https://mp.weixin.qq.com/s/tq3zMbs-ZmSK-trprq82gg

# 「扫盲」 Elasticsearch

Elasticsearch is a real-time, distributed storage, search, and analytics engine

Elasticsearch 是一个**实时**的**分布式存储、搜索、分析**的引擎。

介绍那儿有几个关键字：

## 为什么要用Elasticsearch

在学习一项技术之前，必须先要了解为什么要使用这项技术。所以，为什么要使用Elasticsearch呢？我们在日常开发中，**数据库**也能做到（实时、存储、搜索、分析）。

相对于数据库，Elasticsearch的强大之处就是可以**模糊查询**。

有的同学可能就会说：我数据库怎么就不能模糊查询了？？我反手就给你写一个SQL：

select \* from user where name like '%公众号Java3y%'

这不就可以把**公众号Java3y**相关的内容搜索出来了吗？

的确，这样做的确可以。但是要明白的是：name like %Java3y%这类的查询是不走**索引**的，不走索引意味着：只要你的数据库的量很大（1亿条），你的查询肯定会是**秒**级别的

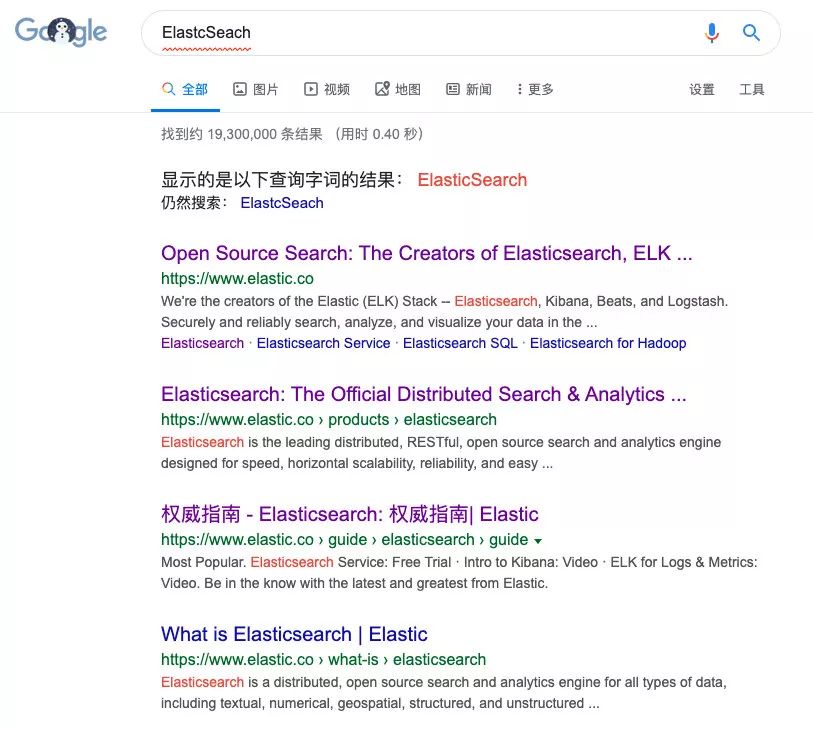
如果对数据库索引还不是很了解的同学，建议复看一下我以前的文章。我觉得我当时写得还不赖（哈哈哈）

