--elasticsearch学习笔记--

数据库做搜索弊端

站内搜索（垂直搜索）：数据量小，简单搜索，可以使用数据库

- 存储问题。电商网站商品上亿条时，涉及到单表数据过大必须拆分表，数据库磁盘占用过大必须分库 （mycat）。
- 性能问题：解决上面问题后，查询“笔记本电脑”等关键词时，上亿条数据的商品名字段逐行扫描，性能 跟不上。
- 不能分词。如搜索“笔记本电脑”，只能搜索完全和关键词一样的数据，那么数据量小时，搜索“笔记电 脑”，“电脑”数据要不要给用户。

互联网搜索，肯定不会使用数据库搜索。数据量太大。PB级

全文检索

全文检索是一种非结构化数据的搜索方式。

倒排索引

倒排索引源于实际应用中需要根据属性的值来查找记录。这种索引表中的每一项都包括一个属性值和具有该属性值的各记录的地址。由于不是由记录来确定属性值，而是由属性值来确定记录的位置，因而称为倒排索引(inverted index)。带有倒排索引的文件我们称为倒排索引文件，简称倒排文件(inverted file)。

倒排索引（英语：Inverted index），也常被称为反向索引、置入档案或反向档案，是一种索引方法，被用来存储在全文搜索下某个单词在一个文档或者一组文档中的存储位置的映射。它是文档检索系统中最常用的数据结构。通过倒排索引，可以根据单词快速获取包含这个单词的文档列表。倒排索引主要由两个部分组成：“单词词典”和“倒排文件”。

倒排索引有两种不同的反向索引形式：

- 一条记录的水平反向索引（或者反向档案索引）包含每个引用单词的文档的列表。
- 一个单词的水平反向索引（或者完全反向索引）又包含每个单词在一个文档中的位置。

后者的形式提供了更多的兼容性（比如短语搜索），但是需要更多的时间和空间来创建。现代搜索引擎的索引都是基于倒排索引。相比“签名文件”、“后缀树”等索引结构，倒排索引是实现单词到文档映射关系的最佳实现方式和最有效的索引结构。

elasticsearch是什么?

elasticsearch是目前最流行的准实时全文检索引擎，具有高速检索大数据的能力。

Elasticsearch是一个基于Lucene的搜索服务器。它提供了一个分布式多用户能力的全文搜索引擎，基于RESTful web接口。

Elasticsearch 是使用java开发，基于Lucene、分布式、通过Restful方式进行交互的近实时搜索平台框架。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。

Elasticsearch的功能

- 分布式的搜索引擎和数据分析引擎

  搜索：互联网搜索、电商网站站内搜索、OA系统查询

  数据分析：电商网站查询近一周哪些品类的图书销售前十；新闻网站，最近3天阅读量最高的十个关键 词，舆情分析。

- 全文检索，结构化检索，数据分析

  全文检索：搜索商品名称包含java的图书select * from books where book_name like "%java%"。

  结构化检索：搜索商品分类为spring的图书都有哪些，select * from books where category_id='spring'

  数据分析：分析每一个分类下有多少种图书，select category_id,count(*) from books group by category_id

- 对海量数据进行近实时的处理

  分布式：ES自动可以将海量数据分散到多台服务器上去存储和检索,经行并行查询，提高搜索效率。相对 的，Lucene是单机应用。

  近实时：数据库上亿条数据查询，搜索一次耗时几个小时，是批处理（batch-processing）。而es只需秒级即可查询海量数据，所以叫近实时。秒级。

Elasticsearch的使用场景

- 维基百科，类似百度百科，“网络七层协议”的维基百科，全文检索，高亮，搜索推荐
- Stack Overflow（国外的程序讨论论坛），相当于程序员的贴吧。遇到it问题去上面发帖，热心网友下面回帖解答。
- GitHub（开源代码管理），搜索上千亿行代码。
- 电商网站，检索商品日志数据分析，logstash采集日志，ES进行复杂的数据分析（ELK技术，elasticsearch+logstash+kibana）
- 商品价格监控网站，用户设定某商品的价格阈值，当低于该阈值的时候，发送通知消息给用户，比如说订阅《java编程思想》的监控，如果价格低于27块钱，就通知我，我就去买。
- BI系统，商业智能（Business Intelligence）。大型连锁超市，分析全国网点传回的数据，分析各个商品在什么季节的销售量最好、利润最高。成本管理，店面租金、员工工资、负债等信息进行分析。从而部署下一个阶段的战略目标。
- 百度搜索，第一次查询，使用es。

- OA、ERP系统站内搜索。

Elasticsearch的特点

- 可拓展性：大型分布式集群（数百台服务器）技术，处理PB级数据，大公司可以使用。小公司数据量小，也可以部署在单机。大数据领域使用广泛。
- 技术整合：将全文检索、数据分析、分布式相关技术整合在一起：lucene（全文检索），商用的数据分析软件（BI软件），分布式数据库（mycat）
- 部署简单：开箱即用，很多默认配置不需关心，解压完成直接运行即可。拓展时，只需多部署几个实例即可，负载均衡、分片迁移集群内部自己实施。
- 接口简单：使用restful api经行交互，跨语言。
- 功能强大：Elasticsearch作为传统数据库的一个补充，提供了数据库所不不能提供的很多功能，如全文检索，同义词处理，相关度排名。

lucene和elasticsearch的关系

Lucene：最先进、功能最强大的搜索库，直接基于lucene开发，非常复杂，api复杂 Elasticsearch：基于lucene，封装了许多lucene底层功能，提供简单易用的restful api接口和许多语言 的客户端，如java的高级客户端（Java High Level REST Client）和底层客户端（Java Low Level REST Client）

elasticsearch核心概念

NRT（Near Realtime）：近实时

- 写入数据时，过1秒才会被搜索到，因为内部在分词、录入索引。
- es搜索时：搜索和分析数据需要秒级出结果。

Cluster：集群

包含一个或多个启动着es实例的机器群。通常一台机器起一个es实例。同一网络下，集名一样的多个es实例自动组成集群，自动均衡分片等行为。默认集群名为“elasticsearch”。

Node：节点

每个es实例称为一个节点。节点名自动分配，也可以手动配置。

Index：索引

包含一堆有相似结构的文档数据。

索引创建规则：

- 仅限小写字母
- 不能包含\、/、 *、?、"、<、>、|、#以及空格符等特殊符号

- 从7.0版本开始不再包含冒号
- 不能以-、_或+开头
- 不能超过255个字节（注意它是字节，因此多字节字符将计入255个限制）

Document：文档

es中的最小数据单元。一个document就像数据库中的一条记录。通常以json格式显示。多个document存储于一个索引（Index）中。

```
book document
{
"book_id": "1",
"book_name": "java编程思想",
"book_desc": "从Java的基础语法到最高级特性（深入的[面向对象]
(https://baike.baidu.com/item/面向对象)概念、多线程、自动项目构建、单元测试和调试等），本书
都能逐步指导你轻松掌握。",
"category_id": "2",
"category_name": "java"
}
```

