# ELK学习笔记（来自龙果学院，内容非原创。）

## 分布式文档系统\_深度图解剖析document数据路由原理

### document路由到shard上是什么意思？

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

举个例子，一个index有3个primary shard，P0，P1，P2，每次增删改查一个document的时候，都会带过来一个routing number，默认就是这个document的\_id（可能是手动指定，也可能是自动生成），routing = \_id，假设\_id=1。会将这个routing值，传入一个hash函数中，产出一个routing值的hash值，假设hash(routing) = 21，然后将hash函数产出的值对这个index的primary shard的数量求余数，21 % 3 = 0，就决定了，这个document就放在P0上。

决定一个document在哪个shard上，最重要的一个值就是routing值，默认是\_id，也可以手动指定，相同的routing值，每次过来，从hash函数中，产出的hash值一定是相同的

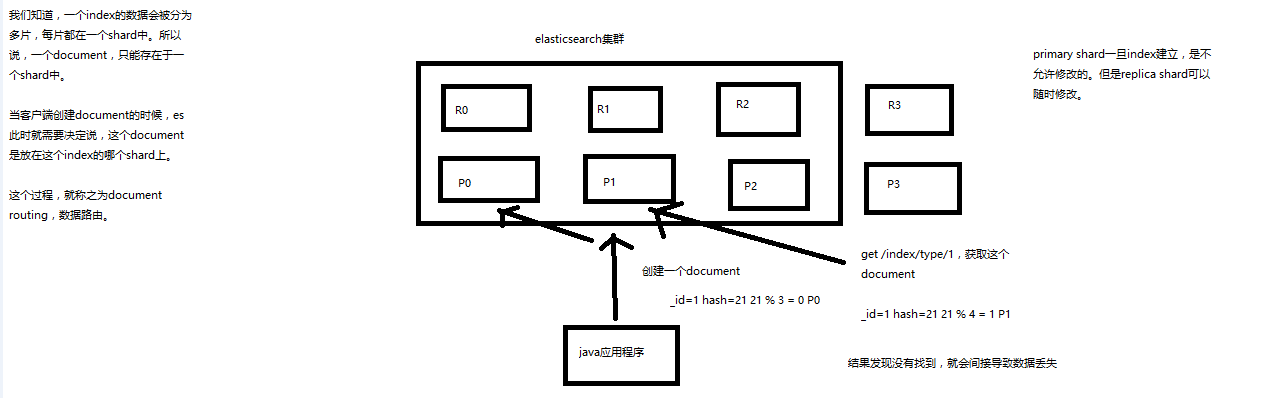
无论hash值是几，无论是什么数字，对number\_of\_primary\_shards求余数，结果一定是在0~number\_of\_primary\_shards-1之间这个范围内的。

### \_id or custom routing value

默认的routing就是\_id，也可以在发送请求的时候，手动指定一个routing value，比如说put /index/type/id?routing=user\_id

手动指定routing value是很有用的，可以保证说，某一类document一定被路由到一个shard上去，那么在后续进行应用级别的负载均衡，以及提升批量读取的性能的时候，是很有帮助的

