【ELK 技术栈第一天 ElasticSearch】

主要内容

1. 什么是 ElasticSearch
2. Linux 安装 ElasticSearch
3. 常用操作命令
4. 分词器和标准化处理
5. ElasticSearch 中的 Mapping 问题
6. Search 搜索详解

学习目标

知识点	要求
什么是 ElasticSearch	掌握
Linux 安装 ElasticSearch	掌握
常用操作命令	掌握
分词器和标准化处理	掌握
ElasticSearch 中的 Mapping 问题	掌握
Search 搜索详解	掌握

ELK 技术栈第一天 ElasticSearch

一、 什么是 Elastic Search

ElasticSearch 是一个基于 Lucene 的搜索服务器。它提供了一个分布式的全文搜索引擎，其对外服务是基于 RESTful web 接口发布的。Elasticsearch 是用 Java 开发的应用，并作为 Apache 许可条款下的开放源码发布，是当前流行的企业级搜索引擎。设计用于云计算中，能够达到近实时搜索，稳定，可靠，快速，安装使用方便。

1 相关概念

1.1 cluster

集群。ElasticSearch 集群由一或多个节点组成，其中有一个主节点，这个主节点是可以通过选举产生的，主从节点是对于集群内部来说的。ElasticSearch 的一个概念就是去中心化，字面上理解就是无中心节点，这是对于集群外部来说的，因为从外部看 ElasticSearch 集群，在逻辑上是个整体，你与集群中的任何一个节点通信和与整个 ElasticSearch 集群通信是等价的。也就是说，主节点的存在不会产生单点安全隐患、并发访问瓶颈等问题。

1.2 shards

primary shard：代表索引的主分片，ElasticSearch 可以把一个完整的索引分成多个 primary shard，这样的好处是可以把一个大的索引拆分成多个分片，分布存储在不同的 ElasticSearch 节点上，从而形成分布式存储，并为搜索访问提供分布式服务，提高并发处理能。primary shard 的数量只能在索引创建时指定，并且索引创建后不能再更改 primary shard 数量。

1.3 replicas

replica shard：代表索引主分片的副本，ElasticSearch 可以设置多个 replica shard。replica shard 的作用：一是提高系统的容错性，当某个节点某个 primary shard 损坏或丢失时可以从副本中恢复。二是提高 ElasticSearch 的查询效率，ElasticSearch 会自动对搜索请求进行负载均衡，将并发的搜索请求发送给合适的节点，增强并发处理能力。

1.4 Index

索引。相当于关系型数据库中的表。其中存储若干相似结构的 Document 数据。如：

客户索引，订单索引，商品索引等。ElasticSearch 中的索引不像数据库表格一样有强制的数据结构约束，在理论上，可以存储任意结构的数据。但了为更好的为业务提供搜索数据支撑，还是要设计合适的索引体系来存储不同的数据。

1.5 Type

类型。每个索引中都必须有唯一的一个 Type，Type 是 Index 中的一个逻辑分类。ElasticSearch 中的数据 Document 是存储在索引下的 Type 中的。

注意：ElasticSearch5.x 及更低版本中，一个 Index 中可以有多个 Type。ElasticSearch6.x 版本之后，type 概念被弱化，一个 index 中只能有唯一的一个 type。且在 7.x 版本之后，删除 type 定义。

1.6 Document

文档。ElasticSearch 中的最小数据单元。一个 Document 就是一条数据，一般使用 JSON 数据结构表示。每个 Index 下的 Type 中都可以存储多个 Document。一个 Document 中可定义多个 field，field 就是数据字段。如：学生数据（{"name":"张三", "age":20, "gender":"男"}）。

1.7 反向索引|倒排索引

对数据进行分析，抽取出数据中的词条，以词条作为 key，对应数据的存储位置作为 value 实现索引的存储。这种索引称为倒排索引。倒排索引是 Document 写入 ElasticSearch 时分析维护的。

如：

数据

商品主键	商品名	商品描述
1	荣耀 10	更贵的手机
2	荣耀 8	相对便宜的手机
3	IPHONE X	要卖肾买的手机

分析结果、倒排索引	
词条	数据
手机	1，2，3
便宜	2
卖肾	3
相对	2
荣耀	1，2
IPHONE	3

2 ElasticSearch 常见使用场景

维基百科：全文检索，高亮显示，搜索推荐

The Guardian（国外的一个新闻网站），此平台可以对用户的行为（点击、浏览、收藏、评论）、社区网络数据（对新闻的评论等）进行数据分析，为新闻的发布者提供相关的公众反馈。

Stack Overflow（国外的程序异常讨论论坛）

Github（开源代码管理），在千亿级别的代码行中搜索信息

电子商务平台等。

3 为什么不用数据库做搜索？

3.1 查询语法复杂度高。

如：电商系统中搜索商品数据 - select * from products where name like '%关键字%' and price bewteen xxx and yyy and .....。不同的用户提供的查询条件不同，需要提供的动态 SQL 过于复杂。

3.2 关键字索引不全面，搜索结果不符合要求

如：电商系统中查询商品数据，条件为商品名包含'***笔记本电脑***'。那么对此关键字的分析结果为-笔记本、电脑、笔记等。对应的查询语法应该为 - select * from products where name like '%笔记本%' or name like '%电脑%' .....