Field:字段

就像数据库中的列（Columns），定义每个document应该有的字段。

Type：类型

每个索引里都可以有一个或多个type，type是index中的一个逻辑数据分类，一个type下的document，都有相同的field。

注意：6.0之前的版本有type（类型）概念，type相当于关系数据库的表，ES官方将在ES9.0版本中彻底删除type。本教程typy都为_doc。

shard：分片

index数据过大时，将index里面的数据，分为多个shard，分布式的存储在各个服务器上面。可以支持海量数据和高并发，提升性能和吞吐量，充分利用多台机器的cpu。

replica：副本

在分布式环境下，任何一台机器都会随时宕机，如果宕机，index的一个分片没有，导致此index不能搜索。所以，为了保证数据的安全，我们会将每个index的分片经行备份，存储在另外的机器上。保证少数机器宕机es集群仍可以搜索。

能正常提供查询和插入的分片我们叫做主分片（primary shard），其余的我们就管他们叫做备份的分片（replica shard）。

es6默认新建索引时，5分片，2副本，也就是一主一备，共10个分片。所以，es集群最小规模为两台。

elasticsearch核念和数据库概念

关系型数据库（比如Mysql）	非关系型数据库（Elasticsearch）
数据库Database	索引Index
表Table	索引Index（原为Type）
数据行Row	文档Document
数据列Column	字段Field
约束 Schema	映射Mapping

elasticsearch相关配置

配置格式是YAML：elasticsearch.yml

常用的配置项如下

```
cluster.name:
    配置elasticsearch的集群名称，默认是elasticsearch。建议修改成一个有意义的名称。
node.name:
    节点名，通常一台物理服务器就是一个节点，es会默认随机指定一个名字，建议指定一个有意义的名称，方便管理
    一个或多个节点组成一个cluster集群，集群是一个逻辑的概念，节点是物理概念，后边章节会详细介绍。
path.conf:
    设置配置文件的存储路径，tar或zip包安装默认在es根目录下的config文件夹，rpm安装默认在/etc/elasticsearch
path.data:
    设置索引数据的存储路径，默认是es根目录下的data文件夹，可以设置多个存储路径，用逗号隔开。
path.logs:
    设置日志文件的存储路径，默认是es根目录下的logs文件夹
path.plugins:
    设置插件的存放路径，默认是es根目录下的plugins文件夹
bootstrap.memory_lock: true
    设置为true可以锁住ES使用的内存，避免内存与swap分区交换数据。
network.host:
    设置绑定主机的ip地址，设置为0.0.0.0表示绑定任何ip，允许外网访问，生产环境建议设置为具体的ip。
http.port: 9200
    设置对外服务的http端口，默认为9200。
transport.tcp.port: 9300    集群结点之间通信端口
node.master:
```

```
    指定该节点是否有资格被选举成为master结点，默认是true，如果原来的master宕机会重新选举新的master。
node.data:
    指定该节点是否存储索引数据，默认为true。
discovery.zen.ping.unicast.hosts: ["host1:port", "host2:port", "..."]
    设置集群中master节点的初始列表。
discovery.zen.ping.timeout: 3s
    设置ES自动发现节点连接超时的时间，默认为3秒，如果网络延迟高可设置大些。
discovery.zen.minimum_master_nodes:
    主结点数量的最少值 ,此值的公式为：(master_eligible_nodes / 2) + 1 ，比如：有3个符合要求的主结点，那么这里要设置为2。
node.max_local_storage_nodes:
    单机允许的最大存储结点数，通常单机启动一个结点建议设置为1，开发环境如果单机启动多个节点可设置大于1。
```

