ES常考知识点

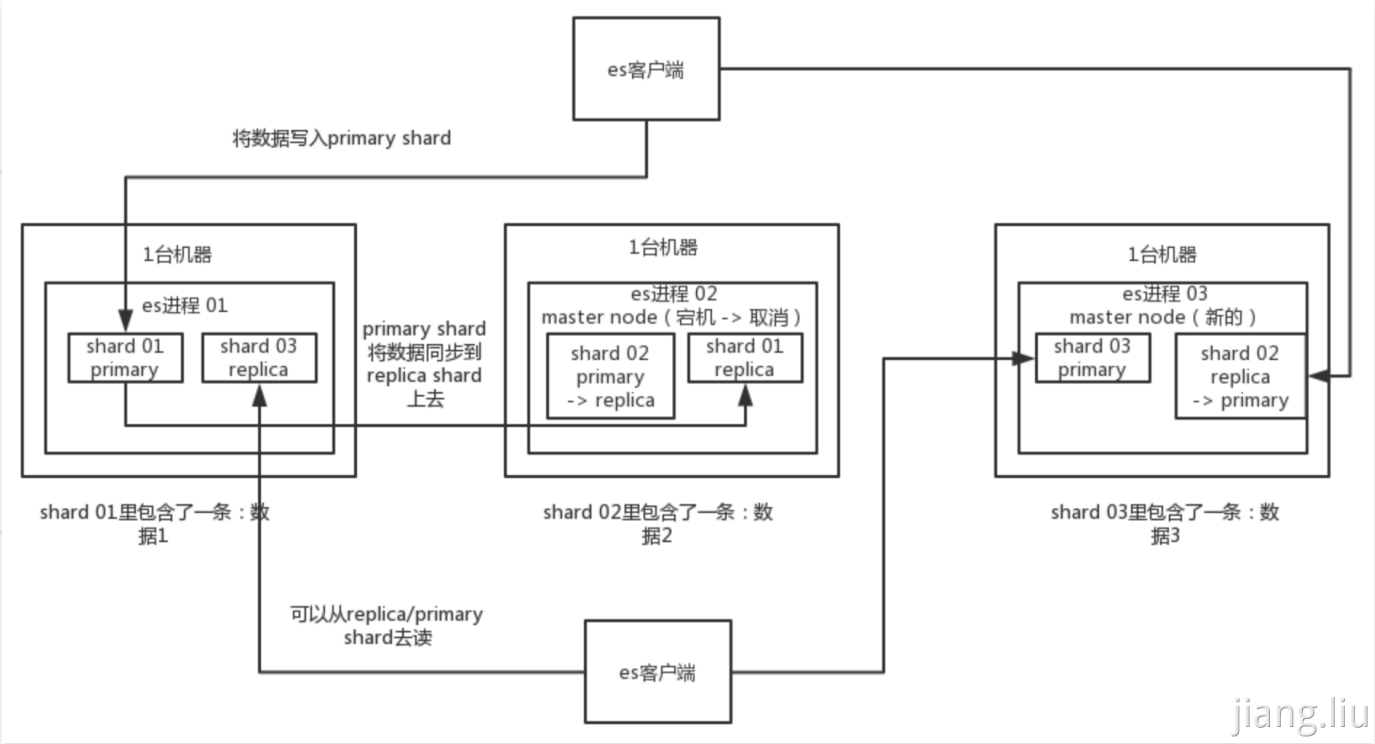
1.说一下ES的分布式架构原理（es是如何实现分布式的）？

核心思想就是在多台机器上启动多个es进程实例，组成了一个es集群。

一个索引，会拆分成多个shard（分片），每个shard存储部分数据。

然后这个shard的数据实际是有多个备份，就是说每个shard都有一个primary shard（主分片），负责写入数据，但是还有几个replica shard（副本分片）。primary shard写入数据之后，会将数据同步到其他几个replica shard上去，（ES写数据是往主分片写数据，读取数据是主分片和副本分片都可以读的）。

通过这个replica的方案，每个shard的数据都有多个备份，如果某个机器宕机了，没关系啊，还有别的数据副本在别的机器上呢。这样就高可用了。



es集群多个节点，会自动选举一个节点为master节点，这个master节点其实就是干一些管理的工作的，比如维护索引元数据，负责切换primary shard和replica shard的身份等工作。

