

* 官网API手册：<https://www.elastic.co>
* 订阅 Elasticsearch 日报，日报每天会推送一些最近技术动向，以及网络上其他人编写的比较不错的内容，订阅地址：<https://tinyletter.com/elastic-daily>
* 英文社区论坛，不可否认官方英文论坛是最活跃的，你可以在这里看大别人都遇到了什么问题，是怎么解决的，这里有官方的工程师回应：<https://discuss.elastic.co/>
* 英文社区博客，官网的博客大部分是关于新产品的信息，但是也有一些对于底层原理的细致讲解，不可错过。地址：<https://www.elastic.co/blog>
* GitHub PR，其他人对 Elasticsearch 提交的改动，对于实际问题很有参考价值：<https://github.com/elastic/elasticsearch/pulls>
* GitHub issues，其他人的提议和 bug 反馈等：<https://github.com/elastic/elasticsearch/issues>
* 中文社区论坛，国内用户交流比较多，英文不好的同学和一些一般性问题也可以来此提问：[https://elasticsearch.cn](https://elasticsearch.cn/)
* 中文社区的精彩分享，线下活动分享的 PPT 会放在这个地方，非常值得借鉴和学习，不可错过：<https://elasticsearch.cn/slides/>
* 社区电台，勇哥会经常对大规模应用 Elasticsearch 的企业做访谈，聊一聊他们的使用情况，遇到的问题及建议，并且普及一些大家可能不了解的知技术点，很多问题和解决方案都是通用的，可以借鉴和参考。订阅地址： <https://www.ximalaya.com/zhubo/111156131/>

# Elasticsearch学习，请先看这一篇！

## 0. 带着问题上路——ES是如何产生的？

### （1）思考：大规模数据如何检索？

如：当系统数据量上了10亿、100亿条的时候，我们在做系统架构的时候通常会从以下角度去考虑问题：

1）用什么数据库好？(mysql、sybase、oracle、达梦、神通、mongodb、hbase…)

2）如何解决单点故障；(lvs、F5、A10、Zookeep、MQ)

3）如何保证数据安全性；(热备、冷备、异地多活)

4）如何解决检索难题；(数据库代理中间件：mysql-proxy、Cobar、MaxScale等;)

5）如何解决统计分析问题；(离线、近实时)