GitHub搜关键字：”索引“

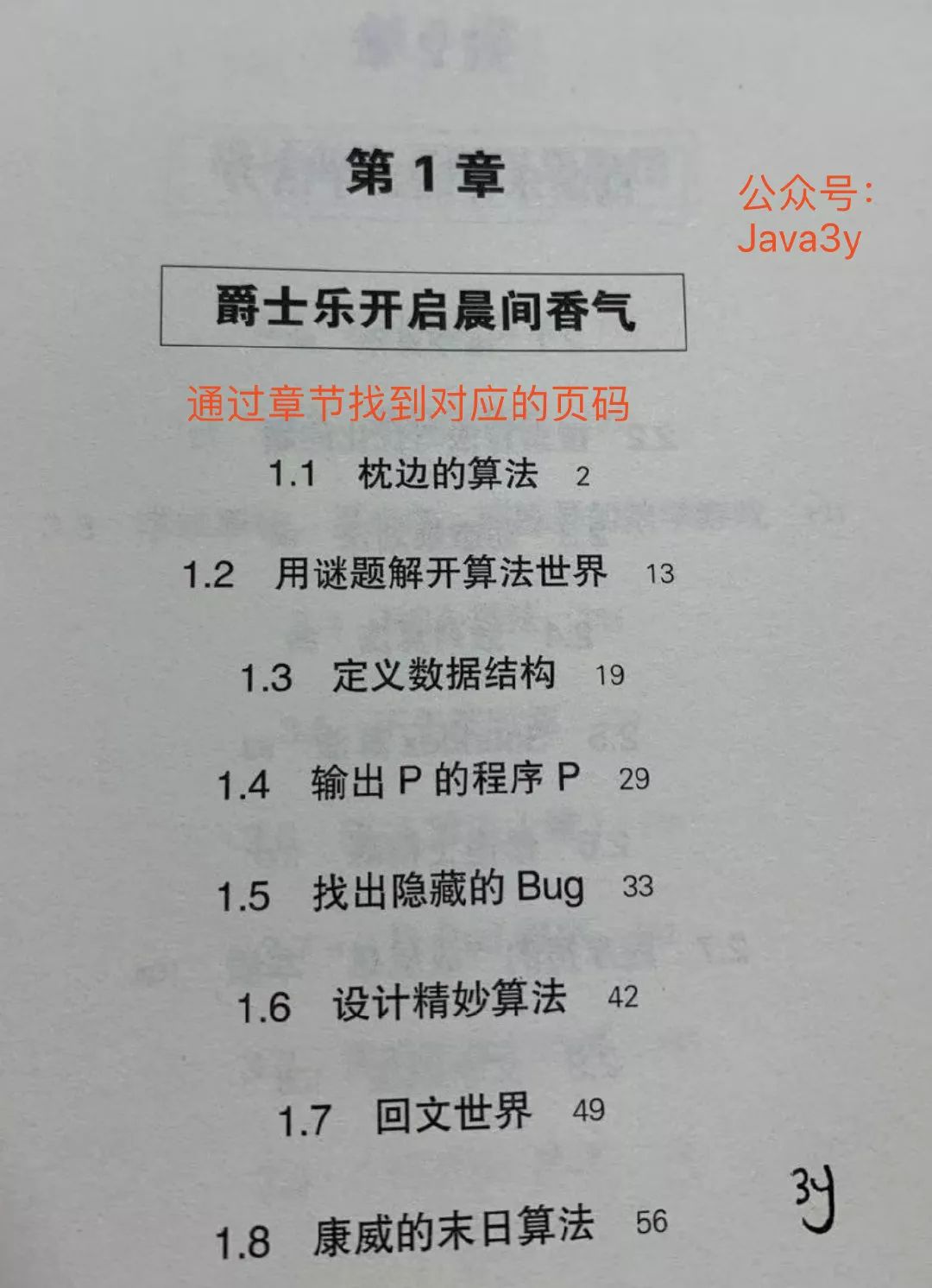
而且，即便给你从数据库根据**模糊匹配**查出相应的记录了，那往往会返回**大量的数据**给你，往往你需要的数据量并没有这么多，可能50条记录就足够了。

还有一个就是：用户输入的内容往往并没有这么的**精确**，比如我从Google输入ElastcSeach（打错字），但是Google还是能估算我想输入的是Elasticsearch

* 分布式
* 搜索
* 分析



于是我们就得知道Elasticsearch是怎么做到实时的，Elasticsearch的架构是怎么样的（分布式）。存储、搜索和分析（得知道Elasticsearch是怎么存储、搜索和分析的）



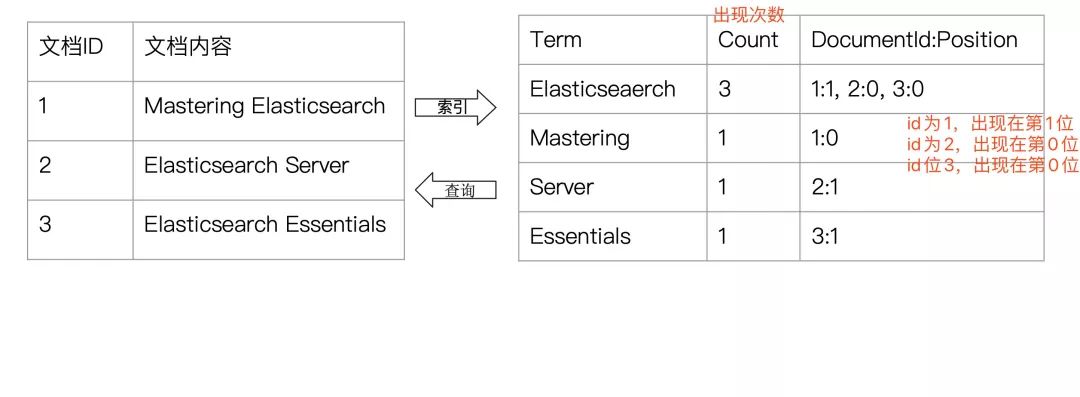
首先我们得知道为什么Elasticsearch为什么可以实现快速的“**模糊匹配**”/“**相关性查询**”，实际上是你写入数据到Elasticsearch的时候会进行**分词**。

