

**要想搜索数据，先在数据库中存一份**

**索引**：对应数据库中的一个**库**

**类型**：对应一张**表（6.0之后废弃）**

**文档**：对应数据库表中的**一行**（通常**json**格式）（6.0规定为固定的名字\_doc）

**字段**：表中的**一列**（json格式中的每一个属性）

**6．0以后废弃了类型的概念，用索引来代替**

**Cluster集群：**集群里有多个节点，有一个主节点，这个主节点可以通过选举产生，主从节点是对于集群内部来说的。es的一个概念是**去中心化**，也就是无中心节点，这是对于集群外部来说的，因为从外部来看es集群，他在逻辑上是个整体，和任何一个节点的通信和与整个es集群通信是等价的。

**节点**：**集群中的一台服务器**

**分片shard**：将**一个索引拆分成多个分片存储**，分布到不同的节点上，提高其**并发**能力，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

**副本replicas**：索引**的副本**，提高可用性和系统的容错性，因为当某个节点某个分片损坏或丢失时可以从副本中恢复。还提高es的查询效率，es会自动对搜索请求进行负载均衡。

**恢复Recovery**：代表数据恢复或者数据重新分布：es在有节点加入或退出的时候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的时候也会进行数据恢复。

**以上也是解决高并发的几个特质**

（1）每个索引会被分成多个分片shards存储(默认5个)。每个分片都会分布式部署在不同的节点上，主分片primary shards定义好后，后面不能做修改。

（2）为了实现数据的**高可用**，主分片可以有对应的备分片replics shards，replic shards分片有负责容错和请求的负载均衡，主分片对应的备分片不能存放同一台服务器上。主分片primary shards可以和其他主节点replics shards放在同一个节点上。

（3）documnet routing(数据路由)：客户端发起创建document的时候，es确定这个document放在这个索引哪个shard上的过程

**路由算法：**shard = hash(routing) % number\_of\_primary\_shards

如果number\_of\_primary\_shards在查询的时候取余发生变化，就无法取到数了，所以索引的主分片数量定义好后不能改

在用搜索引擎搜索时，会**先**将搜索内容**分词**，将一句话拆成多个独立单词，把**搜索独立单词后的数据，合并后返回给客户端**，搜索分词插件通过**git**下载，对于新出的词要写到配置文件里，而且可以加一些扩展停止词，比如并,了, 的什么的，到停止词就结束

**向索引中添加数据**：提交数据用put请求，put后边的**url格式为服务器地址/索引名称/文档名称/数据id**，在下边的框中写请求体，将要添加的数据用json格式写入，如果用命令行来添加数据会很麻烦

**为什么不用数据库搜索**

数据库也可以提供搜索，但是他是**通过特定的列来查找，但用户在搜索时输入的不是这某个列所包含的**，输入的东西**可能包含多个列**，**要通过分词全文匹配**，这是数据库不能实现的，而搜索引擎可以

**为什么用ES**

因为是分布式的实时文件存储，**可以每个字段都存到索引让他之后能被检索到**

搜索引擎可以扩展到上百台服务器，处理PB(内存单位)级数据高度集成化的服务，

应用可以通过简单的 Restful API、各种语言的客户端、命令行交互

**ES读写数据、搜索数据、更新和删除数据**

**写数据**

**1.**客户端向集群某个节点发请求过去，这个节点就是协调节点(coordinating node), 如果没有 协调节点，请求节点就扮演协调节点

**2.**协调节点接受到请求后默认用文档ID(也支持通过路由)来计算，得到文档属于哪个分片，之后把请求转发给对应的节点(有primary shard)。

**路由算法：**



内存缓冲 -> translog/文件系统缓存

**3.**分片所在节点接收到来自协调节点的请求后，primary节点会处理请求，会把数据先写入内存缓冲，然后每隔 1s，把数据refresh到文件系统缓存，到了文件系统缓存数据就能被搜索到(**所以es从写入到能被搜索到，中间有 1s 的延迟)**。同时每隔 5s，把数据写入translog文件(这个机制保证可靠性，如果机器宕机，内存数据全没，所以最多会有5s的数据丢失)，translog大到一定程度，或者默认每隔30mins，会触发commit操作，会把缓冲区的数据清除掉然后都flush到一个新段(segment)磁盘文件中, 段的fsync会创建一个新的提交点，会把内容刷新到磁盘。数据写到段文件segment file后，会建好倒排索引并且把请求写到内存缓冲里。数据会被同步到replica节点。