### （2）传统数据库的应对解决方案

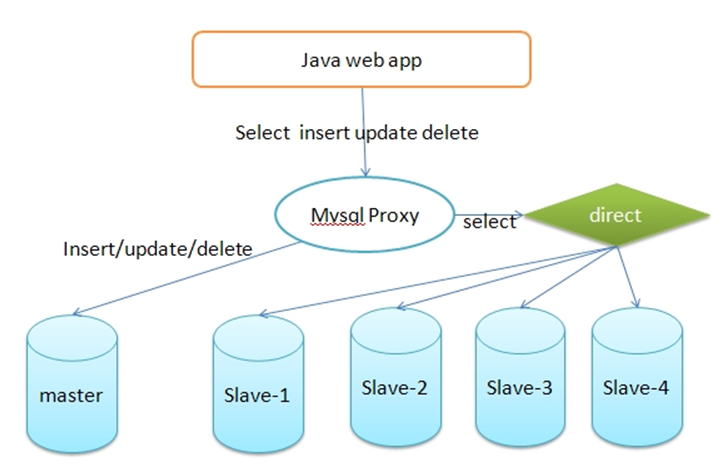
对于关系型数据，我们通常采用以下或类似架构去解决查询瓶颈和写入瓶颈：

解决要点：

1）通过主从备份解决数据安全性问题；

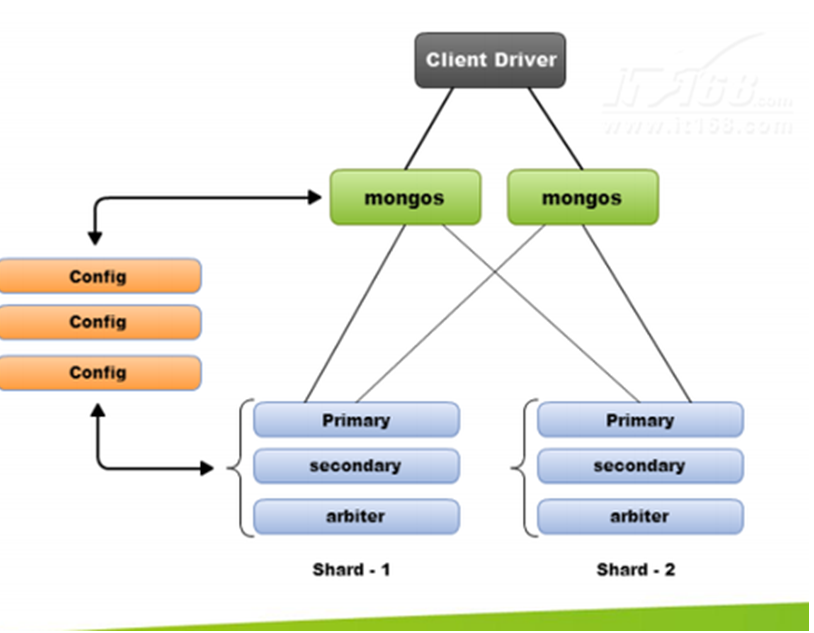
2）通过数据库代理中间件心跳监测，解决单点故障问题；

3）通过代理中间件将查询语句分发到各个slave节点进行查询，并汇总结果



### （3）非关系型数据库的解决方案

对于Nosql数据库，以mongodb为例，其它原理类似：   
解决要点：   
1）通过副本备份保证数据安全性；   
2）通过节点竞选机制解决单点问题；   
3）先从配置库检索分片信息，然后将请求分发到各个节点，最后由路由节点合并汇总结果



### 另辟蹊径——完全把数据放入内存怎么样？

我们知道，完全把数据放在内存中是不可靠的，实际上也不太现实，当我们的数据达到PB级别时，按照每个节点96G内存计算，在内存完全装满的数据情况下，我们需要的机器是：1PB=1024T=1048576G

节点数=1048576/96=10922个

实际上，考虑到数据备份，节点数往往在2.5万台左右。成本巨大决定了其不现实！

从前面讨论我们了解到，把数据放在内存也好，不放在内存也好，都不能完完全全解决问题。

全部放在内存速度问题是解决了，但成本问题上来了。

为解决以上问题，从源头着手分析，通常会从以下方式来寻找方法：

1、存储数据时按有序存储；

2、将数据和索引分离；

3、压缩数据；

这就引出了Elasticsearch。

## 1. ES 基础一网打尽

### 1.1 ES定义

ES=elaticsearch简写， Elasticsearch是一个开源的高扩展的分布式全文检索引擎，它可以近乎实时的存储、检索数据；本身扩展性很好，可以扩展到上百台服务器，处理PB级别的数据。

Elasticsearch也使用Java开发并使用Lucene作为其核心来实现所有索引和搜索的功能，但是它的目的是通过简单的RESTful API来隐藏Lucene的复杂性，从而让全文搜索变得简单。

### 1.2 Lucene与ES关系？

1）Lucene只是一个库。想要使用它，你必须使用Java来作为开发语言并将其直接集成到你的应用中，更糟糕的是，Lucene非常复杂，你需要深入了解检索的相关知识来理解它是如何工作的。

2）Elasticsearch也使用Java开发并使用Lucene作为其核心来实现所有索引和搜索的功能，但是它的目的是通过简单的RESTful API来隐藏Lucene的复杂性，从而让全文搜索变得简单。