要是master节点宕机了，那么会重新选举一个节点为master节点。

如果是非master节点宕机了，那么会由master节点，让那个宕机节点上的primary shard的身份转移到其他机器上的replica shard。接着当你修复了那个宕机节点并重启之后，master节点会控制其他分片将后续修改的数据同步到该节点的分片中，让集群恢复正常。

2. es写入数据的工作原理是？es查询数据的工作原理？

ES 写数据过程

1）客户端随机选择一个`node`发送请求过去，这个`node`就是`coordinating node`（协调节点）。
2）协调节点默认通过`_id`对`document`进行路由（包含Hash的一个算法），找到算出来的`shard`(主分片)，将请求转发给对应的node。
3）实际的`node`上的`primary shard`处理请求之后，然后将数据同步到所有的`replica shard`中去。
4）协调节点判断`primary node`和所有`replica node`都同步完数据之后，就返回响应结果给客户端。

> 当写入一个文档的时候，文档会被存储到一个主分片中。 Elasticsearch 如何知道一个文档应该存放到哪个主分片中呢?
> 是根据下面这个算法决定的：
> shard_num = hash(_routing) % num_primary_shards

> 其中 _routing 是一个可变值，默认是文档的 _id 的值 ，也可以设置成一个自定义的值。 _routing 通过 hash 函数生成一个数字，然后这个数字再除以 num_of_primary_shards （主分片的数量）后得到余数 。这个分布在 0 到 number_of_primary_shards-1 之间的余数，就是我们所寻求的文档所在分片的位置。这就解释了为什么我们要在创建索引的时候就确定好主分片的数量 并且永远不会改变这个数量：因为如果数量变化了，那么所有之前路由的值都会无效，文档就再也找不到了。

==> 写数据底层原理 （数据在主分片中的写入流程）

1）先写入内存buffer，同时将数据写入translog日志文件（每个shard都对应一个translog文件）。在buffer里的时候数据是搜索不到的。

2）如果buffer快满了，或者到一定时间，就会将buffer数据**refresh**到一个新的segment file（磁盘文件）中，这里数据不是直接进入segment file的磁盘文件的，而是先进入os cache（磁盘文件缓存）的。这个过程就是refresh。

> 每隔1秒钟，es将buffer中的数据写入一个新的segment file，每秒钟都会产生一个新的磁盘文件 segment file，这个segment file中就存储最近1秒内buffer中写入的数据。
> 如果buffer里面此时没有数据，那当然不会执行refresh操作，如果buffer里面有数据，默认1秒钟refresh一次，将数据刷入一个新的segment file中。因此es是准实时的，因为写入的数据1秒之后才能被看到。
>
> 可以通过es的restful api或者java api，手动refresh，就是手动将buffer中的数据刷入os cache中，让数据立马就可以被搜索到。

操作系统里面，有一个叫做os cache的东西，即磁盘文件缓存，就是数据写入磁盘文件之前，会先进入os cache，先进入操作系统级别的一个内存缓存中去。只要buffer中的数据被refresh操作，刷入os cache中，就代表这个数据就可以被搜索到了。

3）只要数据进入os cache，此时就可以让这个数据对外提供搜索了。

4）重复1~3步骤，新的数据不断进入buffer和translog，不断refresh，将buffer数据写入一个又一个新的segment file中去，每次refresh之后buffer被清空，translog保留。随着这个过程推进，translog会变得越来越大。当translog达到一定长度的时候，就会触发commit操作。

> buffer中的数据，每隔1秒就被刷到os cache中去，之后这个buffer被清空。所以说buffer的数据始终是可以保持不会填满es进程的内存。
> 每次一条数据写入buffer，同时会写入一条日志到translog日志文件中去，所以这个translog日志文件是不断变大的，大到一定程度（阈值）的时候，就会执行commit操作。

5）commit操作，就是将buffer中现有数据refresh到os cache中去，然后清空buffer，再生成一个commit point文件并写入磁盘（这个文件里面记录着它对应的所有可用的segment file标识）。然后强行将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操作的整个过程在ES中叫`flush`，默认每隔30分钟会自动执行一次commit，但是如果translog过大，也会触发commit。我们也可以手动执行flush操作，就是将所有os cache数据刷到磁盘文件中去。

commit操作：1、写commit point标识文件；2、将os cache数据fsync强刷到磁盘上去；3、清空translog日志文件

> translog日志文件的作用是什么？就是在你执行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日志文件中，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。