jvm.options

设置最小及最大的JVM堆内存大小：

在jvm.options中设置 -Xms和-Xmx：

1） 两个值设置为相等

2） 将Xmx 设置为不超过物理内存的一半。

log4j2.properties

日志文件设置，ES使用log4j，注意日志级别的配置。

ES8默认开启了 ssl 认证

修改elasticsearch.yml配置文件：

xpack.security.http.ssl:enabled 设置成 false

xpack.security.enabled 设置成false

启动

进入bin目录，双击elasticsearch.bat

```
PS C:\Users\mao\Desktop> elasticsearch
warning: ignoring JAVA_HOME=C:\Users\mao\.jdks\openjdk-16.0.2; using bundled JDK
warning: ignoring JAVA_HOME=C:\Users\mao\.jdks\openjdk-16.0.2; using ES_JAVA_HOME
[2022-05-24T21:35:52,947][INFO ][o.e.n.Node               ] [MAO] version[8.1.3], pid[16952], build[default/zip/39afaa3c0fe7db4869a161985e240bd7182d7a07/2022-04-19T08:13:25.444693396Z], OS[Windows 10/10.0/amd64], JVM[Eclipse Adoptium/OpenJDK 64-Bit Server VM/18/18+36]
[2022-05-24T21:35:52,952][INFO ][o.e.n.Node               ] [MAO] JVM home [C:\Program Files\elasticsearch-8.1.3\jdk], using bundled JDK [true]
```

```
[2022-05-24T21:35:52,953][INFO ][o.e.n.Node               ] [MAO] JVM arguments
[-Des.networkaddress.cache.ttl=60, -Des.networkaddress.cache.negative.ttl=10, -
Djava.security.manager=allow, -XX:+AlwaysPreTouch, -Xss1m, -
Djava.awt.headless=true, -Dfile.encoding=UTF-8, -Djna.nosys=true, -XX:-
OmitStackTraceInFastThrow, -XX:+ShowCodeDetailsInExceptionMessages, -
Dio.netty.noUnsafe=true, -Dio.netty.noKeySetOptimization=true, -
Dio.netty.recycler.maxCapacityPerThread=0, -
Dio.netty.allocator.numDirectArenas=0, -Dlog4j.shutdownHookEnabled=false, -
Dlog4j2.disable.jmx=true, -Dlog4j2.formatMsgNoLookups=true, -
Djava.locale.providers=SPI,COMPAT, --add-opens=java.base/java.io=ALL-UNNAMED, -
Xms1g, -Xmx8g, -XX:+UseG1GC, -
Djava.io.tmpdir=C:\Users\mao\AppData\Local\Temp\elasticsearch, -
XX:+HeapDumpOnOutOfMemoryError, -XX:+ExitOnOutOfMemoryError, -
XX:HeapDumpPath=data, -XX:ErrorFile=logs/hs_err_pid%p.log, -
Xlog:gc*,gc+age=trace,safepoint:file=logs/gc.log:utctime,pid,tags:filecount=32,f
ilesize=64m, -XX:MaxDirectMemorySize=4294967296, -
XX:InitiatingHeapOccupancyPercent=30, -XX:G1ReservePercent=25, -Delasticsearch,
-Des.path.home=C:\Program Files\elasticsearch-8.1.3, -Des.path.conf=C:\Program
Files\elasticsearch-8.1.3\config, -Des.distribution.flavor=default, -
Des.distribution.type=zip, -Des.bundled_jdk=true]
[2022-05-24T21:35:56,022][INFO ][o.e.p.PluginsService     ] [MAO] loaded module
[aggs-matrix-stats]
[2022-05-24T21:35:56,023][INFO ][o.e.p.PluginsService     ] [MAO] loaded module
[analysis-common]
[2022-05-24T21:35:56,023][INFO ][o.e.p.PluginsService     ] [MAO] loaded module
[constant-keyword]
[2022-05-24T21:35:56,023][INFO ][o.e.p.PluginsService     ] [MAO] loaded module
[data-streams]
[2022-05-24T21:35:56,023][INFO ][o.e.p.PluginsService     ] [MAO] loaded module
[frozen-indices]
[2022-05-24T21:35:56,024][INFO ][o.e.p.PluginsService     ] [MAO] loaded module
[ingest-common]
[2022-05-24T21:35:56,024][INFO ][o.e.p.PluginsService     ] [MAO] loaded module
[ingest-geoip]
[2022-05-24T21:35:56,024][INFO ][o.e.p.PluginsService     ] [MAO] loaded module
[ingest-user-agent]
[2022-05-24T21:35:56,024][INFO ][o.e.p.PluginsService     ] [MAO] loaded module
[kibana]
[2022-05-24T21:35:56,024][INFO ][o.e.p.PluginsService     ] [MAO] loaded module
[lang-expression]
[2022-05-24T21:35:56,025][INFO ][o.e.p.PluginsService     ] [MAO] loaded module
[lang-mustache]
[2022-05-24T21:35:56,025][INFO ][o.e.p.PluginsService     ] [MAO] loaded module
[lang-painless]
[2022-05-24T21:35:56,025][INFO ][o.e.p.PluginsService     ] [MAO] loaded module
[legacy-geo]
[2022-05-24T21:35:56,025][INFO ][o.e.p.PluginsService     ] [MAO] loaded module
[mapper-extras]
[2022-05-24T21:35:56,026][INFO ][o.e.p.PluginsService     ] [MAO] loaded module
[mapper-version]
[2022-05-24T21:35:56,026][INFO ][o.e.p.PluginsService     ] [MAO] loaded module
[old-lucene-versions]
[2022-05-24T21:35:56,026][INFO ][o.e.p.PluginsService     ] [MAO] loaded module
[parent-join]
[2022-05-24T21:35:56,026][INFO ][o.e.p.PluginsService     ] [MAO] loaded module
[percolator]
[2022-05-24T21:35:56,027][INFO ][o.e.p.PluginsService     ] [MAO] loaded module
[rank-eval]
```

```
[2022-05-24T21:35:56,027][INFO ][o.e.p.PluginsService     ] [MAO] loaded module
[reindex]
[2022-05-24T21:35:56,027][INFO ][o.e.p.PluginsService     ] [MAO] loaded module
[repositories-metering-api]
[2022-05-24T21:35:56,027][INFO ][o.e.p.PluginsService     ] [MAO] loaded module
[repository-azure]
[2022-05-24T21:35:56,027][INFO ][o.e.p.PluginsService     ] [MAO] loaded module
[repository-encrypted]
[2022-05-24T21:35:56,028][INFO ][o.e.p.PluginsService     ] [MAO] loaded module
[repository-gcs]
[2022-05-24T21:35:56,028][INFO ][o.e.p.PluginsService     ] [MAO] loaded module
[repository-s3]
[2022-05-24T21:35:56,028][INFO ][o.e.p.PluginsService     ] [MAO] loaded module
[repository-url]
[2022-05-24T21:35:56,028][INFO ][o.e.p.PluginsService     ] [MAO] loaded module
[runtime-fields-common]
[2022-05-24T21:35:56,028][INFO ][o.e.p.PluginsService     ] [MAO] loaded module
[search-business-rules]
[2022-05-24T21:35:56,029][INFO ][o.e.p.PluginsService     ] [MAO] loaded module
[searchable-snapshots]
[2022-05-24T21:35:56,029][INFO ][o.e.p.PluginsService     ] [MAO] loaded module
[snapshot-based-recoveries]
[2022-05-24T21:35:56,029][INFO ][o.e.p.PluginsService     ] [MAO] loaded module
[snapshot-repo-test-kit]
[2022-05-24T21:35:56,029][INFO ][o.e.p.PluginsService     ] [MAO] loaded module
[spatial]
[2022-05-24T21:35:56,030][INFO ][o.e.p.PluginsService     ] [MAO] loaded module
[transform]
[2022-05-24T21:35:56,030][INFO ][o.e.p.PluginsService     ] [MAO] loaded module
[transport-netty4]
[2022-05-24T21:35:56,030][INFO ][o.e.p.PluginsService     ] [MAO] loaded module
[unsigned-long]
[2022-05-24T21:35:56,030][INFO ][o.e.p.PluginsService     ] [MAO] loaded module
[vector-tile]
[2022-05-24T21:35:56,031][INFO ][o.e.p.PluginsService     ] [MAO] loaded module
[vectors]
