一些基本概念

接近实时（NRT）

- Elasticsearch 是一个接近实时的搜索平台。这意味着，从索引一个文档直到这个文档能够被搜索到有一个很小的延迟（通常是 1 秒）。

集群（cluster）

- 代表一个集群，集群中有多个节点（node），其中有一个为主节点，这个主节点是可以通过选举产生的，主从节点是对于集群内部来说的。es的一个概念就是去中心化，字面上理解就是无中心节点，这是对于集群外部来说的，因为从外部来看es集群，在逻辑上是个整体，你与任何一个节点的通信和与整个es集群通信是等价的。

索引（index）

- ElasticSearch将它的数据存储在一个或多个索引（index）中。用SQL领域的术语来类比，索引就像数据库，可以向索引写入文档或者从索引中读取文档，并通过ElasticSearch内部使用Lucene将数据写入索引或从索引中检索数据。

文档（document）

- 文档（document）是ElasticSearch中的主要实体。对所有使用ElasticSearch的案例来说，他们最终都可以归结为对文档的搜索。文档由字段构成。

映射（mapping）

- 所有文档写进索引之前都会先进行分析，如何将输入的文本分割为词条、哪些词条又会被过滤，这种行为叫做映射（mapping）。一般由用户自己定义规则。

类型（type）

- 每个文档都有与之对应的类型（type）定义。这允许用户在一个索引中存储多种文档类型，并为不同文档提供类型提供不同的映射。

分片（shards）

- 代表索引分片，es可以把一个完整的索引分成多个分片，这样的好处是可以把一个大的索引拆分成多个，分布到不同的节点上。构成分布式搜索。分片的数量只能在索引创建前指定，并且索引创建后不能更改。5.X默认不能通过配置文件定义分片

副本（replicas）

- 代表索引副本，es可以设置多个索引的副本，副本的作用一是提高系统的容错性，当个某个节点某个分片损坏或丢失时可以从副本中恢复。二是提高es的查询效率，es会自动对搜索请求进行负载均衡。

数据恢复（recovery）

- 代表数据恢复或叫数据重新分布，es在有节点加入或退出时会根据机器的负载对索引分片进行重新分配，挂掉的节点重新启动时也会进行数据恢复。
- GET /_cat/health?v　#可以看到集群状态

数据源（River）

- 代表es的一个数据源，也是其它存储方式（如：数据库）同步数据到es的一个方法。它是以插件方式存在的一个es服务，通过读取river中的数据并把它索引到es中，官方的river有couchDB的，RabbitMQ的，Twitter的，Wikipedia的，river这个功能将会在后面的文件中重点说到。

网关（gateway）

- 代表es索引的持久化存储方式，es默认是先把索引存放到内存中，当内存满了时再持久化到硬盘。当这个es集群关闭再重新启动时就会从gateway中读取索引数据。es支持多种类型的gateway，有本地文件系统（默认），分布式文件系统，Hadoop的HDFS和amazon的s3云存储服务。

自动发现（discovery.zen）

- 代表es的自动发现节点机制，es是一个基于p2p的系统，它先通过广播寻找存在的节点，再通过多播协议来进行节点之间的通信，同时也支持点对点的交互。
- 5.X关闭广播，需要自定义

通信（Transport）

- 代表es内部节点或集群与客户端的交互方式，默认内部是使用tcp协议进行交互，同时它支持http协议（json格式）、thrift、servlet、memcached、zeroMQ等的传输协议（通过插件方式集成）。
- 节点间通信端口默认：9300-9400

分片和复制（shards and replicas）

一个索引可以存储超出单个结点硬件限制的大量数据。比如，一个具有10亿文档的索引占据1TB的磁盘空间，而任一节点可能没有这样大的磁盘空间来存储或者单个节点处理搜索请求，响应会太慢。

为了解决这个问题，Elasticsearch提供了将索引划分成多片的能力，这些片叫做分片。当你创建一个索引的时候，你可以指定你想要的分片的数量。每个分片本身也是一个功能完善并且独立的“索引”，这个“索引”可以被放置到集群中的任何节点上。