3.3 效率问题

数据量越大，查询反应效率越低。

二、 Linux 安装 ElasticSearch

使用的 ElasticSearch 的版本是 6.8.4。ElasticSearch6.x 要求 Linux 内核必须是 3.5+版本以上。

在 linux 操作系统中，查看内核版本的命令是： uname -a

课堂使用的 Linux 是 CentOS8。内核使用的是 4.4。

ElasticSearch6.X 版本要求 JDK 版本至少是 1.8.0_131。 提供 1.8.0_161JDK 安装包。

1 为 ElasticSearch 提供完善的系统配置

ElasticSearch 在 Linux 中安装部署的时候，需要系统为其提供若干系统配置。如：应

用可启动的线程数、应用可以在系统中划分的虚拟内存、应用可以最多创建多少文件等。

1.1 修改限制信息

vi /etc/security/limits.conf

是修改系统中允许应用最多创建多少文件等的限制权限。Linux 默认来说，一般限制应用最多创建的文件是 65535 个。但是 ElasticSearch 至少需要 65536 的文件创建权限。修改后的内容为：

* soft nofile 65536

* hard nofile 65536

1.2 修改线程开启限制

在 CentOS6.5 版本中编辑下述的配置文件

vi /etc/security/limits.d/90-nproc.conf

在 CentOS7+版本中编辑配置文件是：

vi /etc/security/limits.conf

是修改系统中允许用户启动的进程开启多少个线程。默认的 Linux 限制 root 用户开启的进程可以开启任意数量的线程，其他用户开启的进程可以开启 1024 个线程。必须修改限制数为 4096+。因为 ElasticSearch 至少需要 4096 的线程池预备。ElasticSearch 在 5.x 版本之后，强制要求在 linux 中不能使用 root 用户启动 ElasticSearch 进程。所以必须使用其他用户启动 ElasticSearch 进程才可以。

```
*               soft    nproc       4096
root            soft    nproc       unlimited
```

注意：Linux 低版本内核为线程分配的内存是 128K。4.x 版本的内核分配的内存更大。如果虚拟机的内存是 1G，最多只能开启 3000+个线程数。至少为虚拟机分配 1.5G 以上的内存。

1.3 修改系统控制权限

CentOS6.5 中的配置文件为：

vi /etc/sysctl.conf

CentOS8 中的配置文件为：

vi /etc/sysctl.d/99-sysctl.conf

系统控制文件是管理系统中的各种资源控制的配置文件。ElasticSearch 需要开辟一个 65536 字节以上空间的虚拟内存。Linux 默认不允许任何用户和应用直接开辟虚拟内存。

新增内容为：

vm.max_map_count=655360

使用命令： sysctl -p 。 让系统控制权限配置生效。

2 安装 ElasticSearch

ElasticSearch 是 java 开发的应用。在 6.8.4 版本中，要求 JDK 至少是 1.8.0_131 版本以上。

ElasticSearch 的安装过程非常简单。解压立刻可以使用。

2.1 解压缩安装压缩包

tar -zxf elasticsearch-6.8.4.tar.gz

2.2 移动 ElasticSearch

mv elasticsearch-6.8.4 /usr/local/es/

2.3 修改 ElasticSearch 应用的所有者

因为 ElasticSearch 不允许 root 用户启动，而课堂案例中，ElasticSearch 是 root 用户解压缩的。所以解压后的 ElasticSearch 应用属于 root 用户。所以我们需要将 ElasticSearch 应用的所有者修改为其他用户。当前课堂案例中虚拟机 Linux 内有 bjsxt 这个用户。

chown -R bjsxt.bjsxt /usr/local/es

2.4 切换用户

su bjsxt

2.5 修改配置

修改 config/elasticsearch 的配置文件，设置可访问的客户端。0.0.0.0 代表任意客户端访问。

vi config/elasticsearch.yml

增加下述内容：

network.host: 0.0.0.0

2.6 启动

前台启动

/usr/local/es/bin/elasticsearch

关闭： ctrl + c

后台启动

/usr/local/es/bin/elasticsearch -d

关闭：

jps 命令查看 ElasticSearch 线程的编号

kill -9 ElasticSearch 线程编号

2.7 测试连接

curl http://localhost:9200

返回如下结果：

```
{
  "name" : "L6WdN7y",
  "cluster_name" : "elasticsearch",
  "cluster_uuid" : "s7_GSd9YQnaH10VQBKCQ5w",
  "version" : {
    "number" : "6.3.1",
    "build_flavor" : "default",
    "build_type" : "tar",
    "build_hash" : "eb782d0",
    "build_date" : "2018-06-29T21:59:26.107521Z",
    "build_snapshot" : false,
    "lucene_version" : "7.3.1",
```

```
    "minimum_wire_compatibility_version" : "5.6.0",
    "minimum_index_compatibility_version" : "5.0.0"
  },
  "tagline" : "You Know, for Search"
}
```

3 搭建集群

修改配置文件$elasticsearch_home/config/elasticsearch.yml

增加配置：

```
# 发现的节点 IP
discovery.zen.ping.unicast.hosts: ["ip1", "ip2"]
# 最小集群数：常用计算公式 - 总数/2 + 1
discovery.zen.minimum_master_nodes: min_nodes_count
```

4 安装 Kibana

Kibana 是一个基于 WEB 的 ElasticSearch 管理控制台。现阶段安装 Kibana 主要是为了方便学习。

在 Linux 中安装 Kibana 很方便。解压，启动即可。Kibana 要求的环境配置是小于 ElasticSearch 的要求的。

```
tar -zxf kibana-6.3.1-linux-x86_64.tar.gz
```

修改 config/kibana.yml

```
vi config/kibana.yml
```

新增内容： server.host: "0.0.0.0"

bin/kibana

访问时，使用浏览器访问 http://192.168.2.119:5601/

三、 常用 ElasticSearch 管理操作

1 查看健康状态

GET _cat/health?v

epoch	timestamp	cluster	status	node.total	node.data	shards
1531290005	14:20:05	elasticsearch	green	1	1	2

pri	relo	init	unassign	pending_tasks	max_task_wait_time	active_shards_percent
2	0	0	0	0	-	100.0%

status：green、yellow、red

green：每个索引的 primary shard 和 replica shard 都是 active 的

yellow 每个索引的 primary shard 都是 active 的 ,但部分的 replica shard 不是 active 的

red：不是所有的索引的 primary shard 都是 active 状态的。

2 创建索引

命令语法：PUT 索引名{索引配置参数}

index 名称必须是小写的，且不能以下划线'_'，'-'，'+'开头。

在 ElasticSearch 中，默认的创建索引的时候，会分配 5 个 primary shard，并为每个 primary shard 分配一个 replica shard（在 ES7 版本后，默认创建 1 个 primary shard）。