**4.**协调节点如果发现primary节点和所有replica node都搞定后，就返回响应结果给客户端。

**读数据**

可以通过 doc id 来查询，会根据 doc id 进行 hash，判断出来当时把 doc id 分配到了哪个 shard 上面去，从那个 shard 去查询。

1.客户端发送请求到任意一个 node，成为 coordinate node。

2.coordinate node 对 doc id 进行哈希路由，将请求转发到对应的 node，此时会使用 round-robin 随机轮询算法，在 primary shard 以及其所有 replica 中随机选择一个，请求负载均衡。

3.接收请求的 node 返回 document 给 coordinate node。

4.coordinate node 返回 document 给客户端。

**搜数据**

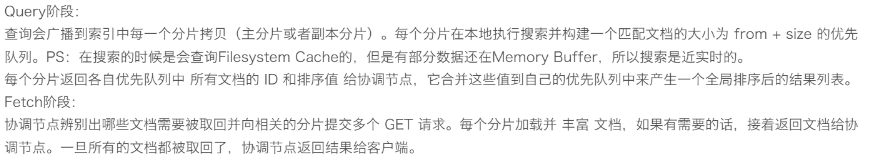
1.客户端发送请求到一个协调节点

2.协调节点把搜索请求转发到所有的shard(主分片或者副本分片)

3.query阶段：每个shard分片在本地执行搜索并且构建一个匹配文档的大小为from+size的优先队列。每个分片把自己优先队列里的所有文档ID和排序值给协调节点，接着协调节点合并这些数据到自己的优先队列来产生一个全局排序后的结果列表

4.fetch阶段：接着由协调节点根据要被取回的doc id去GET请求相关的分片，每个分片自己加载文档，根据需要把文档返回给协调节点，所有文档都被取回了之后协调节点把结果返回给客户端。

Query then fetch过程



**删数据**--也是写操作

因为ES里文档是不可变的，所以删除操作commit 的时候会生成一个 .del 文件，里面将某个 doc 标识为 deleted 状态，那么搜索的时候根据 .del 文件就知道这个 doc 是否被删除了。

该文档依然能匹配查询，但是会在结果中被过滤掉。当段合并时，在.del 文件中被标记为删除的文档将不会被写入新段。

底层：细节：buffer 每 refresh 一次，就会产生一个 segment file，所以默认情况下是 1 秒钟一个 segment file，这样下来 segment file 会越来越多，此时会定期执行 merge。每次 merge 的时候，会将多个 segment file 合并成一个，同时这里会将标识为 deleted 的 doc 给物理删除掉，然后将新的 segment file 写入磁盘，这里会写一个 commit point，标识所有新的 segment file，然后打开 segment file 供搜索使用，同时删除旧的 segment file。

**更新数据**--也是写操作

将原来的 doc 标识为 deleted 状态，然后新写入一条数据。就是在新的文档被创建时，Elasticsearch 会给文档指定一个版本号，当执行更新时，旧版本的文档在.del 文件中被标记为删除，新版本的文档被索引到一个新段。旧版本的文档依然能匹配查询，但是会在结果中被过滤掉。

**底层Lucene**

lucene 是一个 jar 包，里面包含了封装好的各种建立倒排索引的算法代码。用 Java 开发的时候，引入 lucene jar然后基于 lucene的api就可以开发. 里面包括索引创建、索引、搜索

通过 lucene，我们可以将已有的数据建立索引，lucene 会在本地磁盘上面，给我们组织索引的数据结构。

**倒排索引**

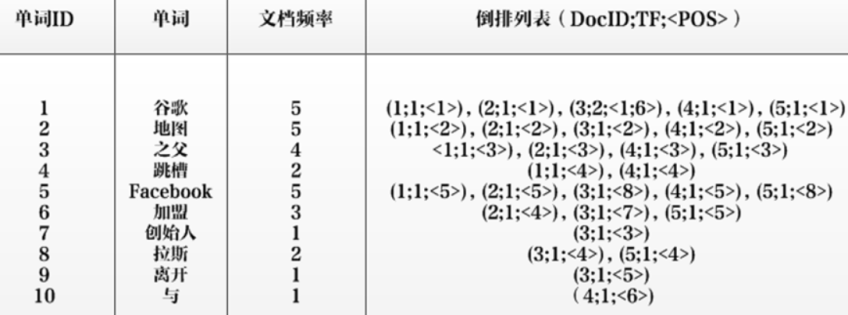
**倒排索引就是关键词到文档 ID 的映射，由词典和倒排表组成，每个关键词都对应一系列的文件，这些文件中都出现了关键词,实现O(1)时间复杂度检索文章**