6）translog其实也是先写入os cache的，默认每隔5秒刷一次到磁盘的translog日志文件中，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。

实际上这里，如果面试官没有问你 es 丢数据的问题，你可以在这里给面试官炫一把，你说其实 es 第一是准实时的，数据写入 1 秒后可以搜索到；但可能会丢数据。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**。

7）如果是删除操作，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被删除了。

8）如果是更新操作，就是将原来的doc标识为deleted状态，然后新写入一条数据。

9）buffer每次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，segment file会越来越多，此时ES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文件，记录所有新的segment file标识，然后打开segment file供搜索使用，同时删除旧的segment file。

> 注意：当我们在这个commit point上进行搜索时，就相当于在它下面的segment中进行搜索

当segment file多到一定程度的时候，es就会自动触发merge操作，将多个segment file给merge成一个segment file。

> es里的写流程，有4个底层的核心概念: **refresh、flush、translog、merge**

所以总结是有三个批次操作：

- **1秒做一次refresh保证近实时搜索，**
- **5秒做一次translog持久化保证数据未持久化前留底，**
- **30分钟做一次数据持久化**

总结一下，数据先写入内存 buffer，然后每隔 1s，将数据 refresh 到 os cache，到了 os cache 数据就能被搜索到（所以我们才说 es 从写入到能被搜索到，中间有 1s 的延迟）。每隔 5s，将数据写入 translog 文件（这样如果机器宕机，内存数据全没，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，将缓冲区的数据都 flush 到 segment file 磁盘文件中。

> 数据写入segment file之后，同时就建立起来了倒排索引。

ES 读数据过程

查询过程：

通过doc_Id来GET某一条数据，在写入了某个document的时候，这个document会自动分配一个全局唯一的id：doc_id，同时也是根据doc_id进行hash路由到对应的primary shard上面去的。也可以手动指定doc_id，比如用订单id，用户id。

通过doc_id来查询的时候，根据doc_id进行hash，判断出来doc_id被分配到了哪个shard上面去，从那个shard去查询document。

1）客户端发送请求到任意一个node，成为coordinate node协调节点，
2）协调节点对document进行路由，将请求转发到对应的node，此时会使用随机轮询算法，在primary shard以及其所有replica shard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

搜索过程：

1）客户端发送请求到一个coordinate node协调节点 。

2）协调节点将搜索请求转发到所有的shard对应的primary shard或replica shard中去（这里系统会采用相应的负载算法，确保各个分片的读负载）。

3）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。

4）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。

3.大数据量下如何提高ES查询效率?

1）性能优化的杀手锏——filesystem cache

> os cache，操作系统的缓存。

往es里写的数据，实际上都写到磁盘文件里去了，磁盘文件里的数据操作系统会自动将里面的数据缓存到os cache里面去。es的搜索引擎严重依赖于底层的filesystem cache，你如果给filesystem cache更多的内存，尽量让内存可以容纳所有的indx segment file索引数据文件，那么你搜索的时候就基本都是走内存的，性能会非常高。我们之前很多的测试和压测，如果走磁盘，搜索性能绝对是秒级。但是如果走filesystem cache，走纯内存的话，性能比走磁盘要高一个数量级，基本上是毫秒级的。

> 比如说，es节点有3台机器，每台机器，看起来内存很多，64G，总内存，64 * 3 = 192g，每台机器给es jvm heap是32G，那么剩下来留给filesystem cache的就是每台机器才32g，总共集群里给filesystem cache的就是32 * 3 = 96g内存。
> ok，那么你往es集群里写入的数据有多少数据量?
> 如果此时，整个磁盘上索引数据文件，在3台机器上，一共占用了1T的磁盘容量，你的es数据量是1T，每台机器的数据量是300g，你觉得性会能好吗？filesystem cache的内存才100g，十分之一的数据可以放内存，其他的都在磁盘，然后你执行搜索操作，大部分操作都是走磁盘，性能肯定差！
> 归根结底，你要让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。比如说，你一共要在es中存储1T的数据，那么你的多台机器留个filesystem cache的内存加起来综合，至少要到512G，至少半数的情况下，搜索是走内存的，性能一般可以到几秒钟。如果最佳的情况下，我们生产环境实践过，仅仅在es中存少量的数据，内存留给filesystem cache就100G，然后我们控制在索引数据在100GB以内，相当于数据几乎全部走内存来搜索，性能非常之高，一般可以在1秒以内。