### primary shard数量不可变的谜底



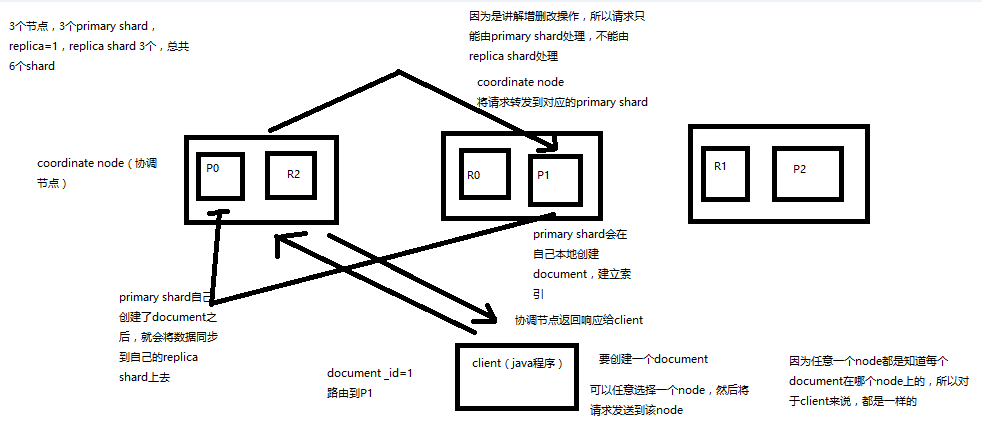
### 分布式文档系统\_document增删改内部原理图解揭秘

（1）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）

（2） coordinating node，对document进行路由，将请求转发给对应的node（有primary shard）

（3）实际的node上的primary shard处理请求，然后将数据同步到replica node

（4）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。



## 分布式文档系统\_图解写一致性原理以及quorum机制深入剖析

（1）consistency，one（primary shard），all（all shard），quorum（default）

我们在发送任何一个增删改操作的时候，比如说put /index/type/id，都可以带上一个consistency参数，指明我们想要的写一致性是什么？

put /index/type/id?consistency=quorum

one：要求我们这个写操作，只要有一个primary shard是active活跃可用的，就可以执行

all：要求我们这个写操作，必须所有的primary shard和replica shard都是活跃的，才可以执行这个写操作

quorum：默认的值，要求所有的shard中，必须是大部分的shard都是活跃的，可用的，才可以执行这个写操作

（2）quorum机制，写之前必须确保大多数shard都可用，int( (primary + number\_of\_replicas) / 2 ) + 1，当number\_of\_replicas>1时才生效

quroum = int( (primary + number\_of\_replicas) / 2 ) + 1

举个例子，3个primary shard，number\_of\_replicas=1，总共有3 + 3 \* 1 = 6个shard

quorum = int( (3 + 1) / 2 ) + 1 = 3

所以，要求6个shard中至少有3个shard是active状态的，才可以执行这个写操作

（3）如果节点数少于quorum数量，可能导致quorum不齐全，进而导致无法执行任何写操作

3个primary shard，replica=1，要求至少3个shard是active，3个shard按照之前学习的shard&replica机制，必须在不同的节点上，如果说只有2台机器的话，是不是有可能出现说，3个shard都没法分配齐全，此时就可能会出现写操作无法执行的情况

es提供了一种特殊的处理场景，就是说当number\_of\_replicas>1时才生效，因为假如说，你就一个primary shard，replica=1，此时就2个shard

(1 + 1 / 2) + 1 = 2，要求必须有2个shard是活跃的，但是可能就1个node，此时就1个shard是活跃的，如果你不特殊处理的话，导致我们的单节点集群就无法工作

（4）quorum不齐全时，wait，默认1分钟，timeout，100，30s

等待期间，期望活跃的shard数量可以增加，最后实在不行，就会timeout

我们其实可以在写操作的时候，加一个timeout参数，比如说put /index/type/id?timeout=30，这个就是说自己去设定quorum不齐全的时候，es的timeout时长，可以缩短，也可以增长