在 ElasticSearch 中 ,默认的限制是 :如果磁盘空间不足 15%的时候 ,不分配 replica shard。如果磁盘空间不足 5%的时候，不再分配任何的 primary shard。ElasticSearch 中对 shard 的分布是有要求的。ElasticSearch 尽可能保证 primary shard 平均分布在多个节点上。Replica shard 会保证不和他备份的那个 primary shard 分配在同一个节点上。

创建默认索引

```
PUT test_index1
```

创建索引时指定分片。

```
PUT test_index2
{
   "settings":{
      "number_of_shards" : 2,
      "number_of_replicas" : 1
   }
}
```

3 修改索引

命令语法：PUT 索引名/_settings{索引配置参数}

注意：索引一旦创建，primary shard 数量不可变化，可以改变 replica shard 数量。

```
PUT test_index2/_settings
{
   "number_of_replicas" : 2
}
```

4 删除索引

命令语法：DELETE 索引名 1[, 索引名 2 ...]

DELETE test_index1

5 查看索引信息

GET _cat/indices?v

health	status	index	uuid	pri	rep	docs.count
yellow	open	test_index	2PJFQBtzTwOUhcy-QjfYmQ	5	1	0

docs.deleted	store.size	pri.store.size
0	460b	460b

6 检查分片信息

查看索引的 shard 信息。

GET _cat/shards?v

index	shard	prirep	state	docs	store	ip	node
test_index2	1	p	STARTED	0	261b	192.168.89.142	mN_pylT
test_index2	1	r	UNASSIGNED				
test_index2	1	r	UNASSIGNED				
test_index2	0	p	STARTED	0	261b	192.168.89.142	mN_pylT
test_index2	0	r	UNASSIGNED				
test_index2	0	r	UNASSIGNED				

7 新增 Document

在索引中增加文档。在 index 中增加 document。

ElasticSearch 有自动识别机制。如果增加的 document 对应的 index 不存在，自动创建 index；如果 index 存在，type 不存在，则自动创建 type。如果 index 和 type 都存在，则使用现有的 index 和 type。

7.1 PUT 语法

此操作为手工指定 id 的 Document 新增方式。

语法：PUT 索引名/类型名/唯一 ID{字段名:字段值}

如：

```
PUT test_index/my_type/1
{
    "name":"test_doc_01",
    "remark":"first test elastic search",
    "order_no":1
}
PUT test_index/my_type/2
{
    "name":"test_doc_02",
    "remark":"second test elastic search",
    "order_no":2
}
PUT test_index/my_type/3
{
    "name":"test_doc_03",
    "remark":"third test elastic search",
    "order_no":3
}
```

结果：

```
{
  "_index": "test_index", 新增的 document 在什么 index 中，
  "_type": "my_type", 新增的 document 在 index 中的哪一个 type 中。
  "_id": "1", 指定的 id 是多少
  "_version": 1, document 的版本是多少，版本从 1 开始递增，每次写操作都会+1
  "result": "created", 本次操作的结果，created 创建，updated 修改，deleted 删除
  "_shards": { 分片信息
    "total": 2, 分片数量只提示 primary shard
    "successful": 1, 数据 document 一定只存放在 index 中的某一个 primary shard 中
    "failed": 0
  },
  "_seq_no": 0, 执行的序列号
  "_primary_term": 1 词条比对。
}
```

如果使用 PUT 语法对同 id 的 Document 执行多次操作。是一种覆盖操作。如果需要 ElasticSearch 辅助检查 PUT 的 Document 是否已存在，可以使用强制新增语法。***使用强制新增语法时，如果 Document 的 id 在 ElasticSearch 中已存在，则会报错。（version conflict, document already exists）***

语法：

PUT 索引名/类型名/唯一 ID/_create{字段名:字段值}

或

PUT 索引名/类型名/唯一 ID?op_type=create{字段名:字段值}。

如：

```
PUT test_index/my_type/1/_create
{
    "name":"new_test_doc_01",
    "remark":"first test elastic search",
    "order_no":1
}
```

7.2 POST 语法

此操作为 ElasticSearch 自动生成 id 的新增 Document 方式。此语法格式和 PUT 请求的数据新增，只有唯一的区别，就是可以自动生成主键 id，其他的和 PUT 请求新增数据完全一致。

语法：POST 索引名/类型名{字段名:字段值}

如：

```
POST test_index/my_type
{
    "name":"test_doc_04",
    "remark":"forth test elastic search",
    "order_no":4
```

```
}
```

8 查询 Document

8.1 GET ID 单数据查询

语法：GET 索引名/类型名/唯一 ID

如：

GET test_index/my_type/1

结果：

```
{
  "_index": "test_index",
  "_type": "my_type",
  "_id": "1",
  "_version": 1,
  "found": true,
  "_source": { 找到的 document 数据内容。
    "name": "test_doc_01",
    "remark": "first test elastic search",
    "order_no":1
  }
}
```

8.2 GET _mget 批量查询

批量查询可以提高查询效率。推荐使用（相对于单数据查询来说）。

语法如下：

```
GET _mget
{
  "docs" : [
    {
      "_index" : "索引名",
      "_type" : "类型名",
      "_id" : "唯一 ID 值"
    }, {}, {}
  ]
}
```

```
GET 索引名/_mget
```

```
{
  "docs" : [
    {
      "_type" : "类型名",
      "_id" : "唯一 ID 值"
    }, {}, {}
  ]
}
```

```
GET 索引名/类型名/_mget
{
  "docs" : [
    {
      "_id" : "唯一 ID 值"
    },
    {
      "_id" : "唯一 ID 值"
    }
  ]
}
```

