全文搜索技术原理入门

Full-text Search Engine Technology Primer

作者：黑夜路人（Black）

(V1.2)

分享人介绍

“黑夜路人”（heiyeluren / Black），热爱技术的程序员，互联网系统架构师，CSDN博客技术专家，国内开源技术社区建设者

联系方式

个人微信

公众号

weibo.com/heiyeluren

blog.csdn.net/heiyeshuwu

github.com/heiyeluren

知识准备

学习本内容，需要具备的相关知识基础

适合用户

两年以上编程经验

对全文检索有兴趣或工作需要

操作系统知识

Linux 基本使用

Linux 常用命令

Linux 软件安装

编程语言知识

熟悉以下任意语言

PHP

Python

Golang

C/C++

Java

数据结构算法

了解如下知识:

字符串、链表、数组、队列、树

查找算法、内外部排序算法

数据库基本存储知识

内容概要

核心内容概要和模块

内容结构：原理 + 实例（实践）

搜索引擎基础	搜索索引系统	搜索排序系统
搜索引擎核心流程 爬虫系统 网页分析 中文处理	索引系统核心 倒排表的结构 倒排表构建和管理	搜索的核心流程 搜索结果相关性排序算法

信息爆炸了！

庞大的信息，杂乱无章的信息，如何快速找到想要的信息；
全球10亿个网站，超过1000亿网页，站内信息众多

传统数据库无法解决?

传统关系型数据库无法解决目前面临的判断复杂的数据搜索的需求

DB	问题	原因
Oracle、MySQL SQL Server	部分数据库无法支持全文检索功能；无法高效率进行检查查询和排序工作	数据库存储方式和索引方式：B+Tree/Hash/LSM Tree
LevelDB、Redis MongoDB、NoSQL	只支持某种特定场景数据查找，部分存储容量有限	数据库本身架构的局限性和应用场景完全不同

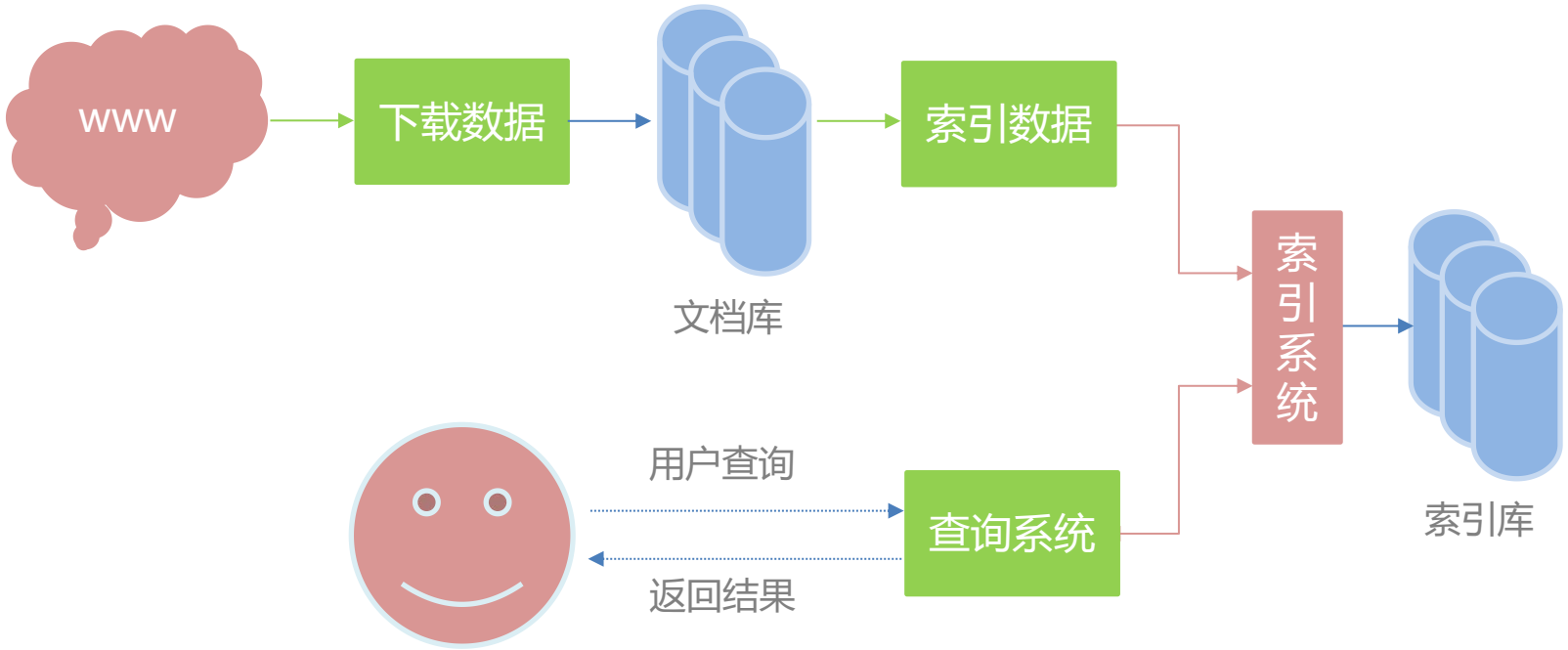
搜索引擎的服务方式

搜索引擎的服务方式可以分为三种：导航式引擎、全网搜索引擎、站内搜索引擎

目录导航式引擎	全网全文搜索引擎	站内全文搜索引擎
Hao123、hao 360等各类网址导航	谷歌、百度360、搜狗	Sphinx、LuceneElasticSearch、Solr
适合简单记录网址	适合全网场景	站内信息检索

