Atitit 搜索的艺术

目录

[1. 索引基础 2](#_Toc21921)

[1.1. 单词-文档矩阵 2](#_Toc22096)

[1.2. 倒排索引基本概念 3](#_Toc22122)

[2. 建立索引 4](#_Toc31253)

[2.1. 两遍文档遍历法（2-Pass In-Memory Inversion） 4](#_Toc25564)

[2.2. 排序法（Sort-based Inversion） 5](#_Toc9763)

[2.3. 归并法（Merge-based Inversion） 5](#_Toc9897)

[3. 索引更新策略 5](#_Toc15474)

[3.1. 完全重建策略（Complete Re-Build） 5](#_Toc11449)

[3.2. 再合并策略（Re-Merge） 6](#_Toc8174)

[3.3. 原地更新策略（In-Place） 6](#_Toc12125)

[3.4. 混合策略（Hybrid） 6](#_Toc26200)

[4. 查询处理 7](#_Toc28735)

[4.1. 一次一文档（Doc at a Time） 7](#_Toc27121)

[4.2. 一次一单词（Term at a Time） 7](#_Toc2774)

[4.3. 跳跃指针（Skip Pointers） 8](#_Toc16808)

[5. 多字段索引 8](#_Toc18013)

[5.1. 多索引方式 8](#_Toc7085)

[5.2. 倒排列表方式 8](#_Toc5281)

[5.3. 扩展列表方式 9](#_Toc5517)

[6. 短语查询 9](#_Toc3105)

[7. 分布式索引（Parallel Indexing） 9](#_Toc17993)

[7.1. 按文档划分（Document Paritioning） 9](#_Toc24665)

[7.2. 按单词划分（Term Paritioning） 9](#_Toc4610)

[7.2.1. 两种方案比较 9](#_Toc12587)

[8. Lucene/Solr/elasticsearch开源搜索引擎 10](#_Toc23604)

[9. 搜索引擎中的常用算法 11](#_Toc4799)

[9.1. 索引建立 查询等算法 11](#_Toc5269)

[9.2. 4.2 倒排列表压缩算法 78 11](#_Toc31072)

[9.3. 6.3 PageRank算法 137 11](#_Toc15372)

[9.4. 10.1 通用去重算法框架 261 12](#_Toc20749)

[10. Ref 12](#_Toc10858)

[10.1. 后来找到了一本书，《这就是搜索引擎》 12](#_Toc1763)

[10.2. atitit 这就是搜索引擎-核心技术详解 读后感总结 艾提拉著.docx 12](#_Toc25279)

# 索引基础

先介绍与搜索引擎有关的一些基本概念，了解这些概念对后续了解工作机制非常重要。

## **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。如下图所示，每列代表一个文档，每行代表一个单词，打对钩的位置代表包含关系。



从纵向看，可以得知每列代表文档包含了哪些单词；从横向看，每行代表了哪些文档包含了某个单词。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上述概念模型，比如倒排索引、签名文件、后缀树等方式。但实验数据表明，倒排索引是单词到文档映射关系的最佳实现方式。

## **倒排索引基本概念**

文档（Document）：以文本形式存在的存储对象。如：网页、Word、PDF、XML等不同格式的文件。  
文档集合（Document Collection）：若干文档构成的集合。如：大量的网页。  
文档编号（Document ID）：搜索引擎内部，唯一标识文档的唯一编号。  
单词编号（Word ID）：搜索引擎内部，唯一标识单词的唯一编号。  
倒排索引（Inverted Index）：实现单词--文档矩阵的一种具体存储形式。倒排索引主要有单词词典和倒排文件组成。  
单词词典（Lexicon）：文档集合中出现过的所有单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及指向倒排列表的指针。  
倒排列表（PostingList）：出现了某个单词的所有文档的文档列表及单词在该文档中出现的位置信息。列表中每条记录称为一个倒排项（Posting）。  
倒排文件（Inverted File）：保存所有单词的倒排列表的文件，倒排文件是存储倒排索引的物理文件。

概念之间的关系如图：

# 建立索引

前面介绍了索引结构，那么，有了数据之后索引是怎么建立的呢？主要有三种建立索引的方法。

## **两遍文档遍历法（2-Pass In-Memory Inversion）**

此方法在内存里完成索引的创建过程。要求内存要足够大。  
**第一遍**  
收集一些全局的统计信息。包括文档集合包含的文档个数N，文档集合内所包含的不同单词个数M，每个单词在多少个文档中出现过的信息DF。  
将所有单词对应的DF值全部相加，就可以知道建立最终索引所需的内存大小是多少。  
获取信息后，根据统计信息分配内存等资源，同事建立好单词相对应倒排列表在内存中的位置信息。