### 1.3 ES主要解决问题：

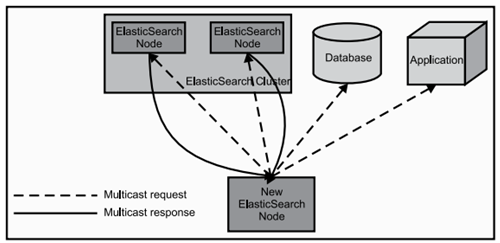
1）检索相关数据；

2）返回统计结果；

3）速度要快。

### 1.4 ES工作原理

当ElasticSearch的节点启动后，它会利用多播(multicast)(或者单播，如果用户更改了配置)寻找集群中的其它节点，并与之建立连接。这个过程如下图所示：



### 1.5 ES核心概念

#### 1）Cluster：集群

ES可以作为一个独立的单个搜索服务器。不过，为了处理大型数据集，实现容错和高可用性，ES可以运行在许多互相合作的服务器上。这些服务器的集合称为集群。

#### 2）Node：节点

形成集群的每个服务器称为节点。

#### 3）Shard：分片

当有大量的文档时，由于内存的限制、磁盘处理能力不足、无法足够快的响应客户端的请求等，一个节点可能不够。这种情况下，数据可以分为较小的分片。每个分片放到不同的服务器上。

当你查询的索引分布在多个分片上时，ES会把查询发送给每个相关的分片，并将结果组合在一起，而应用程序并不知道分片的存在。即：这个过程对用户来说是透明的。

#### 4）Replia：副本

为提高查询吞吐量或实现高可用性，可以使用分片副本。

副本是一个分片的精确复制，每个分片可以有零个或多个副本。ES中可以有许多相同的分片，其中之一被选择更改索引操作，这种特殊的分片称为主分片。

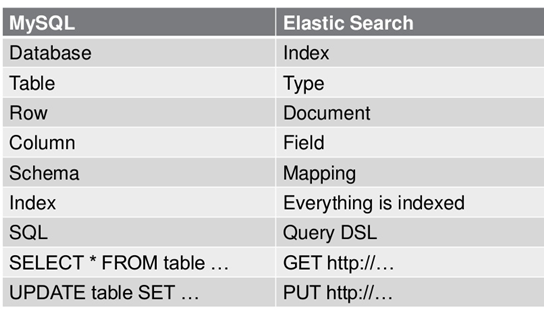
当主分片丢失时，如：该分片所在的数据不可用时，集群将副本提升为新的主分片。

#### 5）全文检索

全文检索就是对一篇文章进行索引，可以根据关键字搜索，类似于mysql里的like语句。

全文索引就是把内容根据词的意义进行分词，然后分别创建索引，例如”你们的激情是因为什么事情来的” 可能会被分词成：“你们“，”激情“，“什么事情“，”来“ 等token，这样当你搜索“你们” 或者 “激情” 都会把这句搜出来。

### 1.6 ES数据架构的主要概念（与关系数据库Mysql对比）



（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。

### 1.7 ELK是什么？

ELK=elasticsearch+Logstash+kibana

elasticsearch：后台分布式存储以及全文检索

logstash: 日志加工、“搬运工”

kibana：数据可视化展示。

ELK架构为数据分布式存储、可视化查询和日志解析创建了一个功能强大的管理链。 三者相互配合，取长补短，共同完成分布式大数据处理工作。

## 2. ES特点和优势

1）分布式实时文件存储，可将每一个字段存入索引，使其可以被检索到。

2）实时分析的分布式搜索引擎。

分布式：索引分拆成多个分片，每个分片可有零个或多个副本。集群中的每个数据节点都可承载一个或多个分片，并且协调和处理各种操作；

负载再平衡和路由在大多数情况下自动完成。

3）可以扩展到上百台服务器，处理PB级别的结构化或非结构化数据。也可以运行在单台PC上（已测试）

4）支持插件机制，分词插件、同步插件、Hadoop插件、可视化插件等。

## 3、ES性能

### 3.1 性能结果展示

（1）硬件配置：

CPU 16核 AuthenticAMD

内存 总量：32GB

硬盘 总量：500GB 非SSD

（2）在上述硬件指标的基础上测试性能如下：

1）平均索引吞吐量： 12307docs/s（每个文档大小：40B/docs）

2）平均CPU使用率： 887.7%（16核，平均每核：55.48%）

3）构建索引大小： 3.30111 GB

4）总写入量： 20.2123 GB

5）测试总耗时： 28m 54s.