2）ES存储必须检索的字段，其他字段存Hbase

比如说现在有一行数据：id name age .... 一共30个字段，但是现在搜索只需要根据id name age这三个字段来搜索，其他字段用不上。如果傻乎乎的往es里写入所有字段，导致70%的数据是搜索用不到的，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。仅仅写入es中要用来检索的少数字段即可，比如说就写入id、name、age三个字段就行了，把其他的字段的数据存储到mysql里面，我们一般是建议用es + hbase的这么一个架构。

> 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，不要做复杂的搜索，就是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id，然后根据id到hbase里去查询每个 id对应的完整的数据，再返回给前端。这样的话，es数据量很少，3个字段的数据，索引中的所有数据都可以放导内存里了。

最好写入es的数据小于等于，或者是略微大于es的filesystem cache的内存容量。然后从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，1T数据都放es，会每次查询都是5~10秒，但是现在性能就特别高，每次查询就是50ms。

3）数据预热

假如说，哪怕你就按上述的方案做了，es集群中每个机器写入的数据量还是超过了filesystem cache一倍，比如说你写入一台机器60g数据，结果filesystem cache就30g，还是有30g数据留在了磁盘上。

> 举个例子，比如说微博，可以把一些大v啊等平时看的人很多的数据给提前在后台搞个程序，每隔一段时间去搜索一下热数据，刷到filesystem cache里去，后面用户实际上来看这个热数据的时候，他们就是直接从内存里搜索了，这样就会很快。
>
> 比如说电商系统，可以将平时浏览量比较多的一些商品，这些热数据，做一个专门的缓存预热子系统，每隔1分钟自己主动访问一次，刷到filesystem cache里去。这样下次别人访问的时候，一定性能会好一些。

4）冷热分离

关于es性能优化，数据拆分，将大量不搜索的字段，拆分到别的地方去存储，这个类似于mysql分库分表的垂直拆分。

es也可以做类似于mysql的水平拆分，将大量的访问很少、频率很低的数据单独写一个索引，然后将访问很频繁的热数据单独写一个索引。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。

> 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，然后你大量访问热数据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都不再相同的机器上，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。

5）document模型设计（索引结构）

在es里面，复杂的关联查询，复杂的查询语法，尽量别用，一旦用了性能一般都不太好。

设计es里的数据模型的时候，将计算结果字段设计好，然后计算逻辑在java程序中去执行。在写ES的java程序里，就已经完成关联计算，将关联好的数据直接写入es中，搜索的时候就不需要利用es的搜索语法去完成join来搜索了。

document模型设计是非常重要的，很多操作不要在搜索的时候才想去执行各种复杂的乱七八糟的操作。es能支持的操作就是那么多，不要考虑用es做一些它不好操作的事情。如果真的有那种操作，尽量在document模型设计的时候，在写入的时候就完成。另外对于一些太复杂的操作，比如join，nested，parent-child搜索都要尽量避免，性能都很差的。

两个思路，在搜索/查询的时候，要执行一些业务强相关的特别复杂的操作：
1）在写入数据的时候，就设计好模型，加几个字段，把处理好的数据写入加的字段里面。
2）自己用java程序封装，es能做的用es做，es做不了的在java程序里面去做，比如说基于es查询结果，再用java封装一些特别复杂的操作。

6）分页性能优化 !!!!!!!!!!!!!!!!!!

es的分页是较坑的，举个例子吧，假如你每页是10条数据，你现在要查询第100页，实际上是会把每个shard上存储的前1000条数据都查到一个协调节点上，如果你有个5个shard，那么就有5000条数据，接着协调节点对这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。

ES它是分布式的，你要查第100页的10条数据，不可能从5个shard，每个shard就查2条数据，最后到协调节点合并成10条数据。你必须从每个shard都查1000条数据过来，然后根据需求进行排序、筛选等等操作，最后再次分页，拿到里面第100页的数据。

