目录

Elasticsearch 分享

1. 资料

右键，在新浏览器窗口中打开
Download PPT: Elasticsearch分享第一期PPT
Download PDF：Elasticsearch第一期文档

部署环境为centos7+,jdk1.8,elasticsearch6.4.0
Download install-jdk.sh: JDK部署脚本
Download install-elasticsearch.sh: Elasticsearch单节点部署脚本

2. 声明

本手册所有操作均基于Elasticsearch6.4版本；本课程是一门原理+实战的课程。
实战演练环境： 为个人阿里云ECS搭建
ES地址
kibana地址

3. 学习资料推荐

《Elasticsearch官方最新文档 》

《Elasticsearch: 权威指南 》

《kibana使用手册官方中文文档 》

1. 倒排索引在elasticsearch中的简单应用

因为演示的ES集群只有两个节点，所以设置一个模板，所有索引不设置副本

```
{
    "order": 0,
    "index_patterns": [
      "*"
    ],
    "settings": {
      "index": {
        "number_of_shards": "2",
        "number_of_replicas": "0"
      }
    },
    "mappings": {},
    "aliases": {}
  }
```

1.1. 创建索引

```
PUT elasticsearch_study_index
{
    "aliases": {},
    "settings": {
      "index": {
        "number_of_shards": "2",
        "number_of_replicas": "0"
      }
    },
    "mappings": {
      "_doc": {
        "properties": {
          "title": {
              "type": "text",
              "analyzer": "whitespace"
          },
          "author":{
              "type": "keyword",
             "ignore_above": 256
          },
          "content": {
            "type": "text",
            "fields": {
              "space": {
                "type": "text",
                "analyzer": "whitespace"
              }
            }
          },
          "descripe": {
            "type": "text",
            "analyzer": "standard"
          }
        }
      }
    }
  }
```

1.2. 插入一条数据

```
POST elasticsearch_study_index/_doc/1
{
  "title": "好未来 是一个以智慧教育和开放平台为主体，探索未来教育新
}
```

1.3. 对数据进行搜索

```
#搜索不到数据
GET elasticsearch_study_index/_search
{
  "query": {
    "match": {
      "title": "好未"
    }
  }
}
# 能够搜索数据
GET elasticsearch_study_index/_search
{
  "query": {
    "match": {
      "title": "好未来"
    }
  }
}
```

查看 `whitespace` 分词结果

```
POST elasticsearch_study_index/_analyze
{
  "analyzer": "whitespace",
  "text": "好未来 是一个以智慧教育和开放平台为主体，探索未来教育新模
}
```

1. 搜索的本质

1. Elasticsearch

1. 分词器

1.1. 自定义分词

```
PUT elasticsearch-analyzer-index
{
  "settings": {
    "analysis": {
      "analyzer": {
        "my_custom_analyzer": {
          "type": "custom",
          "char_filter": [
            "emoticons" ,
            "html_strip"
          ],
          "tokenizer": "punctuation",
          "filter": [
            "lowercase",
            "english_stop",
            "asciifolding"
          ]
        }
      },
      "tokenizer": {
        "punctuation": {
          "type": "pattern",
          "pattern": "[ .,!?。]"
        }
      },
      "char_filter": {
        "emoticons": {
          "type": "mapping",
          "mappings": [
            ":) => _happy_",
            ":( => _sad_",
            "🐂 => 牛"
          ]
        }
      },
      "filter": {
        "english_stop": {
          "type": "stop",
          "stopwords": "_english_"
        }
      }
    }
  },
  "mappings": {
      "_doc": {
        "properties": {
          "title": {
              "type": "text",
```

```
                "analyzer": "my_custom_analyzer",
                "search_analyzer": "my_custom_analyzer"
            },
            "author":{
                "type": "keyword",
               "ignore_above": 256
            },
            "content": {
              "type": "text",
              "fields": {
                "space": {
                  "type": "text",
                  "analyzer": "whitespace"
                }
              }
            },
            "descripe": {
              "type": "text",
              "analyzer": "standard"
            }
          }
        }
    }
}
```

检测分词结果

```
POST elasticsearch-analyzer-index/_analyze
{
  "analyzer": "my_custom_analyzer",
  "text":     "I'm a :) person, and you?<p>好未来🐂</p>"
}

POST elasticsearch-analyzer-index/_analyze
{
  "analyzer": "my_custom_analyzer",
  "text": "Is this <b>déjà vu</b>?"
}
```

响应结如下

```
{
  "tokens": [
    {
      "token": "i'm",
      "start_offset": 0,
      "end_offset": 3,
      "type": "word",
      "position": 0
    },
    {
      "token": "_happy_",
      "start_offset": 6,
      "end_offset": 8,
      "type": "word",
      "position": 2
    },
    {
      "token": "person",
      "start_offset": 9,
      "end_offset": 15,
      "type": "word",
      "position": 3
    },
    {
      "token": "you",
      "start_offset": 21,
      "end_offset": 24,
      "type": "word",
      "position": 5
    },
    {
      "token": """

好未来牛

""",
      "start_offset": 25,
      "end_offset": 37,
      "type": "word",
      "position": 6
    }
  ]
}
```

