1.什么是elasticsearch

1.1倒排索引

- 在搜索引擎中,文档(doc),都会建立一个docid
- 对文档进行分词,然后建立每个单词的id(wordId),对应关系为 wordId-->word-->docId[]

DocId	Doc
1	谷歌地图之父跳槽 Facebook
2	谷歌地图之父加盟 Facebook
3	谷歌地图创始人拉斯离开谷歌加盟 Facebook
4	谷歌地图之父跳槽 Facebook 与 Wave 项目取消有关
5	谷歌地图之父拉斯加盟社交网站 Facebook

倒排索引

WordId	Word	DocIds
1	谷歌	1,2,3,4,5
2	地图	1,2,3,4,5
3	之父	1,2,4,5
4	跳槽	1,4
5	Facebook	1,2,3,4,5
6	加盟	2,3,5
7	创始人	3
8	拉斯	3,5
9	离开	3
10	与	4
..	..	..

1.2lucene

- lucene是一个jar包,封装了各种建立倒排索引的算法代码,java开发直接调用api即可
- 通过lucene可以将已有的数据建立索引,lucene会在本地磁盘,组织索引的数据结构

1.3Elasticsearch

- es 中存储数据的基本单位是索引
- 自动维护数据分布到各个节点的索引的建立,搜索分布请求分布到多个节点上执行

- 自动维护数据的冗余副本,保证机器宕机不会丢失数据
- 提供高级功能

1.4特点

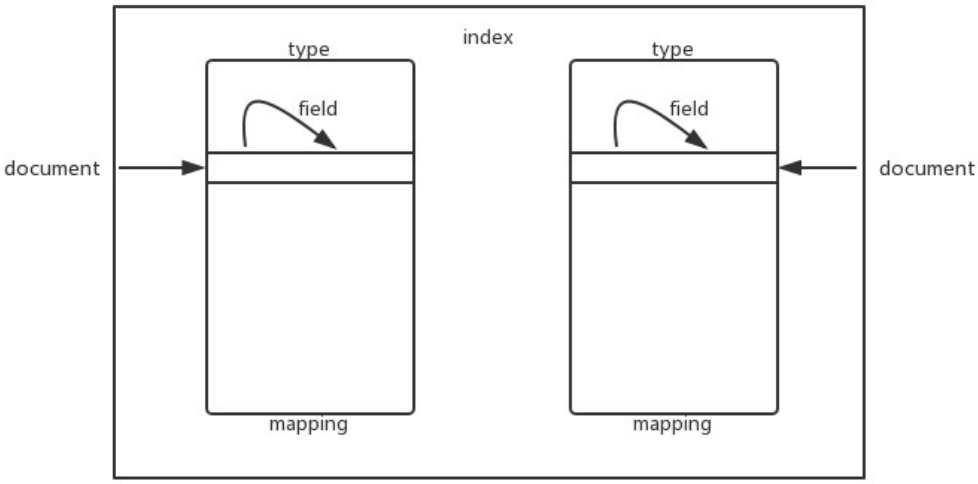
- 大型分布式集群技术,pb级数据