还是以上图为例，上图出现了4次“**算法**”这个词，我们能不能根据这次词为它找他对应的目录？Elasticsearch正是这样干的，如果我们根据上图来做这个事，会得到类似这样的结果：

* 算法  ->2,13,42,56

这代表着“算法”这个词肯定是在第二页、第十三页、第四十二页、第五十六页出现过。这种根据**某个词**(不完整的条件)再查找对应记录，叫做**倒排索引**。

再看下面的图，好好体会一下：



众所周知，世界上有这么多的语言，那Elasticsearch怎么**切分这些词呢？**，Elasticsearch**内置**了一些分词器

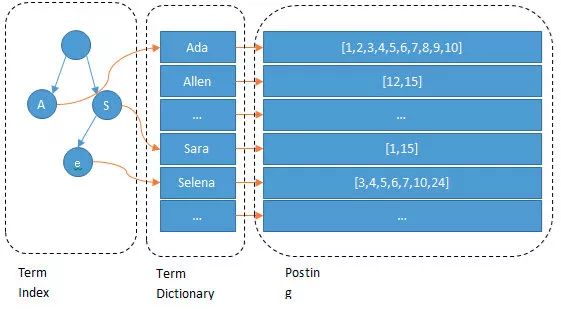
* Standard  Analyzer 。按词切分，将词小写
* Simple Analyzer。按非字母过滤（符号被过滤掉），将词小写
* WhitespaceAnalyzer。按照空格切分，不转小写
* ….等等等

Elasticsearch分词器主要由三部分组成：

* 􏱀􏰉􏰂􏰈􏰂􏰆􏰄Character Filters（文本过滤器，去除HTML）
* Tokenizer（按照规则切分，比如空格）
* TokenFilter（将切分后的词进行处理，比如转成小写）

显然，Elasticsearch是老外写的，内置的分词器都是英文类的，而我们用户搜索的时候往往搜的是中文，现在中文分词器用得最多的就是**IK**。

扯了一大堆，那Elasticsearch的数据结构是怎么样的呢？看下面的图：



我们输入一段文字，Elasticsearch会根据分词器对我们的那段文字进行**分词**（也就是图上所看到的Ada/Allen/Sara..)，这些分词汇总起来我们叫做Term Dictionary，而我们需要通过分词找到对应的记录，这些文档ID保存在PostingList

在Term Dictionary中的词由于是非常非常多的，所以我们会为其进行**排序**，等要查找的时候就可以通过**二分**来查，不需要遍历整个Term Dictionary

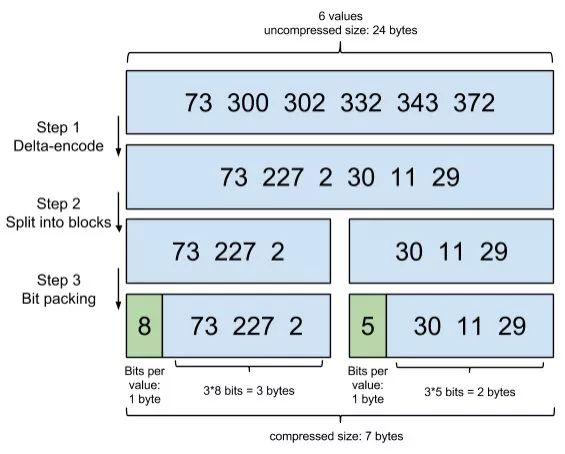
由于Term Dictionary的词实在太多了，不可能把Term Dictionary所有的词都放在内存中，于是Elasticsearch还抽了一层叫做Term Index，这层只存储  部分   **词的前缀**，Term Index会存在内存中（检索会特别快）

