**Elasticsearch学习笔记**

一．一些设置

1. 监控 fielddata（Monitoring fielddata）

无论是仔细监控 fielddata 的内存使用情况， 还是看有无数据被回收都十分重要。高的回收数可以预示严重的资源问题以及性能不佳的原因。

Fielddata 的使用可以被监控：

按索引使用 indices-stats API ：

GET /\_stats/fielddata?fields=\*

按节点使用 nodes-stats API ：

GET /\_nodes/stats/indices/fielddata?fields=\*

按索引节点：

GET /\_nodes/stats/indices/fielddata?level=indices&fields=\*

使用设置 ?fields=\* ，可以将内存使用分配到每个字段。

断路器设置：

PUT /\_cluster/settings

{

"persistent" : {

"indices.breaker.fielddata.limit" : "40%" https://www.elastic.co/guide/cn/elasticsearch/guide/current/images/icons/callouts/1.png

}

}

2.设置分片数

PUT /my\_index/\_settings

{

"number\_of\_replicas": 1

}

3.索引切换(重命名索引)

POST /\_aliases

{

"actions": [

{ "add": { "alias": "logs\_current", "index": "logs\_2014-10" }}, https://www.elastic.co/guide/cn/elasticsearch/guide/current/images/icons/callouts/1.png

{ "remove": { "alias": "logs\_current", "index": "logs\_2014-09" }}, https://www.elastic.co/guide/cn/elasticsearch/guide/current/images/icons/callouts/2.png

{ "add": { "alias": "last\_3\_months", "index": "logs\_2014-10" }}, https://www.elastic.co/guide/cn/elasticsearch/guide/current/images/icons/callouts/3.png

{ "remove": { "alias": "last\_3\_months", "index": "logs\_2014-07" }} https://www.elastic.co/guide/cn/elasticsearch/guide/current/images/icons/callouts/4.png

]

}

|  |  |
| --- | --- |
| [https://www.elastic.co/guide/cn/elasticsearch/guide/current/images/icons/callouts/1.png](https://www.elastic.co/guide/cn/elasticsearch/guide/current/time-based.html#CO290-1) [https://www.elastic.co/guide/cn/elasticsearch/guide/current/images/icons/callouts/2.png](https://www.elastic.co/guide/cn/elasticsearch/guide/current/time-based.html#CO290-2) | 将 logs\_current 由九月切换至十月。 |
| [https://www.elastic.co/guide/cn/elasticsearch/guide/current/images/icons/callouts/3.png](https://www.elastic.co/guide/cn/elasticsearch/guide/current/time-based.html#CO290-3) [https://www.elastic.co/guide/cn/elasticsearch/guide/current/images/icons/callouts/4.png](https://www.elastic.co/guide/cn/elasticsearch/guide/current/time-based.html#CO290-4) | 将十月添加到 last\_3\_months 并且删掉七月。 |

4.迁移索引

./bin/elasticsearch --node.box\_type strong

box\_type 参数是完全随意

PUT /logs\_2014-10-01

{

"settings": {

"index.routing.allocation.include.box\_type" : "strong"

}

}

5.关闭索引

POST /logs\_2014-01-\*/\_flush https://www.elastic.co/guide/cn/elasticsearch/guide/current/images/icons/callouts/1.png

POST /logs\_2014-01-\*/\_close https://www.elastic.co/guide/cn/elasticsearch/guide/current/images/icons/callouts/2.png

POST /logs\_2014-01-\*/\_open https://www.elastic.co/guide/cn/elasticsearch/guide/current/images/icons/callouts/3.png

|  |  |
| --- | --- |
| [https://www.elastic.co/guide/cn/elasticsearch/guide/current/images/icons/callouts/1.png](https://www.elastic.co/guide/cn/elasticsearch/guide/current/retiring-data.html#CO292-1) | 刷写（Flush）所有一月的索引来清空事务日志。 |
| [https://www.elastic.co/guide/cn/elasticsearch/guide/current/images/icons/callouts/2.png](https://www.elastic.co/guide/cn/elasticsearch/guide/current/retiring-data.html#CO292-2) | 关闭所有一月的索引. |
| [https://www.elastic.co/guide/cn/elasticsearch/guide/current/images/icons/callouts/3.png](https://www.elastic.co/guide/cn/elasticsearch/guide/current/retiring-data.html#CO292-3) | 当你需要再次访问它们时，使用 open API 来重新打开它们。 |

6.fielddata设置

## Fielddata and doc values settings

Today we have these settings:

* doc\_values: true|false
* fielddata.format: disabled | doc\_values | paged\_bytes | array
* fielddata.loading: lazy | eager | eager\_global\_ordinals
* fielddata.filters: frequency:{}, regex:{}

These become a lot easier to simplify if we deprecate fielddata for all but analyzed string fields.