- 全文检索,数据分析,分布式技术合在一起

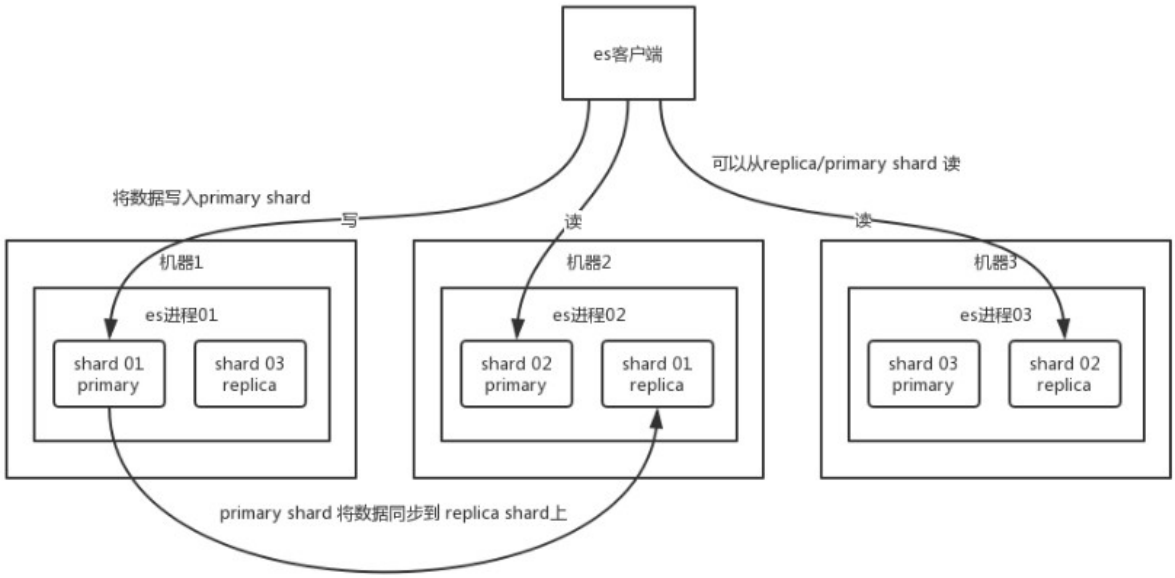
2.es分布式架构原理&es读写过程

2.1es分布式架构原理

`index>>>type>>>mapping>>>document>>>field`

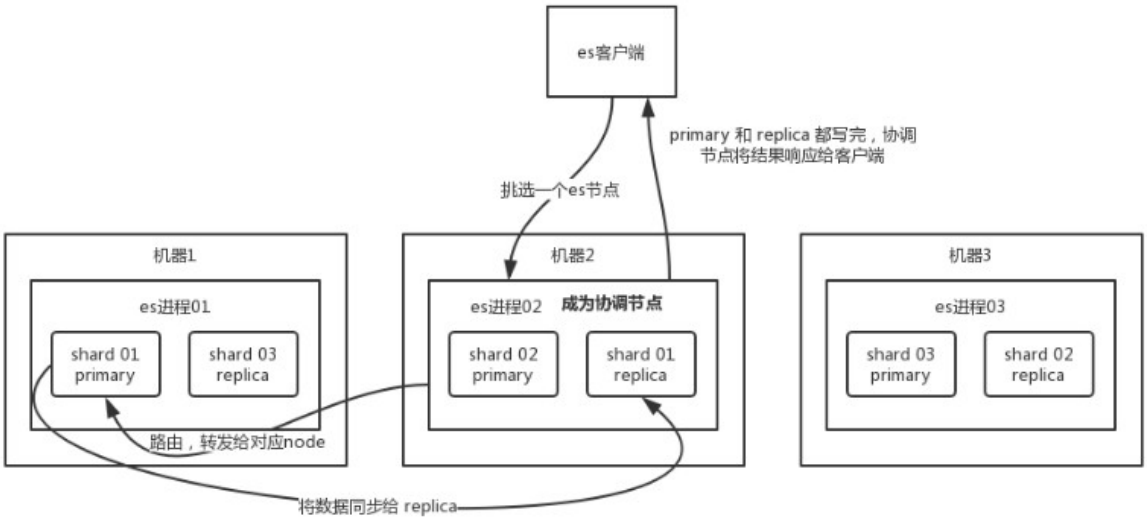
- index索引
- type,是索引下的一个逻辑分类,一个type下有多个document,同一index 不同type下的document有相同的field
- mapping 定义type中每个字段的名称
- document 相当于数据库表中的每一条记录
- field 类似一条记录的各个字段





- 用户发送写请求,es客户端会把请求的document 拆分成多个shard,存放每个es节点上
- 每个shard都有一个parmary shard负责写数据,有几个replica shard在写入数据后会将数据同步到其他的replicashard上,形成备份
- es多个节点会自动推选master,master节点负责维护索引元数据,切换parmary shard 和replica shard
- 如果master 宕机重新选举master
- 如果非master宕机,master会让primary shard对应的replia shard切换为parmary shard,
- 宕机机器重新上线,会同步新的parmary shard 上的数据,而且本身为replia shard