Term Index在内存中是以**FST**（Finite State Transducers）的形式保存的，其特点是**非常节省内存**。FST有两个优点：

* 1）空间占用小。通过对词典中单词前缀和后缀的重复利用，压缩了存储空间；
* 2）查询速度快。O(len(str))的查询时间复杂度。

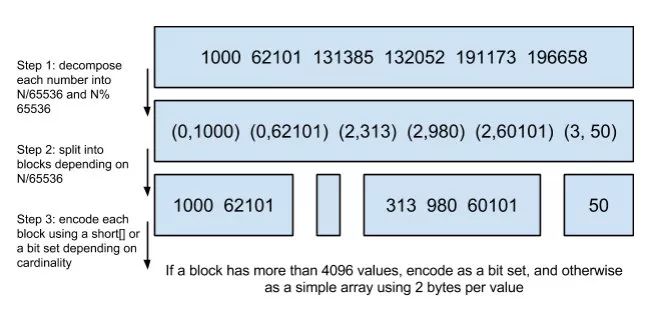
前面讲到了Term Index是存储在内存中的，且Elasticsearch用**FST**（Finite State Transducers）的形式保存（节省内存空间）。Term Dictionary在Elasticsearch也是为他进行排序（查找的时候方便），其实PostingList也有对应的优化。

PostingList会使用Frame Of Reference（**FOR**）编码技术对里边的数据进行压缩，**节约磁盘空间**。



PostingList里边存的是文档ID，我们查的时候往往需要对这些文档ID做**交集和并集**的操作（比如在多条件查询时)，PostingList使用**Roaring Bitmaps**来对文档ID进行交并集操作。

使用**Roaring Bitmaps**的好处就是可以节省空间和快速得出交并集的结果。



所以到这里我们总结一下Elasticsearch的数据结构有什么特点：

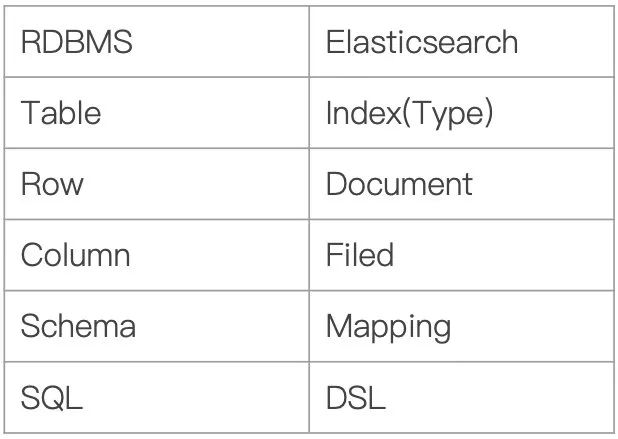


## Elasticsearch的术语和架构

从官网的介绍我们已经知道Elasticsearch是**分布式**存储的，如果看过我的文章的同学，对**分布式**这个概念应该不陌生了。

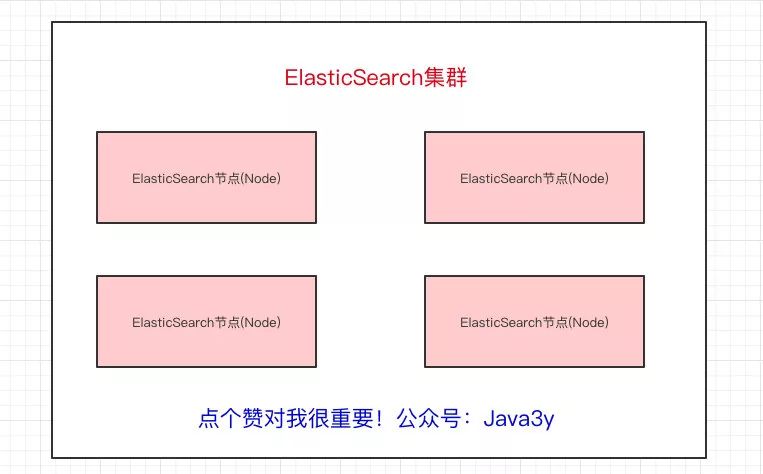
在讲解Elasticsearch的架构之前，首先我们得了解一下Elasticsearch的一些常见术语。

* **Index**：Elasticsearch的Index相当于数据库的Table
* **Type**：这个在新的Elasticsearch版本已经废除（在以前的Elasticsearch版本，一个Index下支持多个Type--有点类似于消息队列一个topic下多个group的概念）
* **Document**：Document相当于数据库的一行记录
* **Field**：相当于数据库的Column的概念
* **Mapping**：相当于数据库的Schema的概念
* **DSL**：相当于数据库的SQL（给我们读取Elasticsearch数据的API）