[2022-05-24T21:35:56,031][INFO ][o.e.p.PluginsService     ] [MAO] loaded module
[wildcard]
[2022-05-24T21:35:56,031][INFO ][o.e.p.PluginsService     ] [MAO] loaded module
[x-pack-aggregate-metric]
[2022-05-24T21:35:56,032][INFO ][o.e.p.PluginsService     ] [MAO] loaded module
[x-pack-analytics]
[2022-05-24T21:35:56,033][INFO ][o.e.p.PluginsService     ] [MAO] loaded module
[x-pack-async]
[2022-05-24T21:35:56,033][INFO ][o.e.p.PluginsService     ] [MAO] loaded module
[x-pack-async-search]
[2022-05-24T21:35:56,033][INFO ][o.e.p.PluginsService     ] [MAO] loaded module
[x-pack-autoscaling]
[2022-05-24T21:35:56,034][INFO ][o.e.p.PluginsService     ] [MAO] loaded module
[x-pack-ccr]
[2022-05-24T21:35:56,034][INFO ][o.e.p.PluginsService     ] [MAO] loaded module
[x-pack-core]
[2022-05-24T21:35:56,035][INFO ][o.e.p.PluginsService     ] [MAO] loaded module
[x-pack-deprecation]
[2022-05-24T21:35:56,035][INFO ][o.e.p.PluginsService     ] [MAO] loaded module
[x-pack-enrich]
[2022-05-24T21:35:56,035][INFO ][o.e.p.PluginsService     ] [MAO] loaded module
[x-pack-eql]
```

```
[2022-05-24T21:35:56,036][INFO ][o.e.p.PluginsService     ] [MAO] loaded module
[x-pack-fleet]
[2022-05-24T21:35:56,036][INFO ][o.e.p.PluginsService     ] [MAO] loaded module
[x-pack-graph]
[2022-05-24T21:35:56,037][INFO ][o.e.p.PluginsService     ] [MAO] loaded module
[x-pack-identity-provider]
[2022-05-24T21:35:56,041][INFO ][o.e.p.PluginsService     ] [MAO] loaded module
[x-pack-ilm]
[2022-05-24T21:35:56,041][INFO ][o.e.p.PluginsService     ] [MAO] loaded module
[x-pack-logstash]
[2022-05-24T21:35:56,041][INFO ][o.e.p.PluginsService     ] [MAO] loaded module
[x-pack-ml]
[2022-05-24T21:35:56,042][INFO ][o.e.p.PluginsService     ] [MAO] loaded module
[x-pack-monitoring]
[2022-05-24T21:35:56,042][INFO ][o.e.p.PluginsService     ] [MAO] loaded module
[x-pack-ql]
[2022-05-24T21:35:56,043][INFO ][o.e.p.PluginsService     ] [MAO] loaded module
[x-pack-rollup]
[2022-05-24T21:35:56,043][INFO ][o.e.p.PluginsService     ] [MAO] loaded module
[x-pack-security]
[2022-05-24T21:35:56,044][INFO ][o.e.p.PluginsService     ] [MAO] loaded module
[x-pack-shutdown]
[2022-05-24T21:35:56,044][INFO ][o.e.p.PluginsService     ] [MAO] loaded module
[x-pack-sql]
[2022-05-24T21:35:56,044][INFO ][o.e.p.PluginsService     ] [MAO] loaded module
[x-pack-stack]
[2022-05-24T21:35:56,045][INFO ][o.e.p.PluginsService     ] [MAO] loaded module
[x-pack-text-structure]
[2022-05-24T21:35:56,045][INFO ][o.e.p.PluginsService     ] [MAO] loaded module
[x-pack-voting-only-node]
[2022-05-24T21:35:56,046][INFO ][o.e.p.PluginsService     ] [MAO] loaded module
[x-pack-watcher]
[2022-05-24T21:35:56,047][INFO ][o.e.p.PluginsService     ] [MAO] no plugins
loaded
[2022-05-24T21:35:56,636][INFO ][o.e.e.NodeEnvironment    ] [MAO] using [1] data
paths, mounts [[(C:)]], net usable_space [74.5gb], net total_space [237.8gb],
types [NTFS]
[2022-05-24T21:35:56,637][INFO ][o.e.e.NodeEnvironment    ] [MAO] heap size
[8gb], compressed ordinary object pointers [true]
[2022-05-24T21:35:56,739][INFO ][o.e.n.Node               ] [MAO] node name
[MAO], node ID [QrII3Cg2Sh-RnBrmVzmV9Q], cluster name [elasticsearch], roles
[data_hot, transform, data_content, data_warm, master, remote_cluster_client,
data, data_cold, ingest, data_frozen, ml]
[2022-05-24T21:36:00,682][INFO ][o.e.x.m.p.l.CppLogMessageHandler] [MAO]
[controller/1664] [Main.cc@123] controller (64 bit): Version 8.1.3 (Build
92d8267e6ebfb7) Copyright (c) 2022 Elasticsearch BV
[2022-05-24T21:36:00,966][INFO ][o.e.x.s.Security         ] [MAO] Security is
enabled
[2022-05-24T21:36:01,316][INFO ][o.e.x.s.a.Realms         ] [MAO] license mode is
[trial], currently licensed security realms are
[reserved/reserved,file/default_file,native/default_native]
[2022-05-24T21:36:01,328][INFO ][o.e.x.s.a.s.FileRolesStore] [MAO] parsed [0]
roles from file [C:\Program Files\elasticsearch-8.1.3\config\roles.yml]

[2022-05-24T21:36:01,896][INFO ][o.e.i.g.DatabaseNodeService] [MAO] deleting
stale file [C:\Users\mao\AppData\Local\Temp\elasticsearch\geoip-
databases\QrII3Cg2Sh-RnBrmVzmV9Q\GeoLite2-ASN.mmdb]
```

```
[2022-05-24T21:36:01,897][INFO ][o.e.i.g.DatabaseNodeService] [MAO] deleting
stale file [C:\Users\mao\AppData\Local\Temp\elasticsearch\geoip-
databases\QrII3Cg2Sh-RnBrmVzmV9Q\GeoLite2-ASN.mmdb_COPYRIGHT.txt]
[2022-05-24T21:36:01,898][INFO ][o.e.i.g.DatabaseNodeService] [MAO] deleting
stale file [C:\Users\mao\AppData\Local\Temp\elasticsearch\geoip-
databases\QrII3Cg2Sh-RnBrmVzmV9Q\GeoLite2-ASN.mmdb_elastic-geoip-database-
service-agreement-LICENSE.txt]
[2022-05-24T21:36:01,899][INFO ][o.e.i.g.DatabaseNodeService] [MAO] deleting
stale file [C:\Users\mao\AppData\Local\Temp\elasticsearch\geoip-
databases\QrII3Cg2Sh-RnBrmVzmV9Q\GeoLite2-ASN.mmdb_LICENSE.txt]
[2022-05-24T21:36:01,899][INFO ][o.e.i.g.DatabaseNodeService] [MAO] deleting
stale file [C:\Users\mao\AppData\Local\Temp\elasticsearch\geoip-
databases\QrII3Cg2Sh-RnBrmVzmV9Q\GeoLite2-City.mmdb]
[2022-05-24T21:36:01,905][INFO ][o.e.i.g.DatabaseNodeService] [MAO] deleting
stale file [C:\Users\mao\AppData\Local\Temp\elasticsearch\geoip-
databases\QrII3Cg2Sh-RnBrmVzmV9Q\GeoLite2-City.mmdb_COPYRIGHT.txt]
[2022-05-24T21:36:01,906][INFO ][o.e.i.g.DatabaseNodeService] [MAO] deleting
stale file [C:\Users\mao\AppData\Local\Temp\elasticsearch\geoip-
databases\QrII3Cg2Sh-RnBrmVzmV9Q\GeoLite2-City.mmdb_elastic-geoip-database-
service-agreement-LICENSE.txt]
[2022-05-24T21:36:01,907][INFO ][o.e.i.g.DatabaseNodeService] [MAO] deleting
stale file [C:\Users\mao\AppData\Local\Temp\elasticsearch\geoip-
databases\QrII3Cg2Sh-RnBrmVzmV9Q\GeoLite2-City.mmdb_LICENSE.txt]
[2022-05-24T21:36:01,908][INFO ][o.e.i.g.DatabaseNodeService] [MAO] deleting
stale file [C:\Users\mao\AppData\Local\Temp\elasticsearch\geoip-
databases\QrII3Cg2Sh-RnBrmVzmV9Q\GeoLite2-City.mmdb_README.txt]
[2022-05-24T21:36:01,908][INFO ][o.e.i.g.DatabaseNodeService] [MAO] deleting