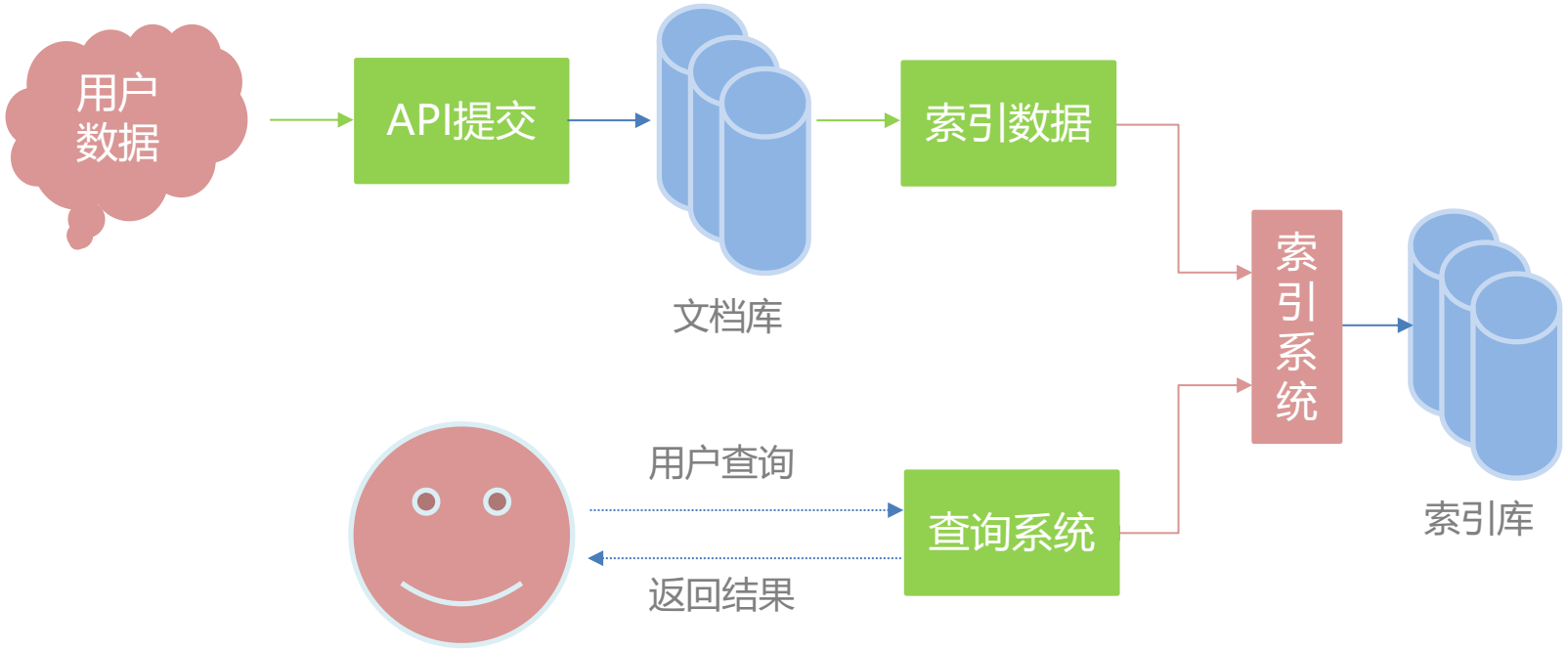
全网搜索引擎工作过程

下载系统→分析系统→索引系统→查询系统



站内搜索引擎工作过程

数据提交→分析系统→索引系统→查询系统



概念引入

蜘蛛、种子站点、URL、反链

1、蜘蛛

Spider、Robots

2、种子站点

蜘蛛爬行开始抓取的起点

3、URL

http://www.hao123.com

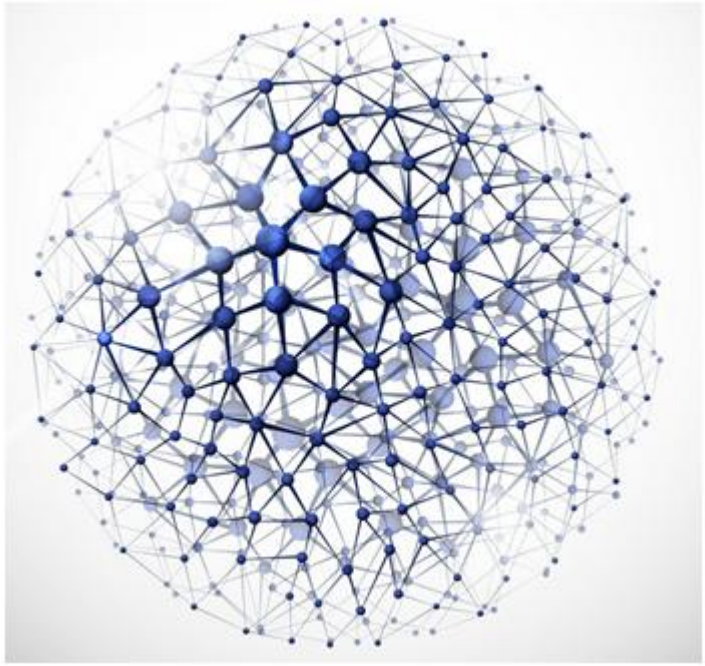
4、反链

正向链接与反向链接

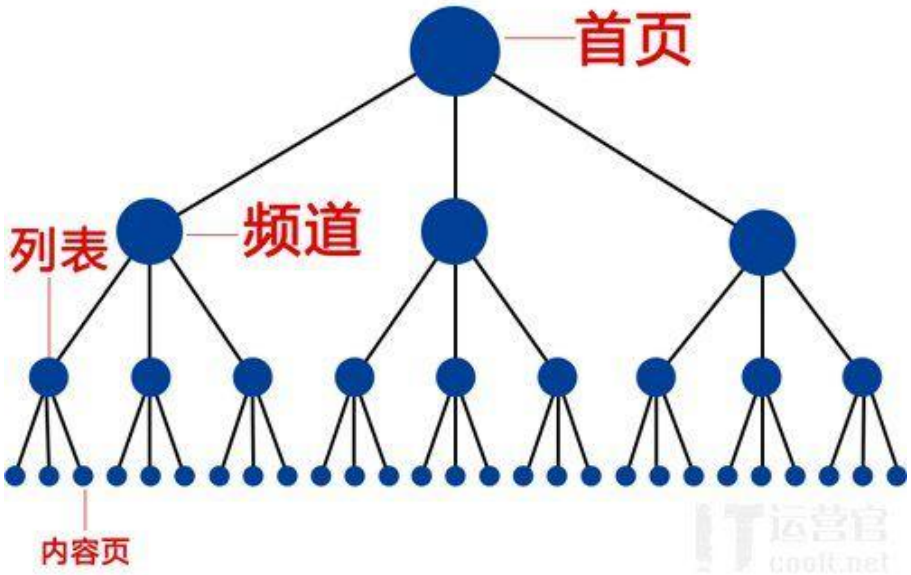
互联网的结构

网页之间的蝴蝶结结构？目录型网页、权威型网页。

互联网的结构

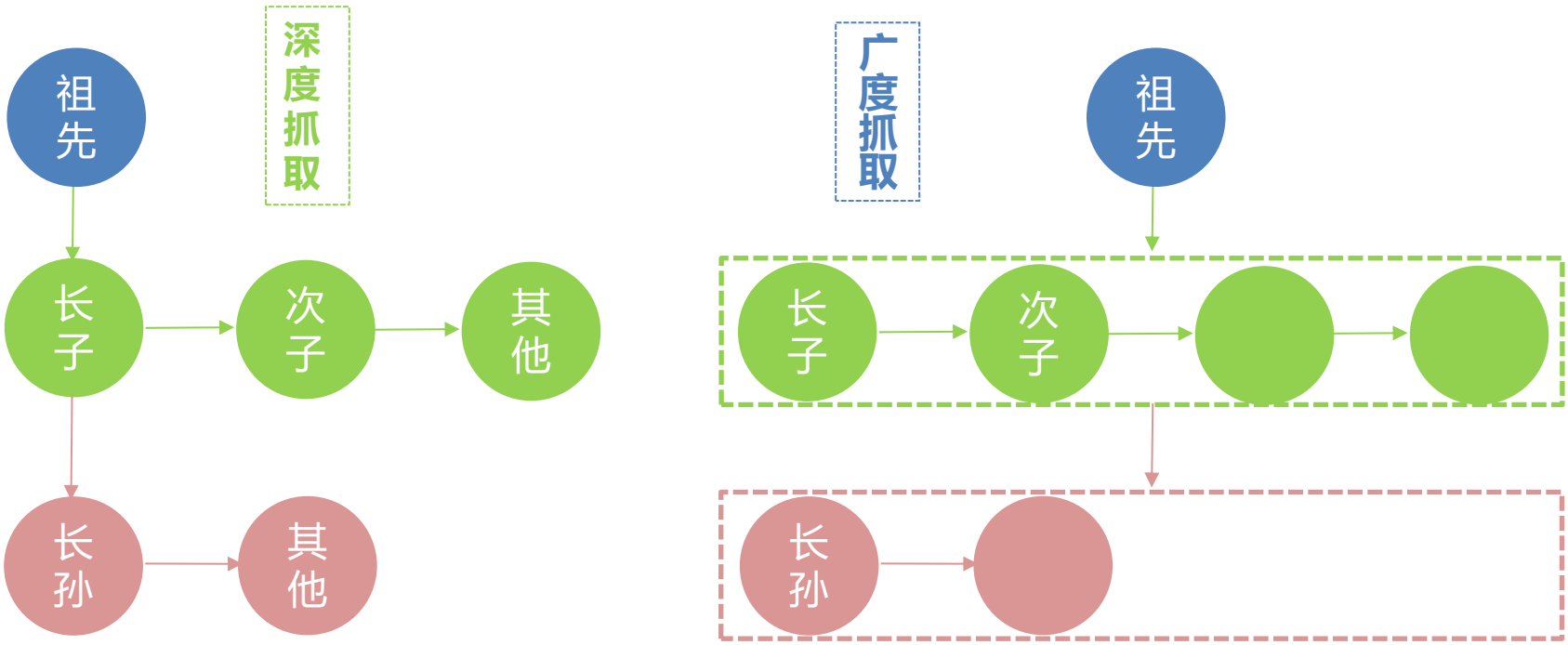


一般网站的结构



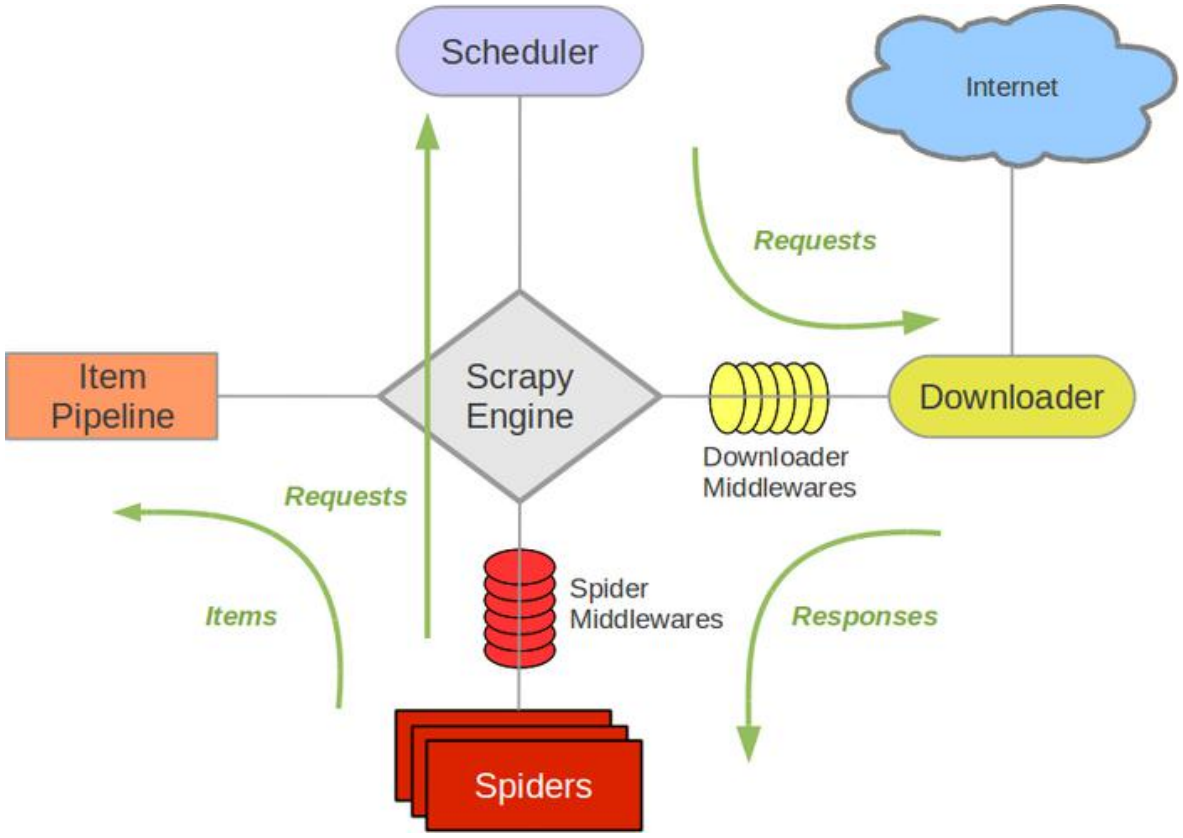
蜘蛛爬行抓取策略

宽度抓取（广度抓取）与**深度**抓取



实例：开源爬虫 Scrapy

常见开源爬虫主要有 Scrapy、Nutch、Labin、Heritrix



Scrapy 架构

1、Scrapy Engine(引擎)
引擎负责控制数据流在系统的所有组件中流动，并在相应动作发生时触发事件

2、Scheduler(调度器):
调度器从引擎接受request并将他们入队，以便之后引擎请求他们时提供给引擎

3、Downloader（下载器）
下载器负责获取页面数据并提供给引擎，而后提供给spider

4、Spider（爬虫）
Spider是Scrapy用户编写用于分析response并提取item(即获取到的item)或额外跟进的URL的类。每个spider负责处理一个特定(或一些)网站

5、Item Pipeline(管道)
Item Pipeline负责处理被spider提取出来的item。典型的处理有清理、 验证及持久化(例如存储到数据库中)

6、Downloader Middlewares（下载中间件）
下载器中间件是在引擎及下载器之间的特定钩子(specific hook)，处理Downloader传递给引擎的response。 其提供了一个简便的机制，通过插入自定义代码来扩展Scrapy功能

7、Spider Middlewares（Spider中间件）
Spider中间件是在引擎及Spider之间的特定钩子(specific hook)，处理spider的输入(response)和输出(items及requests)。 其提供了一个简便的机制，通过插入自定义代码来扩展Scrapy功能

黑

一 搜索引擎系统概述

二 搜索引擎下载系统

三 搜索引擎分析系统

四 搜索引擎索引系统

五 搜索引擎查询系统

概念引入

Html语言、锚文本、半结构化数据

1、Html语言

标签?标签!

2、锚文本

黑夜路人的开源世界

3、半结构化数据

结构化数据？半结构化数据？

半结构化向结构化靠拢

提取文字！奔着结构化的目标提取文字！


```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-stri

    <html xmlns="http://www.w3.org/1999/xhtml">

<head>

    <script type="text/javascript" src="http://c.csdnimg.cn/pubfooter/js/tracking.js" charset="utf-8"></scr

    <script type="text/javascript">
        var protocol = window.location.protocol;
        document.write('<script type="text/javascript" src="' + protocol + '//csdnimg.cn/pubfooter/js/repo
    </script>


 <meta http-equiv="Cache-Control" content="no-siteapp" /><link rel="alternate" media="handheld" href="#" />

    <title>heiyeluren的blog（黑夜路人的开源世界）
        - 博客频道 - CSDN.NET</title>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <meta name="description" content="" />
    <script src="http://static.blog.csdn.net/scripts/jquery.js" type="text/javascript"></script>
      <script type="text/javascript" src="http://static.blog.csdn.net/scripts/jquery-version.js"></script>
    <script type="text/javascript" src="http://static.blog.csdn.net/scripts/ad.js?v=1.1"></script>
        <!--new top-->
                <link rel="stylesheet" href="http://c.csdnimg.cn/public/common/toolbar/css/index.css">

      <!-- ad begin -->
        <script language="javascript" type="text/javascript" src="http://ads.csdn.net/js/tracking.js"></sc
    <!-- ad end-->

    <link rel="Stylesheet" type="text/css" href="http://static.blog.csdn.net/skin/light_blue/css/style.css
    <link id="RSSLink" title="RSS" type="application/rss+xml" rel="alternate" href="/heiyeshuwu/rss/list" /
    <link rel="shortcut icon" href="http://c.csdnimg.cn/public/favicon.ico" />
    <link type="text/css" rel="stylesheet" href="http://static.blog.csdn.net/scripts/SyntaxHighlighter/styl



<script>
    var _hmt = _hmt || [];
    (function () {
        var hm = document.createElement("script");
        hm.src = "//hm.baidu.com/hm.js?6bcd52f51e9b3dce32bec4a3997715ac";
        var s = document.getElementsByTagName("script")[0];
        s.parentNode.insertBefore(hm, s);
    })();
</script>
```

