# 目录

# §1 ElasticSearch

Elasticsearch 是个开源分布式搜索引擎，它的特点有：分布式，零配置，自动发现，索引自动分片，索引副本机制，Restful 风格接口，多数据源，自动搜索负载等。

## §1.1 What's a mapping?

A mapping not only tells ES what is in a field…it tells ES what terms are indexed and searchable.

A mapping is composed of one or more 'analyzers', which are in turn built with one or more 'filters'. When ES indexes your document, it passes the field into each specified analyzer, which pass the field to individual filters.

Filters are easy to understand: a filter is a function that transforms data. Given a string, it returns another string (after some modifications). A function that converts strings to lowercase is a good example of a filter.

An analyzer is simply a group of filters that are executed in-order. So an analyzer may first apply a lowercase filter, then apply a stop-word removal filter. Once the analyzer is run, the remaining terms are indexed and stored in ES.

Which means a mapping is simply a series of instructions on how to transform your input data into searchable, indexed terms.

使用 **curl -X PUT http://localhost:9200/test/item/1 -d '{"name":"zach", "description": "A Pretty cool guy."}'** 时，ES 会生成如下的 mapping:

```
mappings: {
   item: {
      properties: {
         description: { type: string }
         name: { type: string }
      }
   }
}
```

Above, ES guessed "string" as the mapping for ``description''. When ES implicitly creates a "string" mapping, it applies the default global analyzer. Unless you've changed it, the default

analyzer is the Standard Analyzer. This analyzer will apply the standard token filter, lowercase filter and stop token filter to your field. We lost a word ( “A” ) capitalization and punctuation. Importantly, even though ES continues to store the original document in it’ s original form, the only parts that are searchable are the parts that have been run through an analyzer.So, don’ t think of mappings as data-types , think of them as instructions on how you will eventually search your data. If you care about stop-words like “a” , you need to change the analyzer so that they aren’ t removed.

## §1.2 Constructing more complicated mapping

Setting up proper analyzers in ES is all about thinking about the search query. You have to provide instructions to ES about the appropriate transformations so you can search intelligently.

The first thing that happens to an input query is tokenization , breaking an input query into smaller chunks called tokens. There are several tokenizers available, which you should explore on your own when you get a chance. The Standard tokenizer is being used in this example, which is a pretty good tokenizer for most English-language search problems. You can query ES to see how it tokenizes a sample sentence:

**curl -X GET "http://localhost:9200/test/_analyze?tokenizer=standard &pretty=true" -d 'The quick brown fox is jumping over the lazy dog.'**

Ok, so our input query has been turned into tokens. Referring back to the mapping, the next step is to apply filters to these tokens. In order, these filters are applied to each token: Standard Token Filter, Lowercase Filter, ASCII Folding Filter.

**curl -X GET "http://localhost:9200/test/_analyze?filter=standard &pretty=true" -d 'The quick brown fox is jumping over the lazy dog.'**

```
"partial":{
     "search_analyzer":"full_name",
     "index_analyzer":"partial_name",
     "type":"string"
}
```

As you can see, we specify both a search and index analyzer. Huh? These two separate analyzers instruct ES what to do when it is indexing a field, or searching a field. But why are these needed?

The index analyzer is easy to understand. We want to break up our input fields into various tokens so we can later search it. So we instruct ES to use the new partial_name analyzer that we built, so that it can create nGrams for us.

The search analyzer is a little trickier to understand, but crucial to getting good relevance. Imagine querying for “Race” . We want that query to match “race” , “races” and “racecar” . When searching, we want to make sure ES eventually searches with the token “race” . The full_name analyzer will give us the needed token to search with.

If, however, we used the partial_name nGram analyzer, we would generate a list of nGrams as our search query. The search query “Race” would turn into ["ra", "rac", "race"].Those tokens are then used to search the index. As you might guess, “ra” and “rac” will match a lot of things you don’ t want, such as “racket” or “ratify” or “rapport” .

So specifying different index and search analyzers is critical when working with things like ngrams. Make sure you always double check what you expect to query ES with…and what is actually being passed to ES.