2.2写



- 客户端选择一个 node 发送请求过去，这个 node 就是 `coordinating node`（协调节点）。
- `coordinating node` 对 document 进行**路由**，将请求转发给对应的 node（有 primary shard）。
- 实际的 node 上的 `primary shard` 处理请求，然后将数据同步到 `replica node`。
- `coordinating node` 如果发现 `primary node` 和所有 `replica node` 都搞定之后，就返回响应结果给客户端。

2.3读

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。

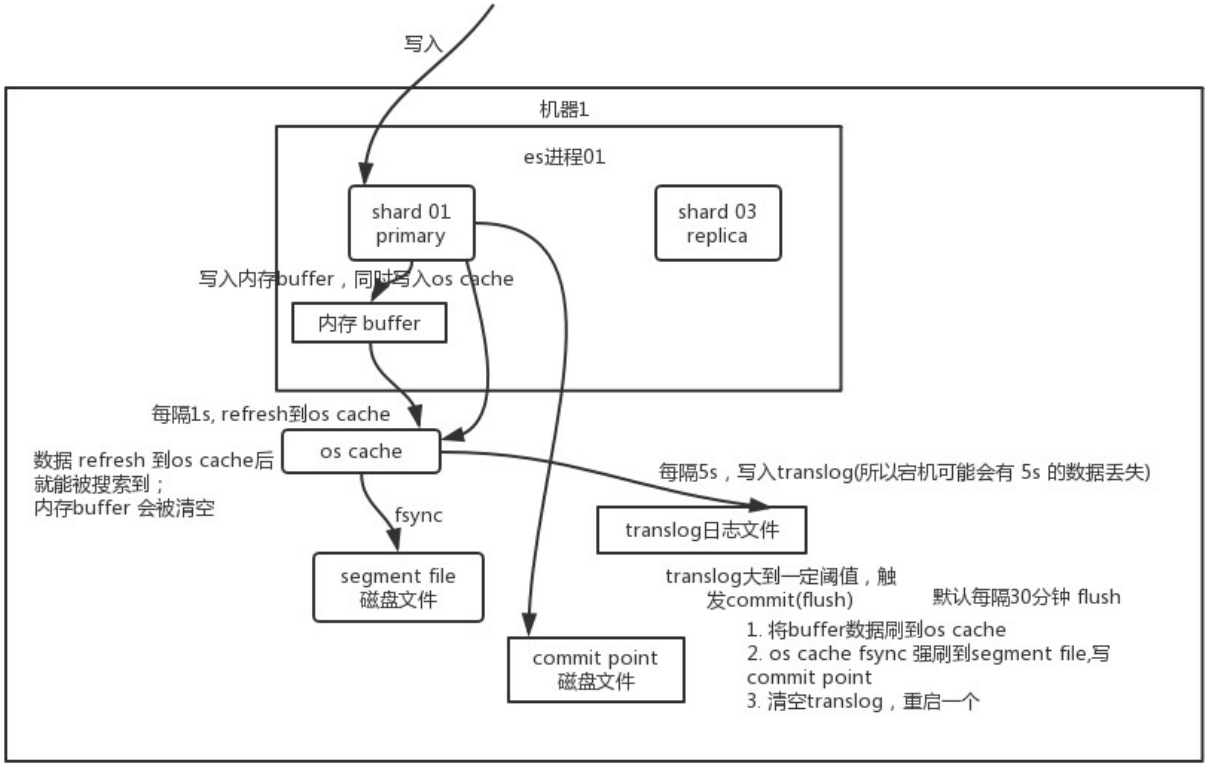
- 客户端发送请求到**任意**一个 node，成为 coordinate node。
- coordinate node 对 doc id 进行哈希路由，将请求转发到对应的 node，此时会使用 round-robin **随机轮询算法**，在 primary shard 以及其所有 replica 中随机选择一个，让读请求负载均衡。
- 接收请求的 node 返回 document 给 coordinate node。
- coordinate node 返回 document 给客户端。

2.4 搜索

你根据 java 关键词来搜索，将包含 java的 document 给搜索出来。es 就会给你返回：java真好玩儿啊，java好难学啊。

- 客户端发送请求到一个 coordinate node。
- 协调节点将搜索请求转发到**所有**的 shard 对应的 primary shard 或 replica shard，都可以。
- query phase：每个 shard 将自己的搜索结果（其实就是一些 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。
- fetch phase：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。