1、锚文本（anchor text）

2、标题（title）

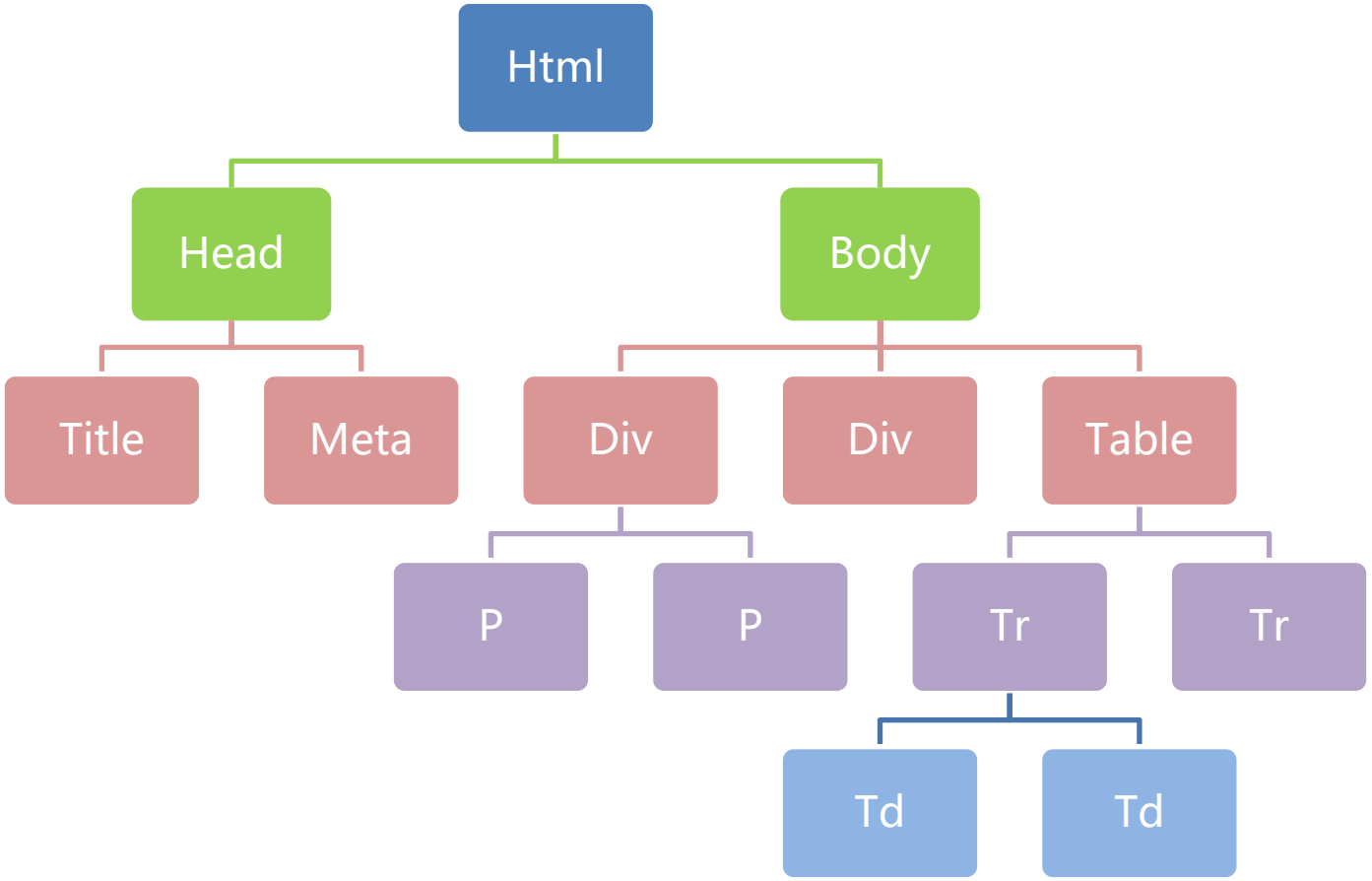
3、正文标题（content title）

4、正文（content）

5、正向链接（link）

网页HTML标签树

将半结构向结构化数据转变，DOM树

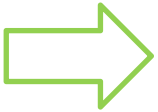


网页去重

常用网页去重算法有Shingling、I-Match、SimHash、Spotsig等算法

Shingling 算法去重过程：

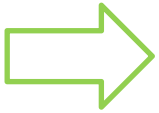
中国足球队在米卢的率领下首次获得世界杯决赛阶段的比赛资格，新浪体育报道。



中国足球队
国足球队在
足球队在米
……

$J=A\cap B/A\cup B$

米卢率领中国足球队首次杀入世界杯决赛阶段，搜狐体育播报。



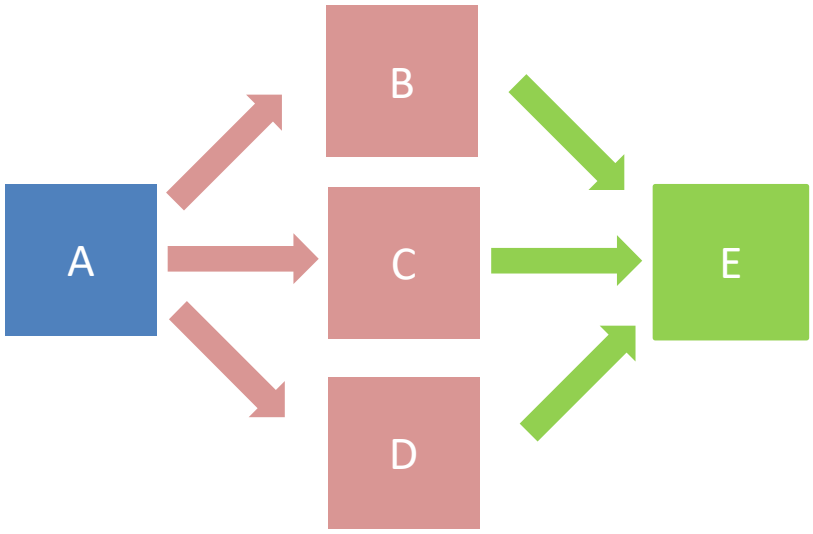
米卢率领中
卢率领中国
率领中国足
……

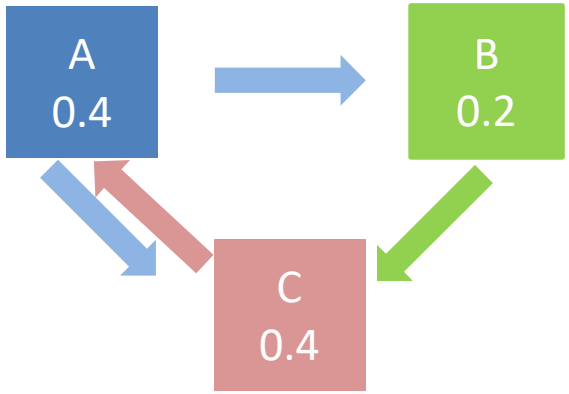
基于链接的分析算法

PageRank、点击反馈

1、PageRank算法

$$PR(A) = (1-d) + d(PR(t1)/C(t1) + \ldots + PR(tn)/C(tn))$$





NLP基础：中文分词技术

自然语言处理：基于词典及统计学的中文分词或更多方式，用户查询意图，相关文档索引入库都需要依赖于NLP

1、基于词典的中文分词

正向最大匹配法
逆向最大匹配法
双向最大匹配法

学历史学好?

2、基于统计学的分词

$P(S) = P(W_1W_2W_3\cdots W_n) = P(W_1) * P(W_2|W_1) * P(W_t|W_1W_2\cdots W_{t-1})$

3、基于深度学习的分词

CNN
RNN
LSTM

中文分词算法(Tokenizer / Segmentation)

针对无法直接切割的语言，亚洲系语言，比如CJK语言的特殊处理，新词发现

基于字典的算法

正向最大匹配
(FMM, Front Maximum Matching)
逆向最大匹配
(RMM, Reverse Maximum Matching)

双向最大匹配法
(BM, Bi-directction Matching method)
最小切算法
(MSM，Minimum Segmentation method)
MMSeg
(Maximum Matching Segmentation)

基于统计机器学习的算法

N元文法模型
(N-gram)
结构化感知器
(SP, Structured Perceptron)

条件随机场算法
(CRF, Conditional Random Field)
隐马尔科夫模型
(HMM, Hidden Markov Model)

基于深度学习的算法

卷积神经网络模型
(CNN，Convolutional Neural Networks)

递归神经网络模型
RNN（Recurrent Neural Networks）
+
LSTM（Long-Short Term Memory）

实例：开源分词库

针对亚洲系语言，比如CJK语言的特殊处理，新词发现

基于字典的算法

- IKAnalyzer (FMM+RMM+BM)
- Jieba (FMM+DP+DAG)
- SCWS分词 (基于字典)
- MMSEG (MMseg算法)
- Friso (基于MMseg)
- Jcseg (基于FMM)
- sego (最短路径/Trie-tree)

基于统计学的算法

- Genius (基于CRF算法)
- Finalseg (基于HMM模型)
- HanLP (HMM+N-gram)
- Ansj (N-gram+CRF+HMM)
- SnowNLP (N-gram+HMM)
- CoreNLP (基于CRF)
- THULAC (基于SP)
- LTP (基于SP)

基于深度学习的算法

- Kcws (基于LSTM+CRF)
- MiNLP (基于CNN+CRF)
- FoolNLTK (基于LSTM)