### 3.2 性能esrally工具（推荐）

使用参考：http://blog.csdn.net/laoyang360/article/details/52155481

## 4、为什么要用ES？

### 4.1 ES国内外使用优秀案例

1） 2013年初，GitHub抛弃了Solr，采取ElasticSearch 来做PB级的搜索。“GitHub使用ElasticSearch搜索20TB的数据，包括13亿文件和1300亿行代码”。

2）维基百科：启动以elasticsearch为基础的核心搜索架构。

3）SoundCloud：“SoundCloud使用ElasticSearch为1.8亿用户提供即时而精准的音乐搜索服务”。

4）百度：百度目前广泛使用ElasticSearch作为文本数据分析，采集百度所有服务器上的各类指标数据及用户自定义数据，通过对各种数据进行多维分析展示，辅助定位分析实例异常或业务层面异常。目前覆盖百度内部20多个业务线（包括casio、云分析、网盟、预测、文库、直达号、钱包、风控等），单集群最大100台机器，200个ES节点，每天导入30TB+数据。

### 4.2 我们也需要

实际项目开发实战中，几乎每个系统都会有一个搜索的功能，当搜索做到一定程度时，维护和扩展起来难度就会慢慢变大，所以很多公司都会把搜索单独独立出一个模块，用ElasticSearch等来实现。

近年ElasticSearch发展迅猛，已经超越了其最初的纯搜索引擎的角色，现在已经增加了数据聚合分析（aggregation）和可视化的特性，如果你有数百万的文档需要通过关键词进行定位时，ElasticSearch肯定是最佳选择。当然，如果你的文档是JSON的，你也可以把ElasticSearch当作一种“NoSQL数据库”， 应用ElasticSearch数据聚合分析（aggregation）的特性，针对数据进行多维度的分析。

【知乎：热酷架构师潘飞】ES在某些场景下替代传统DB

个人以为Elasticsearch作为内部存储来说还是不错的，效率也基本能够满足，在某些方面替代传统DB也是可以的，前提是你的业务不对操作的事性务有特殊要求；而权限管理也不用那么细，因为ES的权限这块还不完善。

由于我们对ES的应用场景仅仅是在于对某段时间内的数据聚合操作，没有大量的单文档请求（比如通过userid来找到一个用户的文档，类似于NoSQL的应用场景），所以能否替代NoSQL还需要各位自己的测试。

如果让我选择的话，我会尝试使用ES来替代传统的NoSQL，因为它的横向扩展机制太方便了。

## 5. ES的应用场景是怎样的？

通常我们面临问题有两个：

1）新系统开发尝试使用ES作为存储和检索服务器；

2）现有系统升级需要支持全文检索服务，需要使用ES。

以上两种架构的使用，以下链接进行详细阐述。

http://blog.csdn.net/laoyang360/article/details/52227541

一线公司ES使用场景：

1）新浪ES 如何分析处理32亿条实时日志 http://dockone.io/article/505

2）阿里ES 构建挖财自己的日志采集和分析体系 http://afoo.me/columns/tec/logging-platform-spec.html

3）有赞ES 业务日志处理 http://tech.youzan.com/you-zan-tong-ri-zhi-ping-tai-chu-tan/

4）ES实现站内搜索 http://www.wtoutiao.com/p/13bkqiZ.html

## 6. 如何部署ES？

### 6.1 ES部署（无需安装）

1）零配置，开箱即用

2）没有繁琐的安装配置

3）java版本要求：最低1.7

我使用的1.8

[root@laoyang config\_lhy]# echo $JAVA\_HOME

/opt/jdk1.8.0\_91

4）下载地址：

https://download.elastic.co/elasticsearch/release/org/elasticsearch/distribution/zip/elasticsearch/2.3.5/elasticsearch-2.3.5.zip