## §1.3 depth into ElasticSearch

当使用如下命令搜索时:

```
curl -XPOST "http://namenode:9200/_search" -d'
{
    "query": {
      "match_all" : {}
    },
    "filter" : {
      "term" : { "director" : "Francis Ford Coppola"}
    }
}'
```

没有任何结果，而使用

```
curl -XPOST "http://namenode:9200/_search" -d'
{
    "query": {
      "match_all" : {}
    },
    "filter" : {
      "term" : { "year" : "1962"}
    }
}'
```

却能输出结果，What's going on here? We've obviously indexed two movies with "Francis Ford Coppola" as director and that's what we see in search results as well. Well, while ES has a JSON object with that data that it returns to us in search results in the form of the _source property .

When we index a document with ElasticSearch it (simplified) does two things: it stores the original data untouched for later retrieval in the form of _source and it indexes each JSON property into one or more fields in a Lucene index. During the indexing it processes each field according to how the field is mapped. If it isn't mapped , default mappings depending on the fields type (string, number etc) is used.

As we haven't supplied any mappings for our index, Elastic-Search uses the default mappings for the director field. This means that in the index the director fields value isn't **"Francis Ford Coppola"**. Instead it's something like **["francis", "ford", "coppola"]**. We can verify that by modifying our filter to instead match "francis" (or "ford" or "coppola").

So, what to do if we want to filter by the exact name of the director? We modify how it's mapped. There are a number of ways to add mappings to ElasticSearch, through a configuration file, as part of a HTTP request that creates and index and by calling the _mapping endpoint. Using the last approach, we could fix the above issue by adding a mapping for the "director" field , to instruct ElasticSearch not to analyze (tokenize etc.) the field at all.

```
curl -X PUT namenode:9200/movies/movie/_mapping -d'
{
   "movie": {
      "properties": {
         "director": {
            "type": "string",
```

```
            "index": "not_analyzed"
         }
      }
   }
}'
```

In many cases it's not possible to modify existing mappings. Often the easiest work around for that is to create a new index with the desired mappings and re-index all of the data into the new index. The second problem is that, even if we could add it, we would have limited our ability to search in the director field. That is, while a search for the exact value in the field would match we wouldn't be able to search for single words in the field.

Luckily, there's a simple solution to our problem. We add a mapping that upgrades the field to a multi field. What that means is that we'll map the field multiple times for indexing. Given that one of the ways we map it match the existing mapping both by name and settings that will work fine and we won't have to create a new index.

```
curl -XPUT "namenode:9200/movies/movie/_mapping" -d'
{
   "movie": {
      "properties": {
         "director": {
            "type": "multi_field",
            "fields": {
               "director": {"type": "string's"},
               "original": {"type" : "string", "index" : "not_analyzed"}
            }
         }
      }
   }
}'
```

## §1.4 ElasticSearch 集群设置

This will print the number of open files the process can open on startup. Alternatively, you can retrieve the max_file_descriptors for each node using the Nodes Info API, with:

**curl -XGET 'namenode:9200/_nodes/process?pretty=true'**，下面是删除 ES 上名为 jdbc 的 index 的 Restfule 命令:

**curl -XDELETE 'http://namenode:9200/jdbc/'**

配置文件在 **{$ES_HOME}/config** 文件夹下，**elasticsearch.yml** 和 **logging.yml**，修改 **elasticsearch.yml** 文件中的 cluster.name，当集群名称

相同时，每个 ES 节点将会搜索它的伙伴节点，因此必须保证集群内每个节点的 cluster.name 相同，下面是关闭 ES 集群的 Restful 命令：

```
# 关闭集群内的某个ES节点'_local'
$ curl -XPOST 'http://namenode:9200/_cluster/nodes/_local/_shutdown'
# 关闭集群内的全部ES节点
$ curl -XPOST 'http://namenode:9200/_shutdown'
```

注意，如果一台机器上不止一个 ES 在运行，那么通过**./bin/elastic-search** 开启的 ES 的 http_address 将会使用 9200 以上的接口（形如 9201,9202,···），而相应的 transport_address 也递增（形如: 9301,9302,···），因此，为使用 9200 端口，可使用上述命令关闭其它 ES 进程，可通过 conf 目录下的 log 文件来查看某些端口是否被占用。**elasticsearch.yml** 文件存在如下配置信息：

