es的内存消耗主要有这几个方面：

## segment

一个segment是一个完备的lucene倒排索引，而倒排索引是通过词典到文档列表的映射关系，快速做查询的。由于词典的size会很大，全部装载到heap里不现实，因此lucene为词典做了一层前缀索引，在lucene4.0之后采用的数据结构是FST。这种数据结构占用空间很小，lucene打开索引的时候将其全量装载到内存中，加快磁盘上词典查询速度的同时减少随机磁盘访问次数。（即tip文件内容）

为了加快数据的访问，每个segment都会有一些索引数据驻留在heap里。因此segment越多，瓜分掉的heap也越多，并且这部分的heap是无法被GC掉的。所以，当一个node的segment memory占用过多的时候，就需啊哟考虑删除、归档数据，或者扩容了。

查看集群所有的segment的memory情况：

<http://127.0.0.1:9200/_cat/segments?v&h=index,shard,segment,size,size.memory>

查看一个索引所有的segment的memory情况：

<http://127.0.0.1:9200/_cat/segments/test?v&h=shard,segment,size,size.memory>

查看一个node上所有的segment占用的memory总和：

<http://127.0.0.1:9200/_cat/nodes?v&h=name,port,sm>

减少节点上的segment memory占用的方法：

1. 删除不用的索引
2. 关闭索引（释放内存），需要的时候再重新打开
3. 定期对不再更新的索引做merge（force merge api），即对segment file强制做合并，可以节省大量的segment memory

## Filter Cache

Filter cache是用来缓存使用过的filter的结果集的，需要注意的是这个缓存也是常驻heap，无法GC的。别人的经验是默认的10% heap设置工作得够好了，如果实际使用中heap没什么压力的情况下，才考虑加大这个设置。

## Field Data cache

在有大量排序、数据聚合的应用场景，可以说field data cache是性能和稳定性的杀手。 对搜索结果做排序或者聚合操作，需要将倒排索引里的数据进行解析，然后进行一次倒排。 这个过程非常耗费时间，因此ES 2.0以前的版本主要依赖这个cache缓存已经计算过的数据，提升性能。但是由于heap空间有限，当遇到用户对海量数据做计算的时候，就很容易导致heap吃紧，集群频繁GC，根本无法完成计算过程。 ES2.0以后，正式默认启用Doc Values特性(1.x需要手动更改mapping开启)，将field data在indexing time构建在磁盘上，经过一系列优化，可以达到比之前采用field data cache机制更好的性能。因此需要限制对field data cache的使用，最好是完全不用，可以极大释放heap压力。

## Bulk Queue

一般来说，Bulk queue不会消耗很多的heap，但是见过一些用户为了提高bulk的速度，客户端设置了很大的并发量，并且将bulk Queue设置到不可思议的大，比如好几千。 Bulk Queue是做什么用的？当所有的bulk thread都在忙，无法响应新的bulk request的时候，将request在内存里排列起来，然后慢慢清掉。 这在应对短暂的请求爆发的时候有用，但是如果集群本身索引速度一直跟不上，设置的好几千的queue都满了会是什么状况呢？ 取决于一个bulk的数据量大小，乘上queue的大小，heap很有可能就不够用，内存溢出了。一般来说官方默认的thread pool设置已经能很好的工作了，建议不要随意去“调优”相关的设置，很多时候都是适得其反的效果。

## Indexing Buffer

Indexing Buffer是用来缓存新数据，当其满了或者refresh/flush interval到了，就会以segment file的形式写入到磁盘。 这个参数的默认值是10% heap size。根据经验，这个默认值也能够很好的工作，应对很大的索引吞吐量。 但有些用户认为这个buffer越大吞吐量越高，因此见过有用户将其设置为40%的。到了极端的情况，写入速度很高的时候，40%都被占用，导致OOM。

## Cluster State Buffer

ES被设计成每个node都可以响应用户的api请求，因此每个node的内存里都包含有一份集群状态的拷贝。这个cluster state包含诸如集群有多少个node，多少个index，每个index的mapping是什么？有多少shard，每个shard的分配情况等等 (ES有各类stats api获取这类数据)。 在一个规模很大的集群，这个状态信息可能会非常大的，耗用的内存空间就不可忽视了。频繁的状态更新都有可能给heap带来压力。 在超大规模集群的情况下，可以考虑分集群并通过tribe node连接做到对用户api的透明，这样可以保证每个集群里的state信息不会膨胀得过大。

## 超大搜索聚合结果集的fetch

ES是分布式搜索引擎，搜索和聚合计算除了在各个data node并行计算以外，还需要将结果返回给汇总节点进行汇总和排序后再返回。无论是搜索，还是聚合，如果返回结果的size设置过大，都会给heap造成很大的压力，特别是数据汇聚节点。超大的size多数情况下都是用户用例不对，比如本来是想计算cardinality，却用了terms aggregation + size:0这样的方式; 对大结果集做深度分页；一次性拉取全量数据等等。

## 小结

1. 倒排词典的索引需要常驻内存，无法GC，需要监控data node上segment memory增长趋势。

2. 各类缓存，field cache, filter cache, indexing cache, bulk queue等等，要设置合理的大小，并且要应该根据最坏的情况来看heap是否够用，也就是各类缓存全部占满的时候，还有heap空间可以分配给其他任务吗？避免采用clear cache等“自欺欺人”的方式来释放内存。

3. 避免返回大量结果集的搜索与聚合。确实需要大量拉取数据的场景，可以采用scan & scroll api来实现。

4. cluster stats驻留内存并无法水平扩展，超大规模集群可以考虑分拆成多个集群通过tribe node连接。

5. 想知道heap够不够，必须结合实际应用场景，并对集群的heap使用情况做持续的监控。