插入数据

```
POST elasticsearch-analyzer-index/_doc/1
{
  "title": "好未来 是一个以智慧教育和开放平台为主体，I'm a :) per
}

POST elasticsearch-analyzer-index/_doc/_bulk?refresh
{"index":{}}
{"author":"李四1" ,"title": "好未来是 一个 以 智慧教育 和 开放平
{"index":{}}
{"author":"李四2" ,"title": "好未来 是 一个 以 智慧教育和 开放平
```

搜索

```
POST elasticsearch-analyzer-index/_search
{
  "query": {
    "match": {
       "title": "好未来牛"
    }
  }
}

POST elasticsearch-analyzer-index/_search
{
  "query": {
    "match": {
       "title": "好未来"
    }
  }
}
```

官方文档

analysis

1. Elasticsearch基本概念

1.1. 集群（cluster）

一个Elasticsearch集群由一个或多个ES节点组成，并提供集群内所有节点的联合索引和搜索能力（所有节点共同存储数据）。

一个集群被命名为唯一的名字（默认为elasticsearch），集群名称非常重要，因为节点需要通过集群的名称加入集群。

集群的的状态分为三种：`green`，`yellow`，`red`

green

集群所有索引的分片（shard）都被正常的分配到节点上；

yellow

集群中有索引存在副本分片（shard）没有被分配；

red

集群中有索引存在主分片（shard）没有被分配；

查看集群健康信息

```
GET _cluster/health?pretty
GET _cat/health?format=json
```

1.2. 节点（node）

一个节点是集群中的一个服务器，用来存储数据并参与集群的索引和搜索。

和集群类似，节点由一个名称来标识，默认情况下，该名称是在节点启动时分配给节点的随机通用唯一标识符（UUID）。

您也可以自定义任意节点的名称，节点名称对于管理工作很重要， 因为通过节点名称可以确定网络中的哪些服务器对应于Elasticsearch集群中的哪些节点。

一个节点可以被添加到指定名称的集群中。默认情况下，每个节点会

被设置加入到名称为elasticsearch的集群中， 这意味着，如果在您在网络中启动了某些节点（假设这些节点可以发现彼此），它们会自动形成并加入名称为elasticsearch的集群中。

一个集群可以拥有任意多的节点。此外，如果在您的网络中没有运行任何Elasticsearch节点， 此时启动一个节点会创建一个名称为elasticsearch的单节点集群。

节点的角色可以，由以下三个参数来决定，不同角色，可以专注于完成不同的事：

```
node.master: true
node.data: true
node.ingest: true
```

Master Node(专有主节点)
如果节点为以下配置，则该节点为专有候选主节点；
如果选举成功，则该节点成为master,master节点一般不用于和应用创建连接，每个节点都保存了集群状态，master节点不占用磁盘IO和CPU，内存使用量一般。
其他没有选举成为master节点的，仍然保持候选主节点的身份，需要和占用的资源比master节点更少。

```
# 具有资格选举成为主节点的，候选节点
node.master: true
node.data: false
node.ingest: false
```

master节点控制整个集群的元数据。只有Master Node节点可以修改节点状态信息及元数据(metadata)的处理， 比如索引的新增、删除、分片路由分配、所有索引和相关 Mapping 、Setting 配置等等。

候选节点：与集群保持心跳，判断Master是否存活，如果Master故障则参加新一轮的Master选举

集群规划： Elasticsearch集群建议master至少三台(生产建议每个es实例部署在不同的设备上),
三个Master节点最多只能故障一台Master节点，数据不会丢失，如果三个节点故障两个节点，则造成数据丢失并无法组成集群。

Data Node(专有数据节点)
数据节点，该节点和索引应用创建连接、接收索引请求，该节点真正存储数据，ES集群的性能取决于该节点的个数（每个节点最优配置的情况下），data节点会占用**大量的CPU、IO和内存**。

data节点的分片执行查询语句获得查询结果后将结果反馈给协调节点，在查询的过程中非常消耗硬件资源，如果在分片配置及优化没做好的情况下, 进行一次查询非常缓慢(硬件配置也要跟上数据量)。

在Elasticsearch集群中，此节点应该是最多的，单个索引在一个data节点实例上分片数保持在3个以内(建议分片数量按照Data节点数量划分比较好，每个节点上存储一个分片)；
每1GB堆内存对应集群的分片保持在20个以内；每个分片大小不要超过30G，纯搜索型建议不超过10GB，日志型建议30GB-50GB。

```
node.master: false
node.data: true
node.ingest: false
```