## 分布式文档系统\_document查询内部原理图解揭秘

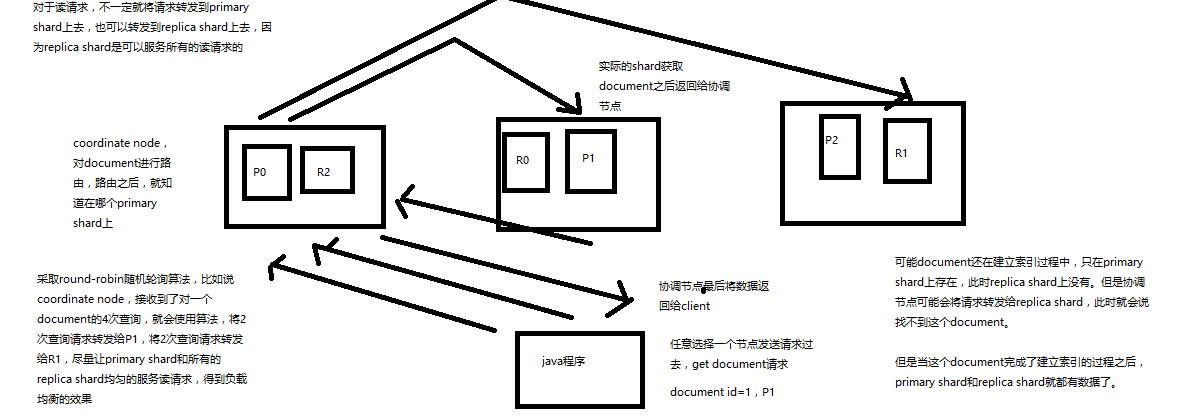
1、客户端发送请求到任意一个node，成为coordinate node

2、coordinate node对document进行路由，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

3、接收请求的node返回document给coordinate node

4、coordinate node返回document给客户端

5、特殊情况：document如果还在建立索引过程中，可能只有primary shard有，任何一个replica shard都没有，此时可能会导致无法读取到document，但是document完成索引建立之后，primary shard和replica shard就都有了



## 分布式文档系统\_bulk api的奇特json格式与底层性能优化关系大揭秘

bulk api奇特的json格式

{"action": {"meta"}}\n

{"data"}\n

{"action": {"meta"}}\n

{"data"}\n

良好的json数组格式

[{

"action": {

},

"data": { }

}]

1、bulk中的每个操作都可能要转发到不同的node的shard去执行

2、如果采用比较良好的json数组格式

允许任意的换行，整个可读性非常棒，读起来很爽，es拿到那种标准格式的json串以后，要按照下述流程去进行处理

（1）将json数组解析为JSONArray对象，这个时候，整个数据，就会在内存中出现一份一模一样的拷贝，一份数据是json文本，一份数据是JSONArray对象

（2）解析json数组里的每个json，对每个请求中的document进行路由

（3）为路由到同一个shard上的多个请求，创建一个请求数组

（4）将这个请求数组序列化

（5）将序列化后的请求数组发送到对应的节点上去

3、耗费更多内存，更多的jvm gc开销

我们之前提到过bulk size最佳大小的那个问题，一般建议说在几千条那样，然后大小在10MB左右，所以说，可怕的事情来了。假设说现在100个bulk请求发送到了一个节点上去，然后每个请求是10MB，100个请求，就是1000MB = 1GB，然后每个请求的json都copy一份为jsonarray对象，此时内存中的占用就会翻倍，就会占用2GB的内存，甚至还不止。因为弄成jsonarray之后，还可能会多搞一些其他的数据结构，2GB+的内存占用。

占用更多的内存可能就会积压其他请求的内存使用量，比如说最重要的搜索请求，分析请求，等等，此时就可能会导致其他请求的性能急速下降

另外的话，占用内存更多，就会导致java虚拟机的垃圾回收次数更多，跟频繁，每次要回收的垃圾对象更多，耗费的时间更多，导致es的java虚拟机停止工作线程的时间更多

4、现在的奇特格式

{"action": {"meta"}}\n

{"data"}\n

{"action": {"meta"}}\n

{"data"}\n

（1）不用将其转换为json对象，不会出现内存中的相同数据的拷贝，直接按照换行符切割json

（2）对每两个一组的json，读取meta，进行document路由

（3）直接将对应的json发送到node上去

5、最大的优势在于，不需要将json数组解析为一个JSONArray对象，形成一份大数据的拷贝，浪费内存空间，尽可能地保证性能

## 初识搜索引擎\_search结果深入解析（search timeout机制揭秘）

1、我们如果发出一个搜索请求的话，会拿到一堆搜索结果，这个搜索结果里的各种数据，都代表了什么含义？

GET /\_search

{

"took": 6,

"timed\_out": false,

"\_shards": {

"total": 6,

"successful": 6,

"failed": 0

},

"hits": {

"total": 10,

"max\_score": 1,

"hits": [

{

"\_index": ".kibana",

"\_type": "config",

"\_id": "5.2.0",

"\_score": 1,

"\_source": {

"buildNum": 14695

}

}

]