9 修改 Document

9.1 替换 Document（全量替换）

和新增的 PUT|POST 语法是一致。

PUT|POST 索引名/类型名/唯一 ID{字段名:字段值}

本操作相当于覆盖操作。全量替换的过程中，ElasticSearch 不会真的修改 Document 中的数据，而是标记 ElasticSearch 中原有的 Document 为 deleted 状态，再创建一个新的 Document 来存储数据，当 ElasticSearch 中的数据量过大时，ElasticSearch 后台回收 deleted 状态的 Document。

如：

```
PUT test_index/my_type/1
{
    "name":"new_test_doc_01",
    "remark":"first test elastic search",
```

```
    "order_no":1
}
```

结果：

```
{
  "_index": "test_index",
  "_type": "my_type",
  "_id": "1",
  "_version": 2,
  "result": "updated",
  "_shards": {
    "total": 2,
    "successful": 1,
    "failed": 0
  },
  "_seq_no": 1,
  "_primary_term": 1
}
```

9.2 更新 Document (partial update)

语法：POST 索引名/类型名/唯一 ID/_update{doc:{字段名:字段值}}

只更新某 Document 中的部分字段。这种更新方式也是标记原有数据为 deleted 状态，创建一个新的 Document 数据 ,将新的字段和未更新的原有字段组成这个新的 Document，并创建。对比全量替换而言，只是操作上的方便，在底层执行上几乎没有区别。

如：

```
POST test_index/my_type/1/_update
{
    "doc":{
        "name":" test_doc_01_for_update"
    }
}
```

结果：

```
{
  "_index": "test_index",
  "_type": "my_type",
  "_id": "1",
  "_version": 5,
```

```
    "result": "updated",
    "_shards": {
      "total": 2,
      "successful": 1,
      "failed": 0
    },
    "_seq_no": 2,
    "_primary_term": 1
}
```

10 删除 Document

ElasticSearch 中执行删除操作时，ElasticSearch 先标记 Document 为 deleted 状态，而不是直接物理删除。当 ElasticSearch 存储空间不足或工作空闲时，才会执行物理删除操作。标记为 deleted 状态的数据不会被查询搜索到。

语法：DELETE 索引名/类型名/唯一 ID

如：

```
DELETE test_index/my_type/1
```

结果：

```
{
  "_index": "test_index",
  "_type": "my_type",
  "_id": "1",
  "_version": 6,
  "result": "deleted",
  "_shards": {
    "total": 2,
    "successful": 1,
    "failed": 0
  },
  "_seq_no": 5,
  "_primary_term": 1
}
```

11 bulk 批量增删改

使用 bulk 语法执行批量增删改。语法格式如下：

POST _bulk

{ "action_type" : { "metadata_name" : "metadata_value" } }

{ document datas | action datas }

语法中的 action_type 可选值为：

create：强制创建，相当于 PUT 索引名/类型名/唯一 ID/_create

index: 普通的 PUT 操作，相当于创建 Document 或全量替换

update: 更新操作(partial update),相当于 POST 索引名/类型名/唯一 ID/_update

delete: 删除操作

案例如下：

新增数据： POST _bulk { "create" : { "_index" : "test_index" , "_type" : "my_type", "_id" : "1" } } { "field_name" : "field value" }
PUT 操作新增或全量替换 POST _bulk { "index" : { "_index" : "test_index", "_type" : "my_type" , "_id" : "2" } } { "field_name" : "field value 2" }
POST 更新数据 POST _bulk { "update" : { "_index" : "test_index", "_type" : "my_type" , "_id" : 2, "_retry_on_conflict" : 3 } } { "doc" : { "field_name" : "partial update field value" } }
DELETE 删除数据 POST _bulk { "delete" : { "_index" : "test_index", "_type" : "my_type", "_id" : "2" } }
批量写操作 POST _bulk { "create" : { "_index" : "test_index" , "_type" : "my_type", "_id" : "10" } } { "field_name" : "field value" } { "index" : { "_index" : "test_index", "_type" : "my_type" , "_id" : "20" } } { "field_name" : "field value 2" } { "update" : { "_index" : "test_index", "_type" : "my_type" , "_id" : 20, "_retry_on_conflict" : 3 } } { "doc" : { "field_name" : "partial update field value" } } { "delete" : { "_index" : "test_index", "_type" : "my_type", "_id" : "2" } }

注意：bulk 语法中要求一个完整的 json 串不能有换行。不同的 json 串必须使用换行分隔。多个操作中，如果有错误情况，不会影响到其他的操作，只会在批量操作返回结果中标记失败。bulk 语法批量操作时，bulk request 会一次性加载到内存中，如果请求数据量太大，性能反而下降（内存压力过高），需要反复尝试一个最佳的 bulk request size。一般从 1000~5000 条数据开始尝试，逐渐增加。如果查看 bulk request size 的话，一般是 5~15MB 之间为好。

bulk 语法要求 json 格式是为了对内存的方便管理，和尽可能降低内存的压力。如果 json 格式没有特殊的限制，ElasticSearch 在解释 bulk 请求时，需要对任意格式的 json 进行解释处理，需要对 bulk 请求数据做 json 对象会 json array 对象的转化，那么内存的占用量至少翻倍，当请求量过大的时候，对内存的压力会直线上升，且需要 jvm gc 进程对垃圾数据做频繁回收，影响 ElasticSearch 效率。

生成环境中，bulk api 常用。都是使用 java 代码实现循环操作。一般一次 bulk 请求，执行一种操作。如：批量新增 10000 条数据等。

四、 分词器（analyzer）和标准化处理（normalization）

1 什么是分词器

分词器是一个字符串解析拆分工具。其作用是分析写入的 Document 中的文本数据 field，并将 field 数据拆分成一个个有完整含义的、不可拆分的单词。

如：I think dogs is human's best friend.在写入此数据的时候，ElasticSearch 会使用分词器分析并拆分数据，将上述的语句切分成若干的单词，分别是：I、 think、 dogs、 human's、 best、 friend。