全文搜索索引

索引，全文检索，文档编号

全文检索大致分为两个部分：索引创建、搜索索引。于是全文检索就存在如下三个问题：

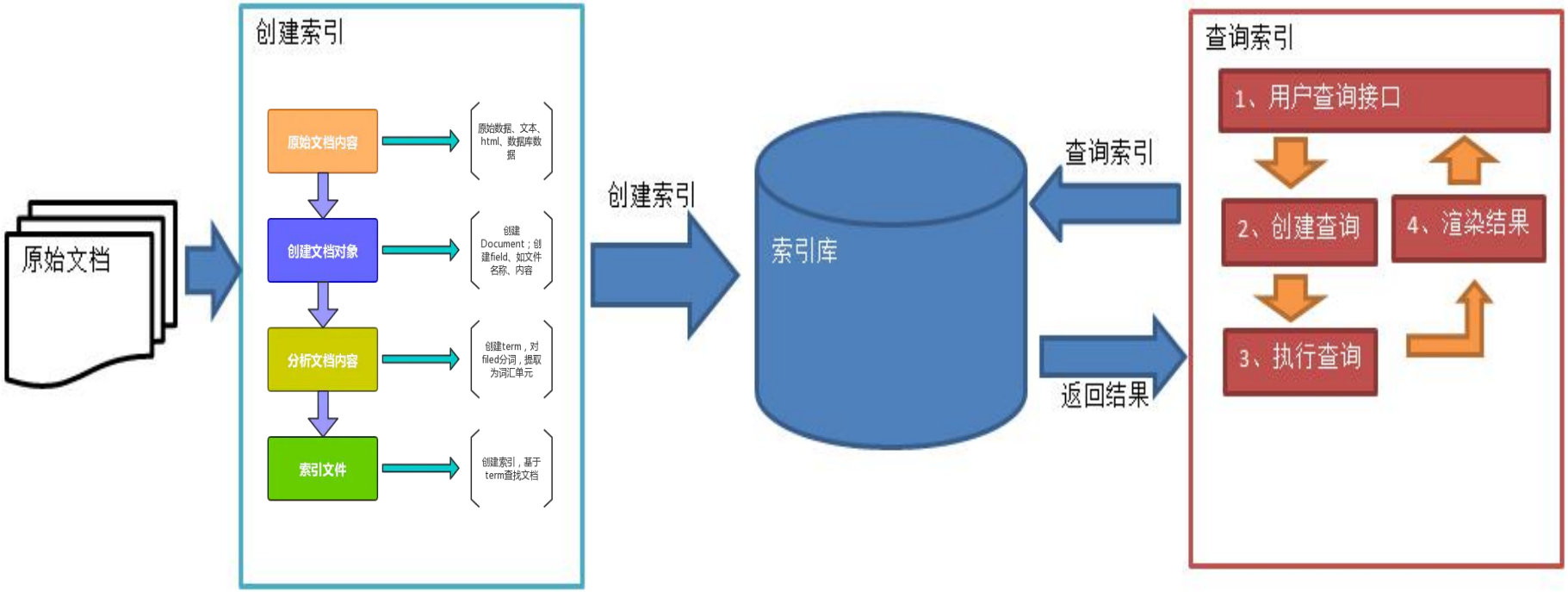
1、索引里存着什么？

2、如何创建索引？

3、如何进行索引搜索？

全文搜索索引和查询过程

索引，全文检索，文档编号



概念引入

索引，全文检索，文档编号

黑

1、索引

索引是描述信息的信息，比如书籍的目录

正排索引、正向索引、前向索引

倒排索引、倒向索引

2、全文检索

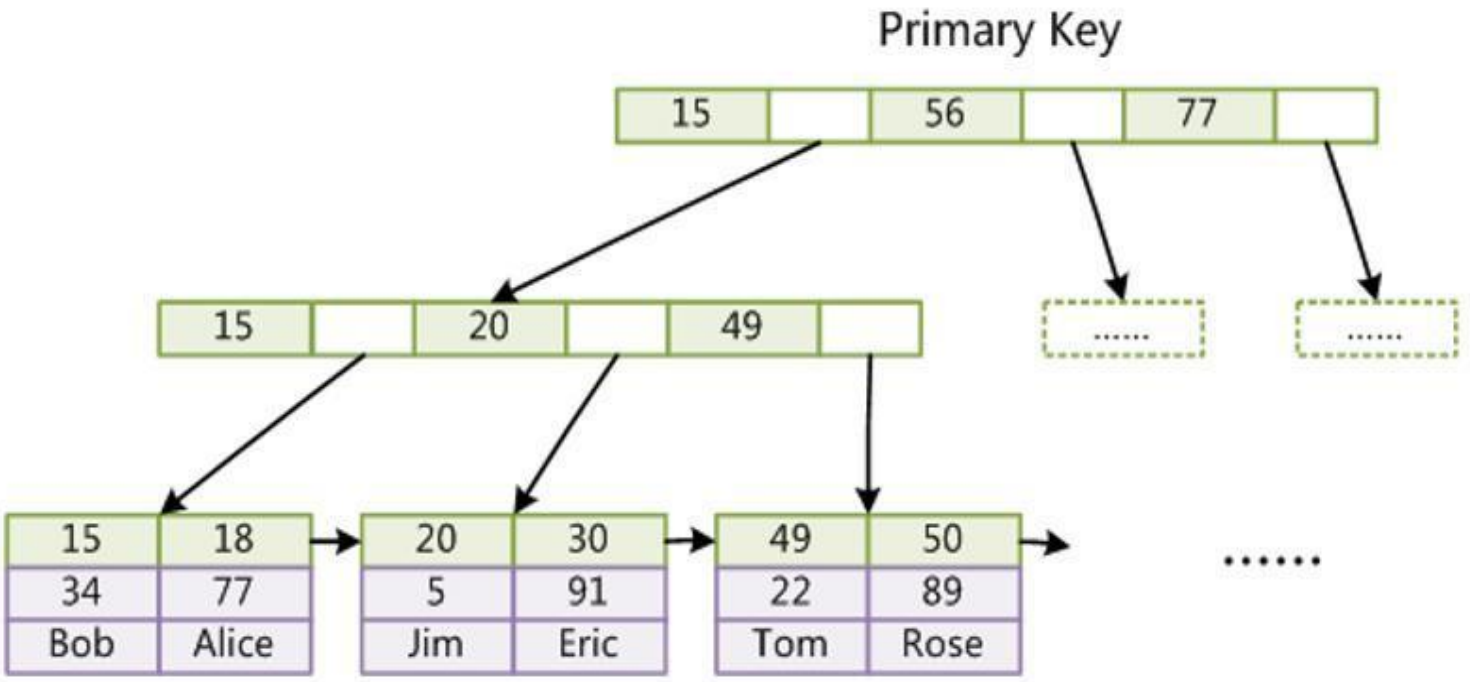
文档的全部文字参与索引，检索词出现的位置可以提供

3、文档编号

每篇文档的一个独立的编号

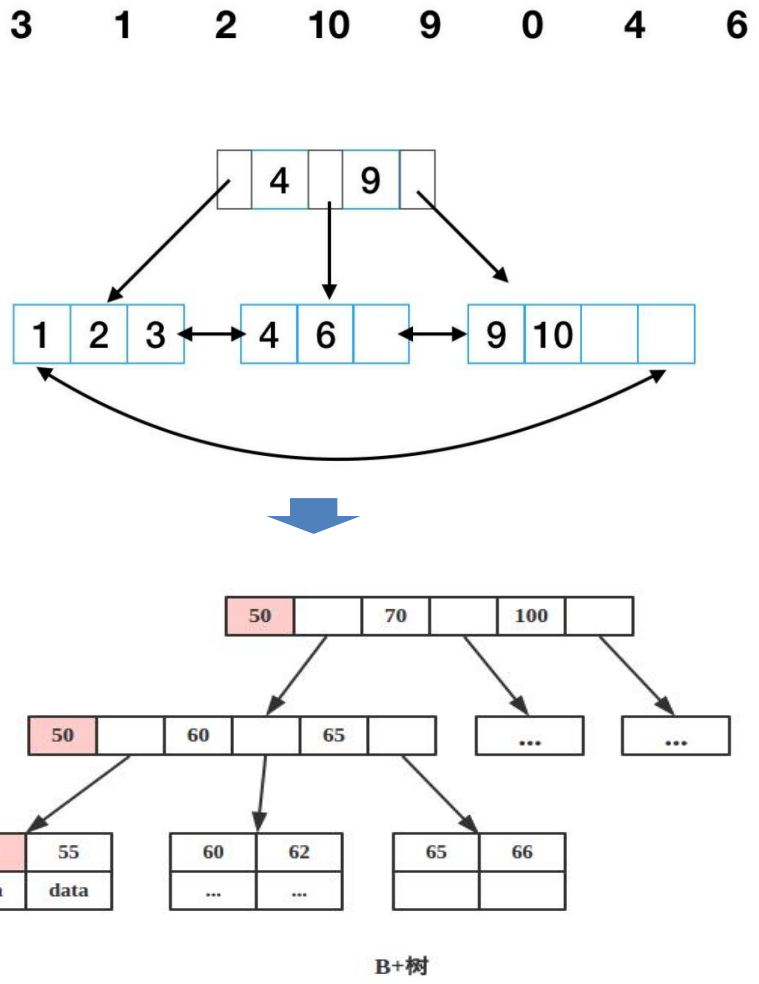
常见数据库索引方式：B+树

Oracle、MySQL、PostgreSQL等数据库和磁盘文件系统主要索引结构

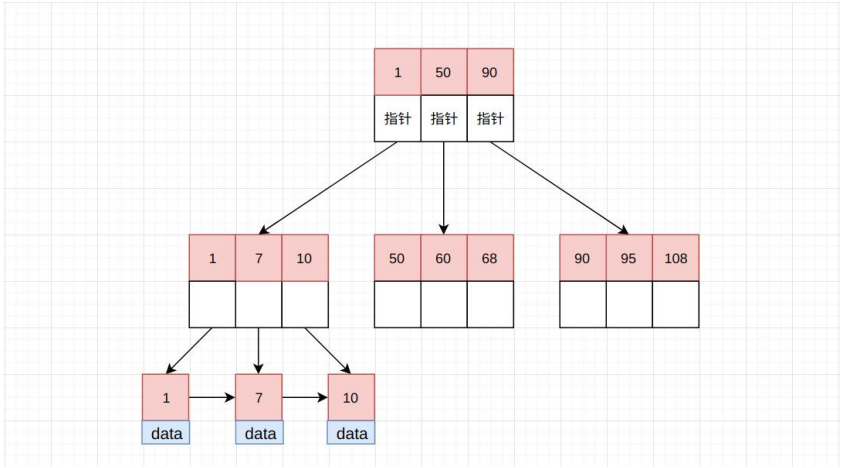


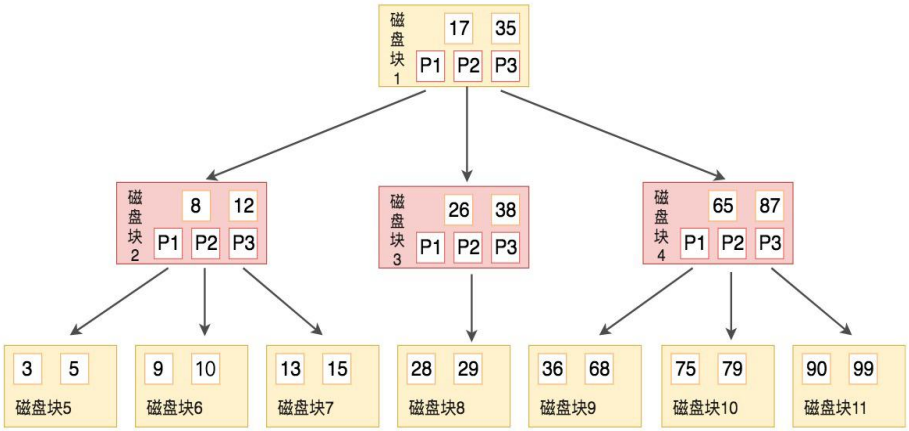
常见数据库索引方式：B+树

Oracle、MySQL、PostgreSQL等数据库和磁盘文件系统主要索引结构



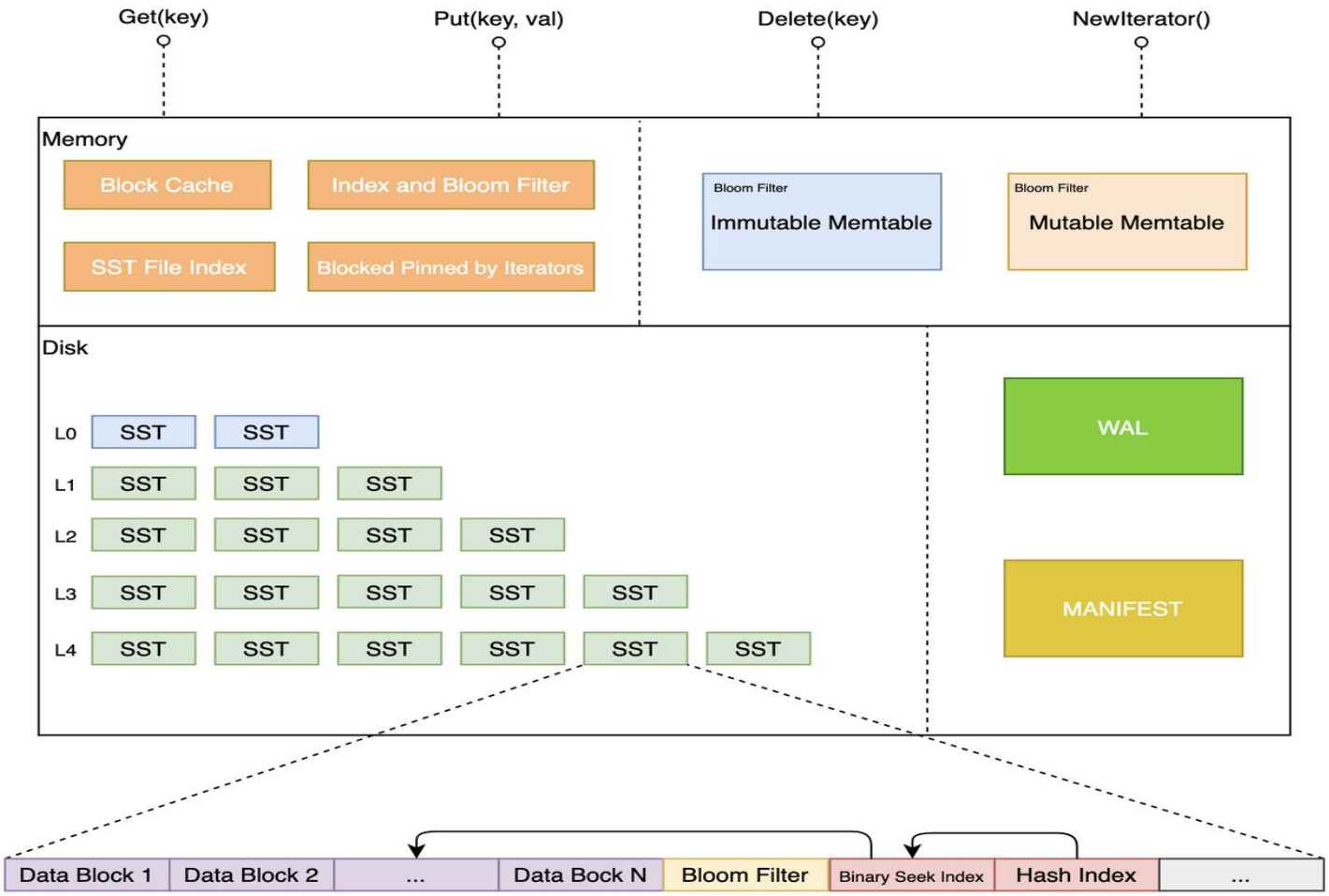
B+树





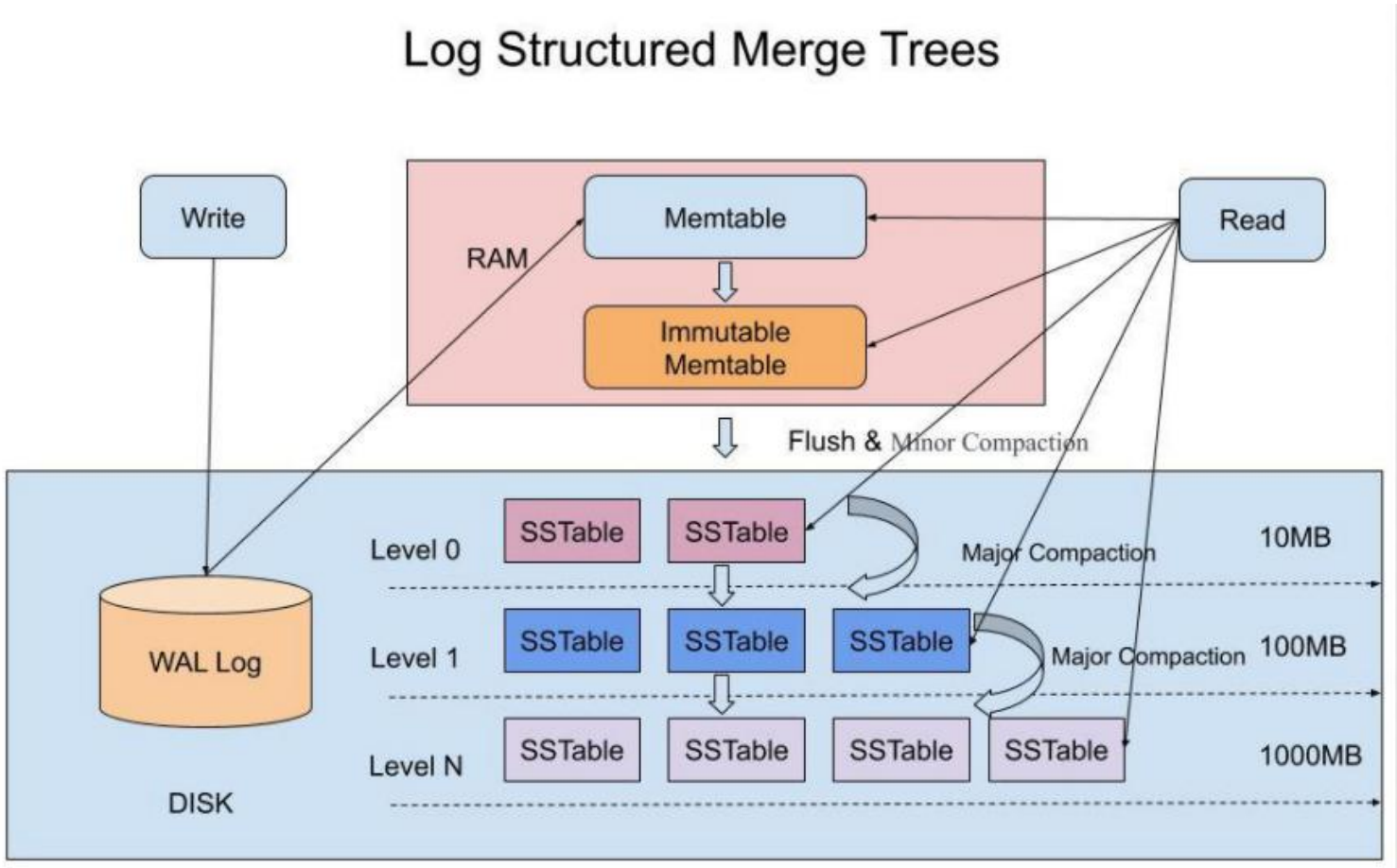
常见数据库索引方式：LSM树

MongoDB、LevelDB、Cassandra 等数据存储系统主要索引结构



常见数据库索引方式：LSM树

MongoDB、LevelDB、Cassandra 等数据存储系统主要索引结构



全文搜索：正向索引与倒排索引

注意正向与倒排索引中的主键与属性

正向索引

Lid	Word id	N hits	Hit List