分片之所以重要，主要有两方面的原因：

- 允许你水平分割/扩展你的内容容量
- 允许你在分片（位于多个节点上）之上进行分布式的、并行的操作，进而提高性能/吞吐量

至于一个分片怎样分布，它的文档怎样聚合回搜索请求，是完全由Elasticsearch管理的，对于作为用户的你来说，这些都是透明的。

在一个网络/云的环境里，失败随时都可能发生。在某个分片/节点因为某些原因处于离线状态或者消失的情况下，故障转移机制是非常有用且强烈推荐的。为此， Elasticsearch允许你创建分片的一份或多份拷贝，这些拷贝叫做复制分片，或者直接叫复制。

复制之所以重要，有两个主要原因：

- 在分片/节点失败的情况下，复制提供了高可用性。复制分片不与原/主要分片置于同一节点上是非常重要的。因为搜索可以在所有的复制上并行运行，复制可以扩展你的搜索量/吞吐量
- 总之，每个索引可以被分成多个分片。一个索引也可以被复制0次（即没有复制） 或多次。一旦复制了，每个索引就有了主分片（作为复制源的分片）和复制分片（主分片的拷贝）。
- 分片和复制的数量可以在索引创建的时候指定。在索引创建之后，你可以在任何时候动态地改变复制的数量，但是你不能再改变分片的数量。
- 5.X默认5:1　5个主分片，1个复制分片

默认情况下，Elasticsearch中的每个索引分配5个主分片和1个复制。这意味着，如果你的集群中至少有两个节点，你的索引将会有5个主分片和另外5个复制分片（1个完全拷贝），这样每个索引总共就有10个分片。

来自：http://www.cnblogs.com/xiaochina/p/6855591.html

MySQL	Elastic Search
Database	Index
Table	Type
Row	Document
Column	Field
Schema	Mapping
Index	Everything is indexed
SQL	Query DSL
SELECT * FROM table …	GET http://…
UPDATE table SET …	PUT http://…

（1）关系型数据库中的数据库（DataBase），等价于ES中的索引（Index）

（2）一个数据库下面有N张表（Table），等价于1个索引Index下面有N多类型（Type），

（3）一个数据库表（Table）下的数据由多行（ROW）多列（column，属性）组成，等价于1个Type由多个文档（Document）和多Field组成。

（4）在一个关系型数据库里面，schema定义了表、每个表的字段，还有表和字段之间的关系。 与之对应的，在ES中：Mapping定义索引下的Type的字段处理规则，即索引如何建立、索引类型、是否保存原始索引JSON文档、是否压缩原始JSON文档、是否需要分词处理、如何进行分词处理等。

（5）在数据库中的增insert、删delete、改update、查search操作等价于ES中的增PUT/POST、删Delete、改_update、查GET.

Elastic 6.x 版只允许每个 Index 包含一个 Type，7.x 版将会彻底移除 Type。

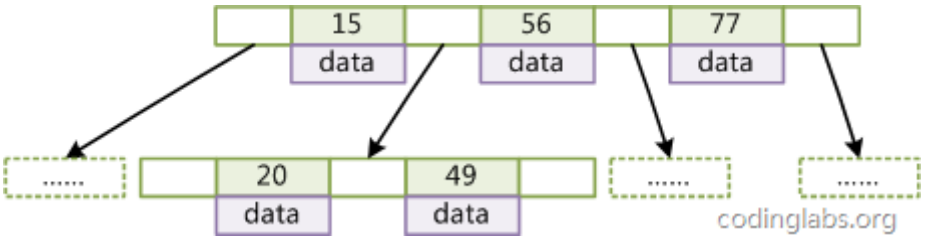
来自:https://blog.csdn.net/laoyang360/article/details/52244917

Elasticsearch是如何做到快速索引的

InfoQ那篇文章里说Elasticsearch使用的**倒排索引**比关系型数据库的B-Tree索引快，为什么呢?