内存建议: 假如一台机器部署了一个ES实例，则ES最大可用内存给到物理内存的50%，最多不可超过32G（如果超过32GB，jvm将会使用长指针，官方建议堆内存分配为26-32GB）。 如果单台机器上部署了多个ES实例，则多个ES实例内存相加等于物理内存的50%，多个ES实例内存相加不宜超过32G。

分片建议（理想情况下）：
如果单个分片每个节点可支撑90G数据，依此可计算出所需data节点数。
如果多个分片按照单个data节点jvm内存最大30G来计算，一个节点的分片保持在600个以内，存储保持在18T以内。

Ingest Node(专有数据预处理节点)
ingest 节点可以看作是数据前置处理转换的节点，支持 pipeline管道 设置，可以使用 ingest 对数据进行过滤、转换等操作，类似于 logstash 中 filter 的作用，功能相当强大。

Ingest节点处理时机——在数据被索引之前，通过预定义好的处理管道对数据进行预处理。默认情况下，所有节点都启用Ingest，因此任何节点都可以处理Ingest任务。我们也可以创建专用的Ingest节点。

通常数据的加工建议，放在专有的数据 ETL组件中

```
node.master: false
node.data: false
node.ingest: true
```

Coordinating Node(专有协调节点)
协调节点，该节点和检索应用创建连接、接受检索请求，但其本身不负责存储数据，可当负责均衡节点，该节点不占用io、cpu和内存。但是如果有不合理的聚合查询，也会导致协调节点oom。

协调节点接受客户端搜索请求后将请求转发到与查询条件相关的多个data节点的分片上，然后多个data节点的分片执行查询语句或者查询结果再返回给协调节点，协调节点把各个data节点的返回结果进行整合、排序等一系列操作后再将最终结果返回给用户请求

增加协调节点可增加检索并发,但检索的速度还是取决于查询所命中的分片个数以及分片中的数据量

```
node.master: false
node.data: false
node.ingest: false
```

查看节点信息

```
GET _cat/nodes?v&format=txt
```

1.3. 索引（index）

一个索引是一个拥有一些相似特征的文档的集合（相当于关系型数据库中的一个数据库）。 例如，您可以拥有一个客户数据的索引，以及一个订单数据的索引。 一个索引通常使用一个名称（所有字母必须小写）来标识， 当针对这个索引的文档执行索引、搜索、更新和删除操作的时候，这个名称被用来指向索引。

> 表 1-1 Elasticsearch与关系型数据库的对应关系

Elasticsearch	关系型数据库
索引（index）	HTTP方法，包括 `GET` 、 `POST` 、 `PUT` 、 `HEAD` 、 `DELETE` 。
文档类型（type），es6版本之后一个index只能有一个type	数据库（Database）
文档（document）	一行数据（Row）
字段（field）	一列数据（Column）
映射（mapping）	数据库的组织和结构（Schema）

在Elasticsearch中索引一词，有多种含义：

> Index Versus
>
> > 你也许已经注意到 索引 这个词在 Elasticsearch 语境中有多种含义， 这里有必要做一些说明：
> > **索引（名词）：**
> > 如前所述，一个 索引 类似于传统关系数据库中的一个 数据库 ，是一个存储关系型文档的地方。 索引 (index) 的复数词为 `indices` 或 `indexes` 。 **索引（动词）：**
> > 索引一个文档 就是存储一个文档到一个 索引 （名词）中以便被检索和查询。 这非常类似于 SQL 语句中的 `INSERT` 关键词，除了文档已存在时，新文档会替换旧文档情况之外。
> > **倒排索引**：
> > 关系型数据库通过增加一个 索引 比如一个 `B树` （B-tree）索引 到指定的列上， 以便提升数据检索速度。 `Elasticsearch` 和 `Lucene` 使用了一个叫做 倒排索引 的结构来达到相同的目的。 默认的，一个文档中的每一个属性都是 被索引 的（有一个倒排索引）和可搜索的。 一个没有倒排索引的属性是不能被搜索到的。

索引的状态分为三种： `green` ， `yellow` ， `red`

green

索引的分片（shard）都被正常的分配到节点上；

yellow

索引存在副本分片（shard）没有被分配；

red

索引存在主分片（shard）没有被分配；

查看索引表的状态

```
GET _cat/indices?health=green
GET _cat/indices/elasticsearch-analyzer-index?health=green
```

查看索引的定义信息

```
GET elasticsearch-analyzer-index
GET elasticsearch-analyzer-index?flat_settings&include_defa
```