Doc1	Word1	m	Hit1，...Hitm
	Word2	n	Hit1，...Hitm
	.....	.....	.....
	Null		
Doc2	.....	.....	.....
	Null		

倒排索引

Word id	nDocs	偏移量
Term1	2	X
Term2	1	y
Term3		
.....		
TermN		z

Doc id	N hits	Hit List
Doc1		
Doc2		
Doc1	3	3,5,7
.....		
DocN		

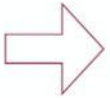
全文搜索：正向索引与倒排索引

来个更简单易懂的正向与倒排吧

文件ID	内容
文件1	关键词1，关键词2
文件2	关键词1，关键词3
文件3	关键词3，关键词2

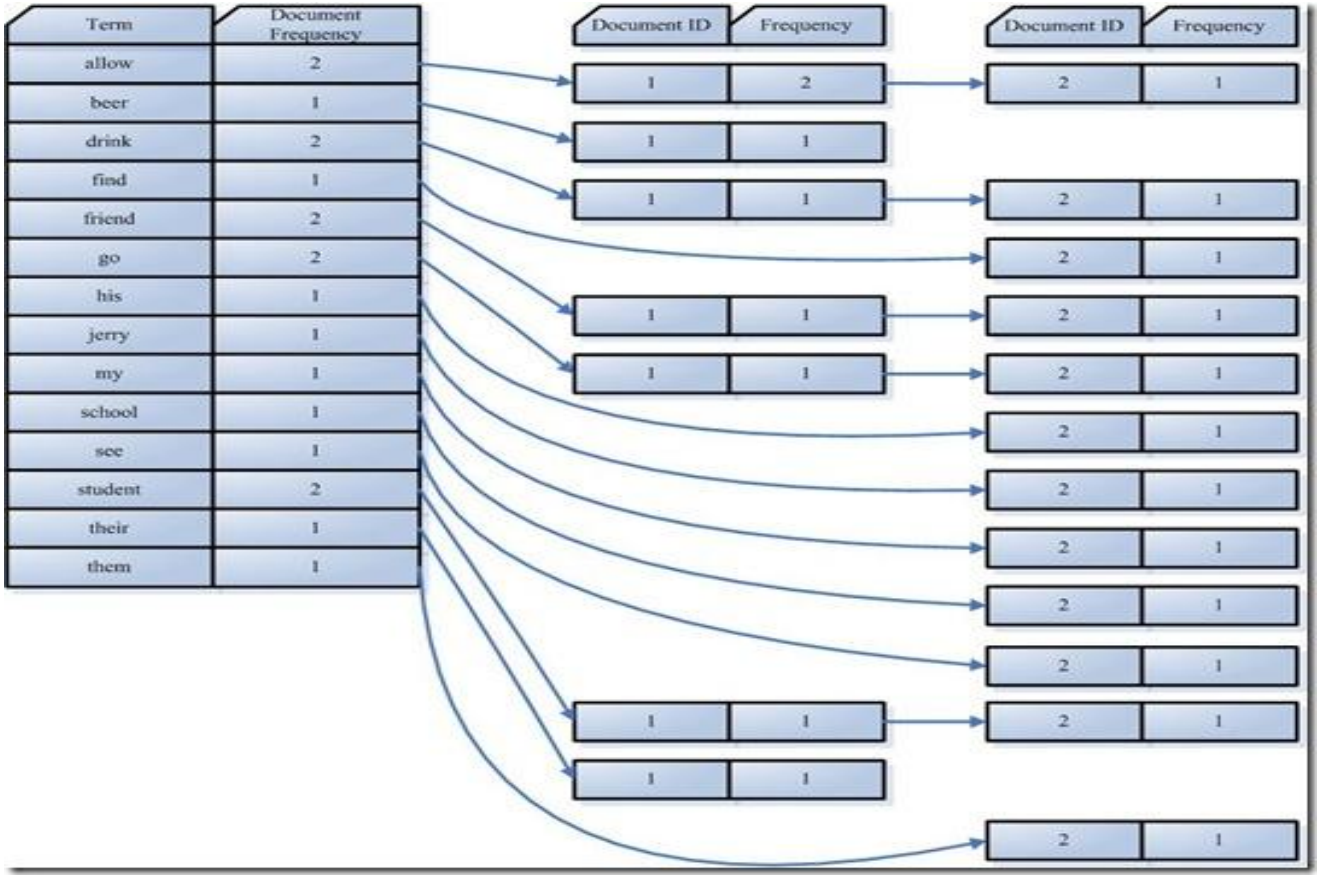
关键词	文件
关键词1	文件1，文件2
关键词2	文件1，文件3
关键词3	文件2，文件3

文档ID	文档内容
1	elasticsearch是最流行的搜索引擎
2	php 是世界上最好的语言
3	搜索引擎是如何诞生的

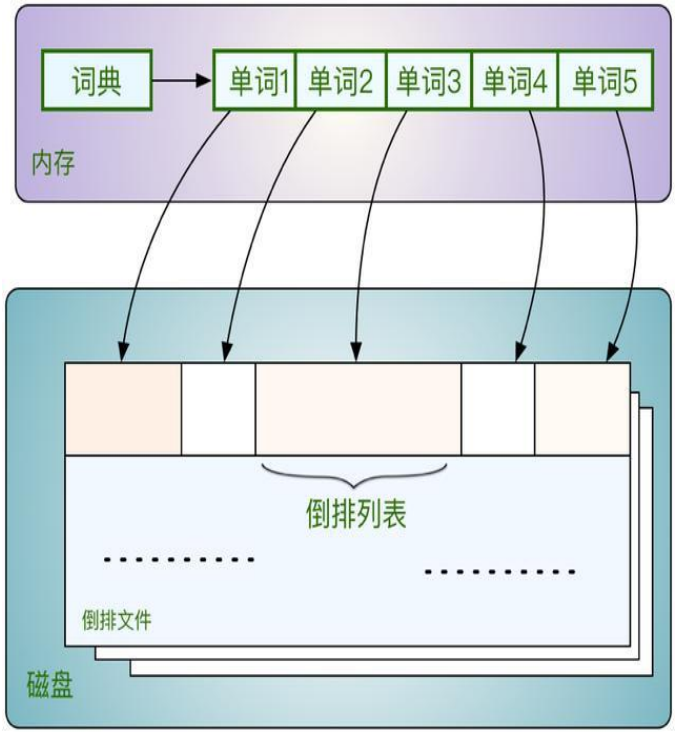


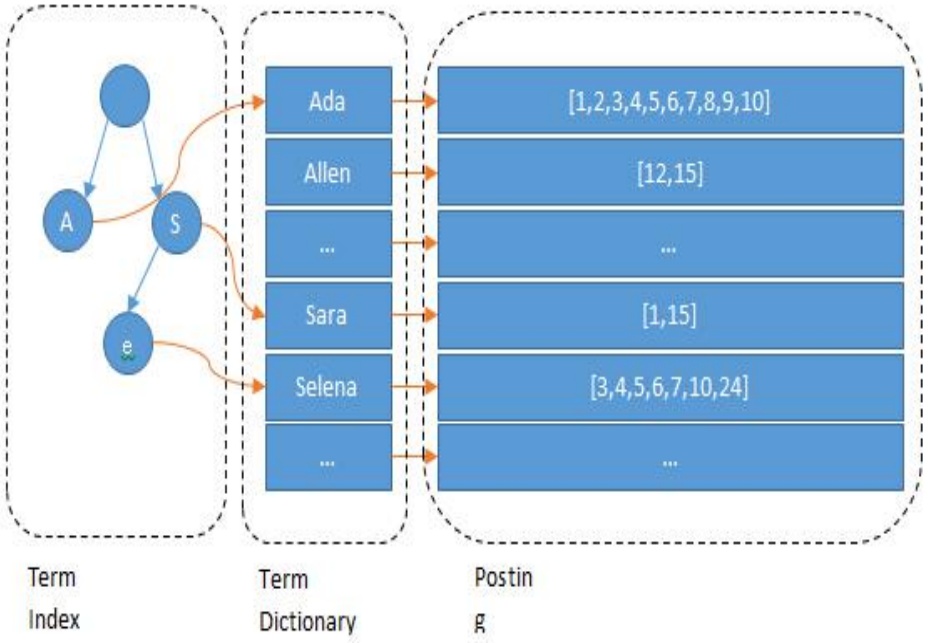
单词	文档 ID列表
elasticsearch	1
流行	1
搜索引擎	1,3
php	2
世界	2
最好	2
语言	2
如何	3
诞生	3

倒排索引的数据结构


Term
Document Frequency
allow 2
beer 1
drink 2
find 1
friend 2
go 2
his 1
jerry 1
my 1
school 1
see 1
student 2
their 1
them 1
Document ID
Frequency
Document ID
Frequency