2 什么是标准化处理

标准化处理是用于完善分词器结果的。

分词器处理的文本结果，通常会有一些不需要的、有异议的、包含时态转化等情况的数据。在上述案例中的分词结果是：i、 think、 dogs、 human's、 best、 friend。其中 i 是很少应用在搜索条件中的单词；dogs 是 dog 单词的复数形式，通常在搜索过程中使用 dog 作为搜索条件更频繁一些；human's 是特殊的标记方式，通常不会在搜索中作为条件出现。那么 ElasticSearch 维护这些单词是没有太大必要的。这个时候就需要标准化处理了。

如：china 搜索时，如果条件为 cn 是否可搜索到。如：dogs，搜索时，条件为 dog 是否可搜索到数据。如果可以使用简写（cn）或者单复数（dog&dogs）搜索到想要的结果，那么称为搜索引擎人性化。

normalization 是为了提升召回率的（recall），就是提升搜索能力的。

normalization 是配合分词器(analyzer)完成其功能的。

3 ElasticSearch 默认提供的常见分词器

要切分的语句：Set the shape to semi-transparent by calling set_trans(5)

standard analyzer - 是 ElasticSearch 中的默认分词器。标准分词器，处理英语语法的分词器。切分后的 key_words：set, the, shape, to, semi, transparent, by, calling, set_trans, 5。这种分词器也是 ElasticSearch 中默认的分词器。切分过程中不会忽略停止词（如：the、a、an 等）。会进行单词的大小写转换、过滤连接符（-）或括号等常见符号。

```
GET _analyze
{
  "text": "Set the shape to semi-transparent by calling set_trans(5)",
```

"analyzer": "standard"

}

simple analyzer - 简单分词器。切分后的 key_words：set, the, shape, to, semi, transparent, by, calling, set, trans。就是将数据切分成一个个的单词。使用较少，经常会破坏英语语法。

GET _analyze

{

"text": "Set the shape to semi-transparent by calling set_trans(5)",

"analyzer": "simple"

}

whitespace analyzer - 空白符分词器。切分后的 key_words：Set, the, shape, to, semi-transparent, by, calling, set_trans(5)。就是根据空白符号切分数据。如：空格、制表符等。使用较少，经常会破坏英语语法。

GET _analyze

{

"text": "Set the shape to semi-transparent by calling set_trans(5)",

"analyzer": "whitespace"

}

language analyzer - 语言分词器 ,如英语分词器(english)等。切分后的 key_words：set, shape, semi, transpar, call, set_tran, 5。根据英语语法分词，会忽略停止词、转换大小写、单复数转换、时态转换等，应用分词器分词功能类似 standard analyzer。

GET _analyze

```
{
  "text": "Set the shape to semi-transparent by calling set_trans(5)",
  "analyzer": "english"
}
```

注意：ElasticSearch 中提供的常用分词器都是英语相关的分词器，对中文的分词都是一字一词。

4 安装中文分词器

IK 中文分词器，很少有直接下载使用的，都需要通过 github 下载源码，本地编译打包。就是 maven 工程中的 package 能力。

github 上提供的源码不是伴随 ES 的每个版本提供，一般只有分词器无效后，才提供新的版本。通常都是伴随 ES 的次版本号提供 IK 分词器版本。下载对应的 IK 分词器源码，本地 package 打包，生成 zip 压缩包，既是 IK 在 ES 中的分词器安装包。

git clone https://github.com/medcl/elasticsearch-analysis-ik.git

git checkout tags/v6.5.0

4.1 安装 IK 分词器

ElasticSearch 是一个开箱即用的工具。插件安装方式也非常简单。

将 IK 分词器的 zip 压缩文件上传到 Linux，并在 ElasticSearch 安装目录的 plugins 目录中手工创建子目录，目录命名为 ik。将 zip 压缩文件解压缩到新建目录 ik 中。重新启动 ElasticSearch 即可。

复制中文分词器 zip 压缩文件到 ElasticSearch 应用目录中：

cp elasticsearch-analysis-ik-6.8.4.zip /opt/es/plugins/

创建 IK 中文分词器的插件子目录：

mkdir /opt/es/plugins/ik/

移动压缩文件到 ik 插件目录中：

mv /opt/es/plugins/elasticsearch-analysis-ik-6.8.4.zip /usr/local/es/plugins/ik/

解压缩：

unzip /opt/es/plugins/ik/elasticsearch-analysis-ik-6.8.4.zip

所有的分词器，都是针对词语的，不是语句的。拆分单元是词语，不是语句。

4.2 测试 IK 分词器

IK 分词器提供了两种 analyzer，分别是 ik_max_word 和 ik_smart。

ik_max_word: 会将文本做最细粒度的拆分，比如会将“中华人民共和国国歌”拆分为“中华人民共和国,中华人民,中华,华人,人民共和国,人民,人,民,共和国,共和,国,国歌”，会穷尽各种可能的组合；

ik_smart: 会做最粗粒度的拆分,比如会将“中华人民共和国国歌”拆分为“中华人民共和国,国歌”。

```
GET _analyze
{
  "text" : "中华人民共和国国歌",
  "analyzer": "ik_max_word"
}

GET _analyze
{
  "text" : "中华人民共和国国歌",
  "analyzer": "ik_smart"
}
```

