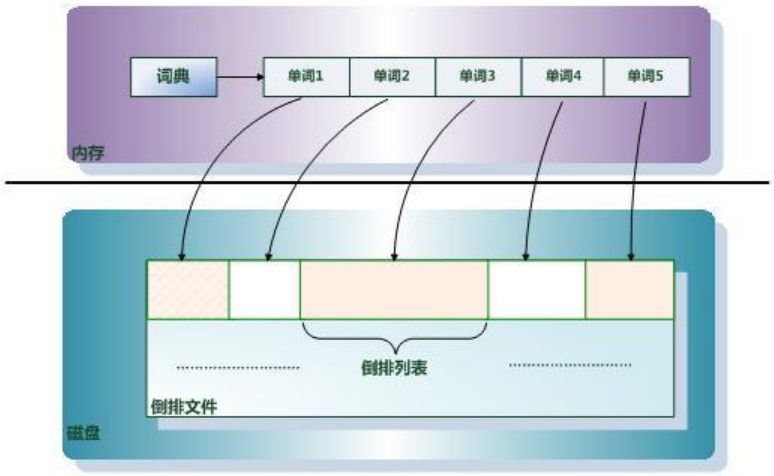
** 单词词典(Lexicon)：搜索引擎的通常索引单位是单词，单词词典是由文档集合中出现过的所有词构成的字符串集合，单词词典内每条索引项记载单词本身的一些信息以及指向“倒排列表”的指针**

** 倒排列表(PostingList)：倒排列表记载了出现过某个单词的所有文档的文档列表及单词在该文中出现的位置信息，每条记录称为一个倒排项(Posting)。根据倒排列表，即可获知哪些文档包含某个词。**

** 倒排文件(Inverted File)：所有单词的倒排列表往往顺序地存储在磁盘的某个文件里，这个文即被称之为倒排文件，倒排文件是存储倒排索引的物理文件。**

** 关于这些概念之间的关系，通过图2可以比较清晰的看出来。**

** **



3.倒排索引简单实例

** 倒排索引从逻辑结构和基本思路上来讲非常简单。下面我们通过具体实例来进行说明，使得读能够对倒排索引有一个宏观而直接的感受。**

** 假设文档集合包含五个文档，每个文档内容如图3所示，在图中最左端一栏是每个文档对应的文编号。我们的任务就是对这个文档集合建立倒排索引。**

** **

** 图3 文档集合**

** 中文和英文等语言不同，单词之间没有明确分隔符号，所以首先要用分词系统将文档自动切分单词序列。这样每个文档就转换为由单词序列构成的数据流，为了系统后续处理方便，需要对每个不的单词赋予唯一的单词编号，同时记录下哪些文档包含这个单词，在如此处理结束后，我们可以得到简单的倒排索引（参考图3-4）。在图4中，“单词ID”一栏记录了每个单词的单词编号，第二栏是对的单词，第三栏即每个单词对应的倒排列表。比如单词“谷歌”，其单词编号为1，倒排列表为{1,2,3,45}，说明文档集合中每个文档都包含了这个单词。**

** **

** 图4 简单的倒排索引**

** 之所以说图4所示倒排索引是最简单的，是因为这个索引系统只记载了哪些文档包含某个单词而事实上，索引系统还可以记录除此之外的更多信息。图5是一个相对复杂些的倒排索引，与图4的基索引系统比，在单词对应的倒排列表中不仅记录了文档编号，还记载了单词频率信息（TF），即这个词在某个文档中的出现次数，之所以要记录这个信息，是因为词频信息在搜索结果排序时，计算查询文档相似度是很重要的一个计算因子，所以将其记录在倒排列表中，以方便后续排序时进行分值计算在图5的例子里，单词“创始人”的单词编号为7，对应的倒排列表内容为：（3:1），其中的3代表文编号为3的文档包含这个单词，数字1代表词频信息，即这个单词在3号文档中只出现过1次，其它单词应的倒排列表所代表含义与此相同。**