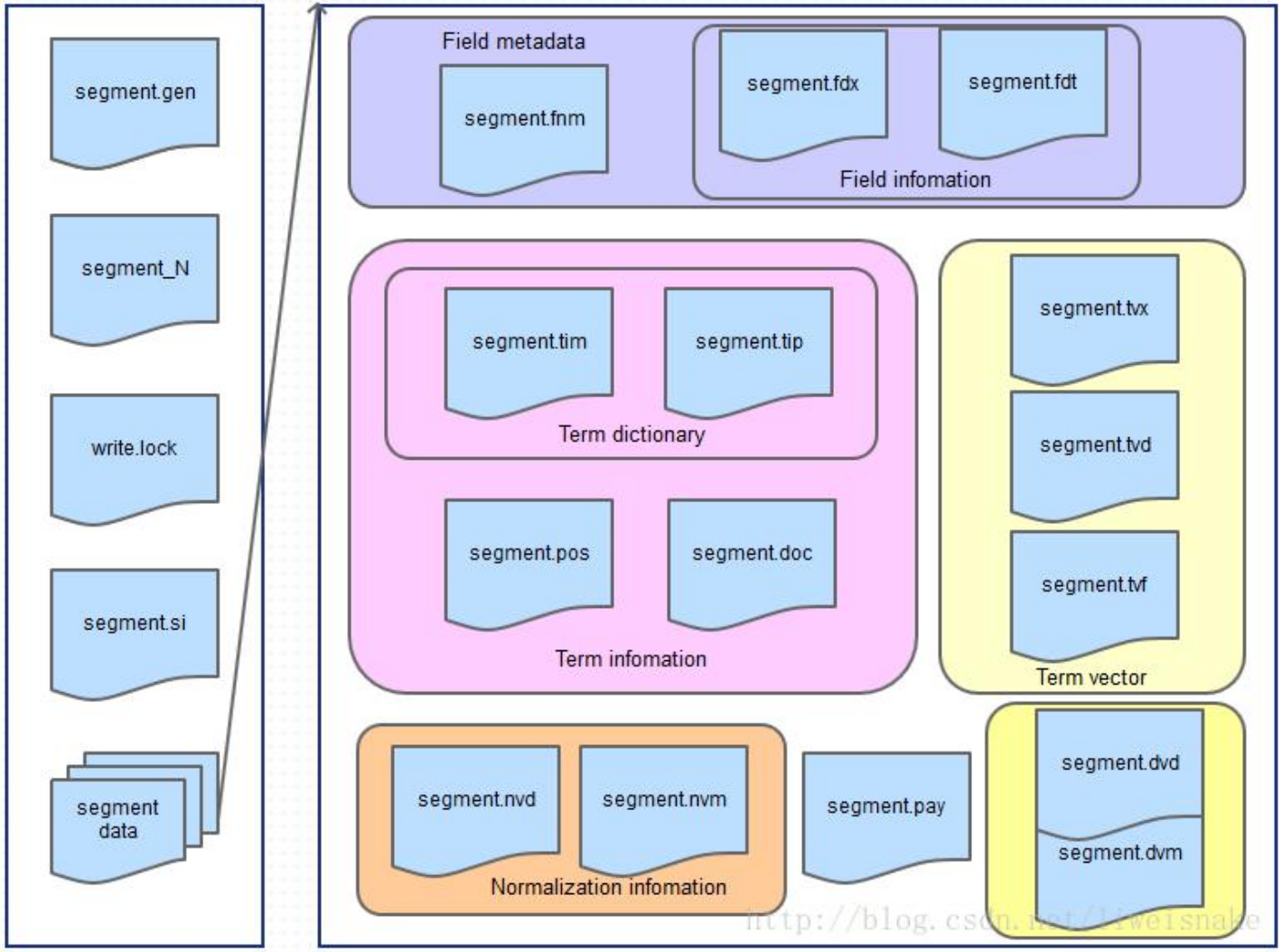
倒排索引存储结构


词典
单词1
单词2
单词3
单词4
单词5
内存
倒排列表
倒排文件
磁盘


A
S
e
Ada
Allen
...
Sara
Selena
...
[1,2,3,4,5,6,7,8,9,10]
[12,15]
...
[1,15]
[3,4,5,6,7,10,24]
...
Term
Index
Term
Dictionary
Postin
g

实例：Lucene倒排索引

Lucene 引擎索引文件组成



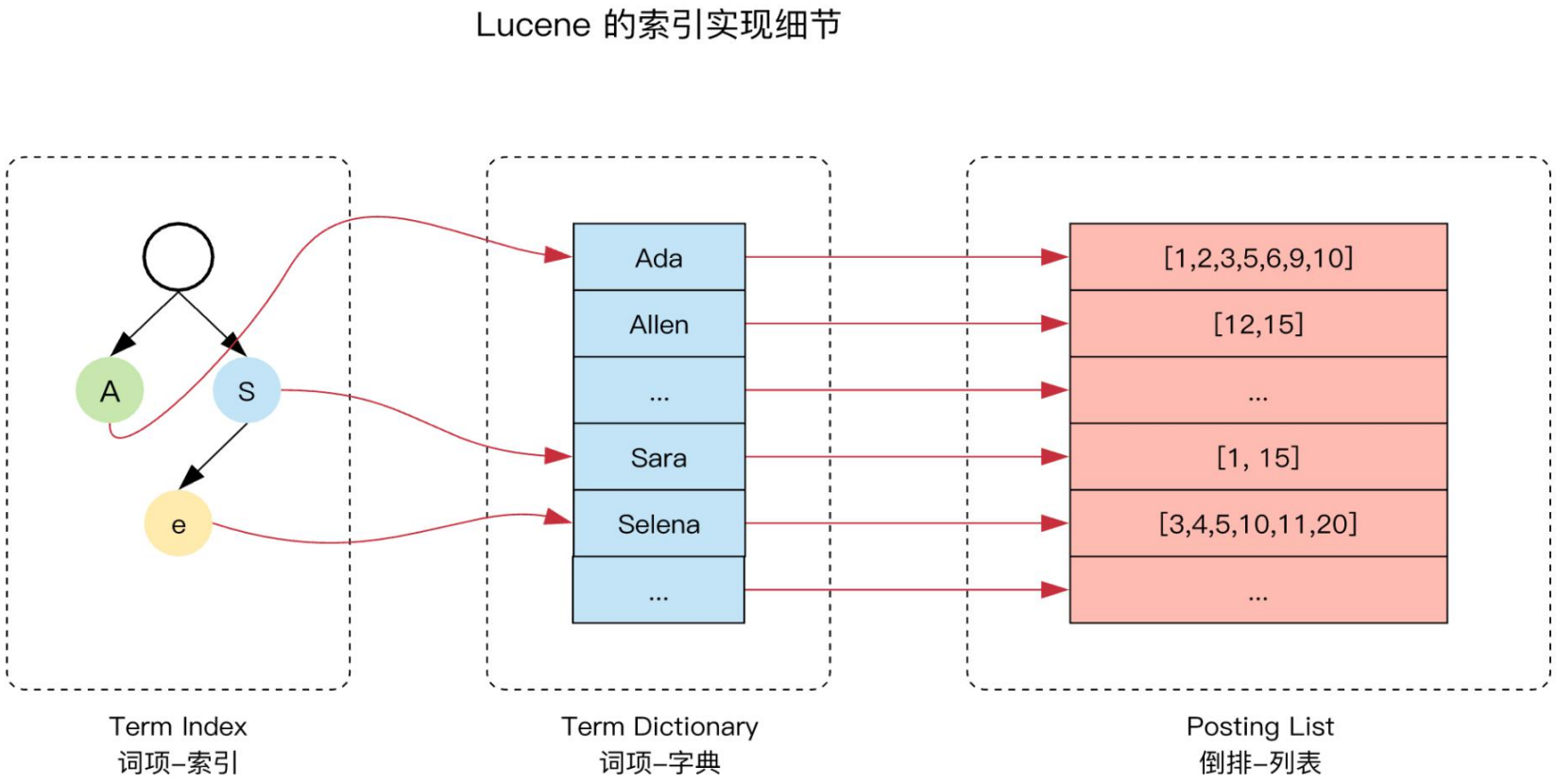
实例：Lucene倒排索引

Lucene 引擎索引文件列表

文件名	扩展名	描述
Segments File	segments.gen, segments_N	段文件，存储提交点的信息
Lock File	write.lock	写锁文件，阻止多个IndexWriter写入同一个文件
Segment Info	.si	段信息文件，存储段的元数据
Compound File	.cfs, .cfe	复合文件，它是一个虚拟文件，由系统中使用较为频繁的索引文...
Fields	.fnm	域文件，存储field信息
Field Index	.fdx	域指针文件，包含到域文件的指针
Field Data	.fdt	域数据文件，包含文档中存储的域数据
Term Dictionary	.tim	项词典，存储项信息
Term Index	.tip	项索引，存储到项词典的索引
Frequencies	.doc	频率，包含每个项以及频率信息
Positions	.pos	位置，存储一个项在索引中的位置信息
Payloads	.pay	载荷，存储额外的预先设置好的元信息如字符偏移或者用户载荷
Norms	.nvd, .nvm	调整因子，编码文档和域的产度及加权
Per-Document Values	.dvd, .dvm	编码额外的打分因子及其他预置的文档信息
Term Vector Index	.tvx	文档向量索引，存储在文档数据文件的偏移
Term Vector Documents	.tvd	包含每个文档的项向量信息
Term Vector Fields	.tvf	包含域级别的项向量信息
Deleted Documents	.del	关于文档被删除的信息