4.3 IK 配置文件

IK 的配置文件在 ElasticSearch 安装目录/plugins/ik/config/中。

配置文件有：

main.dic ： IK 中内置的词典。 main dictionary。记录了 IK 统计的所有中文单词。一行一词。文件中未记录的单词，IK 无法实现有效分词。如：雨女无瓜。不建议修改当前文件中的单词。这个是最核心的中文单词库。就好像，很多的网络词不会收集到辞海中一样。

quantifier.dic ： IK 内置的数据单位词典

suffix.dic ：IK 内置的后缀词典

surname.dic ：IK 内置的姓氏词典

stopword.dic ：IK 内置的英文停用词

preposition.dic ：IK 内置的中文停用词（介词）

IKAnalyzer.cfg.xml ： 用于配置自定义词库的

自定义词库是用户手工提供的特殊词典，类似网络热词，特定业务用词等。

ext_dict - 自定义词库，配置方式为相对于 IKAnalyzer.cfg.xml 文件所在位置的相对路径寻址方式。相当于是用户自定义的一个 main.dic 文件。是对 main.dic 文件的扩展。

ext_stopwords - 自定义停用词，配置方式为相对于 IKAnalyzer.cfg.xml 文件所在位置的相对路径寻址方式。相当于是 preposition.dic 的扩展。

注意：IK 的所有的 dic 词库文件，必须使用 UTF-8 字符集。不建议使用 windows 自带的文本编辑器编辑。Windows 中自带的文本编辑器是使用 GBK 字符集。IK 不识别，是乱码。

五、 ElasticSearch 中的 mapping 问题

Mapping 在 ElasticSearch 中是非常重要的一个概念。决定了一个 index 中的 field 使用什么数据格式存储，使用什么分词器解析，是否有子字段等。

Mapping 决定了 index 中的 field 的特征。

1 mapping 核心数据类型

ElasticSearch 中的数据类型有很多，在这里只介绍常用的数据类型。

文本（字符串）：text

整数：byte、short、integer、long

浮点型：float、double

布尔类型：boolean

日期类型：date

数组类型：array　{a:[]}

对象类型：object　{a:{}}

不分词的字符串（关键字）： keyword

2 dynamic mapping 对字段的类型分配

true or false -> boolean

123 -> long

123.123 -> double

2018-01-01 -> date

hello world -> text

[] -> array

{} -> object

在上述的自动 mapping 字段类型分配的时候，只有 text 类型的字段需要分词器。默认分词器是 standard 分词器。

3 查看索引 mapping

可以通过命令查看已有 index 的 mapping 具体信息，语法如下：

GET 索引名/_mapping

如：

```
GET test_index/_mapping
```

结果：

```
{
  "test_index": { # 索引名
    "mappings": { # 映射列表
      "my_type": { # 类型名
        "properties": { # 字段列表
          "age": { # 字段名
            "type": "long" # 字段类型
          },
          "gender": {
            "type": "text",
            "fields": { # 子字段列表
              "keyword": { # 子字段名
                "type": "keyword", # 子字段类型，keyword 不进行分词处理的文本类型
                "ignore_above": 256 # 子字段存储数据长度
              }
            }
          },
          "name": {
            "type": "text",
            "fields": {
              "keyword": {
                "type": "keyword",
                "ignore_above": 256
              }
            }
          }
        }
      }
    }
  }
}
```

4 custom mapping

可以通过命令，在创建 index 和 type 的时候来定制 mapping 映射，也就是指定字段的类型和字段数据使用的分词器。

手工定制 mapping 时 ,只能***新增 mapping 设置*** ,***不能对已有的 mapping 进行修改***。

如：有索引 a，其中有类型 b，增加字段 f1 的 mapping 定义。后续可以增加字段 f2 的 mapping 定义，但是不能修改 f1 字段的 mapping 定义。

通常都是手工创建 index，并进行各种定义。如：settings,mapping 等。

4.1 创建索引时指定 mapping

语法：

```
PUT  索引名称
{
   "mappings":{
      "类型名称":{
         "properties":{
            "字段名":{
               "type":类型,
               ["analyer":字段的分词器,]
               ["fields":{
                  "子字段名称":{
                     "type":类型,
                     "ignore_above":长度限制
```

```
                }
            }]
        }
    }
}
}
}
```

如：

```
PUT /test_index
{
  "settings": {
    "number_of_shards": 2,
    "number_of_replicas": 1
  },
  "mappings": {
    "test_type":{
      "properties": {
        "author_id" : {
          "type": "byte",
          "index": false
        },
        "title" : {
          "type": "text",
          "analyzer": "ik_max_word",
          "fields": {
            "keyword" : {
              "type": "keyword",
              "ignore_above": 256
            }
          }
        },
        "content" : {
          "type": "text",
          "analyzer": "ik_max_word"
        },
        "post_date" : {
```

```
            "type": "date"
          }
        }
      }
    }
  }
```

"index" - 是否可以作为搜索索引。可选值：true | false

"analyzer" - 指定分词器。

"type" - 指定字段类型

4.2 为已有索引添加*新的字段 mapping*

语法：

PUT 索引名/_mapping/类型名

```
{
  "properties":{
    "新字段名":{
      "type":类型,
      "analyer":字段的分词器,
      "fields":{
        "子字段名":{
          "type":类型,
          "ignore_above":长度
        }
      }
    }
  }
```

}

如：

```
PUT /test_index/_mapping/test_type
{
  "properties" : {
    "new_field" : { "type" : "text" , "analyzer" : "standard" }
  }
}
```

4.3 测试不同的字段的分词器

语法：

GET 索引名称/_analyze

{

"field":"索引中的 text 类型的字段名",

"text":"要分词处理的文本数据"

}

使用索引中的字段对应的分词器，对文本数据做分词处理。

如：

```
GET /test_index/_analyze
{
  "field": "new_field",
  "text": "中华人民共和国国歌"
}

GET /test_index/_analyze
{
  "field": "content",
  "text": "中华人民共和国国歌"
}
```