响应返回

```
"elasticsearch-analyzer-index": {
    "aliases": {
      "elasticsearch-analyzer-index-alias": {}
    },
    "mappings": {
      "_doc": {
        "properties": {
          "author": {
            "type": "keyword",
            "ignore_above": 256
          },
          "content": {
            "type": "text",
            "fields": {
              "space": {
                "type": "text",
                "analyzer": "whitespace"
              }
            }
          },
          "descripe": {
            "type": "text",
            "analyzer": "standard"
          },
          "title": {
            "type": "text",
            "analyzer": "my_custom_analyzer"
          }
        }
      }
    },
    "settings": {
      "index": {
        "number_of_shards": "1",
        "provided_name": "elasticsearch-analyzer-index",
        "creation_date": "1622366213571",
        "analysis": {
          "filter": {
            "english_stop": {
              "type": "stop",
              "stopwords": "_english_"
            }
          },
          "char_filter": {
            "emoticons": {
              "type": "mapping",
              "mappings": [
                ":) => _happy_",
```

```
                ":( => _sad_",
                "🐂 => 牛"
              ]
            }
          },
          "analyzer": {
            "my_custom_analyzer": {
              "filter": [
                "lowercase",
                "english_stop",
                "asciifolding"
              ],
              "char_filter": [
                "emoticons",
                "html_strip"
              ],
              "type": "custom",
              "tokenizer": "punctuation"
            }
          },
          "tokenizer": {
            "punctuation": {
              "pattern": "[ .,!?。]",
              "type": "pattern"
            }
          }
        },
        "number_of_replicas": "0",
        "uuid": "vQ2TkUx3SNCGjBuJZ6AQKA",
        "version": {
          "created": "6040099"
        }
      }
    }
  }
```

1.4. 类型（type）

一个类型通常是一个索引的一个逻辑分类或分区，允许在一个索引下存储不同类型的文档（相当于关系型数据库中的一张表），例如用户类型、订单类型等。目前已经不支持在一个索引下创建多个类型，并且类型概念已经在后续版本中删除， 详情请参见Elasticsearch官方文档。

1.5. 文档（document）

`Elasticsearch` 是 面向文档 的，意味着它存储整个对象或 文档。 `Elasticsearch` 不仅存储文档， 而且 索引 每个文档的内容， 使之可以被检索。在 `Elasticsearch` 中，我们对文档进行索引、检索、排序和过滤—而不是对行列数据。 这是一种完全不同的思考数据的方式，也是 `Elasticsearch` 能支持复杂全文检索的原因。

如果我们知道一个文档的id，那么我们可以通过以下语句直接改文档

```
GET elasticsearch_study_index/_doc/1
```

1.6. 字段（field）

组成文档的最小单位。相当于关系型数据库中的一列数据。

1.7. 映射（mapping）

用来定义一个文档以及其所包含的字段如何被存储和索引， 例如在 `mapping` 中定义字段的名称和类型，以及所使用的分词器。相当于关系型数据库中的 `Schema` 。

1.8. 分片（shards）

代表索引分片， `Elasticsearch` 可以把一个完整的索引分成多个分片,即一个索引表其实是一组 `shard` 的逻辑视图， 这样的好处是可以把一个大的索引拆分成多个，分布到不同的节点上，构成分布式搜索。 分片的数量只能在索引创建前指定，并且索引创建后不能更改。

1.9. 段（segements）

每个shard分片是一个lucene实例，每个分片由多个segment组成。每个segment占用内存，文件句柄等。
服务器总内存除了分配jvm配置的，其余都给了lucene，占用page cache内存，page cache保存对文件数据segment的缓存。free -g可查看内存使用，es节点只有es服务，基本cache就是缓存的segment。

多个段可以合并，减少磁盘的io。
lucene的数据写入会先写如到缓存（buffer）中，当达到一定数量以后，会flush成文一个segment，写入到磁盘当中。每个segement有自己独立的索引，可以单独查询。
segment不会被修改，数据的的写入都是进行批量的追加，避免了随机写的存在，提高了吞吐量。segement可以被删除，但也不是修改segement文件， 而是由另外的文件记录需要被删除的documentId。

index的查询是对多个segement文件的查询，其中也包含了处理被删除文件的处理，并对查询结果进行合并。为了进行查询优化，lucene有策略对多个segment进行优化。

```
#查看集群所有索引segements的情况
GET _cat/segments?v
#查看每个节点segements占用的堆内存
GET _cat/nodes?v&h=ip,ram.percent,sm
#查看指定索引的segements的情况
GET _cat/segments/elasticsearch_study_index?v
```

1. Elasticsearch、Lucene、Segments

分片内部原理

1.1. refresh

默认值为1秒，即数据写入ES后1s可见。

```
"index.refresh_interval": "1s"
```

如果对数据的实时要求不高，可以增加refresh的频率，使indexBuffer中的数据积累的较多的时候，形成大的段（segment），这样可以减少大量小段合并，提升查询性能。

1.2. segments

段合并是一项比较重的操作，会消耗大量磁盘IO；可以限制每个索引表合并线程数目，避免在写入高峰的时候有大量合并操作，影响写入性能

```
"index.merge.scheduler.max_merge_count": "200",
"index.merge.scheduler.max_thread_count": "1",
```