**Proposal for fields that support doc values:**

* doc\_values : true (default) | false
* global\_ordinals : lazy (default) | eager

**Proposal for analyzed string fields:**

* fielddata: disabled (default) | lazy | eager
* global\_ordinals : lazy (default) | eager
* fielddata.filters: frequency:{}, regex:{}

If, in the future, we can automatically figure out which global ordinals need to be built eagerly, then we can remove the global\_ordinals setting.

PUT /music/\_mapping/song

{

"properties": {

"tag": {

"type": "string",

"fielddata": { https://elasticsearch.cn/book/elasticsearch_definitive_guide_2.x/images/icons/callouts/1.png

"filter": {

"frequency": { https://elasticsearch.cn/book/elasticsearch_definitive_guide_2.x/images/icons/callouts/2.png

"min": 0.01, https://elasticsearch.cn/book/elasticsearch_definitive_guide_2.x/images/icons/callouts/3.png

"min\_segment\_size": 500 https://elasticsearch.cn/book/elasticsearch_definitive_guide_2.x/images/icons/callouts/4.png

}

}

}

}

}

}

PUT /music/\_mapping/\_song

{

"tags": {

"type": "string",

"fielddata": {

"loading" : "eager" https://elasticsearch.cn/book/elasticsearch_definitive_guide_2.x/images/icons/callouts/1.png

}

}

}

PUT /music/\_mapping/\_song

{

"tags": {

"type": "string",

"fielddata": {

"loading" : "eager" https://elasticsearch.cn/book/elasticsearch_definitive_guide_2.x/images/icons/callouts/1.png

}

}

}

二、监控、部署

1.集群健康

GET \_cluster/health

{

"cluster\_name": "elasticsearch\_zach",

"status": "green",

"timed\_out": false,

"number\_of\_nodes": 1,

"number\_of\_data\_nodes": 1,

"active\_primary\_shards": 10,

"active\_shards": 10,

"relocating\_shards": 0,

"initializing\_shards": 0,

"unassigned\_shards": 0

}

green

所有的主分片和副本分片都已分配。你的集群是 100% 可用的。

yellow

所有的主分片已经分片了，但至少还有一个副本是缺失的。不会有数据丢失，所以搜索结果依然是完整的。不过，你的高可用性在某种程度上被弱化。如果 *更多的* 分片消失，你就会丢数据了。把 yellow 想象成一个需要及时调查的警告。

red

至少一个主分片（以及它的全部副本）都在缺失中。这意味着你在缺少数据：搜索只能返回部分数据，而分配到这个分片上的写入请求会返回一个异常。

* number\_of\_nodes 和 number\_of\_data\_nodes 这个命名完全是自描述的。
* active\_primary\_shards 指出你集群中的主分片数量。这是涵盖了所有索引的汇总值。
* active\_shards 是涵盖了所有索引的\_所有\_分片的汇总值，即包括副本分片。
* relocating\_shards 显示当前正在从一个节点迁往其他节点的分片的数量。通常来说应该是 0，不过在 Elasticsearch 发现集群不太均衡时，该值会上涨。比如说：添加了一个新节点，或者下线了一个节点。
* initializing\_shards 是刚刚创建的分片的个数。比如，当你刚创建第一个索引，分片都会短暂的处于 initializing 状态。这通常会是一个临时事件，分片不应该长期停留在 initializing 状态。你还可能在节点刚重启的时候看到 initializing 分片：当分片从磁盘上加载后，它们会从 initializing 状态开始。
* unassigned\_shards 是已经在集群状态中存在的分片，但是实际在集群里又找不着。通常未分配分片的来源是未分配的副本。比如，一个有 5 分片和 1 副本的索引，在单节点集群上，就会有 5 个未分配副本分片。如果你的集群是 red 状态，也会长期保有未分配分片（因为缺少主分片）