5）启动

cd /usr/local/elasticsearch-2.3.5

./bin/elasticsearch

bin/elasticsearch -d(后台运行)

### 6.2 ES必要的插件

必要的Head、kibana、IK（中文分词）、graph等插件的详细安装和使用。

http://blog.csdn.net/column/details/deep-elasticsearch.html

### 6.3 ES windows下一键安装

自写bat脚本实现windows下一键安装。

1）一键安装ES及必要插件（head、kibana、IK、logstash等）

2）安装后以服务形式运行ES。

3）比自己摸索安装节省至少2小时时间，效率非常高。

脚本说明：

http://blog.csdn.net/laoyang360/article/details/51900235

## 7. ES对外接口（开发人员关注）

1）JAVA API接口

http://www.ibm.com/developerworks/library/j-use-elasticsearch-java-apps/index.html

2）RESTful API接口

常见的增、删、改、查操作实现：

http://blog.csdn.net/laoyang360/article/details/51931981

## 8.ES遇到问题怎么办？

1）国外：https://discuss.elastic.co/

2）国内：http://elasticsearch.cn/

参考：

[1] http://www.tuicool.com/articles/7fueUbb

[2] http://zhaoyanblog.com/archives/495.html

[3]《Elasticsearch服务器开发》

[4]《实战Elasticsearch、Logstash、Kibana》

[5]《Elasticsearch In Action》

[6]《某ES大牛PPT》

## 9、还有吗？

《死磕 Elasticsearch 方法论》：普通程序员高效精进的 10 大狠招！（免费完整版）

https://blog.csdn.net/laoyang360/article/details/79293493

# Elasticsearch索引原理

## 介绍

*Elasticsearch 是一个分布式可扩展的实时搜索和分析引擎.*

Elasticsearch 是一个建立在全文搜索引擎 Apache Lucene(TM) 基础上的搜索引擎. 当然 Elasticsearch 并不仅仅是 Lucene 那么简单，它不仅包括了全文搜索功能，还可以进行以下工作:

* 分布式实时文件存储，并将每一个字段都编入索引，使其可以被搜索。
* 实时分析的分布式搜索引擎。
* 可以扩展到上百台服务器，处理PB级别的结构化或非结构化数据。

### 基本概念

先说Elasticsearch的文件存储，Elasticsearch是面向文档型[**数据库**](http://lib.csdn.net/base/mysql)，一条数据在这里就是一个文档，用JSON作为文档序列化的格式，比如下面这条用户数据：

{

"name" : "John",

"sex" : "Male",

"age" : 25,

"birthDate": "1990/05/01",

"about" : "I love to go rock climbing",

"interests": [ "sports", "music" ]

}

用[**MySQL**](http://lib.csdn.net/base/mysql)这样的数据库存储就会容易想到建立一张User表，有balabala的字段等，在Elasticsearch里这就是一个*文档*，当然这个文档会属于一个User的*类型*，各种各样的类型存在于一个*索引*当中。这里有一份简易的将Elasticsearch和关系型数据术语对照表:

关系数据库 ⇒ 数据库 ⇒ 表 ⇒ 行 ⇒ 列(Columns)

Elasticsearch ⇒ 索引 ⇒ 类型 ⇒ 文档 ⇒ 字段(Fields)

一个 Elasticsearch 集群可以包含多个索引(数据库)，也就是说其中包含了很多类型(表)。这些类型中包含了很多的文档(行)，然后每个文档中又包含了很多的字段(列)。

Elasticsearch的交互，可以使用[**Java**](http://lib.csdn.net/base/java)API，也可以直接使用HTTP的Restful API方式，比如我们打算插入一条记录，可以简单发送一个HTTP的请求：

PUT /megacorp/employee/1

{

"name" : "John",

"sex" : "Male",

"age" : 25,

"about" : "I love to go rock climbing",

"interests": [ "sports", "music" ]