** **

** 图 5 带有单词频率信息的倒排索引**

** 实用的倒排索引还可以记载更多的信息，图6所示索引系统除了记录文档编号和单词频率信息，额外记载了两类信息，即每个单词对应的“文档频率信息”（对应图6的第三栏）以及在倒排列表记录单词在某个文档出现的位置信息。**

** **

** 图6 带有单词频率、文档频率和出现位置信息倒排索引**

** “文档频率信息”代表了在文档集合中有多少个文档包含某个单词，之所以要记录这个信息，其因与单词频率信息一样，这个信息在搜索结果排序计算中是非常重要的一个因子。而单词在某个文档出现的位置信息并非索引系统一定要记录的，在实际的索引系统里可以包含，也可以选择不包含这个息，之所以如此，因为这个信息对于搜索系统来说并非必需的，位置信息只有在支持“短语查询”的候才能够派上用场。**

** 以单词“拉斯”为例，其单词编号为8，文档频率为2，代表整个文档集合中有两个文档包含这单词，对应的倒排列表为：{(3;1;<4>)，(5;1;<4>)},其含义为在文档3和文档5出现过这个单词，单词率都为1，单词“拉斯”在两个文档中的出现位置都是4，即文档中第四个单词是“拉斯”。**

** 图6所示倒排索引已经是一个非常完备的索引系统，实际搜索系统的索引结构基本如此，区别无是采取哪些具体的数据结构来实现上述逻辑结构。**

** 有了这个索引系统，搜索引擎可以很方便地响应用户的查询，比如用户输入查询词“Facebook，搜索系统查找倒排索引，从中可以读出包含这个单词的文档，这些文档就是提供给用户的搜索结果而利用单词频率信息、文档频率信息即可以对这些候选搜索结果进行排序，计算文档和查询的相似性按照相似性得分由高到低排序输出，此即为搜索系统的部分内部流程，具体实现方案本书第五章会做细描述。**

4.** 单词词典**

** 单词词典是倒排索引中非常重要的组成部分，它用来维护文档集合中出现过的所有单词的相关息，同时用来记载某个单词对应的倒排列表在倒排文件中的位置信息。在支持搜索时，根据用户的查词，去单词词典里查询，就能够获得相应的倒排列表，并以此作为后续排序的基础。****

**** 对于一个规模很大的文档集合来说，可能包含几十万甚至上百万的不同单词，能否快速定位个单词，这直接影响搜索时的响应速度，所以需要高效的数据结构来对单词词典进行构建和查找，常的数据结构包括哈希加链表结构和树形词典结构。****

4.1 哈希加链表

**** 图7是这种词典结构的示意图。这种词典结构主要由两个部分构成：**

** 主体部分是哈希表，每个哈希表项保存一个指针，指针指向冲突链表，在冲突链表里，相同哈值的单词形成链表结构。之所以会有冲突链表，是因为两个不同单词获得相同的哈希值，如果是这样在哈希方法里被称做是一次冲突，可以将相同哈希值的单词存储在链表里，以供后续查找。**

** **

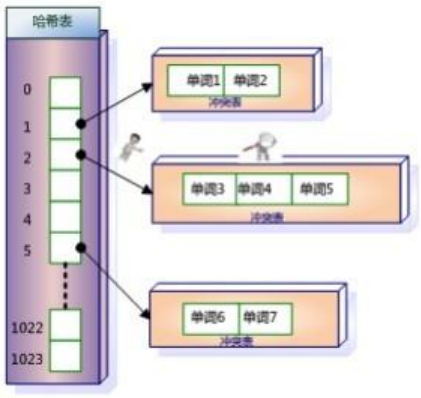


图 1-7　哈希加链表词典结构

**　　在建立索引的过程中，词典结构也会相应地被构建出来。比如在解析一个新文档的时候，对于个在文档中出现的单词T，首先利用哈希函数获得其哈希值，之后根据哈希值对应的哈希表项读取其保存的指针，就找到了对应的冲突链表。如果冲突链表里已经存在这个单词，说明单词在之前解析的档里已经出现过。如果在冲突链表里没有发现这个单词，说明该单词是首次碰到，则将其加入冲突链里。通过这种方式，当文档集合内所有文档解析完毕时，相应的词典结构也就建立起来了。**

