数据存储到分片的规则

shard = hash(routing) % number\_of\_primary\_shards

routing：默认文档id（docId），也可以自定义

number\_of\_primary\_shards：主分片数

创建索引后主分片数不能修改。

如果一致hash就可以减轻这种问题。

文档id作为路由基础，可以做到数据平均分布，性能也不错。为什么要用自定义路由呢？有什么更好的地方么？

根据搜索的机制可以发现，搜索请求会被转发到索引的所有分片上。如果指定了路由，可以在搜索的时候指定分片，避免搜素请求转发到所有分片上。

所有的api（get、index、delete、update、mget）都能接收一个routing参数。可以实现个性化分片映射，某个routing值的数据会被存储在相同的分片上。

自定义路由，参考文档：<https://blog.csdn.net/cnweike/article/details/38531997>

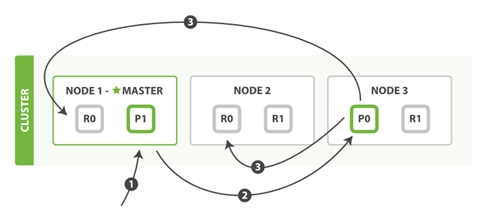
索引数据（添加数据）

分3个阶段

1、客户端发送请求给es一个节点

2、这个节点路由规则计算出数据应该放在第几个分片，es每个节点都知道其他节点的信息，让后将请求转发到主分片所在的节点。

3、主分片处理索引请求，然后并行的将请求转发到复制分片上，所有的分支分片处理成功，主分片节点返回给用户请求的节点（协调节点），然后返回给用户。



删除数据

处理过程跟索引数据的过程一样。

更新数据（文档）

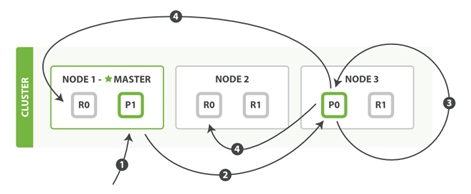
分4个过程

1、客户端发送请求到一个节点

2、该节点通过路由规则计算出数据在第几个分片，然后将请求转发到主分片所在的节点

3、主分片获取文档内容json，修改内容，重新索引文档，当然id没有变，还是在这个分片上。

4、主分片处理成功后，并行的将新文档的版本发送给复本分片，复本分片执行新文档索引操作，成功后返回给主分片，主分片再返回给协调节点，再返回给客户端。



搜索数据

搜索数据跟CRUD不一样，不是处理某一个确定的文档。是对一批不确定的文档进行查找。

搜索会在索引所有的分片上搜索文档。搜索的过程当然用倒排索引。

分两个阶段（搜索阶段，获取结果阶段）

搜索阶段

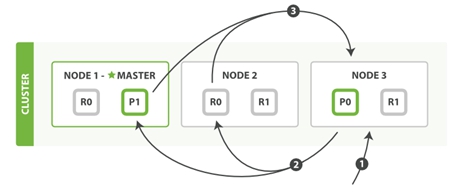
搜索请求会广播到所有的分片（包括主分片和复制分片，广播的过程有个负载均衡的操作），每个分片执行检索，并根据文档匹配度进行优先级排序，形成一个有序的列表，列表的大小由from，size来决定。

分3个过程：

1、客户端将搜索请求发送个一个节点，节点在本地创建一个空的优先级队列，并且配置好分页参数from，size。

2、然后将请求转发到该索引的所有分片（包括主分片和复制分片，所有分片的意思是组成完整的index数据），所有分片执行在本地执行检索，形成一个有序列表。

3、每个分片返回有序队列的sort和\_id给协调节点（接收用户请求的节点），协调节点将所有分片返回的数据进行一个汇总排序，只去from，size规定的\_id。