}

更新，查询也是类似这样的操作，具体操作手册可以参见[Elasticsearch权威指南](http://www.learnes.net/data/README.html)

### 索引

Elasticsearch最关键的就是提供强大的索引能力了，其实InfoQ的这篇[时间序列数据库的秘密(2)——索引](http://www.infoq.com/cn/articles/database-timestamp-02?utm_source=infoq&utm_medium=related_content_link&utm_campaign=relatedContent_articles_clk)写的非常好，我这里也是围绕这篇结合自己的理解进一步梳理下，也希望可以帮助大家更好的理解这篇文章。

Elasticsearch索引的精髓：*一切设计都是为了提高搜索的性能*

另一层意思：为了提高搜索的性能，难免会牺牲某些其他方面，比如插入/更新，否则其他数据库不用混了:)

前面看到往Elasticsearch里插入一条记录，其实就是直接PUT一个json的对象，这个对象有多个fields，比如上面例子中的*name, sex, age, about, interests*，那么在插入这些数据到Elasticsearch的同时，Elasticsearch还默默的为这些字段建立索引–倒排索引，因为Elasticsearch最核心功能是搜索。

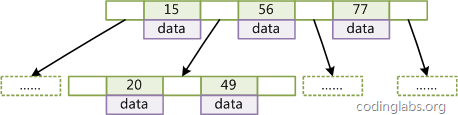
#### Elasticsearch是如何做到快速索引的

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？**

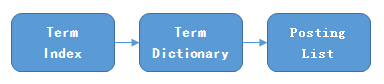
上大学读书时老师教过我们，二叉树查找效率是logN，同时插入新的节点不必移动全部节点，所以用树型结构存储索引，能同时兼顾插入和查询的性能。

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**](http://lib.csdn.net/base/datastructure)：

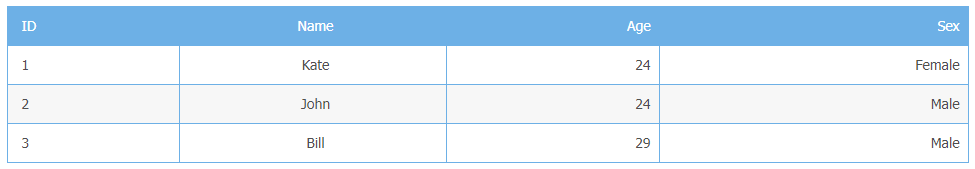


为了提高查询的效率，减少磁盘寻道次数，将多个值作为一个数组通过连续区间存放，一次寻道读取多个数据，同时也降低树的高度。

**什么是倒排索引？**

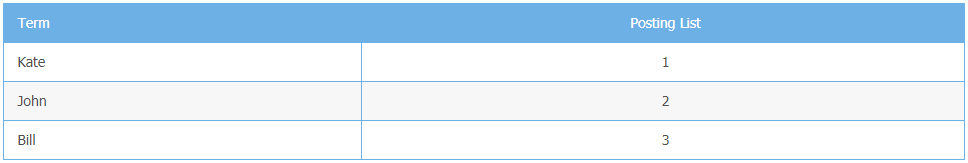


继续上面的例子，假设有这么几条数据(为了简单，去掉about, interests这两个field):

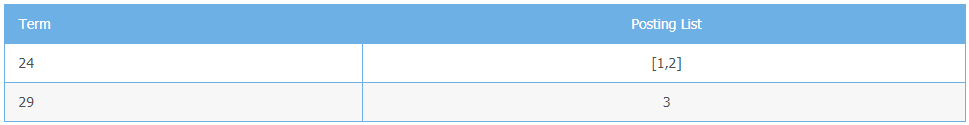


ID是Elasticsearch自建的文档id，那么Elasticsearch建立的索引如下:

**Name:**



**Age:**



**Sex:**



**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。

看到这里，不要认为就结束了，精彩的部分才刚开始…

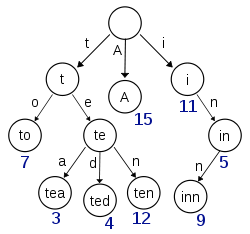
通过posting list这种索引方式似乎可以很快进行查找，比如要找age=24的同学，爱回答问题的小明马上就举手回答：我知道，id是1，2的同学。但是，如果这里有上千万的记录呢？如果是想通过name来查找呢？

**Term Dictionary**

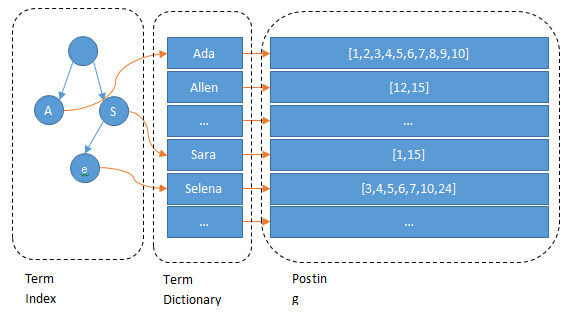
Elasticsearch为了能快速找到某个term，将所有的term排个序，二分法查找term，logN的查找效率，就像通过字典查找一样，这就是**Term Dictionary**。现在再看起来，似乎和传统数据库通过B-Tree的方式类似啊，为什么说比B-Tree的查询快呢？

**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。



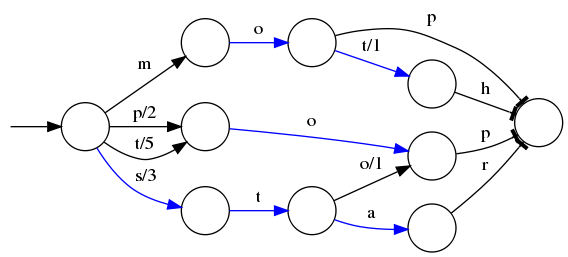
所以term index不需要存下所有的term，而仅仅是他们的一些前缀与Term Dictionary的block之间的映射关系，再结合FST(Finite State Transducers)的压缩技术，可以使term index缓存到内存中。从term index查到对应的term dictionary的block位置之后，再去磁盘上找term，大大减少了磁盘随机读的次数。

这时候爱提问的小明又举手了:”那个FST是神马东东啊?”

一看就知道小明是一个上大学读书的时候跟我一样不认真听课的孩子，数据结构老师一定讲过什么是FST。但没办法，我也忘了，这里再补下课：

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

假设我们现在要将mop, moth, pop, star, stop and top(term index里的term前缀)映射到序号：0，1，2，3，4，5(term dictionary的block位置)。最简单的做法就是定义个Map<String, Integer>，大家找到自己的位置对应入座就好了，但从内存占用少的角度想想，有没有更优的办法呢？答案就是：**FST**([理论依据在此，但我相信99%的人不会认真看完的](http://www.cs.nyu.edu/~mohri/pub/fla.pdf))



⭕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。

后面的更精彩，看累了的同学可以喝杯咖啡……

**压缩技巧**

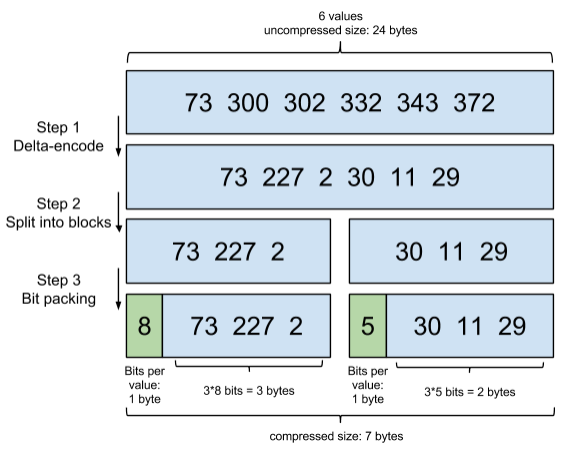
Elasticsearch里除了上面说到用FST压缩term index外，对posting list也有压缩技巧。 小明喝完咖啡又举手了:”posting list不是已经只存储文档id了吗？还需要压缩？”