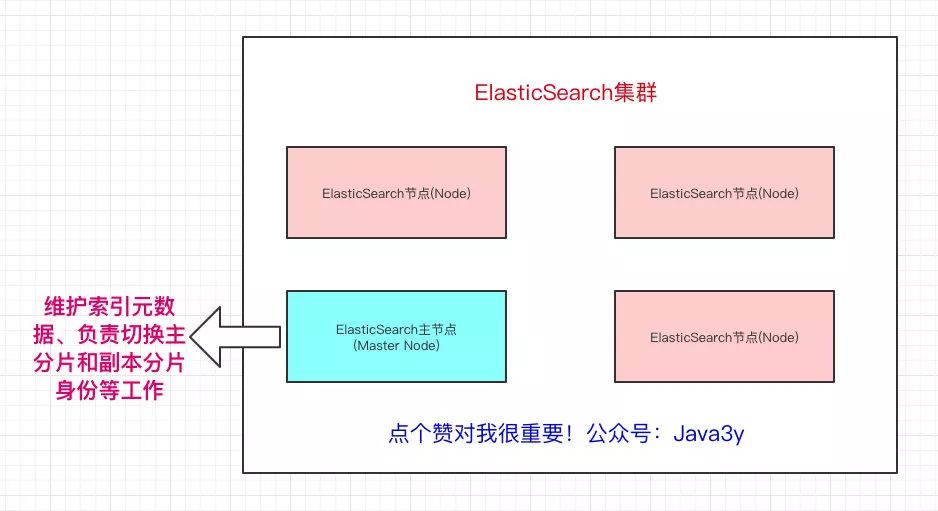


相信大家看完上面的对比图，对Elasticsearch的一些术语就不难理解了。那Elasticsearch的架构是怎么样的呢？下面我们来看看：

一个Elasticsearch集群会有多个Elasticsearch节点，所谓节点实际上就是运行着Elasticsearch进程的机器。

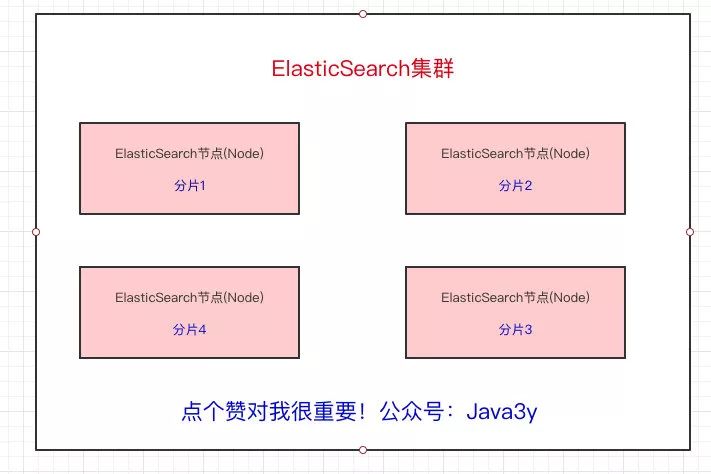


在众多的节点中，其中会有一个Master Node，它主要负责维护索引元数据、负责切换主分片和副本分片身份等工作（后面会讲到分片的概念），如果主节点挂了，会选举出一个新的主节点。



从上面我们也已经得知，Elasticsearch最外层的是Index（相当于数据库 表的概念）；一个Index的数据我们可以分发到不同的Node上进行存储，这个操作就叫做**分片**。

比如现在我集群里边有4个节点，我现在有一个Index，想将这个Index在4个节点上存储，那我们可以设置为4个分片。这4个分片的数据**合起来**就是Index的数据