GET \_cluster/health?level=indices

有关每个索引的细节（状态、分片数、未分配分片数等等）

GET \_cluster/health?level=shards

列出每个索引里每个分片的状态和位置

2.节点信息

GET \_nodes/stats



索引(indices) 部分列出了这个节点上所有索引的聚合过的统计值 ：

"indices": {

"docs": {

"count": 6163666,

"deleted": 0

},

"store": {

"size\_in\_bytes": 2301398179,

"throttle\_time\_in\_millis": 122850

},

返回的统计值被归入以下部分：

* docs 展示节点内存有多少文档，包括还没有从段里清除的已删除文档数量。
* store 部分显示节点耗用了多少物理存储。这个指标包括主分片和副本分片在内。如果限流时间很大，那可能表明你的磁盘限流设置得过低（在[段和合并](https://www.elastic.co/guide/cn/elasticsearch/guide/current/indexing-performance.html#segments-and-merging)里讨论过）。

"indexing": {

"index\_total": 803441,

"index\_time\_in\_millis": 367654,

"index\_current": 99,

"delete\_total": 0,

"delete\_time\_in\_millis": 0,

"delete\_current": 0

},

"get": {

"total": 6,

"time\_in\_millis": 2,

"exists\_total": 5,

"exists\_time\_in\_millis": 2,

"missing\_total": 1,

"missing\_time\_in\_millis": 0,

"current": 0

},

"search": {

"open\_contexts": 0,

"query\_total": 123,

"query\_time\_in\_millis": 531,

"query\_current": 0,

"fetch\_total": 3,

"fetch\_time\_in\_millis": 55,

"fetch\_current": 0

},

"merges": {

"current": 0,

"current\_docs": 0,

"current\_size\_in\_bytes": 0,

"total": 1128,

"total\_time\_in\_millis": 21338523,

"total\_docs": 7241313,

"total\_size\_in\_bytes": 5724869463

},

* indexing 显示已经索引了多少文档。这个值是一个累加计数器。在文档被删除的时候，数值不会下降。还要注意的是，在发生内部 *索引* 操作的时候，这个值也会增加，比如说文档更新。

还列出了索引操作耗费的时间，正在索引的文档数量，以及删除操作的类似统计值。

* get 显示通过 ID 获取文档的接口相关的统计值。包括对单个文档的 GET 和 HEAD 请求。
* search 描述在活跃中的搜索（ open\_contexts ）数量、查询的总数量、以及自节点启动以来在查询上消耗的总时间。用 query\_time\_in\_millis / query\_total 计算的比值，可以用来粗略的评价你的查询有多高效。比值越大，每个查询花费的时间越多，你应该要考虑调优了。

fetch 统计值展示了查询处理的后一半流程（query-then-fetch 里的 *fetch* ）。如果 fetch 耗时比 query 还多，说明磁盘较慢，或者获取了太多文档，或者可能搜索请求设置了太大的分页（比如， size: 10000 ）。

* merges 包括了 Lucene 段合并相关的信息。它会告诉你目前在运行几个合并，合并涉及的文档数量，正在合并的段的总大小，以及在合并操作上消耗的总时间。

在你的集群写入压力很大时，合并统计值非常重要。合并要消耗大量的磁盘 I/O 和 CPU 资源。如果你的索引有大量的写入，同时又发现大量的合并数，一定要去阅读[索引性能技巧](https://www.elastic.co/guide/cn/elasticsearch/guide/current/indexing-performance.html)。

注意：文档更新和删除也会导致大量的合并数，因为它们会产生最终需要被合并的段 *碎片* 。

"filter\_cache": {

"memory\_size\_in\_bytes": 48,

"evictions": 0

},

"fielddata": {

"memory\_size\_in\_bytes": 0,

"evictions": 0

},

"segments": {

"count": 319,

"memory\_in\_bytes": 65812120

},

...

* filter\_cache 展示了已缓存的过滤器位集合所用的内存数量，以及过滤器被驱逐出内存的次数。过多的驱逐数 *可能* 说明你需要加大过滤器缓存的大小，或者你的过滤器不太适合缓存（比如它们因为高基数而在大量产生，就像是缓存一个 now 时间表达式）。

不过，驱逐数是一个很难评定的指标。过滤器是在每个段的基础上缓存的，而从一个小的段里驱逐过滤器，代价比从一个大的段里要廉价的多。有可能你有很大的驱逐数，但是它们都发生在小段上，也就意味着这些对查询性能只有很小的影响。

把驱逐数指标作为一个粗略的参考。如果你看到数字很大，检查一下你的过滤器，确保他们都是正常缓存的。不断驱逐着的过滤器，哪怕都发生在很小的段上，效果也比正确缓存住了的过滤器差很多。