2.5写入磁盘



- 数据先写入buffer中
- buffer快满的了,或者过一段时间,内存中的buffer数据将会被refresh到segment file中,此时要先进入os cache中,这个过程就是refresh
- 如果buffer没有数据,不会执行refresh工作,如果有每一秒执行一次,把数据refresh到segment file中
- es是near real-time,可以通过restful api 或者java api手动执行一次refesh,手动讲buffer中的数据刷新到os cache中,让数据立马搜索到,只要数据输入os cache,buffer就会被清空,不会保留在buffer中,数据在

translog保留,

- commit 操作发生第一步，就是将 buffer 中现有数据 `refresh` 到 `os cache` 中去，清空 buffer。然后，将一个 `commit point`写入磁盘文件，里面标识着这个 `commit point` 对应的所有 `segment file`，同时强行将 `os cache` 中目前所有的数据都 `fsync` 到磁盘文件中去。最后**清空** 现有 translog 日志文件，重启一个 translog，此时 commit 操作完成。
- 这个 commit 操作叫做 `flush`。默认 30 分钟自动执行一次 `flush`，但如果 translog 过大，也会触发 `flush`。flush 操作就对应着 commit 的全过程，我们可以通过 es api，手动执行 flush 操作，手动将 os cache 中的数据 fsync 强刷到磁盘上去。
- translog 日志文件的作用是什么？你执行 commit 操作之前，数据要么是停留在 buffer 中，要么是停留在 os cache 中，无论是 buffer 还是 os cache 都是内存，一旦这台机器死了，内存中的数据就全丢了。所以需要将数据对应的操作写入一个专门的日志文件 `translog` 中，一旦此时机器宕机，再次重启的时候，es 会自动读取 translog 日志文件中的数据，恢复到内存 buffer 和 os cache 中去。
- translog 其实也是先写入 os cache 的，默认每隔 5 秒刷一次到磁盘中去，所以默认情况下，可能有 5 秒的数据会仅仅停留在 buffer 或者 translog 文件的 os cache 中，如果此时机器挂了，**会丢失** 5 秒钟的数据。但是这样性能比较好，最多丢 5 秒的数据。也可以将 translog 设置成每次写操作必须是直接 `fsync` 到磁盘，但是性能会差很多。
- 其实 es 第一是准实时的，数据写入 1 秒后可以搜索到；可能会丢失数据的。有 5 秒的数据，停留在 buffer、translog os cache、segment file os cache 中，而不在磁盘上，此时如果宕机，会导致 5 秒的数据丢失。

2.6删除,更新数据

删除操作

- commit的时候生成.del文件,里面将某个doc标识为deleted状态,搜索的时候根据.del文件就知道doc被删除了
- 如果是更新操作,就是将.doc标识为deleted状态,然后重新写入一条数据
- buffer每refresh一次,就会产生一个segment file,所以默认是一秒钟一个segment file, segment file越来越多
- 此时定期执行merge 操作,会将多个segment file 合并为一个,同时删除标识我deleted 状态的doc(**物理删除**)
- 然后将性的segment file 写入磁盘,这里会写一个commit point,标识新的segement file ,然后open segement file 供搜索使用, del old segement file

3.性能问题

3.1filesystem cache

- es写的数据都是保存在磁盘中的,查询的时候.操作系统会把磁盘中的数据缓存到 filesystem cache中

- **es搜索引擎严重依赖底层的filesystem cache**
- 如果给filesystem cache 内存大,容纳的idx segment file 的索引数据文件多,那么搜索的时候走内存插叙就比较多,效率就高
- 要让 es 性能要好，最佳的情况下，就是你的机器的内存，至少可以容纳你的总数据量的一半。
- 最佳的情况下，是仅仅在 es 中就存少量的数据，就是你要**用来搜索的那些索引**，