配置的源码位置
`org.elasticsearch.indexorg.elasticsearch.indexMergeSchedulerConfig`

```
 public static final Setting<Integer> MAX_THREAD_COUNT_SETT
        new Setting<>("index.merge.scheduler.max_thread_cou
            (s) -> Integer.toString(Math.max(1, Math.min(4,
            (s) -> Setting.parseInt(s, 1, "index.merge.sche
            Property.IndexScope);
    public static final Setting<Integer> MAX_MERGE_COUNT_SE
        new Setting<>("index.merge.scheduler.max_merge_coun
            (s) -> Integer.toString(MAX_THREAD_COUNT_SETTIN
            (s) -> Setting.parseInt(s, 1, "index.merge.sche
    public static final Setting<Boolean> AUTO_THROTTLE_SETT
        Setting.boolSetting("index.merge.scheduler.auto_thr
```

1.2.1. Stored Fields

Stored Fields是一个简单的键值对key-value 默认情况下所有字段存储在_source字段下；

```
PUT elasticsearch-store-fields
{
   "mappings": {
      "_doc": {
         "properties": {
            "counter": {
               "type": "integer",
               "store": false
            },
            "tags": {
               "type": "keyword",
               "store": true
            }
         }
      }
   }
}
```

```
PUT elasticsearch-store-fields/_doc/1
{
    "counter" : 1,
    "tags" : ["red"]
}
```

```
GET elasticsearch-store-fields/_doc/1?stored_fields=tags,co
GET elasticsearch-store-fields/_search?stored_fields=tags,c
```

1.2.2. Document Values

以文件字段为单位进行列式存储；适用场景：排序、聚合、权重记分；

1.3. translog

官方文档 translog中的数据只有在fsync和提交时才会被持久化到磁盘。在硬件失败的情况下，在translog提交之前的数据都会丢失。 默认情况下，如果index.translog.durability被设置为async的话，Elasticsearch每5秒钟同步并提交一次translog。 或者如果被设置为request（默认）的话，每次index，delete，update，bulk请求时就同步一次translog。 更准确地说，如果设置为request, Elasticsearch只会在成功地在主分片和每个已分配的副本分片上fsync并提交translog之后， 才会向客户端报告index、delete、update、bulk成功。

可以动态控制每个索引的translog行为：

- index.translog.sync_interval ： translog多久被同步到磁盘并提交一次。默认5秒。这个值不能小于100ms
- index.translog.durability ： 是否在每次index，delete，update，bulk请求之后立即同步并提交translog。接受下列参数：
  - request ： （默认）fsync and commit after every request。这就意味着，如果发生崩溃，那么所有只要是已经确认的写操作都已经被提交到磁盘。
  - async ： 在后台每sync_interval时间进行一次fsync和commit。意味着如果发生崩溃，那么所有在上一次自动提交以后的已确认的写操作将会丢失。
- index.translog.flush_threshold_size ： 当操作达到多大时执行刷新，默认512mb。也就是说，操作在translog中不断累积，当达到这个阈值时，将会触发刷新(flush)操作。
- index.translog.retention.size ： translog文件达到多大时执行执行刷新。默认512mb。
- index.translog.retention.age ： translog最长多久提交一次。默认12h。

translog-N.tlog - 真正的日志文件，N表示generation（代）的意思，通过它跟索引文件关联 tranlog.ckp - 日志的元数据文件，长度总是20个字节，记录3个信息：偏移量 & 事务操作数量 & 当前代