**第二遍**  
逐个单词建立倒排列表信息。获得包含某个单词的每个文档的文档ID，以及这个单词在文档中的出现次数TF，然后不断填充第一遍扫描时所分配的内存。当第二遍扫描结束的时候，分配的内存正好被填充满，每个单词用指针所指向的内存区域“片段”，其起始位置和终止位置之间的数据就是这个单词对应的倒排列表。

## **排序法（Sort-based Inversion）**

在建立索引过程中，始终在内存中分配固定大小的空间，用来存放词典信息和索引的中间结果，当分配的空间被消耗光的时候，把中间结果写入磁盘，清空内存里中间结果所占空间，以用做下一轮存放索引中间结果的存储区。参考下图：

## **归并法（Merge-based Inversion）**

归并法与排序法类似，不同的是，每次将内存中数据写入磁盘时，包括词典在内的所有中间结果都被写入磁盘，这样内存所有内容都可以被清空，后续建立索引可以使用全部的定额内存。归并法的示意图如下所示：

# **索引更新策略**

动态索引可以满足实时搜索的需求，但是随着加入文档越来越多，临时索引消耗的内存也会随之增加。因此要考虑将临时索引的内容更新到磁盘索引中，以释放内存空间来容纳后续的文档，此时就需要考虑合理有效的索引更新策略。

## **完全重建策略（Complete Re-Build）**

对所有文档重新建立索引。新索引建立完成后，老的索引被遗弃释放，之后对用户查询的响应完全由新的索引负责。在重建过程中，内存中仍然需要维护老的索引对用户的查询做出响应。如图所示

## **再合并策略（Re-Merge）**

有新文档进入搜索系统时，搜索系统在内存维护临时倒排索引来记录其信息，当新增文档达到一定数量，或者指定大小的内存被消耗完，则把临时索引和老文档的倒排索引进行合并，以生成新的索引。过程如下图所示：

## **原地更新策略（In-Place）**

原地更新策略的出发点是为了解决再合并策略的缺点。

在索引合并时，并不生成新的索引文件，而是直接在原先老的索引文件里进行追加操作，将增量索引里单词的倒排列表项追加到老索引相应位置的末尾，这样就可达到上述目标，即只更新增量索引里出现的单词相关信息，其他单词相关信息不变动。

为了能够支持追加操作，原地更新策略在初始建立的索引中，会在每个单词的倒排列表末尾预留出一定的磁盘空间，这样，在进行索引合并时，可以将增量索引追加到预留空间中。如下图：

## **混合策略（Hybrid）**

将单词根据其不同性质进行分类，不同类别的单词，对其索引采取不同的索引更新策略。常见做法：根据单词的倒排列表长度进行区分，因为有些单词经常在不同文档中出现，所以其对应的倒排列表较长，而有些单词很少见，则其倒排列表就较短。根据这一性质将单词划分为长倒排列表单词和短倒排列表单词。长倒排列表单词采取原地更新策略，而短倒排列表单词则采取再合并策略。

因为长倒排列表单词的读/写开销明显比短倒排

# **查询处理**

建立好索引之后，如何用倒排索引来响应用户的查询呢？主要有下面三种查询处理机制。

## **一次一文档（Doc at a Time）**

以倒排列表中包含的文档为单位，每次将其中某个文档与查询的最终相似性得分计算完毕，然后开始计算另外一个文档的最终得分，直到所有文档的得分计算完毕为止。然后根据文档得分进行大小排序，输出得分最高的K个文档作为搜索结果输出，即完成了一次用户查询的响应。实际实现中，只需在内存中维护一个大小为K的优先级队列。如下图所示是一次一文档的计算机制示意图：

## **一次一单词（Term at a Time）**

与一次一文档不同，一次一单词采取“先横向再纵向”的方式，首先将某个单词对应的倒排列表中的每个文档ID都计算一个部分相似性得分，也就是说，在单词-文档矩阵中首先进行横向移动，在计算完某个单词倒排列表中包含的所有文档后，接着计算下一个单词倒排列表中包含的文档ID，即进行纵向计算，如果发现某个文档ID已经有了得分，则在原先得分基础上累加。当所有单词都处理完毕后，每个文档最终的相似性得分计算结束，之后按照大小排序，输出得分最高的K个文档作为搜索结果。 下图是一次一单词的运算机制。

## **跳跃指针（Skip Pointers）**

基本思想：将一个倒排列表数据化整为零，切分为若干个固定大小的数据块，一个数据块作为一组，对于每个数据块，增加元信息来记录关于这个块的一些信息，这样即使是面对压缩后的倒排列表，在进行倒排列表合并的时候也能有两个好处