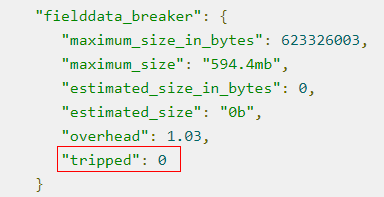
* field\_data 显示 fielddata 使用的内存， 用以聚合、排序等等。这里也有一个驱逐计数。和 filter\_cache 不同的是，这里的驱逐计数是很有用的：这个数应该或者至少是接近于 0。因为 fielddata 不是缓存，任何驱逐都消耗巨大，应该避免掉。如果你在这里看到驱逐数，你需要重新评估你的内存情况，fielddata 限制，请求语句，或者这三者。
* segments 会展示这个节点目前正在服务中的 Lucene 段的数量。 这是一个重要的数字。大多数索引会有大概 50–150 个段，哪怕它们存有 TB 级别的数十亿条文档。段数量过大表明合并出现了问题（比如，合并速度跟不上段的创建）。注意这个统计值是节点上所有索引的汇聚总数。记住这点。

memory 统计值展示了 Lucene 段自己用掉的内存大小。 这里包括底层数据结构，比如倒排表，字典，和布隆过滤器等。太大的段数量会增加这些数据结构带来的开销，这个内存使用量就是一个方便用来衡量开销的度量值。

heap\_used\_percent 指标是值得关注的一个数字。Elasticsearch 被配置为当 heap 达到 75% 的时候开始 GC。如果你的节点一直 >= 75%，你的节点正处于 内存压力 状态。这是个危险信号，不远的未来可能就有慢 GC 要出现了。如果 heap 使用率一直 >=85%，你就麻烦了。Heap 在 90–95% 之间，则面临可怕的性能风险，此时最好的情况是长达 10–30s 的 GC，最差的情况就是内存溢出（OOM）异常。



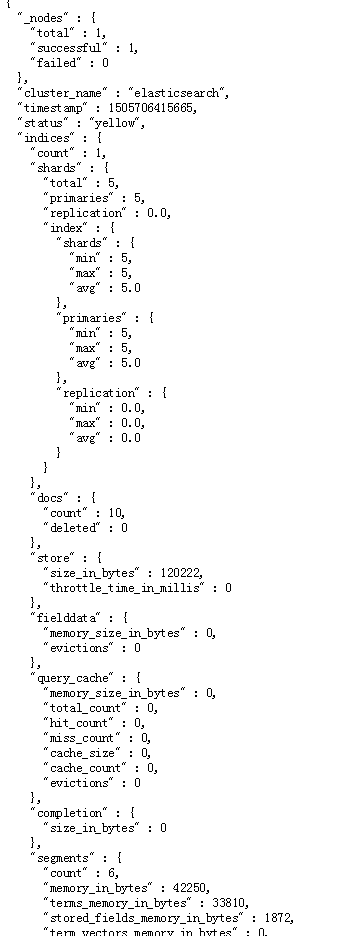
total\_opened 数很大而且还在一直上涨，这是一个明确信号，说明你的 HTTP 客户端里有没启用 keep-alive 长连接的



 tripped 指标。如果这个数字很大或者持续上涨，这是一个信号，说明你的请求需要优化，或者你需要添加更多内存（单机上添加，或者通过添加新节点的方式）。

**3.集群统计**

GET \_cluster/stats



GET \_cluster/pending\_tasks

等待中的任务，唯一可能成为瓶颈的是集群状态非常大 而且更新频繁。

**4.Cat信息**

GET /\_cat/health?v 集群健康

GET /\_cat/nodes?v 集群节点

**5.最小主节点数**

临时（Transient）

这些变更在集群重启之前一直会生效。一旦整个集群重启，这些配置就被清除。

永久（Persistent）

这些变更会永久存在直到被显式修改。即使全集群重启它们也会存活下来并覆盖掉静态配置文件里的选项。

discovery.zen.minimum\_master\_nodes: 2 elasticsearch.yml 文件中这样配置

PUT /\_cluster/settings

{

"persistent" : {

"discovery.zen.minimum\_master\_nodes" : 2 https://www.elastic.co/guide/cn/elasticsearch/guide/current/images/icons/callouts/1.png

},

"transient" : {

"indices.store.throttle.max\_bytes\_per\_sec" : "50mb" https://www.elastic.co/guide/cn/elasticsearch/guide/current/images/icons/callouts/2.png