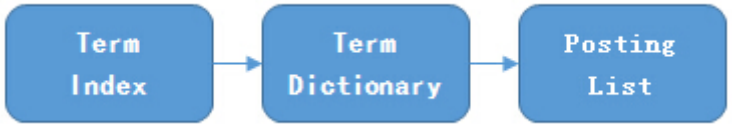
什么是B-Tree索引?

上大学读书时老师教过我们，二叉树查找效率是logN，同时插入新的节点不必移动全部节点，所以用树型结构存储索引，能同时兼顾插入和查询的性能。因此在这个基础上，再结合磁盘的读取特性(顺序读/随机读)，传统关系型数据库采用了B-Tree/B+Tree这样的数据结构:



为了提高查询的效率，减少磁盘寻道次数，将多个值作为一个数组通过连续区间存放，一次寻道读取多个数据，同时也降低树的高度。

什么是倒排索引?



继续上面的例子，假设有这么几条数据(为了简单，去掉about, interests这两个field):

ID	Name	Age	Sex

| 1 | Kate | 24 | Female
| 2 | John | 24 | Male
| 3 | Bill | 29 | Male

ID是Elasticsearch自建的文档id，那么Elasticsearch建立的索引如下：

Name:

Term	Posting List
Kate	1
John	2
Bill	3

Age:

Term	Posting List
24	[1,2]
29	3

Sex:

Term	Posting List
Female	1
Male	[2,3]

Posting List

Elasticsearch分别为每个field都建立了一个倒排索引，Kate, John, 24, Female这些叫term，而[1,2]就是**Posting List**。Posting list就是一个int的数组，存储了所有符合某个term的文档id。

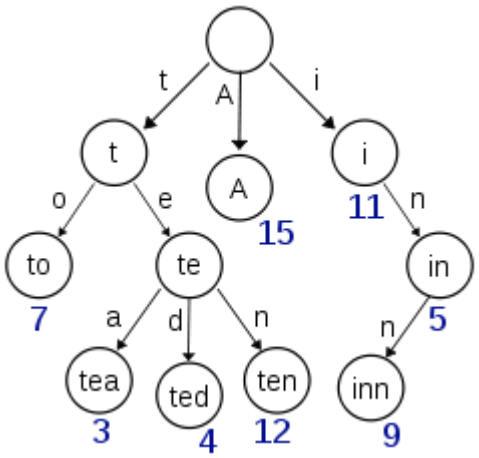
通过posting list这种索引方式似乎可以很快进行查找，比如要找age=24的同学，爱回答问题的小明马上就举手回答：我知道，id是1，2的同学。但是，如果这里有上千万的记录呢？如果是想通过name来查找呢？