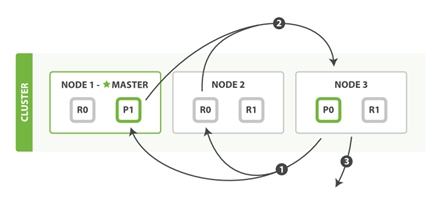
获取数据阶段（fetch phrase）

这个阶段是一个批量get数据的阶段。

1、搜索阶段确定了哪些数据要返回，发送一个mget请求到相关的shard（查询阶段用到的那些分片）。

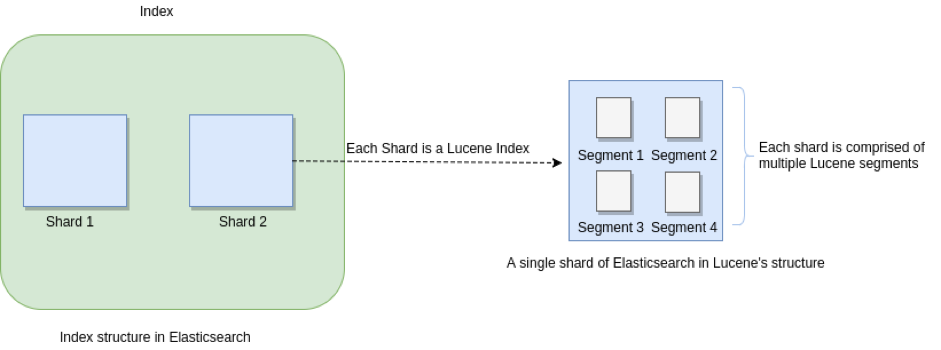
2、相关的shard获取文档内容返回给协调节点。

3、协调节点返回给客户端。



索引分片内部原理

index由shard构成，shard由segment构成，sement由document构成



shard是Lucene索引

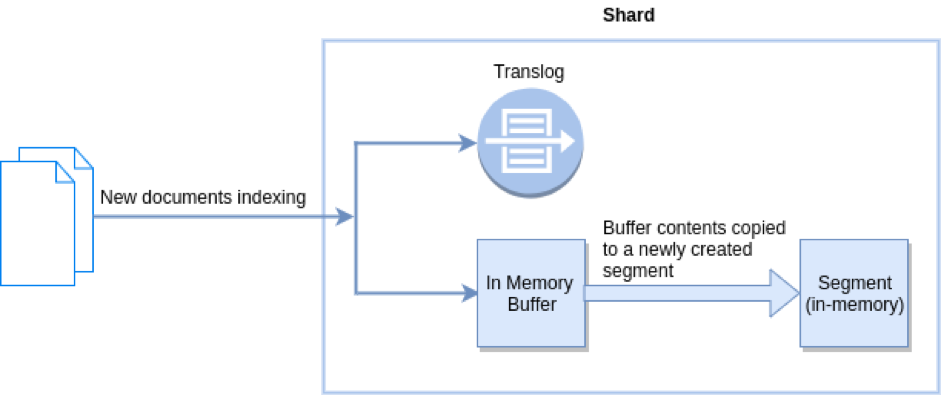
segment是倒排索引的集合

每次创建document都会归属到新的segment，不会修改原来的segment。

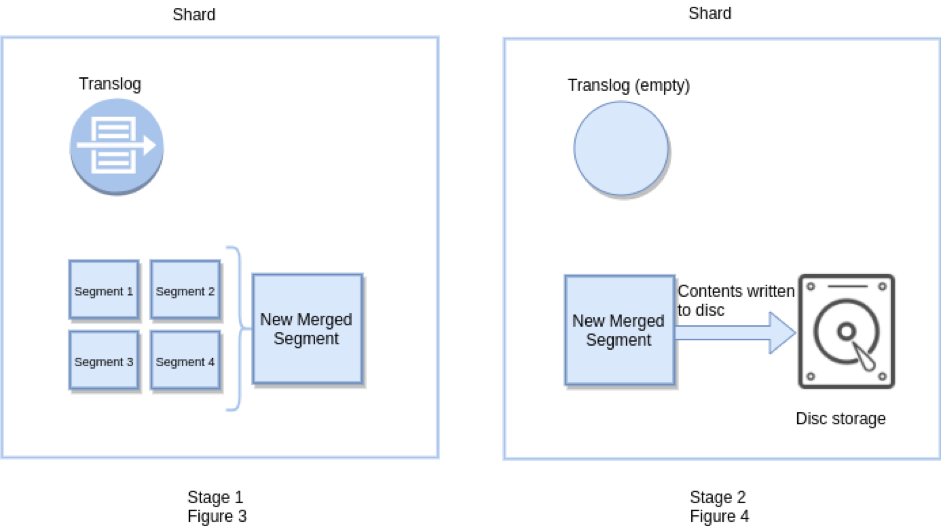
每次删除操作只会标记document为删除，不会立刻物理删除。

每个shard对应一个translog文件

\_refresh操作默认每秒执行一次，将内存buffer中的数据写入segment中，在segment中才可以被检索。



flush，将内存buffer中的数据全部写入所有的segments，并将内存中所有的segments同步到磁盘，并清空translog日志。



**近实时搜索**

fsync将段segment写入磁盘，是一个很耗新能的IO操作。

内存————————pagecache——磁盘

buffer，segment

refresh操作，将内存中bufer的数据写入内存中的segment，并且打开segment，这样才可被搜索到， refresh操作默认每秒执行一次。

这就是为什么叫近实时搜索：文档变化不是立即可被搜索，但是会在一秒之内可被搜索到。

如果对文档做了改动，查询搜索却没看到变化。这时需要一次refresh

POST /index/\_refresh 刷新某一个索引

POST /\_refresh 刷新所有索引

修改refresh频率

PUT /index {"settings":{"refresh\_interval":"30s"}}

在大量索引的过程中，可能不需要每秒refresh一次。

PUT /index {"settings":{"refresh\_interval":"-1"}} 关闭refresh

refresh操作，每秒执行一次

1、将buffer中的数据写入新的segment中（文件系统缓存），

2、打开segment，使数据可被搜索

3、清空缓存buffer中的数据。

**持久化数据**

flush（fsync）

提交点

在前后两次flush数据到磁盘的中间时间，数据变化操作都写到translog中，当然在内存buffer和pagecache有一份这段时间完整的数据。

translog 在磁盘上。

segment在文件系统的pagecache上。

flush的过程

1、将内存buffer中的数据写入到文件系统pagecache上的segment中

2、清空内存buffer中的数据

3、将文件系统pagecache上的segment，fsync到磁盘

4、删除旧的translog（此时内存中的数据和translog上的数据相同，且已经写入磁盘）

决定flush的参数，不同版本可能不同。

index.translog.flush\_threshold\_ops,执行多少次操作后执行一次flush，默认无限制

index.translog.flush\_threshold\_size，translog的大小超过这个参数后flush，默认512mb

index.translog.flush\_threshold\_period,多长时间强制flush一次,默认30m

index.translog.interval,es多久去检测一次translog是否满足flush条件