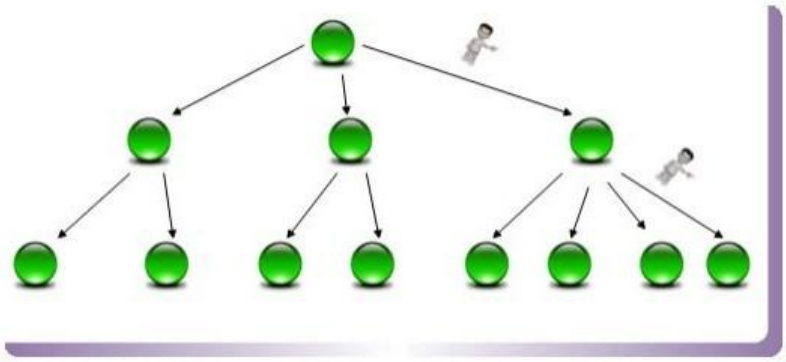
**　　在响应用户查询请求时，其过程与建立词典类似，不同点在于即使词典里没出现过某个单词，不会添加到词典内。以图7为例，假设用户输入的查询请求为单词3，对这个单词进行哈希，定位到哈表内的2号槽，从其保留的指针可以获得冲突链表，依次将单词3和冲突链表内的单词比较，发现单词在冲突链表内，于是找到这个单词，之后可以读出这个单词对应的倒排列表来进行后续的工作，如果有找到这个单词，说明文档集合内没有任何文档包含单词，则搜索结果为空。**

4.2　树形结构**

****　　B树（或者B+树）是另外一种高效查找结构，图8是一个 B树结构示意图。B树与哈希方式查不同，需要字典项能够按照大小排序（数字或者字符序），而哈希方式则无须数据满足此项要求。****

****　　B树形成了层级查找结构，中间节点用于指出一定顺序范围的词典项目存储在哪个子树中，到根据词典项比较大小进行导航的作用，最底层的叶子节点存储单词的地址信息，根据这个地址就可提取出单词字符串。**

** **



** 图8　B树查找结构 **

总结

文档编号	文档内容
1	谷歌地图之父跳槽Facebook
2	谷歌地图之父加盟Facebook
3	谷歌地图创始人拉斯离开谷歌加盟Facebook
4	谷歌地图之父跳槽Facebook与Wave项目取消有关
5	谷歌地图之父拉斯加盟社交网站Facebook

单词ID	单词	文档频率	倒排列表（DocID;TF;<POS>）
1	谷歌	5	(1;1;<1>), (2;1;<1>), (3;2;<1;6>), (4;1;<1>), (5;1;<1>)
2	地图	5	(1;1;<2>), (2;1;<2>), (3;1;<2>), (4;1;<2>), (5;1;<2>)
3	之父	4	<1;1;<3>), (2;1;<3>), (4;1;<3>), (5;1;<3>)
4	跳槽	2	(1;1;<4>), (4;1;<4>)
5	Facebook	5	(1;1;<5>), (2;1;<5>), (3;1;<8>), (4;1;<5>), (5;1;<8>)
6	加盟	3	(2;1;<4>), (3;1;<7>), (5;1;<5>)
7	创始人	1	(3;1;<3>)
8	拉斯	2	(3;1;<4>), (5;1;<4>)
9	离开	1	(3;1;<5>)
10	与	1	(4;1;<6>)

单词ID：记录每个单词的单词编号；

单词：对应的单词；

文档频率：代表文档集合中有多少个文档包含某个单词

倒排列表：包含单词ID及其他必要信息

DocId：单词出现的文档id

TF：单词在某个文档中出现的次数

POS：单词在文档中出现的位置

**** 以单词“加盟”为例，其单词编号为6，文档频率为3，代表整个文档集合中有三个文档包含个单词，对应的倒排列表为{(2;1;<4>),(3;1;<7>),(5;1;<5>)}，含义是在文档2，3，5出现过这个单词在每个文档的出现过1次，单词“加盟”在第一个文档的POS是4，即文档的第四个单词是“加盟”，他的类似。****

****这个倒排索引已经是一个非常完备的索引系统，实际搜索系统的索引结构基本如此。**

原文链接：1- 搜索引擎基础倒排索引