例子

```
/elasticsearch/data/nodes/0/indices/gynPAM1CSeqeMHxAseKQSw/
```

1. segment 合并相关参数释意

1.1. 合并策略

org.elasticsearch.index.MergePolicyConfig.java

```
public static final double          DEFAULT_EXPUNGE_DELETE
   public static final ByteSizeValue   DEFAULT_FLOOR_SEGME
   public static final int             DEFAULT_MAX_MERGE_A
   public static final int             DEFAULT_MAX_MERGE_A
   public static final ByteSizeValue   DEFAULT_MAX_MERGED_
   public static final double          DEFAULT_SEGMENTS_PE
   public static final double          DEFAULT_RECLAIM_DEL
   public static final Setting<Double> INDEX_COMPOUND_FORM
       new Setting<>("index.compound_format", Double.toStr
           Property.Dynamic, Property.IndexScope);

   public static final Setting<Double> INDEX_MERGE_POLICY_
       Setting.doubleSetting("index.merge.policy.expunge_d
           Property.Dynamic, Property.IndexScope);
   public static final Setting<ByteSizeValue> INDEX_MERGE_
       Setting.byteSizeSetting("index.merge.policy.floor_s
           Property.Dynamic, Property.IndexScope);
   public static final Setting<Integer> INDEX_MERGE_POLICY
       Setting.intSetting("index.merge.policy.max_merge_at
           Property.Dynamic, Property.IndexScope);
   public static final Setting<Integer> INDEX_MERGE_POLICY
       Setting.intSetting("index.merge.policy.max_merge_at
           Property.Dynamic, Property.IndexScope);
   public static final Setting<ByteSizeValue> INDEX_MERGE_

       Setting.byteSizeSetting("index.merge.policy.max_mer
           Property.Dynamic, Property.IndexScope);
   public static final Setting<Double> INDEX_MERGE_POLICY_
       Setting.doubleSetting("index.merge.policy.segments_
           Property.Dynamic, Property.IndexScope);
   public static final Setting<Double> INDEX_MERGE_POLICY_
       Setting.doubleSetting("index.merge.policy.reclaim_d
           Property.Dynamic, Property.IndexScope);
```

```
 如果合并的段是小于总索引的此百分比，然后将其写在复合格式，否则它被写入
默认值0.1。该属性可用于系统出现文件句柄数量太多错误时使用，但是会降低搜
index.compound_format:
index.merge.policy.expunge_deletes_allowed: 默认值为10，该值用
index.merge.policy.floor_segment:  默认2MB，小于该值的segment
index.merge.policy.max_merge_at_once:  默认10,一次最多合并多少
index.merge.policy.max_merge_at_once_explicit: 默认30, 显式调
index.merge.policy.max_merged_segment:   默认5GB，超过该值的s
index.merge.policy.segments_per_tier:  每一轮merge的segment的
默认是10.注意，这个值的设置要大于等于max_merge_at_once。否则将会导
index.merge.policy.reclaim_deletes_weight:  ;考虑merge的segm
默认值为2.0，该属性指定了删除文档在合并操作中的重要程度。如果属性值设置
```

1.2. merge调度

org.elasticsearch.inde.MergeSchedulerConfig

```
public static final Setting<Integer> MAX_THREAD_COUNT_SETTI
        new Setting<>("index.merge.scheduler.max_thread_cou
            (s) -> Integer.toString(Math.max(1, Math.min(4,
            (s) -> Setting.parseInt(s, 1, "index.merge.sche
            Property.IndexScope);
    public static final Setting<Integer> MAX_MERGE_COUNT_SE
        new Setting<>("index.merge.scheduler.max_merge_coun
            (s) -> Integer.toString(MAX_THREAD_COUNT_SETTIN
            (s) -> Setting.parseInt(s, 1, "index.merge.sche
    public static final Setting<Boolean> AUTO_THROTTLE_SETT
        Setting.boolSetting("index.merge.scheduler.auto_thr
```

```
控制并发的merge线程数，如果存储是并发性能较好的SSD，可以用系统默认的m
当节点配置的cpu核数较高时，merge占用的资源可能会偏高，影响集群的性能，
index.merge.scheduler.max_thread_count:

index.merge.scheduler.max_merge_count:
MAX_THREAD_COUNT_SETTING.get(s) + 5

index.merge.scheduler.auto_throttle: true
```

1. 文档操作

1. 文档操作

1.1. 创建索引

```
PUT operator_document_index-2021-05-31
{
    "aliases": {
      "operator_document_index-2021.05.31": {}
    },
    "mappings": {
      "log": {
        "dynamic": "true",
        "_all": {
          "enabled": false
        },
        "properties": {
           "time": {
            "type": "date"
          },
          "price": {
            "type": "long"
          },
          "productID": {
            "type": "text",
            "fields": {
              "keyword": {
                "type": "keyword",
                "ignore_above": 256
              }
            }
          }
        }
      }
    },
    "settings": {
      "index": {
        "mapping": {
          "total_fields": {
            "limit": "2000000"
          }
        },
        "refresh_interval": "20s",
        "indexing": {
          "slowlog": {
            "level": "info",
            "threshold": {
              "index": {
                "warn": "10s",
                "trace": "500ms",
                "debug": "2s",
                "info": "5s"
              }
```

```
            },
            "source": "1000"
          }
        },
        "translog": {
          "flush_threshold_size": "1024mb",
          "sync_interval": "120s",
          "durability": "async"
        },
        "max_result_window": "5000",
        "number_of_replicas": "0",
        "routing": {
          "allocation": {
            "exclude": {
              "zone": "data"
            },
            "total_shards_per_node": "1"
          }
        },
        "search": {
          "slowlog": {
            "level": "TRACE",
            "threshold": {
              "fetch": {
                "warn": "200ms",
                "trace": "200ms",
                "debug": "100ms",
                "info": "100ms"
              },
              "query": {
                "warn": "100ms",
                "trace": "200ms",
                "debug": "200ms",
                "info": "1s"
              }
            }
          }
        },
        "number_of_shards": "1",
        "merge": {
          "scheduler": {
            "max_thread_count": "1",
            "max_merge_count": "200"
          }
        }
      }
    }
  }
```

```
GET _cat/shards/operator_document_index-2021-05-31?v

PUT operator_document_index-2021-05-31/_settings
{
  "index":{
    "routing": {
          "allocation": {
            "exclude": {
              "zone": "client"
            },
            "total_shards_per_node": "1"
          }
        }
  }
}
```