index.translog.sync\_interval 控制translog多久fsync到磁盘,最小为100ms

index.translog.durability translog是每5秒钟刷新一次还是每次请求都fsync，这个参数有2个取值:

request(每次请求都执行fsync,es要等translog fsync到磁盘后才会返回成功)

async(默认值,translog每隔5秒钟fsync一次)

translog 提供所有还没有被刷到磁盘的操作的一个持久化纪录。当 Elasticsearch 启动的时候， 它会从磁盘中使用最后一个提交点去恢复已知的段，并且会重放 translog 中所有在最后一次提交后发生的变更操作。

translog 也被用来提供实时 CRUD 。当你试着通过ID查询、更新、删除一个文档，它会在尝试从相应的段中检索之前， 首先检查 translog 任何最近的变更。这意味着它总是能够实时地获取到文档的最新版本。

在重启节点或关闭索引之前执行 [flush](https://www.elastic.co/guide/cn/elasticsearch/guide/cn/translog.html#flush-api) 有益于你的索引。当 Elasticsearch 尝试恢复或重新打开一个索引， 它需要重放 translog 中所有的操作，所以如果日志越短，恢复越快。

translog持久化策略，有2种

1、异步flush到磁盘

2、同步flush到磁盘

默认每5秒同步到磁盘中。可以通过下面的参数设置

PUT /my\_index/\_settings {

"index.translog.durability": "async",

"index.translog.sync\_interval": "5s"

}

每次都写操作都flush translog到磁盘

"index.translog.durability": "request"

写入数据，同时写入buffer和translog中

段合并

每秒都会refresh，每次refresh会产生一个segment，segment的数量会迅速增大，会有一些问题。

1、每一个段都会消耗文件句柄、内存和cpu运行周期。

2、更重要的是，每个搜索请求都必须轮流检查每个段；所以段越多，搜索也就越慢。

段合并可以减少段的数量。

段合并的过程，将一部分大小相近的小段合并到更大的段中。不会中断搜索、索引过程。

合并的过程中，旧的已经删除的document不会copy到大的segment中。

合并结束

1、老的小的segment会被删除。新的段flush到磁盘，并写入一个新的提交点。

2、新的段被打开可以搜索

3、老段被删除。

合并大的段需要消耗大量的I/O和CPU资源，如果任其发展会影响搜索性能。Elasticsearch在默认情况下会对合并流程进行资源限制，所以搜索仍然 有足够的资源很好地执行。

合并索引中的数据到max\_num\_segments指定数目的段中。

POST /logstash-2014-10/\_optimize?max\_num\_segments=1

不会变化的数据，历史日志数据，就可以这样操作，来提升搜索性能。

提升es写入速度

1、批量索引用bulk批量提交

2、禁用\_all字段

3、关闭segment合并

4、分片副本设置为0

5、关闭refresh

6、设置flush为异步，translog文件

数据写入后，在还原配置。

es面试总结的相关文档：<https://blog.csdn.net/feifeidepop/article/details/83281050>

es的from和size也存在深度分页问题。

scroll-scan

在大的数据结果集返回时使用。

elasticsearch选主

倒排索引

倒排索引由所有文档中所有不重复词的列表构成。每个词有一个文档列表

分词、标准化

相关性算法

数据一致性

写quorum/one/all

读sync/async