六、 Search 搜索详解

1 搜索学习测试数据

```
PUT test_search
{
  "mappings": {
    "test_type" : {
      "properties": {
        "dname" : {
          "type" : "text",
          "analyzer": "standard"
        },
        "ename" : {
          "type" : "text",
          "analyzer": "standard"
        },
        "eage" : {
          "type": "long"
        },
        "hiredate" : {
          "type": "date"
        },
        "gender" : {
          "type" : "keyword"
        }
      }
    }
  }
}
POST test_search/test_type/_bulk
{ "index": {}}
{ "dname" : "Sales Department", "ename" : "张三", "eage":20, "hiredate" : "2019-01-01", "gender" : "男性" }
{ "index": {}}
{ "dname" : "Sales Department", "ename" : "李四", "eage":21, "hiredate" : "2019-02-01", "gender" : "男性" }
{ "index": {}}
{ "dname" : "Development Department", "ename" : "王五", "eage":23, "hiredate" : "2019-01-03", "gender" : "男性" }
{ "index": {}}
{ "dname" : "Development Department", "ename" : "赵六", "eage":26, "hiredate" : "2018-01-01", "gender" : "男性" }
```

```
{ "index": {}}
{ "dname" : "Development Department", "ename" : " 韩 梅 梅 ", "eage":24, "hiredate" : "2019-03-01", "gender" : "女性" }
{ "index": {}}
{ "dname" : "Development Department", "ename" : " 钱 虹 ", "eage":29, "hiredate" : "2018-03-01", "gender" : "女性" }
```

2 query string search

search 的参数都是类似 http 请求头中的字符串参数提供搜索条件的。

GET

[/index_name/type_name/]_search[?parameter_name=parameter_value&...]

2.1 全搜索

timeout 参数：是超时时长定义。代表每个节点上的每个 shard 执行搜索时最多耗时多久。不会影响响应的正常返回。只会影响返回响应中的数据数量。

如：索引 a 中，有 10 亿数据。存储在 5 个 shard 中，假设每个 shard 中 2 亿数据，执行全数据搜索的时候，需要耗时 1000 毫秒。定义 timeout 为 10 毫秒，代表的是 shard 执行 10 毫秒，搜索出多少数据，直接返回。

在商业项目中，是禁止全数据搜索的。必须指定搜索的索引，类型和关键字。如果没有指定索引或类型，则代表开发目的不明确，需要重新做用例分析。如果没有关键字，称为索引内全搜索，也叫魔鬼搜索。

语法：

```
GET [索引名/类型名/]_search?timeout=10ms
```

结果：

```
{
  "took": 144, #请求耗时多少毫秒
  "timed_out": false, #是否超时。默认情况下没有超时机制，也就是客户端等待 ElasticSearch 搜索
```

```
结束（无论执行多久），提供超时机制的话，ElasticSearch 则在指定时长内处理搜索，在指定时长结束的时候，将搜索的结果直接返回（无论是否搜索结束）。指定超时的方式是传递参数，参数单位是：毫秒-ms。秒-s。分钟-m。
    "_shards": {
      "total": 1, #请求发送到多少个 shard 上
      "successful": 1,#成功返回搜索结果的 shard
      "skipped": 0, #停止服务的 shard
      "failed": 0 #失败的 shard
    },
    "hits": {
      "total": 1, #返回了多少结果
      "max_score": 1, #搜索结果中，最大的相关度分数，相关度越大分数越高，_score 越大，排位越靠前。
      "hits": [ #搜索到的结果集合，默认查询前 10 条数据。
        {
          "_index": "test_index", #数据所在索引
          "_type": "my_type", #数据所在类型
          "_id": "1", #数据的 id
          "_score": 1, #数据的搜索相关度分数
          "_source": { #  数据的具体内容。
            "field": "value"
          }
        }
      ]
    }
  }
```

2.2 multi index 搜索

所谓的 multi-index 就是从多个 index 中搜索数据。相对使用较少，只有在复合数据搜索的时候，可能出现。一般来说，如果真使用复合数据搜索，都会使用_all。

如：搜索引擎中的无条件搜索。（现在的应用中都被屏蔽了。使用的是默认搜索条件，执行数据搜索。 如： 电商中的搜索框默认值， 搜索引擎中的类别）

无条件搜索，在搜索应用中称为“魔鬼搜索”，代表的是，搜索引擎会执行全数据检索，效率极低，且对资源有非常高的压力。

语法：

```
GET _search
```

```
GET 索引名 1,索引名 2/_search # 搜索多个 index 中的数据

GET 索引名/类型名/_search # 所属一个 index 中 type 的数据

GET prefix_*/_search # 通配符搜索
GET *_suffix/_search

GET 索引名 1,索引名 2/类型名/_search # 搜索多个 index 中 type 的数据

GET _all/_search # _all 代表所有的索引
```

2.3 条件搜索

query string search 搜索是通过 HTTP 请求的请求头传递参数的，默认的 HTTP 请求头字符集是 ISO-8859-1，请求头传递中文会有乱码。

语法：

```
GET 索引名/_search?q=字段名:搜索条件
```

2.4 分页搜索

默认情况下，ElasticSearch 搜索返回结果是 10 条数据。从第 0 条开始查询。

语法：

```
GET 索引名/_search?size=10 # size 查询数据的行数

GET 索引名/_search?from=0&size=10 # from 从第几行开始查询，行号从 0 开始。
```

2.5 +/-搜索

语法：

```
GET 索引名/_search?q=字段名:条件

GET 索引名/_search?q=+字段名:条件

GET 索引名/_search?q=-字段名:条件
```

+ ：和不定义符号含义一样，就是搜索指定的字段中包含 key words 的数据

- ： 与+符号含义相反，就是搜索指定的字段中不包含 key words 的数据

2.6 排序

语法：GET 索引名/_search?sort=字段名:排序规则

排序规则： asc(升序) | desc(降序)

```
GET test_search/_search?sort=eage:asc

GET test_search/_search?sort=eage:desc
```