# **多字段索引**

即对文档的多个字段进行索引。  
实现多字段索引的方式：多索引方式、倒排列表方式和扩展列表方式。

## **多索引方式**

针对每个不同的字段，分别建立一个索引，当用户指定某个字段作为搜索范围时，可以从相应的索引里提取结果。当用户没有指定特定字段时，搜索引擎会对所有字段都进行查找并合并多个字段的相关性得分，这样效率较低。多索引方式示意图如下：

## **倒排列表方式**

将字段信息存储在某个关键词对应的倒排列表内，在倒排列表中每个文档索引项信息的末尾追加字段信息，这样在读出用户查询关键词的倒排列表的同时，就可以根据字段信息，判断关键词是否在某个字段出现，以此来进行过滤。倒排列表方式示意图如下：

IMG_256

## **扩展列表方式**

这是用得比较多的支持多字段索引的方法。为每个字段建立一个列表，该列表记录了每个文档这个字段对应的出现位置信息。下图是扩展列表的示意图：

# **短语查询**

短语查询的本质是如何在索引中维护单词之间的顺序关系或者位置信息。较常见的支持短语查询技术包括：位置信息索引、双词索引和短语索引。也可将三者结合使用。

# **分布式索引（Parallel Indexing）**

当搜索引擎需要处理的文档集合太多的时候，就需要考虑分布式解决方案。每台机器维护整个索引的一部分，有多台机器协作来完成索引的建立和对查询的响应。

## **按文档划分（Document Paritioning）**

将整个文档集合切割成若干个子集合，而每台机器负责对某个文档子集合建立索引，并响应查询请求。按文档划分示意图如下：

## **按单词划分（Term Paritioning）**

每个索引服务器负责词典中部分单词的倒排列表的建立和维护。按单词划分示意图如下

### ****两种方案比较****

按文档比较常用，按单词划分只在特殊应用场合才使用。  
按单词划分的不足：  
**可扩展性**  
搜索引擎处理的文档是经常变动的。如果按文档来对索引划分，只需要增加索引服务器，操作起来很方便。但如果是按单词进行索引划分，则对几乎所有的索引服务器都有直接影响，因为新增文档可能包含所有词典单词，即需要对每个单词的倒排列表进行更新，实现起来相对复杂。

**负载均衡**  
常用单词的倒排列表非常庞大，可能会达到几十M大小。如果按文档划分，这种单词的倒排列表会比较均匀地分布在不同的索引服务器上，而按单词进行索引划分，某个常见单词的倒排列表全部内容都由一台索引服务器维护。如果该单词同时是一个流行词汇，那么该服务器会成为负载过大的性能瓶颈。

**容错性**  
假设某台服务器出现故障。如果按文档进行划分，那么只影响部分文档子集合，其他索引服务器仍然能响应。但如果按单词进行划分，若索引服务器发生故障，则某些单词的倒排列表无法访问，用户查询这些单词的时候，会发现没有搜索结果，直接影响用户体验。

**对查询处理方式的支持**  
按单词进行索引一次只能查询一个单词，而按文档划分的不受此限制。

# Lucene/Solr/elasticsearch开源搜索引擎

搜索引擎

# 搜索引擎中的常用算法

## 索引建立 查询等算法

## 4.2 倒排列表压缩算法 78

4.2.1 评价索引压缩算法的指标 79

4.2.3 Elias Gamma算法与Elias Delta算法 81

4.2.3 Elias Gamma算法与Elias Delta算法 81

4.2.4 Golomb算法与Rice算法 81

4.2.4 Golomb算法与Rice算法 81

4.2.5 变长字节算法（Variable Byte） 83

4.2.6 SimpleX 系列算法 84

4.2.7 PForDelta算法 86

6.2 两个概念模型及算法之间的关系 133

6.2.3 链接分析算法之间的关系 136

## 6.3 PageRank算法 137

6.4 HITS算法（Hypertext Induced Topic Selection） 140

6.4.3 HITS算法 142

6.4.4 HITS算法存在的问题 144

6.4.5 HITS算法与PageRank算法比较 145

6.4.5 HITS算法与PageRank算法比较 145

6.5 SALSA算法 146

6.7 Hilltop算法 156

6.7.1 Hilltop算法的一些基本定义 157

6.7.2 Hilltop算法 158

6.8 其他改进算法 162

6.8.3 PHITS算法（Probability Analogy of HITS） 163

6.8.4 BFS算法（Backward Forward Step） 163

7.9.1 数据划分算法（Partitioning Algorithm） 207

8.6.1 TrustRank算法 237

8.6.2 BadRank算法 238

## 10.1 通用去重算法框架 261

10.2 Shingling算法 262

10.3 I-Match算法 265

10.4 SimHash算法 268

10.5 SpotSig算法 272

# Ref

## 后来找到了一本书，《这就是搜索引擎》

## atitit 这就是搜索引擎-核心技术详解 读后感总结 艾提拉著.docx