1.2. 索引文档

```
POST  operator_document_index-2021-05-31/log/1
{ "price" : 10, "productID" : "XHDK-A-1293-#fJ3","time" : '

POST  operator_document_index-2021-05-31/log/_bulk
{ "index": { "_id": 1 }}
{ "price" : 10, "productID" : "XHDK-A-1293-#fJ3","time" : '
{ "index": { "_id": 2 }}
{ "price" : 20, "productID" : "KDKE-B-9947-#kL5","time" : '
{ "index": { "_id": 3 }}
{ "price" : 30, "productID" : "JODL-X-1937-#pV7","time" : '
{ "index": { "_id": 4 }}
{ "price" : 30, "productID" : "QQPX-R-3956-#aD8","time" : '
```

```
GET _cat/indices/operator_document_index-2021-05-31?v
```

返回,docs.deleted

```
health status index                               uuid
green  open   operator_document_index-2021-05-31 7dazxoVrT
```

1.3. 获取文档

```
GET operator_document_index-2021-05-31/log/1

http://39.104.94.193:9200/operator_document_index-2021-05-3
```

该文件修订时间： 2021-06-22

1. 简单搜索

1. 分页查询

插入数据

```
POST  page_search-2021-06-23/log/_bulk
{ "index": { "_id": 1 }}
{ "price" : 10, "productID" : "XHDK-A-1293-#fJ3","page" : 1
{ "index": { "_id": 2 }}
{ "price" : 20, "productID" : "KDKE-B-9947-#kL5","page" : 2
{ "index": { "_id": 3 }}
{ "price" : 30, "productID" : "JODL-X-1937-#pV7","page" : 3
```

1.1. scroll

scroll

```
POST page_search-2021-06-23/_search?scroll=1m
{
    "size": 1,
    "query": {
        "match" : {
            "title" : "mac"
        }
    },
    "sort": [
    "_id"
  ]
}
```

scroll阶段

```
POST /_search/scroll
{
    "scroll" : "1m",
    "scroll_id" : "首次search返回的scroll_id"
}
```

```
DELETE /_search/scroll/_all

DELETE /_search/scroll
{
    "scroll_id" : "DXF1ZXJ5QW5kRmV0Y2gBAAAAAAAAAD4WYm9laVYt
}

DELETE /_search/scroll
{
    "scroll_id" : [
      "DXF1ZXJ5QW5kRmV0Y2gBAAAAAAAAAD4WYm9laVYtZndUQlNsdDcv
      "DnF1ZXJ5VGhlbkZldGNoBQAAAAAAAAABFmtSWWRRW"
    ]
}

DELETE /_search/scroll/DXlZkMWFBAAAAAAAAAAIWa1JZZFFZQmtTaj
```

1.2. search_after

Search After

```
POST page_search-2021-06-23/_search?scroll=1m
{
    "size": 1,
    "query": {
        "match" : {
            "title" : "mac"
        }
    },
    "sort": [
    "_id"
  ]
}
```

```
POST page_search-2021-06-23/_search
{
    "size": 1,
    "query": {
        "match" : {
            "title" : "mac"
        }
    },
    "sort": [
    "_id"
  ]
}
```

```
POST page_search-2021-06-23/_search
{
    "size": 1,
    "query": {
        "match" : {
            "title" : "mac"
        }
    },
   "search_after": [1],
    "sort": [
        {"_id": "asc"}
    ]
}
```

1. 简单搜索

创建索引

```
PUT cars
{
    "aliases": {},
    "mappings": {
      "_doc": {
        "properties": {
          "color": {
            "type": "text",
            "fields": {
              "keyword": {
                "type": "keyword",
                "ignore_above": 256
              }
            }
          },
          "make": {
            "type": "text",
            "fields": {
              "keyword": {
                "type": "keyword",
                "ignore_above": 256
              }
            }
          },
          "price": {
            "type": "long"
          },
          "sold": {
            "type": "date"
          }
        }
      }
    }
  }
```

插入数据 ~~~~

```
POST /cars/_doc/_bulk
{ "index": {}}
{ "price" : 10000, "color" : "red", "make" : "honda", "sold
{ "index": {}}
{ "price" : 20000, "color" : "red", "make" : "honda", "sold
{ "index": {}}
{ "price" : 30000, "color" : "green", "make" : "ford", "sol
{ "index": {}}
{ "price" : 15000, "color" : "blue", "make" : "toyota", "so
{ "index": {}}
{ "price" : 12000, "color" : "green", "make" : "toyota", "s
{ "index": {}}
{ "price" : 20000, "color" : "red", "make" : "honda", "sold
{ "index": {}}
{ "price" : 80000, "color" : "red", "make" : "bmw", "sold"
{ "index": {}}
{ "price" : 25000, "color" : "blue", "make" : "ford", "sold
```