stale file [C:\Users\mao\AppData\Local\Temp\elasticsearch\geoip-
databases\QrII3Cg2Sh-RnBrmVzmV9Q\GeoLite2-Country.mmdb]
[2022-05-24T21:36:01,909][INFO ][o.e.i.g.DatabaseNodeService] [MAO] deleting
stale file [C:\Users\mao\AppData\Local\Temp\elasticsearch\geoip-
databases\QrII3Cg2Sh-RnBrmVzmV9Q\GeoLite2-Country.mmdb_COPYRIGHT.txt]
[2022-05-24T21:36:01,910][INFO ][o.e.i.g.DatabaseNodeService] [MAO] deleting
stale file [C:\Users\mao\AppData\Local\Temp\elasticsearch\geoip-
databases\QrII3Cg2Sh-RnBrmVzmV9Q\GeoLite2-Country.mmdb_elastic-geoip-database-
service-agreement-LICENSE.txt]
[2022-05-24T21:36:01,911][INFO ][o.e.i.g.DatabaseNodeService] [MAO] deleting
stale file [C:\Users\mao\AppData\Local\Temp\elasticsearch\geoip-
databases\QrII3Cg2Sh-RnBrmVzmV9Q\GeoLite2-Country.mmdb_LICENSE.txt]
[2022-05-24T21:36:02,472][INFO ][o.e.t.n.NettyAllocator   ] [MAO] creating
NettyAllocator with the following configs: [name=elasticsearch_configured,
chunk_size=1mb, suggested_max_allocation_size=1mb, factors=
{es.unsafe.use_netty_default_chunk_and_page_size=false, g1gc_enabled=true,
g1gc_region_size=4mb}]
[2022-05-24T21:36:02,499][INFO ][o.e.i.r.RecoverySettings ] [MAO] using rate
limit [40mb] with [default=40mb, read=0b, write=0b, max=0b]
[2022-05-24T21:36:02,529][INFO ][o.e.d.DiscoveryModule    ] [MAO] using discovery
type [multi-node] and seed hosts providers [settings]
[2022-05-24T21:36:03,571][INFO ][o.e.n.Node               ] [MAO] initialized
[2022-05-24T21:36:03,572][INFO ][o.e.n.Node               ] [MAO] starting ...
[2022-05-24T21:36:03,621][INFO ][o.e.x.s.c.f.PersistentCache] [MAO] persistent
cache index loaded
[2022-05-24T21:36:03,622][INFO ][o.e.x.d.l.DeprecationIndexingComponent] [MAO]
deprecation component started
[2022-05-24T21:36:03,812][INFO ][o.e.t.TransportService   ] [MAO] publish_address
{127.0.0.1:9300}, bound_addresses {127.0.0.1:9300}, {[::1]:9300}
```

```
[2022-05-24T21:36:04,207][WARN ][o.e.b.BootstrapChecks    ] [MAO] initial heap
size [1073741824] not equal to maximum heap size [8589934592]; this can cause
resize pauses
[2022-05-24T21:36:04,209][INFO ][o.e.c.c.Coordinator      ] [MAO] cluster UUID
[9R1mejGTSbCX20057B9IdQ]
[2022-05-24T21:36:04,296][INFO ][o.e.c.s.MasterService    ] [MAO] elected-as-
master ([1] nodes joined)[{MAO}{QrII3Cg2Sh-RnBrmVzmV9Q}{xKQM768NSTSeKF_YYHTyJw}
{127.0.0.1}{127.0.0.1:9300}{cdfhilmrstw} completing election,
_BECOME_MASTER_TASK_, _FINISH_ELECTION_], term: 18, version: 365, delta: master
node changed {previous [], current [{MAO}{QrII3Cg2Sh-RnBrmVzmV9Q}
{xKQM768NSTSeKF_YYHTyJw}{127.0.0.1}{127.0.0.1:9300}{cdfhilmrstw}]}
[2022-05-24T21:36:04,381][INFO ][o.e.c.s.ClusterApplierService] [MAO] master node
 changed {previous [], current [{MAO}{QrII3Cg2Sh-RnBrmVzmV9Q}
{xKQM768NSTSeKF_YYHTyJw}{127.0.0.1}{127.0.0.1:9300}{cdfhilmrstw}]}, term: 18,
version: 365, reason: Publication{term=18, version=365}
[2022-05-24T21:36:04,509][INFO ][o.e.h.AbstractHttpServerTransport] [MAO]
publish_address {172.18.128.1:9200}, bound_addresses {127.0.0.1:9200},
{[::1]:9200}, {172.18.128.1:9200}, {172.23.16.1:9200}, {192.168.202.1:9200},
{192.168.73.1:9200}, {172.28.0.1:9200}, {172.30.64.1:9200}, {172.30.176.1:9200}
[2022-05-24T21:36:04,509][INFO ][o.e.n.Node               ] [MAO] started
[2022-05-24T21:36:04,736][INFO ][o.e.l.LicenseService     ] [MAO] license
[ffaa99ef-66ed-4580-95bf-7a6049a44720] mode [basic] - valid
[2022-05-24T21:36:04,738][INFO ][o.e.x.s.a.Realms         ] [MAO] license mode is
[basic], currently licensed security realms are
[reserved/reserved,file/default_file,native/default_native]
[2022-05-24T21:36:04,745][INFO ][o.e.g.GatewayService     ] [MAO] recovered [10]
indices into cluster_state
[2022-05-24T21:36:05,552][INFO ][o.e.i.g.DatabaseNodeService] [MAO] retrieve
geoip database [GeoLite2-ASN.mmdb] from [.geoip_databases] to
[C:\Users\mao\AppData\Local\Temp\elasticsearch\geoip-databases\QrII3Cg2Sh-
RnBrmVzmV9Q\GeoLite2-ASN.mmdb.tmp.gz]
[2022-05-24T21:36:05,554][INFO ][o.e.i.g.DatabaseNodeService] [MAO] retrieve
geoip database [GeoLite2-Country.mmdb] from [.geoip_databases] to
[C:\Users\mao\AppData\Local\Temp\elasticsearch\geoip-databases\QrII3Cg2Sh-
RnBrmVzmV9Q\GeoLite2-Country.mmdb.tmp.gz]
[2022-05-24T21:36:05,556][INFO ][o.e.i.g.DatabaseNodeService] [MAO] retrieve
geoip database [GeoLite2-City.mmdb] from [.geoip_databases] to
[C:\Users\mao\AppData\Local\Temp\elasticsearch\geoip-databases\QrII3Cg2Sh-
RnBrmVzmV9Q\GeoLite2-City.mmdb.tmp.gz]
[2022-05-24T21:36:05,718][INFO ][o.e.c.r.a.AllocationService] [MAO]
current.health="GREEN" message="Cluster health status changed from [RED] to
[GREEN] (reason: [shards started [[.kibana-event-log-8.1.3-000001][0]]])."
previous.health="RED" reason="shards started [[.kibana-event-log-8.1.3-000001]
[0]]"
[2022-05-24T21:36:05,746][INFO ][o.e.i.g.DatabaseNodeService] [MAO] successfully
loaded geoip database file [GeoLite2-Country.mmdb]
[2022-05-24T21:36:05,806][INFO ][o.e.i.g.DatabaseNodeService] [MAO] successfully
loaded geoip database file [GeoLite2-ASN.mmdb]
[2022-05-24T21:36:06,530][INFO ][o.e.i.g.DatabaseNodeService] [MAO] successfully
loaded geoip database file [GeoLite2-City.mmdb]
[2022-05-24T21:36:17,573][INFO ][o.e.i.g.DatabaseNodeService] [MAO] retrieve
geoip database [GeoLite2-ASN.mmdb] from [.geoip_databases] to
[C:\Users\mao\AppData\Local\Temp\elasticsearch\geoip-databases\QrII3Cg2Sh-
RnBrmVzmV9Q\GeoLite2-ASN.mmdb.tmp.gz]
[2022-05-24T21:36:28,200][INFO ][o.e.i.g.GeoIpDownloader  ] [MAO] successfully
downloaded geoip database [GeoLite2-ASN.mmdb]
```

```
[2022-05-24T21:36:28,296][INFO ][o.e.i.g.DatabaseReaderLazyLoader] [MAO] evicted
[0] entries from cache after reloading database
[C:\Users\mao\AppData\Local\Temp\elasticsearch\geoip-databases\QrII3Cg2Sh-
RnBrmVzmV9Q\GeoLite2-ASN.mmdb]
[2022-05-24T21:36:28,297][INFO ][o.e.i.g.DatabaseNodeService] [MAO] successfully
loaded geoip database file [GeoLite2-ASN.mmdb]
```