- 比如说你现在有一行数据。`id,name,age ....` 30 个字段。但是你现在搜索，只需要根据 `id,name,age` 三个字段来搜索。如果你傻乎乎往 es 里写入一行数据所有的字段，就会导致说 `90%` 的数据是不用来搜索的，结果硬是占据了 es 机器上的 `filesystem cache` 的空间，单条数据的数据量越大，就会导致 `filesystem cahce` 能缓存的数据就越少。其实，仅仅写入 es 中要用来检索的**少数几个字段**就可以了，比如说就写入es `id,name,age` 三个字段，然后你可以把其他的字段数据存在 mysql/hbase 里，我们一般是建议用 `es + hbase` 这么一个架构。
- hbase 的特点是**适用于海量数据的在线存储**，就是对 hbase 可以写入海量数据，但是不要做复杂的搜索，做很简单的一些根据 id 或者范围进行查询的这么一个操作就可以了。从 es 中根据 name 和 age 去搜索，拿到的结果可能就 20 个 `doc id`，然后根据 `doc id` 到 hbase 里去查询每个 `doc id` 对应的**完整的数据**，给查出来，再返回给前端
- 写入 es 的数据最好小于等于，或者是略微大于 es 的 filesystem cache 的内存容量。然后你从 es 检索可能就花费 20ms，然后再根据 es 返回的 id 去 hbase 里查询，查 20 条数据，可能也就耗费个 30ms，可能你原来那么玩儿，1T 数据都放es，会每次查询都是 5~10秒，现在可能性能就会很高，每次查询就是 50ms。

3.2数据预热

- 对于热数据可以在后台搞一个**系统**,定时的进行访问,导致数据存于systemfile cache中

3.3热冷分离

- 最好是将**冷数据写入一个索引中，然后热数据写入另外一个索引中**，这样可以确保热数据在被预热之后，尽量都让他们留在 `filesystem os cache` 里，**别让冷数据给冲刷掉**。

  你看，假设你有 6 台机器，2 个索引，一个放冷数据，一个放热数据，每个索引 3 个shard。3 台机器放热数据 index；另外 3 台机器放冷数据 index。然后这样的话，你大量的时候是在访问热数据 index，热数据可能就占总数据量的 10%，此时数据量很少，几乎全都保留在 `filesystem cache` 里面了，就可以确保热数据的访问性能是很高的。但是对于冷数据而言，是在别的 index 里的，跟热数据 index 不在相同的机器上，大家互相之间都没什么联系了。如果有人访问冷数据，可能大量数据是在磁盘上的，此时性能差点，就 10% 的人去访问冷数据，90% 的人在访问热数据，也无所谓了。

3.4document模型设计

- es中尽量不要使用复杂关联查询,性能不好
- 在java系统中做好关联,关联好的数据直接写入`elasticsearch`

3.5分页性能优化

每页10条数据,如果查询第100页,实际上,es会把每个shard上存储的前10*100的数据都查询到协调节点,如果有5个shard 就有5000条数据查询到协调节点上,协调节点对这 5000 条数据进行一些合并、处理，再获取到最终第 100 页的 10 条数据。

- 不允许深度分页,默认深度分页性能很惨
- 类似于app里的商品推荐商品不断下拉一页页,可以使用scroll api
- scroll 会一次性给你生成**所有数据的一个快照**，然后每次翻页就是通过**游标移动**，获取下一页下一页这样子，性能会比上面说的那种分页性能也高很多很多，基本上都是毫秒级的
- **不能随意跳到任何一页的场景**

4.部署

面试题

es 生产集群的部署架构是什么？每个索引的数据量大概有多少？每个索引大概有多少个分片？

答:

其实这个问题没啥，如果你确实干过 es，那你肯定了解你们生产 es 集群的实际情况，部署了几台机器？有多少个索引？每个索引有多大数据量？每个索引给了多少个分片？你肯定知道！

但是如果你确实没干过，也别虚，我给你说一个基本的版本，你到时候就简单说一下就好了。

- es 生产集群我们部署了 5 台机器，每台机器是 6 核 64G 的，集群总内存是 320G。
- 我们 es 集群的日增量数据大概是 2000 万条，每天日增量数据大概是 500MB，每月增量数据大概是 6 亿，15G。目前系统已经运行了几个月，现在 es 集群里数据总量大概是 100G 左右。
- 目前线上有 5 个索引（这个结合你们自己业务来，看看自己有哪些数据可以放 es 的），每个索引的数据量大概是 20G，所以这个数据量之内，我们每个索引分配的是 8 个 shard，比默认的 5 个 shard 多了 3 个shard。

大概就这么说一下就行了。