ES搜索引擎里，每个文档都有一个对应的文档 ID，文档内容被表示为一系列关键词的集合。比如文档 1 经过分词，提取了 20 个关键词，每个关键词都会记录它在文档中出现的次数和出现位置。

如果**正牌索引**，我们每次寻找关键字查询，就得搜索所有的⽂档去看是否有这个关键字，这样查询效率太慢了。

于是有了倒排索引，是通过关键字去查⽂档，我们建⽴⼀个索引库，⾥⾯的key是关键字，value是每个⽂档的id，倒排在构建索引的时候较为耗时且维护成本较⾼，但是搜索耗时短，所以我们可以定时去更新索引库

**底层是FST**(Finite State Transducer)数据结构,**优点**是1.空间占用小(对词典里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))



单词“加盟”为例，其单词编号为6，文档频率为3，代表整个文档集合中有三个文档包含这个单词，对应的倒排列表为{(2;1;<4>),(3;1;<7>),(5;1;<5>)}，含义是在文档2，3，5出现过这个单词，在每个文档的出现过1次，单词“加盟”在第一个文档的POS是4，即文档的第四个单词是“加盟”

* 所有词项对应一个或多个文档；
* 倒排索引中的词项****根据字典顺序升序排列****
* 查docid用跳表

**ES集群**

**ES为什么要实现集群**：在单台ES服务器节点上，随着业务量发展索引文件慢慢增多，会影响到效率和内存存储问题等。ES集群**可以把单个索引的分片到多个不同分布式物理机器上存储来实现高可用、容错性**等。

ES集群的索引可能由多个分片构成，而且每个分片可以有多个副本。通过把一个单独的索引分成多个分片，我们就可以处理不能在一个单一的服务器上面运行的大型索引，简单的说就是索引的大小过大，导致效率问题。不能运行的原因可能是内存也可能是存储。**由于每个分片可以有多个副本，通过将副本分配到多个服务器，可以提高查询的负载能力。**