检查ES是否启动成功

浏览器访问 http://localhost:9200/

结果：

```
{
  "name" : "MAO",
  "cluster_name" : "elasticsearch",
  "cluster_uuid" : "9R1mejGTSbCX2OO57B9IdQ",
  "version" : {
    "number" : "8.1.3",
    "build_flavor" : "default",
    "build_type" : "zip",
    "build_hash" : "39afaa3c0fe7db4869a161985e240bd7182d7a07",
    "build_date" : "2022-04-19T08:13:25.444693396Z",
    "build_snapshot" : false,
    "lucene_version" : "9.0.0",
    "minimum_wire_compatibility_version" : "7.17.0",
    "minimum_index_compatibility_version" : "7.0.0"
  },
  "tagline" : "You Know, for Search"
}
```

- name: node名称，取自机器的hostname
- cluster_name: 集群名称（默认的集群名称就是elasticsearch）
- version.number: 7.3.0，es版本号
- version.lucene_version:封装的lucene版本号

查询集群状态

浏览器访问 http://localhost:9200/_cluster/health

结果：

```
{
    "cluster_name": "elasticsearch",
    "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,
```

```
    "delayed_unassigned_shards": 0,
    "number_of_pending_tasks": 0,
    "number_of_in_flight_fetch": 0,
    "task_max_waiting_in_queue_millis": 0,
    "active_shards_percent_as_number": 100
}
```