翻页的时候，翻的越深，每个shard返回的数据就越多，而且协调节点处理的时间越长。非常坑爹。所以用es做分页的时候，你会发现越翻到后面越慢。

我们之前也是遇到过这个问题，用es作分页，前几页就几十毫秒，翻到10页之后，几十页的时候，基本上就要5~10秒才能查出来一页数据了。

- **不允许深度分页**/默认深度分页性能很惨，系统不允许翻那么深的页，默认翻的越深，性能就越差。
- 类似于app里的推荐商品不断下拉出来一页一页的。类似于微博中，下拉刷微博，刷出来一页一页的，你可以用**scroll api滚动查询**。scroll会一次性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，性能会比上面说的那种分页性能也高很多很多。但是唯一的缺点就是，它适合于那种类似微博下拉翻页，不能随意跳到任何一页的场景。同时这个scroll是要保留一段时间内的数据快照的，你需要确保用户不会持续不断翻页翻几个小时。 无论翻多少页，性能基本上都是毫秒级的。因为scroll api是只能一页一页往后翻的，是不能说，先进入第10页，然后去120页，回到58页，不能随意乱跳页。所以现在很多产品，都是不允许你随意翻页的。

4.生产ES集群怎么部署的？多少数据量？多少分片？

参考下面的三点进行回答：

1. es生产集群我们部署了5台机器，每台机器是8核64G的，集群总内存是320G。
2. 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。
3. 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。

5.elasticsearch 的倒排索引

传统的我们的检索是通过文章，逐个遍历找到对应关键词的位置。

而倒排索引，是通过分词策略，形成了词和文章的映射关系表，这种词典+映射表即为倒排索引。有了倒排索引，就能实现 O(1) 时间复杂度的效率检索文章了，极大的提高了检索效率。

倒排索引由两部分组成——**词典**和**倒排表**。

倒排索引的底层实现是基于：FST（Finite State Transducer）数据结构。

lucene 从 4+版本后开始大量使用的数据结构是 FST。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。

6. ES跨索引查询

Elasticsearch 跨索引查询的方式可依据业务场景灵活选择，下面介绍几种：

直接型

明确指定多个索引名称，如：index_01,index_02

Java 复制代码

```
GET /index_01,index_02/_search
{
  "query" : {
    "match": {
      "test": "data"
    }
  }
}
```

模糊型

不限定死索引名称，这种方式一般采用通配符：GET /index_*/_search

```java
GET /index_*/_search
{
  "query" : {
    "match": {
      "test": "data"
    }
  }
}
```

计算型

索引名称通过计算表达式指定，类似正则表达式，也可以同时指定多个索引，如下：logstash-{now/d}表示当前日期：

```java
# 索引名称如：index-2024.03.22
# GET /<index-{now/d}>/_search
GET /%3Cindex-%7Bnow%2Fd%7D%3E/_search{
  "query" : {
    "match": {
      "test": "data"
    }
  }
}
```

也可以为多个索引指定别名，当需要全部查询时，直接通过别名进行查询即可。

7. ES 滚动查询

Scroll是先做一次初始化搜索**把所有符合搜索条件的结果缓存起来生成一个快照**，然后持续地、批量地从快照里拉取数据直到没有数据剩下。而这时对索引数据的插入、删除、更新都不会影响遍历结果，因此scroll 并不适合用来做实时搜索。

Scan是搜索类型，告诉Elasticsearch不用对结果集进行排序，只要分片里还有结果可以返回，就返回一批结果。scroll- scan使用中不能跳页获取结果，必须一页接着一页获取。

为了使用scroll-scan，需要执行一个初始化搜索请求，将search_type设置成scan，并且传递一个scroll参数来告诉 Elasticsearch缓存应该持续多长时间，在缓存持续时间内初始化搜索请求后对索引的修改不会反应到快照中。每次搜索请求后都会返回一个scrollId，是一个 64 位的字符串编码，后续会使用此scrollId来获取数据。scroll时间指的是本次数据处理所需要的时间，如果超过此时间，继续使用该scrollId搜索数据则会报错。在使用scroll-scan时可以指定返回结果集大小，在 scan 的时候，size 作用在每个分片上，所以将会在每批次中得到最大为 `size * 主分片数` 个文档。

扫描式滚动与标准滚动不同：

- 不算分，关闭排序。结果会按照在索引中出现的顺序返回。
- 不支持聚合。
- 初始 search 请求的响应不会在 hits 数组中包含任何结果。第一批结果就会按照第一个 scroll 请求返回。
- 参数 size 控制了每个分片上而非每个请求的结果数目，所以 size 为 10 的情况下，如果命中了 5 个分片，那么每个 scroll 请求最多会返回 50 个结果。