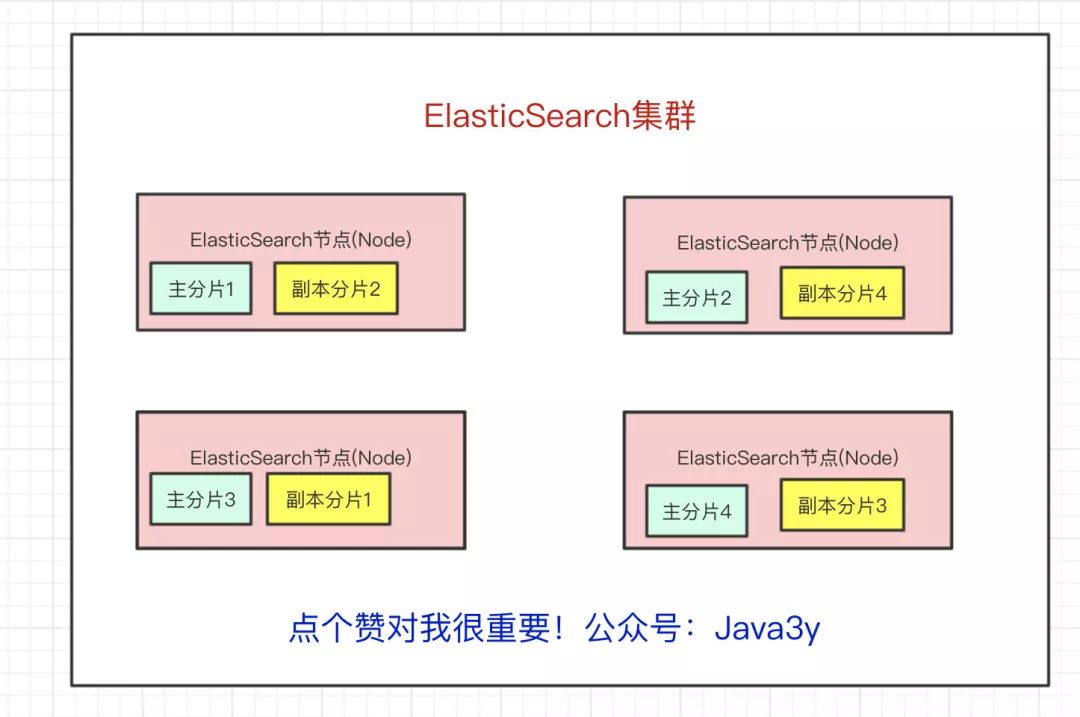
为什么要分片？原因也很简单：

* 如果一个Index的数据量太大，只有一个分片，那只会在一个节点上存储，随着数据量的增长，一个节点未必能把一个Index存储下来。
* 多个分片，在写入或查询的时候就可以并行操作（从各个节点中读写数据，提高吞吐量）

现在问题来了，如果某个节点挂了，那部分数据就丢了吗？显然Elasticsearch也会想到这个问题，所以分片会有主分片和副本分片之分（为了实现**高可用**）

数据写入的时候是**写到主分片**，副本分片会**复制**主分片的数据，读取的时候**主分片和副本分片都可以读**。

Index需要分为多少个主分片和副本分片都是可以通过配置设置的



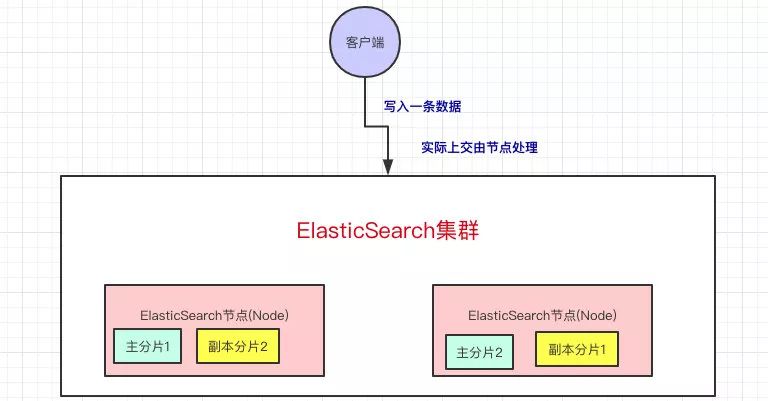
如果某个节点挂了，前面所提高的Master Node就会把对应的副本分片提拔为主分片，这样即便节点挂了，数据就不会丢。