Status：集群状态。Green 所有分片可用。Yellow所有主分片可用。Red主分片不可用，集群不可用。

Kibana

kibana是es数据的前端展现，数据分析时，可以方便地看到数据。作为开发人员，可以方便访问es。

启动Kibana：bin\kibana.bat

结果：

```
PS C:\Users\mao\Desktop> kibana
[2022-05-24T22:07:56.816+08:00][INFO ][plugins-service] Plugin "metricsEntities"
is disabled.
[2022-05-24T22:07:56.883+08:00][INFO ][http.server.Preboot] http server running
at http://localhost:5601
[2022-05-24T22:07:56.921+08:00][INFO ][plugins-system.preboot] Setting up [1]
plugins: [interactiveSetup]
[2022-05-24T22:07:56.933+08:00][INFO ][preboot] "interactiveSetup" plugin is
holding setup: Validating Elasticsearch connection configuration…
[2022-05-24T22:07:56.964+08:00][INFO ][root] Holding setup until preboot stage
is completed.
[2022-05-24T22:07:56.995+08:00][WARN ][config.deprecation] The default mechanism
for Reporting privileges will work differently in future versions, which will
affect the behavior of this cluster. Set "xpack.reporting.roles.enabled" to
"false" to adopt the future behavior before upgrading.
[2022-05-24T22:07:57.160+08:00][INFO ][plugins-system.standard] Setting up [112]
plugins:
[translations,licensing,globalSearch,globalSearchProviders,features,mapsEms,lice
nseApiGuard,usageCollection,taskManager,telemetryCollectionManager,telemetryColl
ectionXpack,kibanaUsageCollection,sharedUX,share,embeddable,uiActionsEnhanced,sc
reenshotMode,screenshotting,banners,telemetry,newsfeed,fieldFormats,expressions,
dataViews,charts,esUiShared,bfetch,data,savedObjects,presentationUtil,expression
Shape,expressionRevealImage,expressionRepeatImage,expressionMetric,expressionIma
ge,customIntegrations,home,searchprofiler,painlessLab,grokdebugger,management,wa
tcher,licenseManagement,advancedSettings,spaces,security,savedObjectsTagging,rep
orting,lists,fileUpload,ingestPipelines,encryptedSavedObjects,dataEnhanced,cloud
,snapshotRestore,eventLog,actions,alerting,triggersActionsUi,transform,stackAler
ts,ruleRegistry,savedObjectsManagement,console,controls,graph,fleet,indexManagem
ent,remoteClusters,crossClusterReplication,indexLifecycleManagement,visualizatio
ns,canvas,visTypeXy,visTypeVislib,visTypeVega,visTypeTimeseries,rollup,visTypeTi
melion,visTypeTagcloud,visTypeTable,visTypeMetric,visTypeHeatmap,visTypeMarkdown
,dashboard,maps,dashboardEnhanced,expressionTagcloud,expressionPie,visTypePie,ex
pressionMetricVis,expressionHeatmap,expressionGauge,dataViewFieldEditor,lens,cas
es,timelines,discover,osquery,observability,discoverEnhanced,dataVisualizer,ml,u
ptime,securitySolution,infra,upgradeAssistant,monitoring,logstash,enterpriseSear
ch,apm,dataViewManagement]
[2022-05-24T22:07:57.177+08:00][INFO ][plugins.taskManager] TaskManager is
identified by the Kibana UUID: 25e0275f-cbbb-424d-86e9-06058cb8dfb3
```

```
[2022-05-24T22:07:57.317+08:00][WARN ][plugins.security.config] Generating a
random key for xpack.security.encryptionKey. To prevent sessions from being
invalidated on restart, please set xpack.security.encryptionKey in the
kibana.yml or use the bin/kibana-encryption-keys command.
[2022-05-24T22:07:57.318+08:00][WARN ][plugins.security.config] Session cookies
will be transmitted over insecure connections. This is not recommended.
[2022-05-24T22:07:57.337+08:00][WARN ][plugins.security.config] Generating a
random key for xpack.security.encryptionKey. To prevent sessions from being
invalidated on restart, please set xpack.security.encryptionKey in the
kibana.yml or use the bin/kibana-encryption-keys command.
[2022-05-24T22:07:57.338+08:00][WARN ][plugins.security.config] Session cookies
will be transmitted over insecure connections. This is not recommended.
[2022-05-24T22:07:57.357+08:00][WARN ][plugins.reporting.config] Generating a
random key for xpack.reporting.encryptionKey. To prevent sessions from being
invalidated on restart, please set xpack.reporting.encryptionKey in the
kibana.yml or use the bin/kibana-encryption-keys command.
[2022-05-24T22:07:57.367+08:00][WARN ][plugins.encryptedSavedObjects] Saved
objects encryption key is not set. This will severely limit Kibana
functionality. Please set xpack.encryptedSavedObjects.encryptionKey in the
kibana.yml or use the bin/kibana-encryption-keys command.
[2022-05-24T22:07:57.381+08:00][WARN ][plugins.actions] APIs are disabled
because the Encrypted Saved Objects plugin is missing encryption key. Please set
xpack.encryptedSavedObjects.encryptionKey in the kibana.yml or use the
bin/kibana-encryption-keys command.
[2022-05-24T22:07:57.396+08:00][WARN ][plugins.alerting] APIs are disabled
because the Encrypted Saved Objects plugin is missing encryption key. Please set
xpack.encryptedSavedObjects.encryptionKey in the kibana.yml or use the
bin/kibana-encryption-keys command.
[2022-05-24T22:07:57.419+08:00][INFO ][plugins.ruleRegistry] Installing common
resources shared between all indices
[2022-05-24T22:07:58.081+08:00][INFO ][plugins.screenshotting.config] Chromium
sandbox provides an additional layer of protection, and is supported for Win32
OS. Automatically enabling Chromium sandbox.
[2022-05-24T22:07:58.995+08:00][INFO ][savedobjects-service] Waiting until all
Elasticsearch nodes are compatible with Kibana before starting saved objects
migrations...
[2022-05-24T22:07:58.996+08:00][INFO ][savedobjects-service] Starting saved
objects migrations
[2022-05-24T22:07:59.377+08:00][INFO ][savedobjects-service] [.kibana] INIT ->
OUTDATED_DOCUMENTS_SEARCH_OPEN_PIT. took: 283ms.
[2022-05-24T22:07:59.439+08:00][INFO ][savedobjects-service]
[.kibana_task_manager] INIT -> OUTDATED_DOCUMENTS_SEARCH_OPEN_PIT. took: 343ms.
[2022-05-24T22:07:59.468+08:00][INFO ][savedobjects-service] [.kibana]
OUTDATED_DOCUMENTS_SEARCH_OPEN_PIT -> OUTDATED_DOCUMENTS_SEARCH_READ. took:
91ms.
[2022-05-24T22:07:59.479+08:00][INFO ][savedobjects-service]
[.kibana_task_manager] OUTDATED_DOCUMENTS_SEARCH_OPEN_PIT ->
OUTDATED_DOCUMENTS_SEARCH_READ. took: 40ms.
[2022-05-24T22:07:59.494+08:00][INFO ][plugins.screenshotting.chromium] Browser
executable: H:\opensoft\kibana-8.1.3\x-
pack\plugins\screenshotting\chromium\chrome-win\chrome.exe
[2022-05-24T22:07:59.521+08:00][INFO ][savedobjects-service]
[.kibana_task_manager] OUTDATED_DOCUMENTS_SEARCH_READ ->
OUTDATED_DOCUMENTS_SEARCH_CLOSE_PIT. took: 42ms.
[2022-05-24T22:07:59.524+08:00][INFO ][savedobjects-service] [.kibana]
OUTDATED_DOCUMENTS_SEARCH_READ -> OUTDATED_DOCUMENTS_SEARCH_CLOSE_PIT. took:
56ms.
```

```
[2022-05-24T22:07:59.529+08:00][INFO ][savedobjects-service]
[.kibana_task_manager] OUTDATED_DOCUMENTS_SEARCH_CLOSE_PIT ->
UPDATE_TARGET_MAPPINGS. took: 8ms.
[2022-05-24T22:07:59.532+08:00][INFO ][savedobjects-service] [.kibana]
OUTDATED_DOCUMENTS_SEARCH_CLOSE_PIT -> UPDATE_TARGET_MAPPINGS. took: 8ms.
[2022-05-24T22:07:59.576+08:00][INFO ][savedobjects-service]
[.kibana_task_manager] UPDATE_TARGET_MAPPINGS ->
UPDATE_TARGET_MAPPINGS_WAIT_FOR_TASK. took: 47ms.
[2022-05-24T22:07:59.623+08:00][INFO ][savedobjects-service] [.kibana]
UPDATE_TARGET_MAPPINGS -> UPDATE_TARGET_MAPPINGS_WAIT_FOR_TASK. took: 91ms.
[2022-05-24T22:07:59.867+08:00][INFO ][savedobjects-service] [.kibana]
UPDATE_TARGET_MAPPINGS_WAIT_FOR_TASK -> DONE. took: 244ms.
[2022-05-24T22:07:59.867+08:00][INFO ][savedobjects-service] [.kibana] Migration
completed after 773ms
[2022-05-24T22:07:59.869+08:00][INFO ][savedobjects-service]
[.kibana_task_manager] UPDATE_TARGET_MAPPINGS_WAIT_FOR_TASK -> DONE. took:
293ms.
[2022-05-24T22:07:59.870+08:00][INFO ][savedobjects-service]
[.kibana_task_manager] Migration completed after 774ms
[2022-05-24T22:08:00.097+08:00][INFO ][plugins-system.preboot] Stopping all
plugins.
[2022-05-24T22:08:00.099+08:00][INFO ][plugins-system.standard] Starting [112]
plugins:
[translations,licensing,globalSearch,globalSearchProviders,features,mapsEms,lice
nseApiGuard,usageCollection,taskManager,telemetryCollectionManager,telemetryColl
ectionXpack,kibanaUsageCollection,sharedUX,share,embeddable,uiActionsEnhanced,sc
reenshotMode,screenshotting,banners,telemetry,newsfeed,fieldFormats,expressions,
dataViews,charts,esUiShared,bfetch,data,savedObjects,presentationUtil,expression
Shape,expressionRevealImage,expressionRepeatImage,expressionMetric,expressionIma
ge,customIntegrations,home,searchprofiler,painlessLab,grokdebugger,management,wa
tcher,licenseManagement,advancedSettings,spaces,security,savedObjectsTagging,rep
orting,lists,fileUpload,ingestPipelines,encryptedSavedObjects,dataEnhanced,cloud
,snapshotRestore,eventLog,actions,alerting,triggersActionsUi,transform,stackAler
ts,ruleRegistry,savedObjectsManagement,console,controls,graph,fleet,indexManagem
ent,remoteClusters,crossClusterReplication,indexLifecycleManagement,visualizatio
ns,canvas,visTypeXy,visTypeVislib,visTypeVega,visTypeTimeseries,rollup,visTypeTi
melion,visTypeTagcloud,visTypeTable,visTypeMetric,visTypeHeatmap,visTypeMarkdown
,dashboard,maps,dashboardEnhanced,expressionTagcloud,expressionPie,visTypePie,ex
pressionMetricVis,expressionHeatmap,expressionGauge,dataViewFieldEditor,lens,cas
es,timelines,discover,osquery,observability,discoverEnhanced,dataVisualizer,ml,u
ptime,securitySolution,infra,upgradeAssistant,monitoring,logstash,enterpriseSear
ch,apm,dataViewManagement]
[2022-05-24T22:08:02.827+08:00][INFO ][plugins.fleet] Beginning fleet setup
[2022-05-24T22:08:02.850+08:00][INFO ][plugins.monitoring.monitoring] config
sourced from: production cluster
[2022-05-24T22:08:04.150+08:00][INFO ][http.server.Kibana] http server running
at http://localhost:5601
[2022-05-24T22:08:04.355+08:00][INFO ][plugins.monitoring.monitoring.kibana-
monitoring] Starting monitoring stats collection
[2022-05-24T22:08:04.445+08:00][INFO ][plugins.ruleRegistry] Installed common
resources shared between all indices
[2022-05-24T22:08:04.445+08:00][INFO ][plugins.ruleRegistry] Installing
resources for index .alerts-observability.uptime.alerts
[2022-05-24T22:08:04.447+08:00][INFO ][plugins.ruleRegistry] Installing
resources for index .alerts-security.alerts
[2022-05-24T22:08:04.448+08:00][INFO ][plugins.ruleRegistry] Installing
resources for index .preview.alerts-security.alerts
```

```
[2022-05-24T22:08:04.448+08:00][INFO ][plugins.ruleRegistry] Installing
resources for index .alerts-observability.logs.alerts
[2022-05-24T22:08:04.449+08:00][INFO ][plugins.ruleRegistry] Installing
resources for index .alerts-observability.metrics.alerts
[2022-05-24T22:08:04.450+08:00][INFO ][plugins.ruleRegistry] Installing
resources for index .alerts-observability.apm.alerts
[2022-05-24T22:08:04.546+08:00][INFO ][plugins.ruleRegistry] Installed resources
for index .alerts-observability.apm.alerts
[2022-05-24T22:08:04.547+08:00][INFO ][plugins.ruleRegistry] Installed resources
for index .alerts-observability.logs.alerts
[2022-05-24T22:08:04.548+08:00][INFO ][plugins.ruleRegistry] Installed resources
for index .alerts-observability.uptime.alerts
[2022-05-24T22:08:04.549+08:00][INFO ][plugins.ruleRegistry] Installed resources
for index .alerts-observability.metrics.alerts
[2022-05-24T22:08:04.550+08:00][INFO ][plugins.ruleRegistry] Installed resources
for index .alerts-security.alerts
[2022-05-24T22:08:04.607+08:00][INFO ][plugins.ruleRegistry] Installed resources
for index .preview.alerts-security.alerts
[2022-05-24T22:08:04.635+08:00][INFO ][plugins.fleet] Deleting preconfigured
output fleet-default-output
[2022-05-24T22:08:04.900+08:00][INFO ][plugins.fleet] Fleet setup completed
[2022-05-24T22:08:04.906+08:00][INFO ][plugins.securitySolution] Dependent
plugin setup complete - Starting ManifestTask
[2022-05-24T22:08:05.408+08:00][INFO ][status] Kibana is now degraded
[2022-05-24T22:08:07.461+08:00][INFO ]
[plugins.securitySolution.endpoint:metadata-check-transforms-task:0.0.1] no
endpoint installation found
[2022-05-24T22:08:11.101+08:00][INFO ][status] Kibana is now available (was
degraded)
```