Term Dictionary

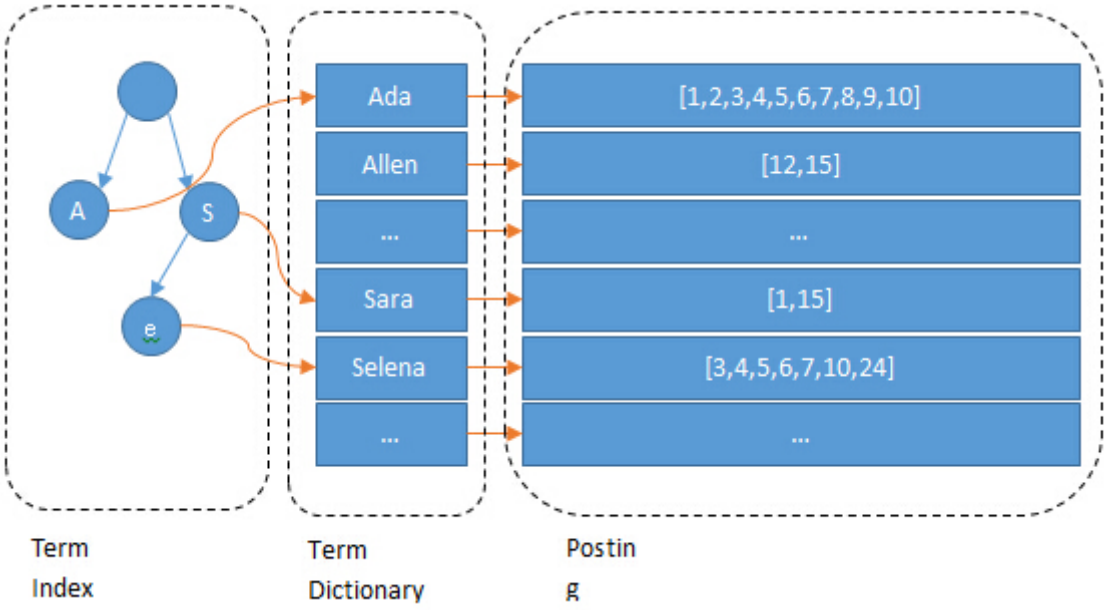
Elasticsearch为了能快速找到某个term，将所有的term排个序，二分法查找term，logN的查找效率，就像通过字典查找一样，这就是**Term Dictionary**。现在再看起来，似乎和传统数据库通过B-Tree的方式类似啊，为什么说比B-Tree的查询快呢？

Term Index

B-Tree通过减少磁盘寻道次数来提高查询性能，Elasticsearch也是采用同样的思路，直接通过内存查找term，不读磁盘，但是如果term太多，term dictionary也会很大，放内存不现实，于是有了**Term Index**，就像字典里的索引页一样，A开头的有哪些term，分别在哪页，可以理解term index是一颗树：

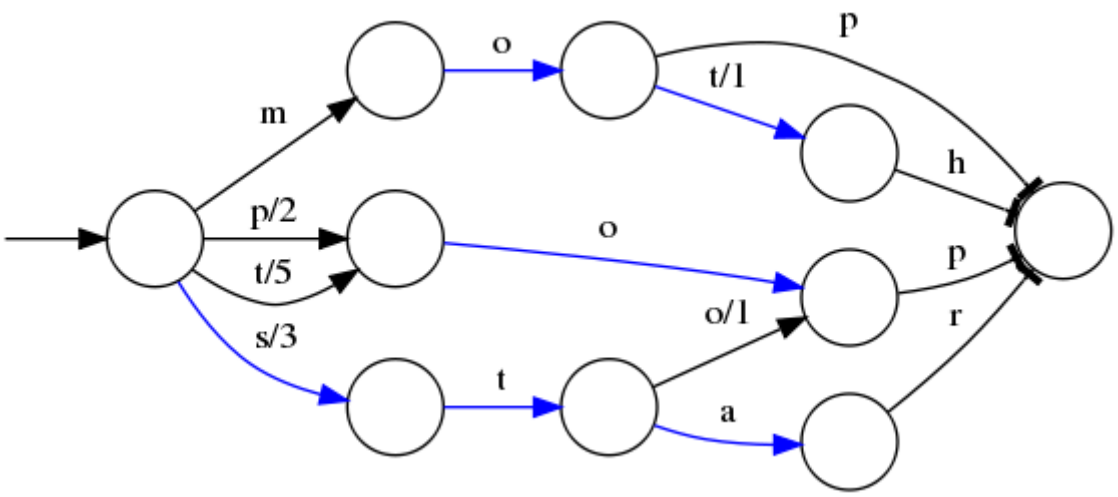


这棵树不会包含所有的term，它包含的是term的一些前缀。通过term index可以快速地定位到term dictionary的某个offset，然后从这个位置再往后顺序查找



所以term index不需要存下所有的term，而仅仅是他们的一些前缀与Term Dictionary的block之间的映射关系，再结合FST(Finite State Transducers)的压缩技术，可以使term index缓存到内存中。从term index查到对应的term dictionary的block位置之后，再去磁盘上找term，大大减少了磁盘随机读的次数。

> FSTs are finite-state machines that **map** a **term (byte sequence)** to an arbitrary **output**.