嗯，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？

**Frame Of Reference**

*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：



如果数学不是体育老师教的话，还是比较容易看出来这种压缩技巧的。

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。

**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]

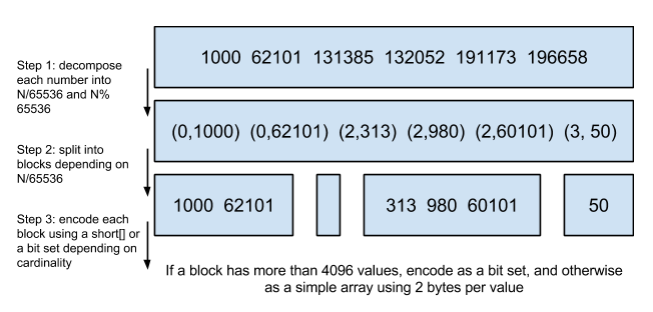
对应的bitmap就是：

[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，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。

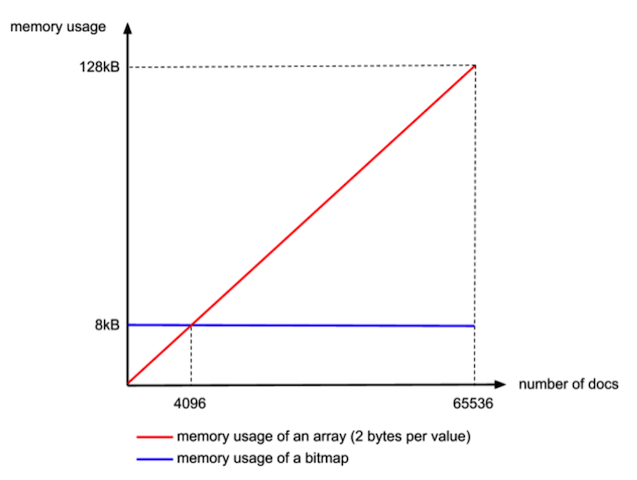


细心的小明这时候又举手了:”为什么是以65535为界限?”

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来区分采用数组还是bitmap的阀值呢？

这个是从内存大小考虑的，当block块里元素超过4096后，用bitmap更剩空间： 采用bitmap需要的空间是恒定的: 65536/8 = 8192bytes 而如果采用short[]，所需的空间是: 2\*N(N为数组元素个数) 小明手指一掐N=4096刚好是边界:

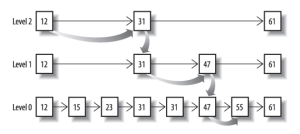


联合索引

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？

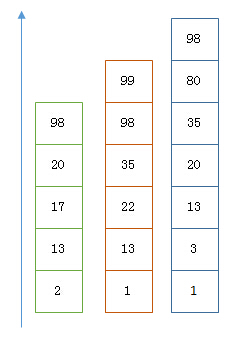
* 利用跳表(Skip list)的数据结构快速做“与”运算，或者
* 利用上面提到的bitset按位“与”

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***](http://lib.csdn.net/base/datastructure)*，用及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

关于最后一点，个人认为有多个因素：

其中一个（也许不是最重要的）因素：上面看到的压缩算法，都是对Posting list里的大量ID进行压缩的，那如果ID是顺序的，或者是有公共前缀等具有一定规律性的ID，压缩比会比较高；

另外一个因素：可能是最影响查询性能的，应该是最后通过Posting list里的ID到磁盘中查找Document信息的那步，因为Elasticsearch是分Segment存储的，根据ID这个大范围的Term定位到Segment的效率直接影响了最后查询的性能，如果ID是有规律的，可以快速跳过不包含该ID的Segment，从而减少不必要的磁盘读次数，具体可以参考这篇[如何选择一个高效的全局ID方案](http://blog.mikemccandless.com/2014/05/choosing-fast-unique-identifier-uuid.html)（评论也很精彩）