}

}

|  |  |
| --- | --- |
| [https://www.elastic.co/guide/cn/elasticsearch/guide/current/images/icons/callouts/1.png](https://www.elastic.co/guide/cn/elasticsearch/guide/current/_changing_settings_dynamically.html#CO304-1) | 这个永久设置会在全集群重启时存活下来。 |
| [https://www.elastic.co/guide/cn/elasticsearch/guide/current/images/icons/callouts/2.png](https://www.elastic.co/guide/cn/elasticsearch/guide/current/_changing_settings_dynamically.html#CO304-2) | 这个临时设置会在第一次全集群重启后被移除。 |

elasticsearch.yml 文件中这样配置

gateway.recover\_after\_nodes: 8

gateway.expected\_nodes: 10

gateway.recover\_after\_time: 5m

等待集群至少存在 8 个节点，等待 5 分钟或者10 个节点上线后，才进行数据恢复

**6.集群使用单播**

discovery.zen.ping.unicast.hosts: ["host1", "host2:port"]

**7.禁止内存交换**

bootstrap.mlockall: true

 elasticsearch.yml 文件中设置

**8.文件描述数**

GET /\_nodes/process



|  |
| --- |
| max\_file\_descriptors 字段显示 Elasticsearch 进程可以访问的可用文件描述符数量。 |

Elasticsearch 对各种文件混合使用了 NioFs（ 注：非阻塞文件系统）和 MMapFs （ 注：内存映射文件系统）。请确保你配置的最大映射数量，以便有足够的虚拟内存可用于 mmapped 文件。这可以暂时设置：

sysctl -w vm.max\_map\_count=262144

或者你可以在 /etc/sysctl.conf 通过修改 vm.max\_map\_count 永久设置它。

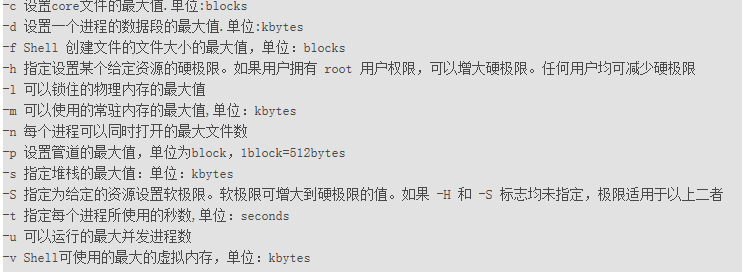
index.merge.scheduler.max\_thread\_count: 1

设置允许 max\_thread\_count + 2 个线程同时进行磁盘操作，index.translog.flush\_threshold\_size 设置，从默认的 512 MB 到更大一些的值，比如 1 GB更少的磁盘 I/O 开销和更好的索引速率

Linux系统：ulimit -a

**ulimit -u 65535**

**ulimit -n 65535**



9.延迟分片推迟

PUT /\_all/\_settings https://www.elastic.co/guide/cn/elasticsearch/guide/current/images/icons/callouts/1.png

{

"settings": {

"index.unassigned.node\_left.delayed\_timeout": "5m"https://www.elastic.co/guide/cn/elasticsearch/guide/current/images/icons/callouts/2.png//5分钟

}

}

瞬时中断的问题，Elasticsearch 可以推迟分片的分配

10.事务日志

translog 叫事务日志：一个文档被索引之后，就会被添加到内存缓冲区，并且 追加到了 translog ，内存缓冲区的文档被写入到一个新的段中使其可被搜索，内存缓冲区被清空，translog 变得越来越大--索引被刷新（flush），刷新（refresh）完成后, 缓存被清空但是事务日志不会，translog 提供所有还没有被刷到磁盘的操作的一个持久化纪录，写入磁盘后tanslog清空

PUT /my\_logs

{

"settings": {

"refresh\_interval": "30s" https://www.elastic.co/guide/cn/elasticsearch/guide/current/images/icons/callouts/1.png

}

}

PUT /my\_index/\_settings

{

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

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

}

**11.IO设置**

默认值是 20 MB/s，对机械磁盘应该是个不错的设置

PUT /\_cluster/settings

{

"persistent" : {

"indices.store.throttle.max\_bytes\_per\_sec" : "100mb"

}

}

如果你使用的是机械磁盘而非 SSD，你需要添加下面这个配置到你的 elasticsearch.yml 里：

index.merge.scheduler.max\_thread\_count: 1

机械磁盘在并发 I/O 支持方面比较差，所以我们需要降低每个索引并发访问磁盘的线程数。这个设置允许 max\_thread\_count + 2 个线程同时进行磁盘操作，也就是设置为 1 允许三个线程。