⭕表示一种状态

-->表示状态的变化过程，上面的字母/数字表示状态变化和权重

将单词分成单个字母通过⭕和-->表示出来，0权重不显示。如果⭕后面出现分支，就标记权重，最后整条路径上的权重加起来就是这个单词对应的序号。

> FSTs are finite-state machines that map a term (**byte sequence**) to an arbitrary output.

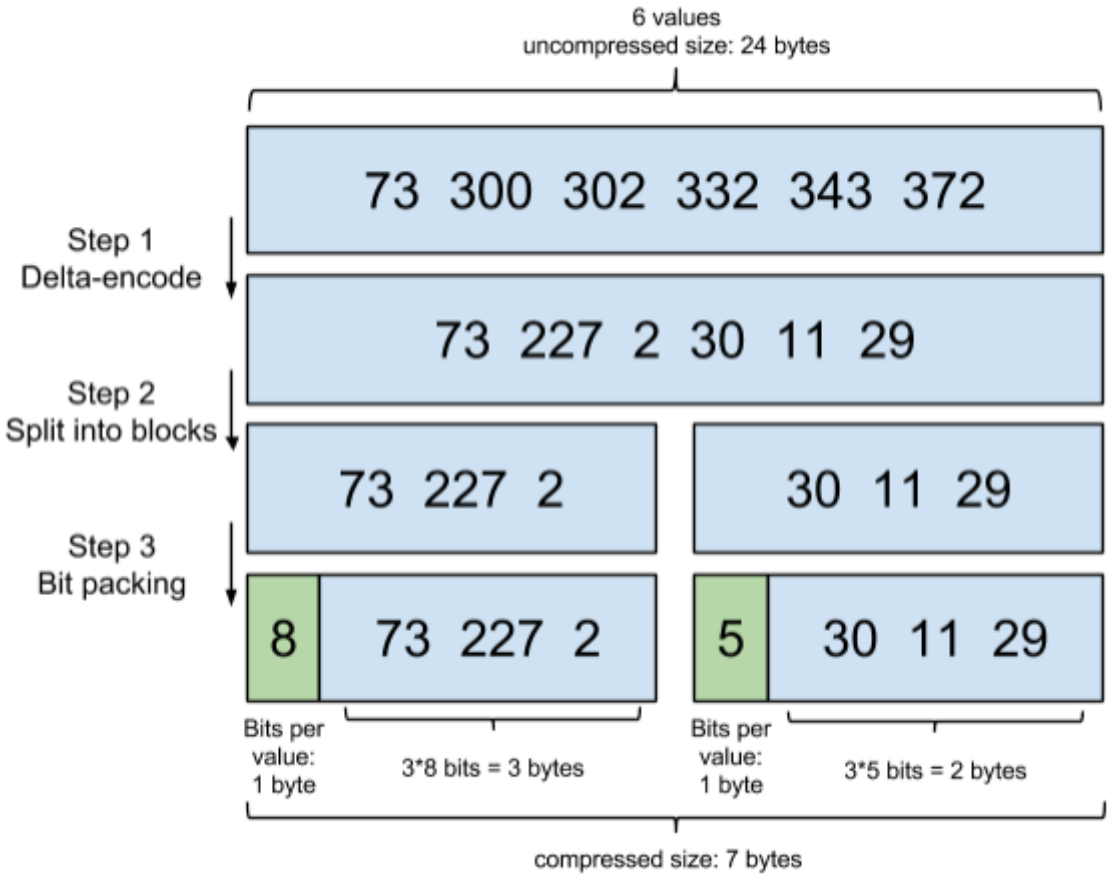
FST以字节的方式存储所有的term，这种压缩方式可以有效的缩减存储空间，使得term index足以放进内存，但这种方式也会导致查找时需要更多的CPU资源。(存储的是种类,而不是内容,用图的路径来表示内容,那就是图算法了,根据权重找路径,懵逼~)

压缩技巧

Elasticsearch里除了上面说到用FST压缩term index外，对posting list也有压缩技巧。我们再看回最开始的例子，如果Elasticsearch需要对同学的性别进行索引(这时传统关系型数据库已经哭晕在厕所……)，会怎样？如果有上千万个同学，而世界上只有男/女这样两个性别，每个posting list都会有至少百万个文档id。 Elasticsearch是如何有效的对这些文档id压缩的呢?