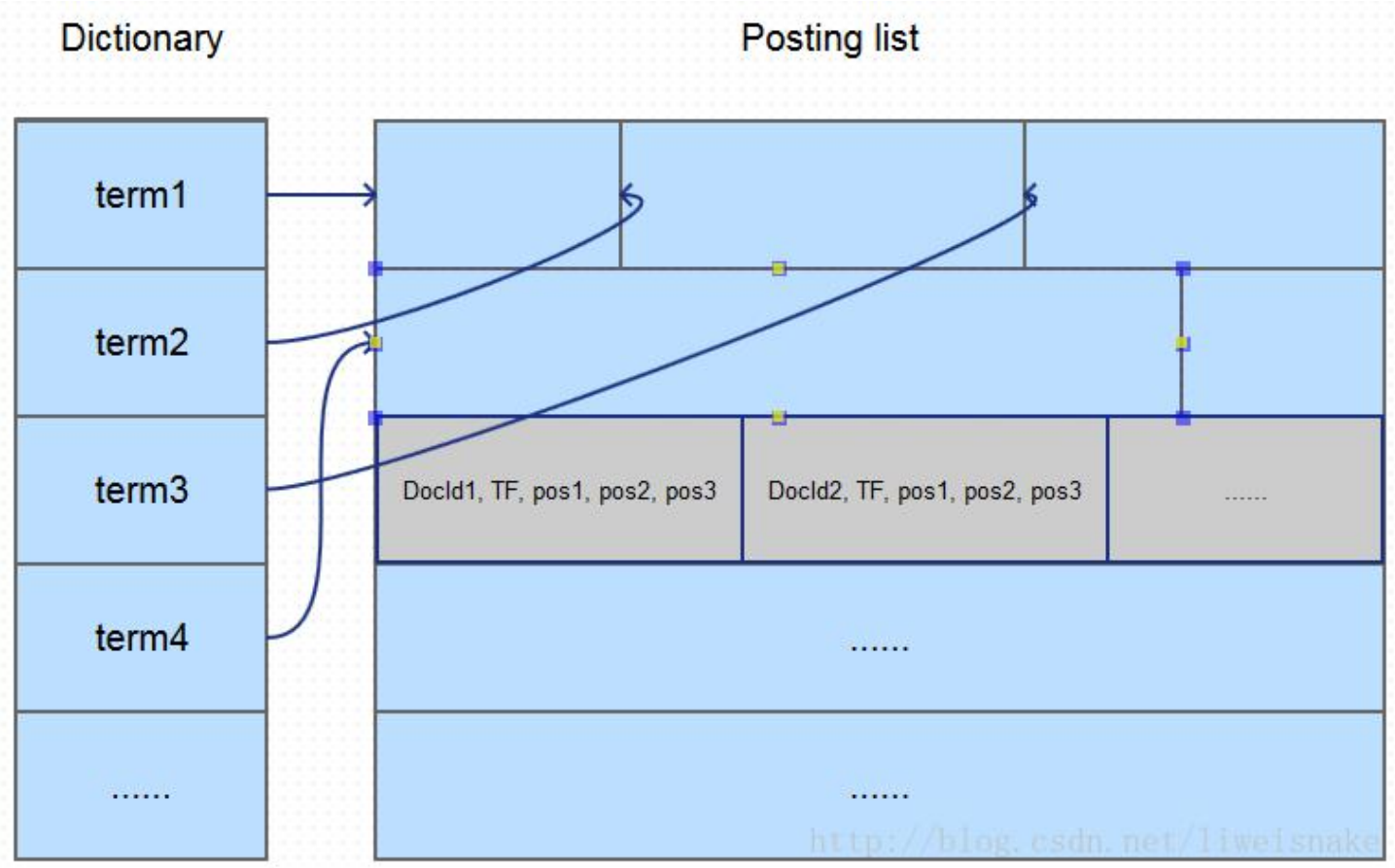
实例：Lucene倒排索引

Lucene 索引实现



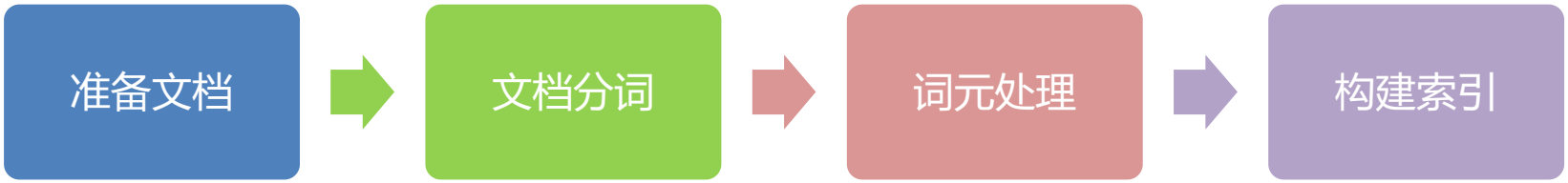
实例：Lucene倒排索引

Lucene 倒排索引存储结构



创建索引大体过程

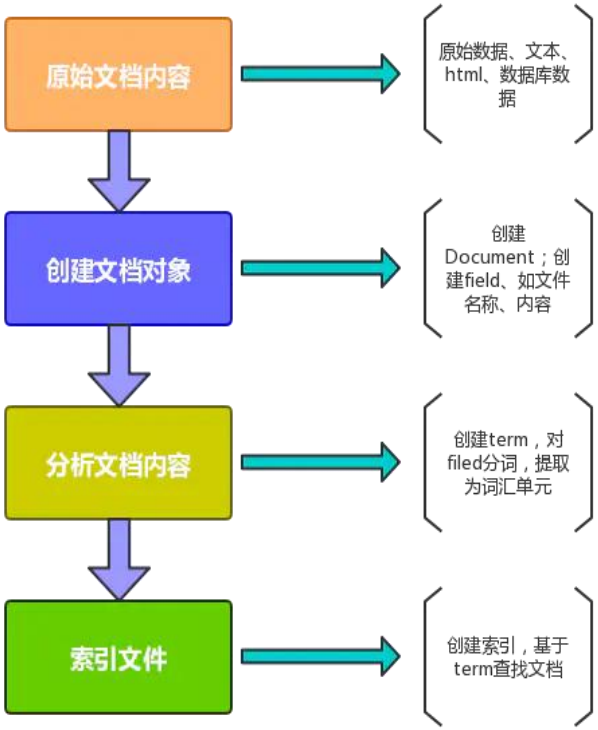
来个更简单易懂的正向与倒排吧

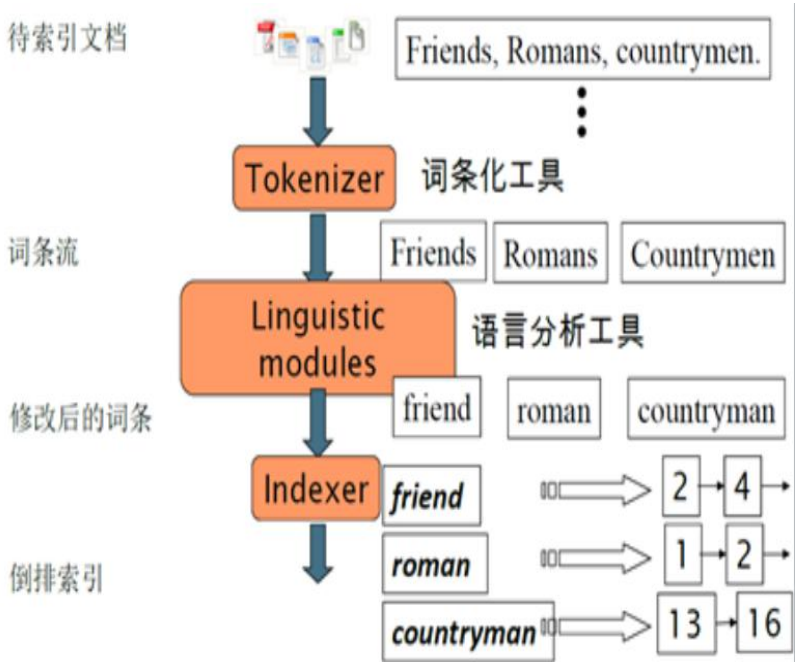


- **准备文档**：收集待处理的文档(Document)
- **文档分词**：将原始文档进行分词处理 (Tokenizer)得到词元(Token)
- **词元处理**：语言组件处理词元 (Token，全角半角转换/变为小写/复数变单数/动词变原型)进行语言学预处理，得到词项(Term)
- **构建索引**：将得到的词项(Term)传递给索引组件(Indexer)建立倒排索引(Posting List):
  1. 利用所得到的Term创建一个字典(Hash、二叉查找树、Tire树)
  2. 对字典进行字母排序
  3. 合并相同词Term成为文档倒排(Posting List)链表

实例：Lucene 索引创建过程

以Lucene作为示例查看构建索引过程





分布式：索引分布式

索引的分布式存储方式对比

文档划分（Document Partitioning）

- 方式：每个服务器存储所有词典，但是只字典的某部分倒排索引和文档数据。
- 优点：可以轻易通过扩展服务器增加文档存储
- 缺点：容易服务器出现冷热不均现象

按单词划分（Term Partitioning）

- 方式：每个服务器存储若干term和其对应所有倒排索引记录
- 优点：存储结构简单方便，更新处理简单，如果是简单单个单词查询比较容易处理
- 缺点：扩展性比较不便，每次扩展都需要调整所有倒排索引记录；服务器出故障会导致该服务器所有term无法搜索；负载均衡会导致冷热不均情况；查询只能一次一个单词，灵活性稍差

概念引入

查询词？检索词？有何区别？

黑

1、查询词

查询词是用户丢在搜索框中进行查询的词

2、检索词

检索词是搜索引擎丢进索引库查询结果时检索的词。

3、查询词→检索词

查询词向检索词的转变中，需要经过分词处理。

全网搜索引擎检索过程

查询的同时搜索引擎在进行快速的计算

黑

查询 → 布尔模型 → 权重计算 → 结果输出

1、布尔模型

最简单也最实用的 0和1

2、权重计算

内容+链接+用户点击，权重算法

3、结果呈现

Url、标题、描述、快照日期、图片

全文搜索基本查询方式

倒排索引检索过程

搜索引擎基本功能：（谷歌/百度搜索为例）

- **布尔逻辑检索**：支持布尔逻辑运算，但检索词和算符之间是否要加空格则不一样，网易的“有道”必须加，而新浪的“爱问”则可以加，也可以不加。（使用 AND/OR/NOT 区分与或非检索）
- **字符串检索/多词检索**：精确检索方式，将检索词用双引号括起来，作为一个完整的字符串进行检索，如“安徽科技学院图书馆”。空格区分多词检索
- **截词检索**：一般搜索引擎都支持，但多提供右截词，符号为“*”。中文搜索引擎中，检索词和“*”所代表的词多构成词组。
- **字段限制**：在搜索引擎中，一律使用前缀限制（=后应加空格）。不同的搜索引擎使用的前缀代码不完全相同，如题名字段，Google使用“TI”，百度使用“Title”
- **其他**：文档类型限制、指定目录(网站)、标题搜索INTITLE、URL范围、引号书名号精确搜索等等

全文搜索：查询过程

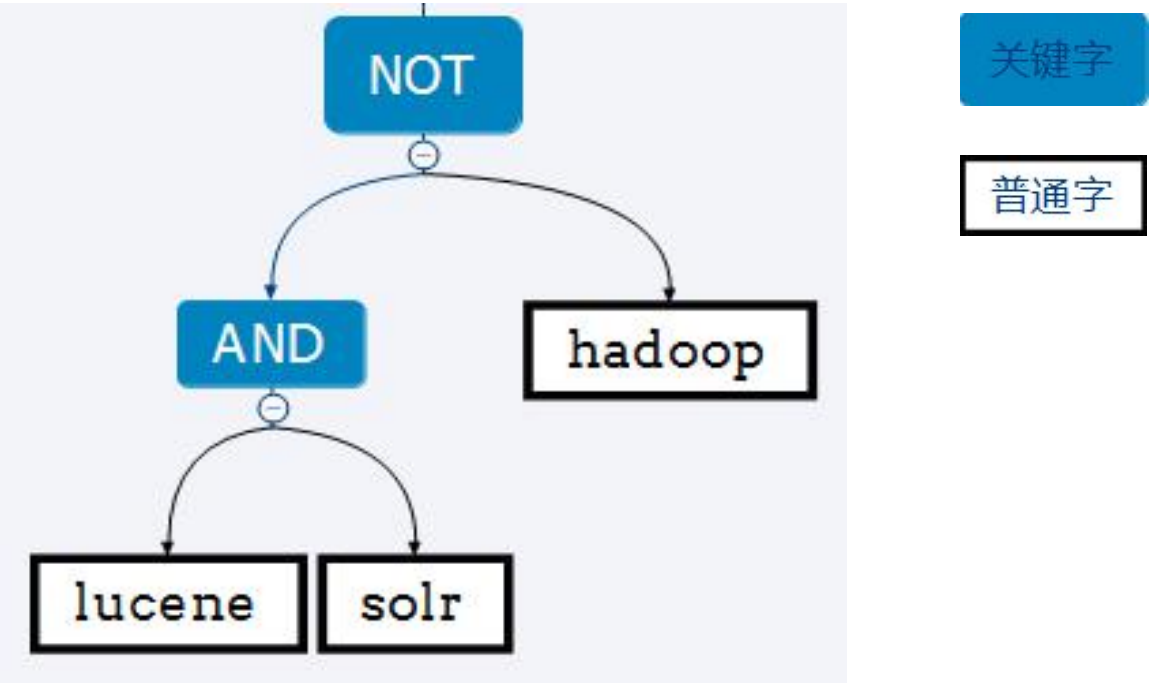
倒排索引检索过程

- 用户输入查询语句
- 对查询语句进行词法分析语法分析及语言处理
  - 词法：单词和关键字
  - 语法分析：根据与语法规则形成语法树
  - 语言处理：和索引过程几乎相同
- 搜索索引得到符合语法树的文档
- 得到查询语句的相关性对结果排序

语法处理成语法树

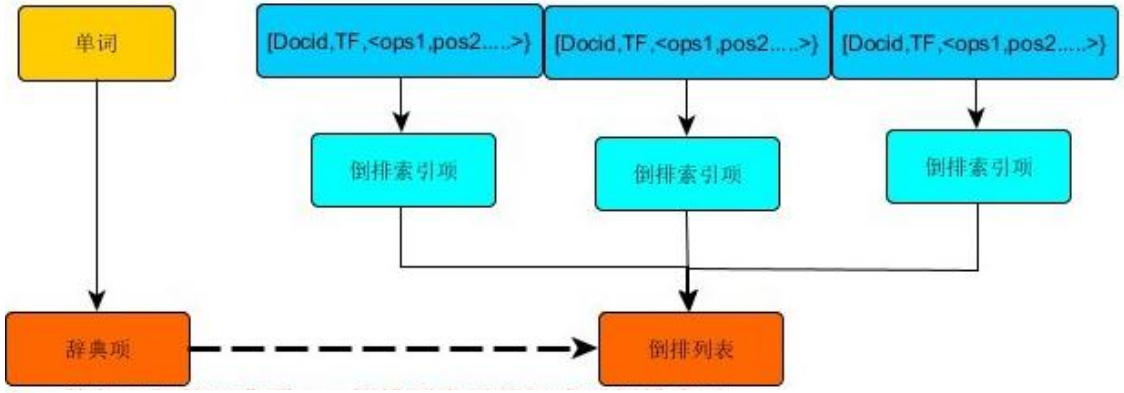
lucence AND solr NOT hadoop语法树

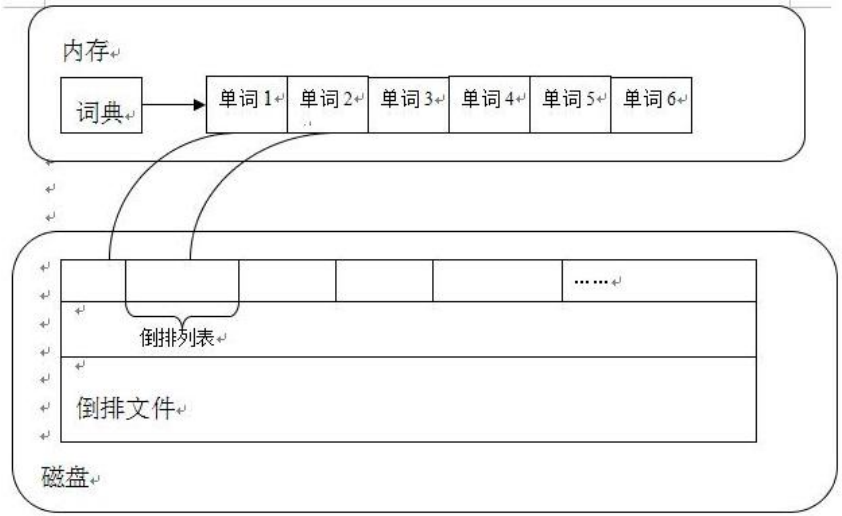
查询：结果中包含 lucene和solr，但是不包含hadoop



从倒排索引中检索

索引存储结构及检索





全文检索排序技术

如何让用户更想要的搜索结果排到前面，相关性计算，权重计算

布尔模型

布尔检索

(Boolean Search)