}

}

took：整个搜索请求花费了多少毫秒

hits.total：本次搜索，返回了几条结果

hits.max\_score：本次搜索的所有结果中，最大的相关度分数是多少，每一条document对于search的相关度，越相关，\_score分数越大，排位越靠前

hits.hits：默认查询前10条数据，完整数据，\_score降序排序

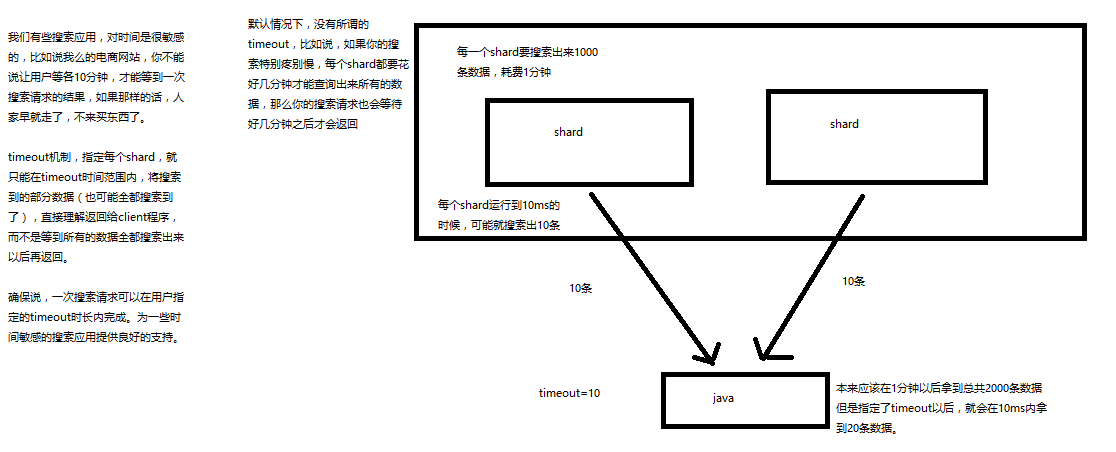
shards：shards fail的条件（primary和replica全部挂掉），不影响其他shard。默认情况下来说，一个搜索请求，会打到一个index的所有primary shard上去，当然了，每个primary shard都可能会有一个或多个replic shard，所以请求也可以到primary shard的其中一个replica shard上去。

2、搜索的timeout机制，底层的原理，画图讲解

timeout：默认无timeout，latency平衡completeness，手动指定timeout，timeout查询执行机制

timeout=10ms，timeout=1s，timeout=1m

GET /\_search?timeout=10m



## 初识搜索引擎\_multi-index&multi-type搜索模式解析以及搜索原理初步图解

1、multi-index和multi-type搜索模式

如何一次性搜索多个index和多个type下的数据

/\_search：所有索引，所有type下的所有数据都搜索出来

/index1/\_search：指定一个index，搜索其下所有type的数据

/index1,index2/\_search：同时搜索两个index下的数据

/\*1,\*2/\_search：按照通配符去匹配多个索引

/index1/type1/\_search：搜索一个index下指定的type的数据

/index1/type1,type2/\_search：可以搜索一个index下多个type的数据

/index1,index2/type1,type2/\_search：搜索多个index下的多个type的数据

/\_all/type1,type2/\_search：\_all，可以代表搜索所有index下的指定type的数据

2、初步图解一下简单的搜索原理