到这里我们可以简单总结一下Elasticsearch的架构了：

## Elasticsearch 写入的流程

上面我们已经知道当我们向Elasticsearch写入数据的时候，是写到主分片上的，我们可以了解更多的细节。

客户端写入一条数据，到Elasticsearch集群里边就是由**节点**来处理这次请求：

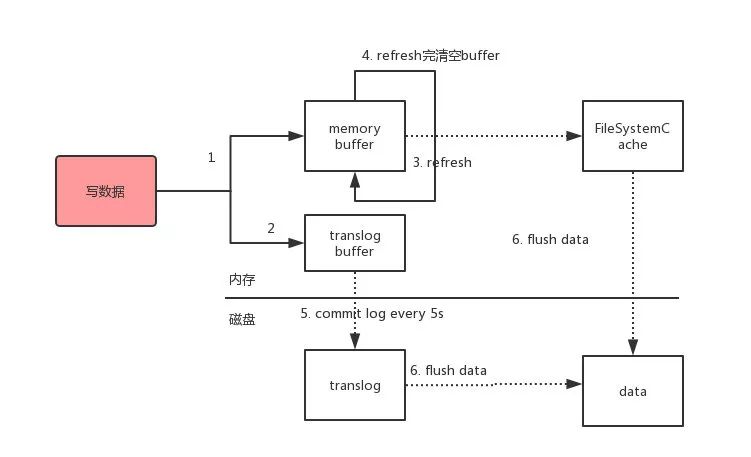


集群上的每个节点都是coordinating node（**协调节点**），协调节点表明这个节点可以做**路由**。比如**节点1**接收到了请求，但发现这个请求的数据应该是由**节点2**处理（因为主分片在**节点2**上），所以会把请求转发到**节点2**上。

* coodinate（**协调**）节点通过hash算法可以计算出是在哪个主分片上，然后**路由到对应的节点**
* shard = hash(document\_id) % (num\_of\_primary\_shards)

路由到对应的节点以及对应的主分片时，会做以下的事：

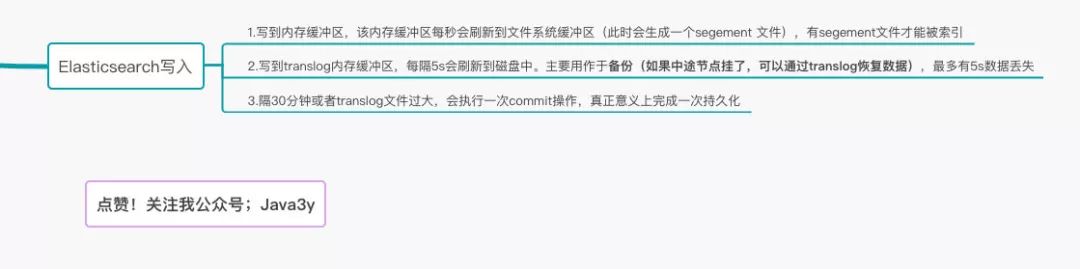
1. 将数据写到内存缓存区
2. 然后将数据写到translog缓存区
3. 每隔**1s**数据从buffer中refresh到FileSystemCache中，生成segment文件，一旦生成segment文件，就能通过索引查询到了
4. refresh完，memory buffer就清空了。
5. 每隔**5s**中，translog 从buffer flush到磁盘中
6. 定期/定量从FileSystemCache中,结合translog内容flush index到磁盘中。



解释一下：

* Elasticsearch会把数据先写入内存缓冲区，然后每隔**1s**刷新到文件系统缓存区（当数据被刷新到文件系统缓冲区以后，数据才可以被检索到）。所以：Elasticsearch写入的数据需要**1s**才能查询到
* 为了防止节点宕机，内存中的数据丢失，Elasticsearch会另写一份数据到**日志文件**上，但最开始的还是写到内存缓冲区，每隔**5s**才会将缓冲区的刷到磁盘中。所以：Elasticsearch某个节点如果挂了，可能会造成有**5s**的数据丢失。
* 等到磁盘上的translog文件大到一定程度或者超过了30分钟，会触发**commit**操作，将内存中的segement文件异步刷到磁盘中，完成持久化操作。

说白了就是：写内存缓冲区（**定时**去生成segement，生成translog），能够**让数据能被索引、被持久化**。最后通过commit完成一次的持久化。



等主分片写完了以后，会将数据并行发送到副本集节点上，等到所有的节点写入成功就返回**ack**给协调节点，协调节点返回**ack**给客户端，完成一次的写入。

## Elasticsearch更新和删除

Elasticsearch的更新和删除操作流程：

* 给对应的doc记录打上.del标识，如果是删除操作就打上delete状态，如果是更新操作就把原来的doc标志为delete，然后重新新写入一条数据

前面提到了，每隔**1s**会生成一个segement 文件，那segement文件会越来越多越来越多。Elasticsearch会有一个**merge**任务，会将多个segement文件**合并**成一个segement文件。