你可以增加 index.translog.flush\_threshold\_size 设置，从默认的 512 MB 到更大一些的值，比如 1 GB。这可以在一次清空触发的时候在事务日志里积累出更大的段。而通过构建更大的段，清空的频率变低，大段合并的频率也变低。这一切合起来导致更少的磁盘 I/O 开销和更好的索引速率

**设置堆内存(系统内存的50%，最大32G)**

export ES\_HEAP\_SIZE=10g

通常来说，设置 ES\_HEAP\_SIZE 环境变量，比直接写 -Xmx -Xms 更好一点。

### Swapping 是性能的坟墓

这是显而易见的， 但是还是有必要说的更清楚一点：内存交换 到磁盘对服务器性能来说是 致命 的。想想看：一个内存操作必须能够被快速执行。

如果内存交换到磁盘上，一个 100 微秒的操作可能变成 10 毫秒。 再想想那么多 10 微秒的操作时延累加起来。 不难看出 swapping 对于性能是多么可怕。

最好的办法就是在你的操作系统中完全禁用 swap。这样可以暂时禁用：

sudo swapoff -a

如果需要永久禁用，你可能需要修改 /etc/fstab 文件，这要参考你的操作系统相关文档。

如果你并不打算完全禁用 swap，也可以选择降低 swappiness 的值。 这个值决定操作系统交换内存的频率。 这可以预防正常情况下发生交换，但仍允许操作系统在紧急情况下发生交换。

对于大部分Linux操作系统，可以在 sysctl 中这样配置：

vm.swappiness=1

net.core.somaxconn=65535

vm.max\_map\_count=262144

fs.file-max=518144

|  |  |
| --- | --- |
| [https://www.elastic.co/guide/cn/elasticsearch/guide/current/images/icons/callouts/1.png](https://www.elastic.co/guide/cn/elasticsearch/guide/current/heap-sizing.html#CO302-1) | swappiness 设置为 1 比设置为 0 要好，因为在一些内核版本 swappiness 设置为 0 会触发系统 OOM-killer（注：Linux 内核的 Out of Memory（OOM）killer 机制）。 |

最后，如果上面的方法都不合适，你需要打开配置文件中的 mlockall 开关。 它的作用就是允许 JVM 锁住内存，禁止操作系统交换出去。在你的 elasticsearch.yml 文件中，设置如下：

bootstrap.mlockall: true

**三、版本升级**

* 滚动升级（Rolling upgrade）

    Rolling upgrade的备份过程可以让用户在一个时间内只升级集群中的某一个特定的节点。由于Elasticsearch集群具有非常优秀的容灾机制，因此，在删除集群中的某一个节点时，数据并不会丢失，而是可以由其余节点上的拷贝恢复。

    不建议在一个集群中长时间的运行多个版本的Elasticsearch实例，因为当删除的节点恢复时，将来自多个版本实例的数据汇聚到同一个节点会有可能会导致节点无法工作。

    接下来来叙述Rolling upgrade升级的操作步骤：

* + 关闭shard 的实时分配选项，这样做的目的在于当集群shutdown之后可以快速的启动。这个参数默认是开启的，默认情况下当实例启动时，会尝试从其他节点实例上拷贝相关的shard副本至本地，这样会浪费大量的时间和耗费高额的IO资源。如果实时分配选项关闭了，那么当新的实例启动，尝试加入集群的时候，它不会从其他实例上拷贝shard副本。当实例完全启动之后，则应该再将该选项开启，以提供长期的容灾。

　　　　curl -XPUT localhost:9200/\_cluster/settings -d ‘{

"transient" : {

"cluster.routing.allocation.enable" : "none"

}

}‘

* + 关闭所要升级版本的节点实例，并将其移除集群

curl -XPOST ‘http://localhost:9200/\_cluster/nodes/\_local/\_shutdown‘

* + 移除节点之后，等待剩余节点数据转移完成，直到确定所有的shard都被正确地分配。
  + 升级节点的Elasticsearch版本，最简单和最安全的办法就是下载一个全新的Elasticsearch版本到本地，并将原来Elasticsearch的配置文件复制到新的版本中，最好能建立一个Elasticsearch的软连接到最新版本文件所在的目录，这样可以方便将来使用。
  + 启动已经升级好的节点ES实例，并检查其是否正确地加入到集群中。
  + 重新开启shard reallocation选项（实时分配选项）

curl -XPUT localhost:9200/\_cluster/settings -d ‘{

"transient" : {

"cluster.routing.allocation.enable" : "all"

}

}‘