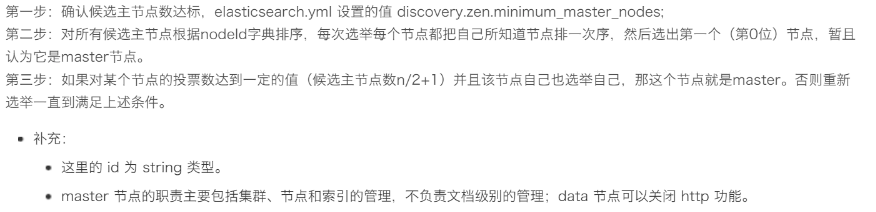
**master选举**

集群中可能会有多个master-eligible node，这个时候就要进行master选举，保证只有一个当选master。

**前置条件**：



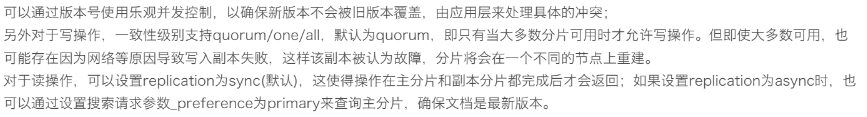
**选举流程：**



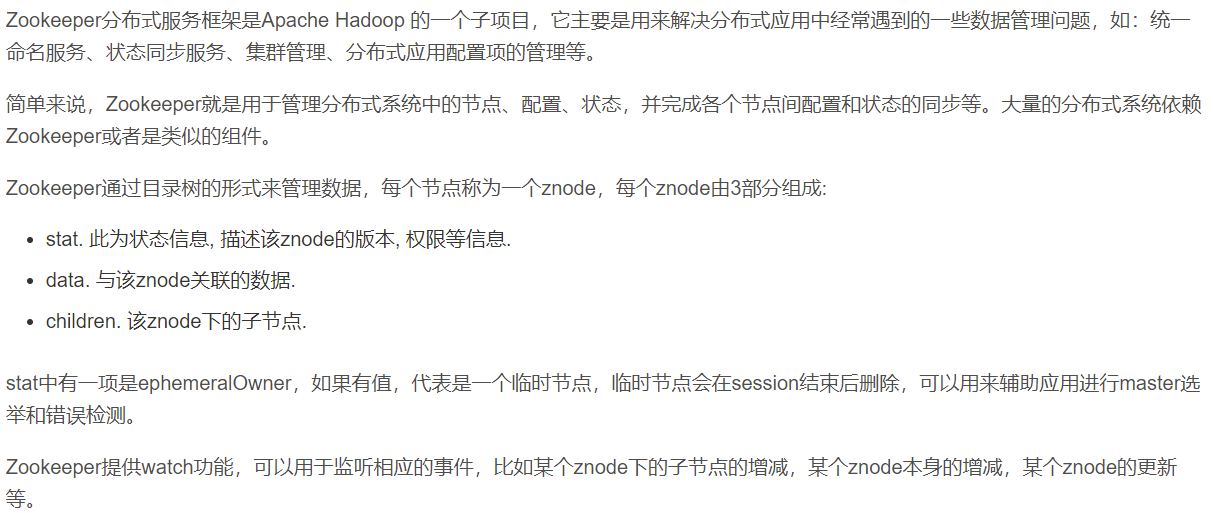
**脑裂和解决方案:**

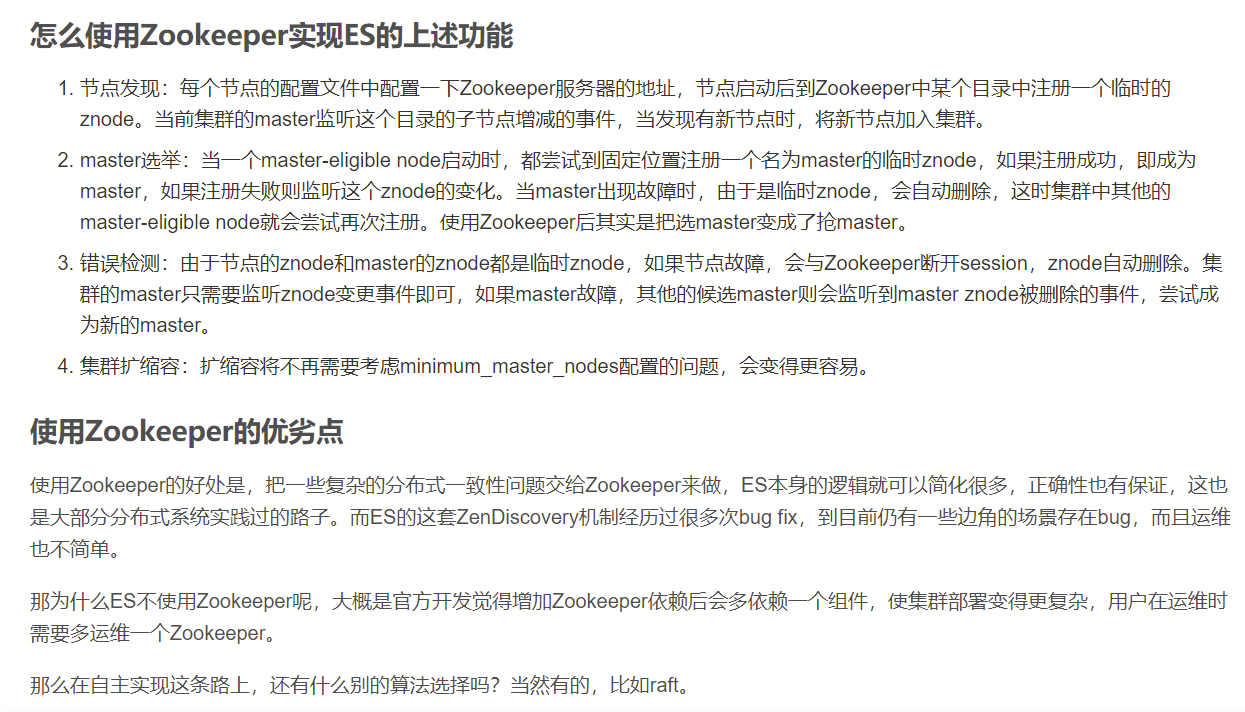


**并发情况怎么保证读写一致**



**ZooKeeper介绍和用ZooKeeper实现ES**





**索引多了怎么调优和部署**

1. 动态索引层面

基于模板+时间+rollover api 滚动创建索引，举例：设计阶段定义：blog 索引的模板格式为：blog\_index\_时间戳的形式，每天递增数据。

这样做的好处：不至于数据量激增导致单个索引数据量非常大，接近于上线 2 的32 次幂-1，索引存储达到了 TB+甚至更大。一旦单个索引很大，存储等各种风险也随之而来，所以要提前考虑+及早避免。

2. 存储层面

冷热数据分离存储，热数据（比如最近 3 天或者一周的数据），其余为冷数据。对于冷数据不会再写入新数据，可以考虑定期 force\_merge 加 shrink 压缩操作，节省存储空间和检索效率。

3. 部署层面

结合 ES 自身的支持动态扩展的特点，动态新增机器的方式可以缓解集群压力，注意：如果之前主节点等规划合理，不需要重启集群也能完成动态新增的。