在合并的过程中，会把带有delete状态的doc给**物理删除**掉。



## Elasticsearch查询

查询我们最简单的方式可以分为两种：

* 根据ID查询doc
* 根据query（搜索词）去查询匹配的doc

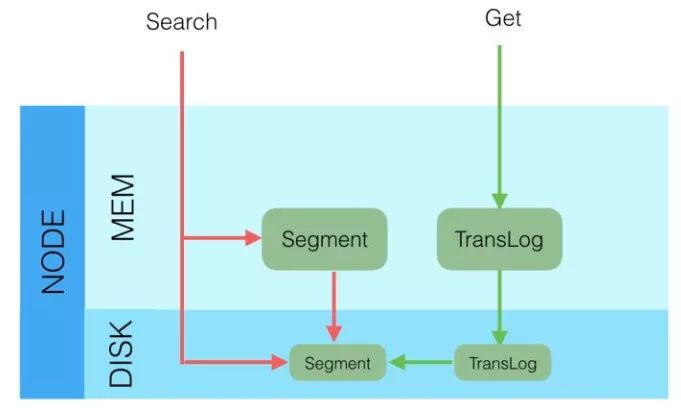
public TopDocs search(Query query, int n);  
public Document doc(int docID);

根据**ID**去查询具体的doc的流程是：

* 检索内存的Translog文件
* 检索硬盘的Translog文件
* 检索硬盘的Segement文件

根据**query**去匹配doc的流程是：

* 同时去查询内存和硬盘的Segement文件

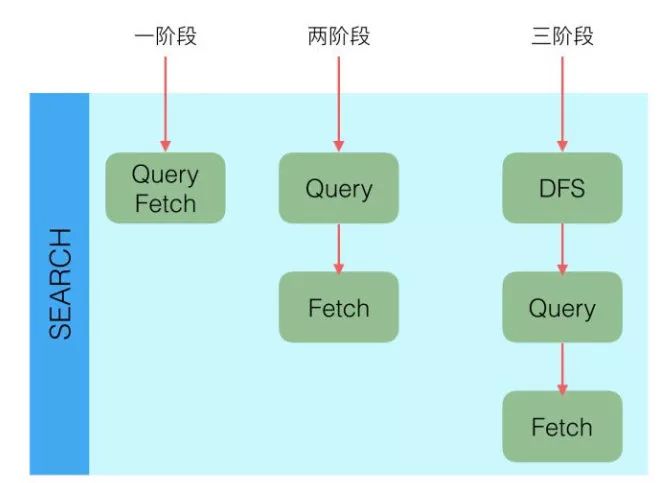


从上面所讲的写入流程，我们就可以知道：Get（通过ID去查Doc是实时的），Query（通过query去匹配Doc是近实时的）

* 因为segement文件是每隔一秒才生成一次的

Elasticsearch查询又分可以为三个阶段：

* QUERY\_AND\_FETCH（查询完就返回整个Doc内容）
* QUERY\_THEN\_FETCH（先查询出对应的Doc id ，然后再根据Doc id 匹配去对应的文档）
* DFS\_QUERY\_THEN\_FETCH（先算分，再查询）
* 「这里的分指的是 **词频率和文档的频率**（Term Frequency、Document Frequency）众所周知，出现频率越高，相关性就更强」



一般我们用得最多的就是**QUERY\_THEN\_FETCH**，第一种查询完就返回整个Doc内容（QUERY\_AND\_FETCH）只适合于只需要查一个分片的请求。

**QUERY\_THEN\_FETCH**总体的流程流程大概是：

* 客户端请求发送到集群的某个节点上。集群上的每个节点都是coordinate node（协调节点）
* 然后协调节点将搜索的请求转发到**所有分片上**（主分片和副本分片都行）
* 每个分片将自己搜索出的结果(doc id)返回给协调节点，由协调节点进行数据的合并、排序、分页等操作，产出最终结果。
* 接着由协调节点根据 doc id 去各个节点上**拉取实际**的 document 数据，最终返回给客户端。

**Query Phase阶段**时节点做的事：

* 协调节点向目标分片发送查询的命令（转发请求到主分片或者副本分片上）
* 数据节点（在每个分片内做过滤、排序等等操作），返回doc id给协调节点

**Fetch Phase阶段**时节点做的是：

* 协调节点得到数据节点返回的doc id，对这些doc id做聚合，然后将目标数据分片发送抓取命令（希望拿到整个Doc记录）
* 数据节点按协调节点发送的doc id，拉取实际需要的数据返回给协调节点

主流程我相信大家也不会太难理解，说白了就是：**由于Elasticsearch是分布式的，所以需要从各个节点都拉取对应的数据，然后最终统一合成给客户端**

只是Elasticsearch把这些活都干了，我们在使用的时候无感知而已。



## 最后

这篇文章主要对Elasticsearch简单入了个门，实际使用肯定还会遇到很多坑，但我目前就到这里就结束了。