1.1. 轻量搜索

URI Search 轻量搜索

```
GET cars/_search?q=color:red

GET cars/_search?q=color:red&analyzer=whitespace&df=make
```

uri支持的参数

Name	描述
q	对应Query DSL 的 `query_string` ,更多信息请参考Query String Query
df	查询中未定义字段前缀时使用的默认字段
analyzer	分析查询字符串时要使用的分析器名称。
analyze_wildcard	是否应该分析通配符和前缀查询。默认为false。
batched_reduce_size	协调节点上应立即减少的碎片结果数.如果请求中可能存在大量shard，则应将此值用作保护机制，以减少每个搜索请求的内存开销。
default_operator	要使用的默认运算符可以是AND或OR。默认为OR。
lenient	如果设置为true，将导致忽略基于格式的失败（如向数字字段提供文本）。默认为false。
explain	对于每次命中，包含如何计算命中分数的说明。
_source	数据库的组织和结构（Schema）
stored_fields	数据库（Database）
sort	一行数据（Row）
track_scores	一列数据（Column）
track_total_hits	数据库的组织和结构（Schema）
lenient	一行数据（Row）
explain	一列数据（Column）
timeout	数据库的组织和结构（Schema）
terminate_after	数据库（Database）
from	一行数据（Row）
size	一列数据（Column）
search_type	数据库的组织和结构（Schema）

Name	描述
allow_partial_search_results	数据库的组织和结构（Schema）

1. 去重统计

cardinality
cardinality-en

```
GET /cars/_doc/_search
{
    "size" : 0,
    "aggs" : {
        "distinct_colors" : {
            "cardinality" : {
              "field" : "color.keyword",
              "precision_threshold" : 100
            }
        }
    }
}
#sql语句写法
POST /_xpack/sql?format=txt
{
    "query": "SELECT COUNT(DISTINCT color.keyword) AS car_
}
```

1. 精准值查询

精确值查找 Term相关查询

```
POST /my_store/products/_bulk
{ "index": { "_id": 1 }}
{ "price" : 10, "productID" : "XHDK-A-1293-#fJ3" }
{ "index": { "_id": 2 }}
{ "price" : 20, "productID" : "KDKE-B-9947-#kL5" }
{ "index": { "_id": 3 }}
{ "price" : 30, "productID" : "JODL-X-1937-#pV7" }
{ "index": { "_id": 4 }}
{ "price" : 30, "productID" : "QQPX-R-3956-#aD8" }
```

数字精确查询

```
GET /my_store/products/_search
{
    "query" : {
        "constant_score" : {
            "filter" : {
                "term" : {
                    "price" : 20
                }
            }
        }
    }
}

POST /_xpack/sql?format=txt
{
    "query": "SELECT * FROM  my_store WHERE  price = 20"
}
```

文本精确查询

```
GET /my_store/products/_search
{
    "query" : {
        "constant_score" : {
            "filter" : {
                "term" : {
                    "productID.keyword": "XHDK-A-1293-#fJ3"
                }
            }
        }
    }
}

# 查看报错,xpack插件对sql支持的不是特别好
POST /_xpack/sql?format=txt
{
    "query": "SELECT * FROM  my_store WHERE  productID.keyw
}
```

1. 组合过滤查询

组合过滤器

1.1. 布尔过滤

一个 bool 过滤器由三部分组成：

```
{
   "bool" : {
      "must" :     [],
      "should" :   [],
      "must_not" : [],
   }
}
```

must
所有的语句都 必须（must） 匹配，与 AND 等价。
must_not
所有的语句都 不能（must not） 匹配，与 NOT 等价。
should
至少有一个语句要匹配，与 OR 等价。

```
GET /my_store/products/_search
{
   "query" : {
             "bool" : {
               "should" : [
                  { "term" : {"price" : 20}},
                  { "term" : {"productID.keyword" : "XHDK-A-
               ],
               "must_not" : {
                  "term" : {"price" : 30}
               }
            }
         }
}
```

嵌套查询

```
POST /_xpack/sql?format=txt
{
    "query": "SELECT * FROM my_store WHERE  productID.keywo
}
```

```
GET /my_store/_search
{
   "query" : {
            "bool" : {
              "should" : [
                { "term" : {"productID.keyword": "KDKE-B-99
                { "bool" : {
                  "must" : [
                    { "term" : {"productID.keyword": "JODL-
                    { "term" : {"price" : 30}}
                  ]
                }}
              ]
           }
     }
}
```