> 增量编码压缩，将大数变小数，按字节存储

首先，Elasticsearch要求posting list是有序的(为了提高搜索的性能，再任性的要求也得满足)，这样做的一个好处是方便压缩，看下面这个图例：



原理就是通过增量，将原来的大数变成小数仅存储增量值，再精打细算按bit排好队，最后通过字节存储，而不是大大咧咧的尽管是2也是用int(4个字节)来存储。(不是很懂最后一行中的绿色底色的8和5是什么意思?)

Roaring bitmaps

说到Roaring bitmaps，就必须先从bitmap说起。Bitmap是一种数据结构，假设有某个posting list：

[1,3,4,7,10]

对应的bitmap就是：

[1,0,1,1,0,0,1,0,0,1]

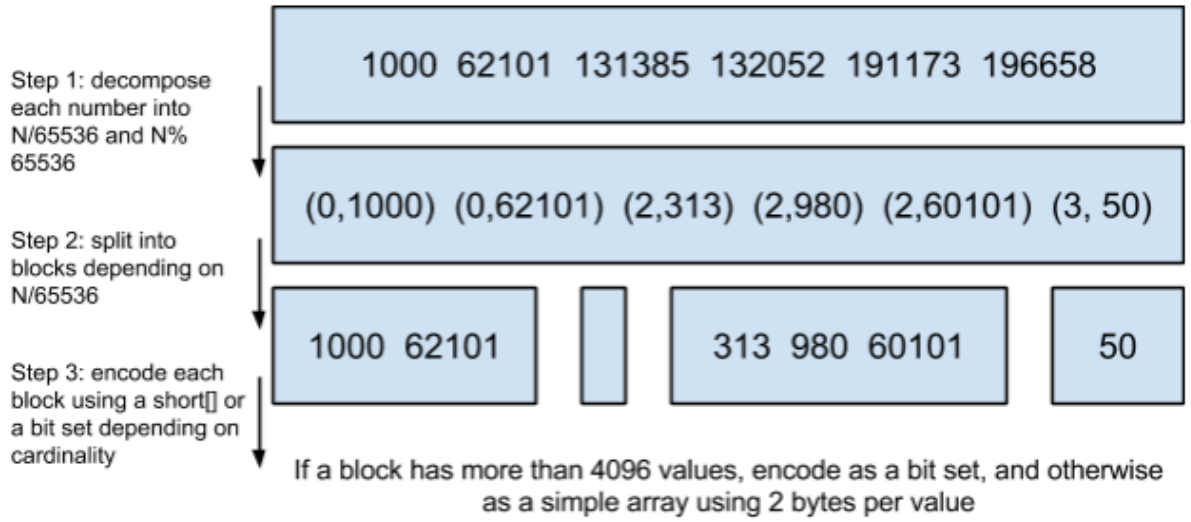
非常直观，用0/1表示某个值是否存在，比如10这个值就对应第10位，对应的bit值是1，这样用一个字节就可以代表8个文档id，(有点像哈希算法)

旧版本(5.0之前)的Lucene就是用这样的方式来压缩的，但这样的压缩方式仍然不够高效，如果有1亿个文档，那么需要12.5MB的存储空间，这仅仅是对应一个索引字段(我们往往会有很多个索引字段)。于是有人想出了Roaring bitmaps这样更高效的数据结构。

Bitmap的缺点是存储空间随着文档个数线性增长，Roaring bitmaps需要打破这个魔咒就一定要用到某些指数特性：

将posting list按照65535为界限分块，比如第一块所包含的文档id范围在0~65535之间，第二块的id范围是65536~131071，以此类推。再用<商，余数>的组合表示每一组

id，这样每组里的id范围都在0~65535内了，剩下的就好办了，既然每组id不会变得无限大，那么我们就可以通过最有效的方式对这里的id存储。



为什么是以65535为界限?"