3 query DSL

DSL - Domain Specified Language ， 特殊领域的语言。

请求参数是请求体传递的。在 ElasticSearch 中，请求体的字符集默认为 UTF-8。

语法格式：

GET 索引名/_search

{

"command":{ "parameter_name" : "parameter_value"}

}

3.1 查询所有数据

```
GET 索引名/_search
{
    "query" : { "match_all" : {} }
}
```

3.2 match search

全文检索。要求查询条件拆分后的任意词条与具体数据匹配就算搜索结果。

```
GET 索引名/_search
```

```
{
  "query": {
    "match": {
      "字段名": "搜索条件"
    }
  }
}
```

3.3 phrase search

短语检索。要求查询条件必须和具体数据完全匹配才算搜索结果。其特征是：1-搜索条件不做任何分词解析；2-在搜索字段对应的倒排索引(正排索引)中进行精确匹配，不再是简单的全文检索。

```
GET 索引名/_search
{
  "query": {
    "match_phrase": {
      "字段名": "搜索条件"
    }
  }
}
```

3.4 range

范围比较搜索

```
GET 索引名/类型名/_search
{
  "query" : {
    "range" : {
      "字段名" : {
        "gt" : 搜索条件 1,
        "lte" : 搜索条件 2
      }
    }
  }
}
```

3.5 term

词组比较，词组搜索。

忽略搜索条件分词，在 ElasticSearch 倒排索引中进行精确匹配。

```
GET 索引名/类型名/_search
{
  "query" : {
    "term" : {
      "字段名": "搜索条件"
    }
  }
}

GET 索引名/类型名/_search
{
  "query" : {
    "terms" : {
      "字段名": ["搜索条件 1", "搜索条件 2"]
    }
  }
}
```

3.6 多条件复合搜索

在一个请求体中，有多个搜索条件，就是复合搜索。如：搜索数据，条件为部门名称是 Sales Department，员工年龄在 20 到 26 之间，部门员工姓名叫张三。上述条件中，部门名称为可选条件，员工年龄必须满足要求，部门员工姓名为可选要求。这种多条件搜索就是符合搜索。

```
GET 索引名/类型名/_search
{
  "query": {
    "bool": {
      "must": [ #数组中的多个条件必须同时满足
        {
          "range": {
            "字段名": {
              "lt": 条件
            }
          }
        }
      ],
      "must_not":[ #数组中的多个条件必须都不满足
```

```
            {
              "match": {
                "字段名": "条件"
              }
            },
            {
              "range": {
                "字段名": {
                  "gte": "搜索条件"
                }
              }
            }
          ]
          "should": [#  数组中的多个条件有任意一个满足即可。
            {
              "match": {
                "字段名": "条件"
              }
            },
            {
              "range": {
                "字段名": {
                  "gte": "搜索条件"
                }
              }
            }
          ]
        }
      }
    }
```

3.7 排序

在 ElasticSearch 的搜索中，默认是使用相关度分数实现排序的。可以通过搜索语法实现定制化排序。

```
GET  索引名/类型名/_search
{
  "query": {
    [搜索条件]
  },
  "sort": [
    {
```

```
            "字段名 1": {
              "order": "asc"
            }
          },
          {
            "字段名 2": {
              "order": "desc"
            }
          }
      ]
    }
```

注意：在 ElasticSearch 中，如果使用 text 类型的字段作为排序依据，会有问题。ElasticSearch 需要对 text 类型字段数据做分词处理。如果使用 text 类型字段做排序，ElasticSearch 给出的排序结果未必友好，毕竟分词后，先使用哪一个单词做排序都是不合理的。所以 ElasticSearch 中默认情况下不允许使用 text 类型的字段做排序，如果需要使用字符串做结果排序，则可使用 keyword 类型字段作为排序依据，因为 keyword 字段不做分词处理。

3.8 分页

DSL 分页也是使用 from 和 size 实现的。

```
GET 索引名称/_search
{
  "query":{
    "match_all":{}
  },
  "from": 起始下标,
  "size": 查询记录数
}
```

3.9 highlight display

在搜索中，经常需要对搜索关键字做高亮显示，这个时候就可以使用 highlight 语法。

语法：

```
GET 索引名/_search
{
  "query": {
    "match": {
      "字段名": "条件"
    }
  },
  "highlight": {
    "fields": {
      "要高亮显示的字段名": {
        "fragment_size": 5, #每个分段长度，默认 20
        "number_of_fragments": 1 #返回多少个分段，默认 3
      }
    },
    "pre_tags": ["前缀"],
    "post_tags": ["后缀"]
  }
}
```

```
// 演示案例
GET test_search/_search
{
  "query": {
    "bool": {
      "should": [
        {
          "match": {
            "dname": "Development department"
          }
        },
        {
          "match": {
            "gender": "男性"
          }
        }
      ]
    }
  },
  "highlight": {
    "fields": {
      "dname": {
        "fragment_size": 20,
        "number_of_fragments": 1
      },
      "gender": {
```

```
                "fragment_size": 20,
                "number_of_fragments": 1
            }
        },
        "pre_tags":["<span style='color:red'>"],
        "post_tags":["</span>"]
    },
    "from": 2,
    "size": 2
}
```

fragment_size：代表字段数据如果过长，则分段，每个片段数据长度为多少。长度不是字符数量，是 ElasticSearch 内部的数据长度计算方式。默认不对字段做分段。

number_of_fragments：代表搜索返回的高亮片段数量，默认情况下会将拆分后的所有片段都返回。

pre_tags：高亮前缀

post_tags：高亮后缀

很多搜索结果显示页面中都不会显示完整的数据 ,这样在数据过长的时候会导致页面效果不佳，都会按照某一个固定长度来显示搜索结果，所以 fragment_size 和 number_of_fragments 参数还是很常用的。