(1) node.master: true,node.data: true，允许节点存储数据，同时作为主节点；

(2) node.master: true,node.data: false，节点不存储数据，但作为集群的协调者；

(3) node.master: false,node.data: true，允许节点存储数据，但不作为主节点；

(4) node.master: false,node.data: false，节点不存储数据，也不作为协调者，但作为搜索任务的一个承担者；

(5) cluster.name: HadoopSearch，node.name: "ES-Slave-02",HadoopSearch 必须相同，但 node.name 每个节点可以自由设置；

如想将 ES 作为一个服务，需要从 github 上下载 elasticsearch-servicewrapper，然后调用 chkconfig，将其添加到/etc/rc[0~6].d/中。

```
curl -L https://github.com/elasticsearch/elasticsearch-servicewrapper/
archive/master.zip > master.zip
unzip master.zip
cd elasticsearch-servicewrapper-master/
mv service /opt/elasticsearch/bin
/opt/elasticsearch/bin/service/elasticsearch install
## 如果想卸载该服务调用：
/opt/elasticsearch/bin/service/elasticsearch remove
## 如果想让ES开机启动
chkconfig elasticsearch on
## 如果想现在开启ES服务
service elasticsearch start
```

配置完后，可通过 *curl -X GET 'http://192.168.50.75:9200/_cluster/nodes?pretty'* 命令，查询集群下的节点信息。

为连接 hive 与 ES，运行 hive 后，在 hive 命令行内执行 **add jar /opt/elasticsearch-hadoop-1.3.0/dist/elasticsearch-hadoop-1.3.0.jar;** 或者 hdfs 上的 jar 包:**add jar hdfs://namenode:9000/elasticsearch-hadoop-1.3.0.jar** 可加载 elasticsearch-hadoop 插件，使用该插件的具体操作如下:

```
DROP TABLE IF EXISTS artist_1;
CREATE EXTERNAL TABLE artists_1 (
  cardid STRING, date STRING, time STRING)
STORED BY 'org.elasticsearch.hadoop.hive.ESStorageHandler'
TBLPROPERTIES('es.resource' = 'liubo/artists/',
              'es.host' = '192.168.50.75',
              'es.mapping.names' = 'text:time'
);
-- 集群下应使用'192.168.50.75'，而非'localhost'(es-hadoop的默认值)
-- insert data to Elasticsearch from another hive table
INSERT OVERWRITE TABLE artists_1
SELECT * FROM cable.temptable;
```

下面的代码是将 Mysql 中的表导入到 ES 中，建立名为 jdbc 的 index，表名称为 jiangsu。

```
curl -XPUT 'localhost:9200/_river/jiangsu/_meta' -d '{
    "type" : "jdbc",
    "jdbc" : {
        "driver" : "com.mysql.jdbc.Driver",
        "url" : "jdbc:mysql://192.168.50.75:3306/jsyx",
        "user" : "root",
        "password" : "123456",
        "sql" : "select * from jiangsu"
    },
    "index" : {
        "index" : "jdbc",
        "type" : "jiangsu"
    }
}'
```

## §1.5 ES 性能优化

一个 Elasticsearch 节点会有多个线程池，但重要的是下面四个:

- 索引（index）: 主要是索引数据和删除数据操作（默认是 cached 类型）;
- 搜索（search）: 主要是获取，统计和搜索操作（默认是 cached 类型）;
- 批量操作（bulk）: 对索引的批量操作，现在尚且不清楚它是不是原子性的，如果是原子的，则放在 MapReduce 里是没有问题的;

- 更新（refresh）：主要是更新操作，如当一个文档被索引后，何时能够通过搜索看到该文档；

在生成索引的过程中，需要修改如下配置参数：

- index.store.type： mmapfs。因为内存映射文件机制能更好地利用 OS 缓存；
- indices.memory.index_buffer_size： 30% 默认值为 10%，表示 10% 的内存作为 indexing buffer；
- index.translog.flush_threshold_ops： 50000，当写日志数达到 50000 时，做一次同步；
- index.refresh_interval:30s，默认值为 1s，新建的索引记录将在 1 秒钟后查询到；

```
curl -XPUT 'http://namenode:9200/hivetest/?pretty' -d '{
    "settings" : {
       "index" : {
       "refresh_interval" : "30s",
       "index.store.type": "mmapfs",
       "indices.memory.index_buffer_size": "30%",
       "index.translog.flush_threshold_ops": "50000"
        }
    }
}'
```