程序员的世界里除了1024外，65535也是一个经典值，因为它=2^16-1，正好是用2个字节能表示的最大数，一个short的存储单位，注意到上图里的最后一行"If a block has more than 4096 values, encode as a bit set, and otherwise as a simple array using 2 bytes per value"，如果是大块，用节省点用bitset存，小块就豪爽点，2个字节我也不计较了，用一个short[]存着方便。

那为什么用4096来区分大块还是小块呢?

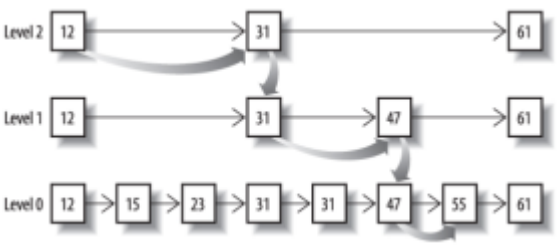
个人理解：都说程序员的世界是二进制的，4096*2bytes = 8192bytes < 1KB, 磁盘一次寻道可以顺序把一个小块的内容都读出来，再大一位就超过1KB了，需要两次读。

联合索引

上面说了半天都是单field索引，如果多个field索引的联合查询，倒排索引如何满足快速查询的要求呢?

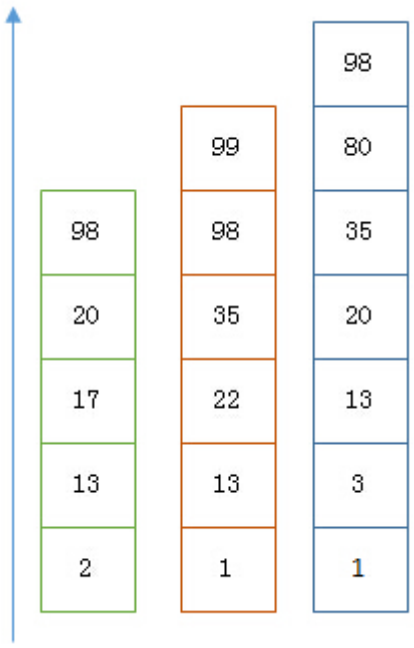
- 利用跳表(Skip list)的数据结构快速做"与"运算，或者
- 利用上面提到的bitset按位"与"

先看看跳表的数据结构：



将一个有序链表level0，挑出其中几个元素到level1及level2，每个level越往上，选出来的指针元素越少，查找时依次从高level往低查找，比如55，先找到level2的31，再找到level1的47，最后找到55，一共3次查找，查找效率和2叉树的效率相当，但也是用了一定的空间冗余来换取的。

假设有下面三个posting list需要联合索引：



如果使用跳表，对最短的posting list中的每个id，逐个在另外两个posting list中查找看是否存在，最后得到交集的结果。

如果使用bitset，就很直观了，直接按位与，得到的结果就是最后的交集。

总结和思考

Elasticsearch的索引思路：

> 将磁盘里的东西尽量搬进内存，减少磁盘随机读取次数(同时也利用磁盘顺序读特性)，结合各种奇技淫巧的压缩算法，用极其苛刻的态度使用内存。

所以，对于使用Elasticsearch进行索引时需要注意：

- 不需要索引的字段，一定要明确定义出来，因为默认是自动建索引的
- 同样的道理，对于String类型的字段，不需要analysis的也需要明确定义出来，因为默认也是会analysis的
- 选择有规律的ID很重要，随机性太大的ID(比如java的UUID)不利于查询

来自:https://www.cnblogs.com/dreamroute/p/8484457.html