向量空间模型

(Vector Space Model)

TF-IDF特征权重计算

余弦相似性计算

(Cosine Similarity)

概率检索模型

BM25

BM25F

其他模型

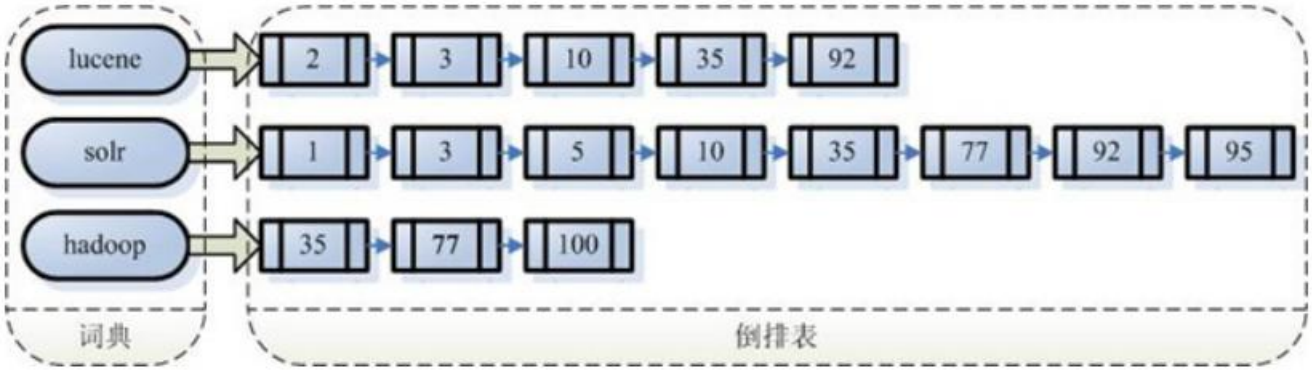
语言模型方法

机器学习模型

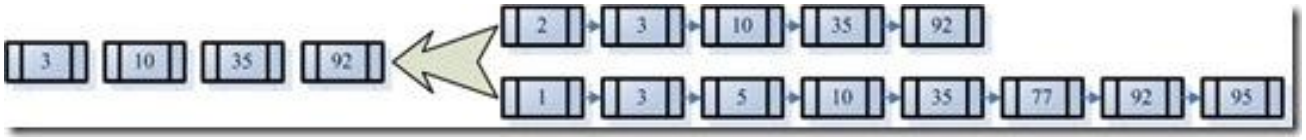
布尔模型：布尔检索

查找：lucene AND solr，布尔模型及索引合并操作

倒排索引的布尔模型操作，合并前：



通过合并链表查出既包含Lucence又包含Solr的文件，合并后：

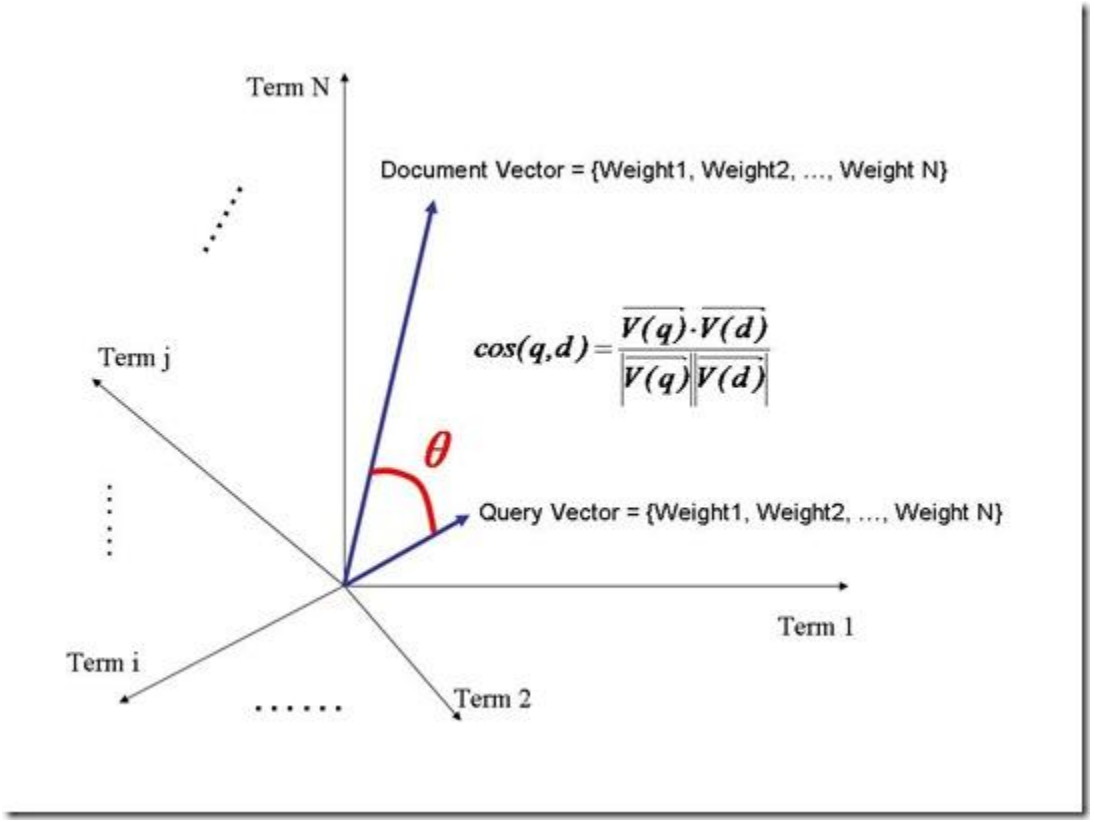


向量空间模型：余弦相似性

向量空间模型（VSM），把相似性问题转成相关性问题，集合权重计算

把对文本内容的处理简化为向量空间中的向量运算，并且它以空间上的相似度表达语义的相似度，简单易用。

流程：文档 -> 向量 -> 使用余弦来计算相似度



向量空间模型：权重计算 TF-IDF

向量空间模型中针对特征向量的权重计算算法：TF-IDF

TF-IDF：字词的重要性随着它在文件中出现的次数成正比增加，但同时会随着它在语料库中出现的频率成反比下降。

$$\text{TF-IDF} = \text{词频(TF)} \times \text{逆文档频率（IDF）}$$

1. TF（词频，Term Freqency）

某个词或短语在一篇文章中出现的次数越多，越相关，这个数字是对 词数(term count)的归一化，以防止它偏向长的文件。（同一个词语在长文件里可能会比短文件有更高的词数，而不管该词语重要与否）

$$\text{词频(TF)} = \frac{\text{某个词在文章中的出现次数}}{\text{该文出现次数最多的词的出现次数}}$$

2. IDF（逆向文件频率，Inverse Document Frequency）

整个文档集合中包含某个词的文档数量越少，这个词越重要。某一特定词语的IDF，可以由总文件数目除以包含该词语之文件的数目，再将得到的商取对数得到。

$$\text{逆文档频率(IDF)} = \log\left(\frac{\text{语料库的文档总数}}{\text{包含该词的文档数} + 1}\right)$$

原版算法：

$$\mathrm{tf}(t,d) = 0.5 + 0.5 \cdot \frac{f_{t,d}}{\max\{f_{t',d} : t' \in d\}}$$

$$\mathrm{idf}(t,D) = \log \frac{N}{|\{d \in D : t \in d\}|}$$

概率模型：Okapi BM25算法

概率模型，二元论方法的实现，BM25算法是TF-IDF算法的升级，与向量空间模型对比，整个算法体系思路不同，虽然实现方式类似

Okapi **BM25** 是到目前为止被认为简单可靠的排名算法之一，源于概率相关模型，不是向量空间模型。Okapi BM25 在 TF-IDF 的基础上增加了两个可调参数，k1 和 b，，分别代表 “词语频率饱和度（term frequency saturation）” 和 “字段长度规约”，

- 词语频率饱和度(term frequency saturation)：在一个文档集合中，最相关的小部分文档中出现term频次的饱和度，不能单一以出现最多term就认为是相关性高的。Okapi BM25 有个 k1 参数，它用于调节饱和度变化的速率。k1 参数的值一般介于 1.2 到 2.0 之间。数值越低则饱和的过程越快速。

- 字段长度归约（Field-length normalization）将文档的长度归约化到全部文档的平均长度上。这对于单字段集合（single-field collections）很有用，可以将不同长度的文档统一到相同的比较条件上。对于双字段集合（例如 “title” 和 “body”）更加有意义，它同样可以将 title 和 body 字段统一到相同的比较条件上。字段长度归约用 b 来表示，它的值在 0 和 1 之间，1 意味着全部归约化，0 则不进行归约化，推荐0.75。字段长度归约是为了让内容较短的字段发挥更大的作用，而内容较长的字段权重相对降低，比如HTML中<title>标签中的内容比<body>标签中权重要高。

概率模型：Okapi BM25算法

概率模型，二元论方法的实现，BM25算法是TF-IDF算法的升级，与向量空间模型对比，整个算法体系思路不同，虽然实现方式类似

Okapi BM25 算法：

可以简单认为是TF-IDF算法的升级，BM25中IDF是TD-IDF中的IDF，f是TD-IDF中的TF，|D|是文档D的长度，avgdl是语料库全部文档的平均长度。k_1是词频饱和度b是字段长度规约，参考上页描述。（K1的值一般在1.2~2.0之间，一般为1.2，b值者用公式计算，一般是0.75）

（字段长度规约公式参考：numTerms是该文档里面关键词所在的字段的词条数量）

NORM(D)= $1/\sqrt{numTerms+1}$

BM25算法：

$$\text{score}(D,Q) = \sum_{i=1}^{n} \text{IDF}(q_i) \cdot \frac{f(q_i, D) \cdot (k_1 + 1)}{f(q_i, D) + k_1 \cdot (1 - b + b \cdot \frac{|D|}{\text{avgdl}})},$$

实例：ElasticSearch 相关性算法

ES中是同时集成了多种算法，包括TF-IDF、向量空间模型余弦相似性、BM25等等，并可选择，默认是TF-IDF 算法优化版

ElasticSearch 默认采用 TF-IDF，因为经过验证简单高效，ElasticSearch 中是 TF-IDF + 字段长度规约 结合使用。也可以选择 Okapi BM25算法，BM25算法新增了词语频率饱和度参数，并且词语频率饱和度和字段长度规约参数可以自己调节。另外也可以使用向量空间模型（ vector space model ）算法。

TF公式：词 t 在文档 d 的词频（ tf ）是该词在文档中出现次数的平方根。

$$TF(t \in d) = \sqrt{frequency}$$

IDF公式：词 t 的逆向文档频率（ idf ）是：索引中文档数量除以所有包含该词的文档数，然后求其对数。

$$IDF(t) = 1 + log\left(\frac{numDocs}{docFreq + 1}\right)$$

字段长度规约：字段长度归一值（ norm ）是字段中词数平方根的倒数。

$$NORM(D) = 1/\sqrt{numTerms + 1}$$

开源搜索引擎

想进行更多学习，可以参考一些开源搜索引擎源码进行学习

开源引擎	官网	开发语言	类型	特点
Lucene **Elasticsearch** **Nutch** **Solr**	http://lucene.apache.org http://www.elasticsearch.org https://nutch.apache.org https://solr.apache.org	Java	ES：分布式引擎 Lucene：索引引擎 Nutch：抓取引擎 Solr：分布式引擎(类ES)	核心引擎基于Lucene 支持分布式 查询语言简单 索引效率高 支持中文分词技术 支持BM25/TF-IDF/空间向量 等相关性策略
Sphinx	http://sphinxsearch.com	C/C++	全文索引引擎	可作为MySQL/PG的全文搜索 引擎
Xapian	https://xapian.org	C/C++	全文索引引擎	倒排检索 布尔查询 相关性打分排序
Meili Search	https://github.com/meilisearch/MeiliSearch	Rust	全文索引引擎	轻量高性能引擎 倒排索引 中文分词排序 相关性排序
Wukong	https://github.com/huichen/wukong	Golang	全文索引引擎	高效索引性能 高性能中文分词 BM25相关计算排序 支持简单分布式
Bleve search	https://github.com/blevesearch/bleve	Golang	全文索引引擎	倒排索引 支持TF-IDF评分 支持多类查询聚合
Riot	https://github.com/go-ego/riot	Golang	全文索引引擎	轻量级高性能引擎 支持中文分词 支持BM25相关性计算 支持分布式搜索

谢谢

Thanks