浏览器访问 http://localhost:5601 进入Dev Tools界面

文档（document）的数据格式

（1）应用系统的数据结构都是面向对象的，具有复杂的数据结构

（2）对象存储到数据库，需要将关联的复杂对象属性插到另一张表，查询时再拼接起来。

（3）es面向文档，文档中存储的数据结构，与对象一致。所以一个对象可以直接存成一个文档。

（4）es的document用json数据格式来表达。

例如：班级和学生关系

```
public class Student
{
  private String id;
  private String name;
  private String classInfoId;
}

private class ClassInfo
{
  private String id;
  private String className;
}
```

数据库中要设计所谓的一对多，多对一的两张表，外键等。查询出来时，还要关联，mybatis写映射文件，很繁琐。

而在es中，一个学生存成文档如下：

```
{
    "id":"1",
    "name": "张三",
    "last_name": "zhang",
    "classInfo":
    {
        "id": "1",
        "className": "三年二班",
    }
}
```

简单的集群管理

快速检查集群的健康状况

es提供了一套api，叫做cat api，可以查看es中各种各样的数据

GET http://localhost:9200/_cat/health?v

结果：

```
epoch      timestamp cluster       status node.total node.data shards pri relo
init unassign pending_tasks max_task_wait_time active_shards_percent
1653402721 14:32:01  elasticsearch green           1         1     14  14    0
 0        0             0                  -                100.0%
```

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

yellow：每个索引的primary shard都是active状态的，但是部分replica shard不是active状态，处于不可用的状态

red：不是所有索引的primary shard都是active状态的，部分索引有数据丢失了

快速查看集群中有哪些索引

GET /_cat/indices?v

```
health status index                         uuid                   pri rep
docs.count docs.deleted store.size pri.store.size
green  open   kibana_sample_data_ecommerce DY7cNTGNSxOPPVU2Awqu5A   1   0
4675            0      4.1mb          4.1mb
green  open   kibana_sample_data_logs      jrrXk1ITQvW8lQDYmWB7wg   1   0
 14074            0      8.1mb          8.1mb
yellow open   demo_index                    1hX6S-rxTAmBCy_R9Rw8Xg   1   1
   0            0       225b           225b
```

简单的索引操作

创建索引：PUT /demo_index?pretty

```
{
  "acknowledged" : true,
  "shards_acknowledged" : true,
  "index" : "demo_index"
}
```

查看索引：GET /demo_index?pretty

```
{
    "demo_index": {
        "aliases": {},
        "mappings": {},
        "settings": {
            "index": {
                "routing": {
                    "allocation": {
                        "include": {
                            "_tier_preference": "data_content"
                        }
                    }
                },
                "number_of_shards": "1",
                "provided_name": "demo_index",
                "creation_date": "1653402994280",
                "number_of_replicas": "1",
                "uuid": "1hX6S-rxTAmBCy_R9Rw8Xg",
                "version": {
                    "created": "8010399"
                }
            }
        }
    }
}
```

删除索引：DELETE /demo_index?pretty

```
{
    "acknowledged": true
}
```

CRUD操作

建立索引

首先建立图书索引 book

语法：put /index

PUT /book

结果：

```
{
    "acknowledged": true,
    "shards_acknowledged": true,
    "index": "book"
}
```

新增文档

语法：PUT /index/type/id

语法：PUT /index/type/id

```
PUT /book/_doc/1

{
"name": "Bootstrap开发",
"description": "Bootstrap是由Twitter推出的一个前台页面开发css框架，是一个非常流行的开发框架，此框架集成了多种页面效果。此开发框架包含了大量的CSS、JS程序代码，可以帮助开发者（尤其是不擅长css页面开发的程序人员）轻松的实现一个css，不受浏览器限制的精美界面css效果。",
"studymodel": "201002",
"price":38.6,
"timestamp":"2019-08-25 19:11:35",
"pic":"group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg",
"tags": [ "bootstrap", "dev"]
}
```

结果：

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

```
PUT /book/_doc/2
{
"name": "java编程思想",
"description": "java语言是世界第一编程语言，在软件开发领域使用人数最多。",
"studymodel": "201001",
"price":68.6,
"timestamp":"2019-08-25 19:11:35",
"pic":"group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg",
"tags": [ "java", "dev"]
}
```

结果:

```
{
    "_index": "book",
    "_id": "2",
    "_version": 1,
    "result": "created",
    "_shards": {
        "total": 2,
        "successful": 1,
        "failed": 0
    },
    "_seq_no": 2,
    "_primary_term": 1
}
```

```
PUT /book/_doc/3
{
"name": "spring开发基础",
"description": "spring 在java领域非常流行，java程序员都在用。",
"studymodel": "201001",
"price":88.6,
"timestamp":"2019-08-24 19:11:35",
"pic":"group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg",
"tags": [ "spring", "java"]
}
```

结果：

```
{
    "_index": "book",
    "_id": "3",
    "_version": 1,
    "result": "created",
    "_shards": {
        "total": 2,
        "successful": 1,
        "failed": 0
    },
    "_seq_no": 3,
    "_primary_term": 1
}
```

查询文档

语法：GET /index/type/id

查看图书:GET /book/_doc/1 就可看到json形式的文档。方便程序解析。

get /book/_doc/1

结果：

```
{
    "_index": "book",
    "_id": "1",
    "_version": 2,
    "_seq_no": 1,
    "_primary_term": 1,
    "found": true,
    "_source": {
        "name": "Bootstrap开发",
        "description": "Bootstrap是由Twitter推出的一个前台页面开发css框架，是一个非常流行的开发框架，此框架集成了多种页面效果。此开发框架包含了大量的CSS、JS程序代码，可以帮助开发者（尤其是不擅长css页面开发的程序人员）轻松的实现一个css，不受浏览器限制的精美界面css效果。",
        "studymodel": "201002",
        "price": 38.6,
        "timestamp": "2019-08-25 19:11:35",
        "pic": "group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg",
```

```
        "tags": [
            "bootstrap",
            "dev"
        ]
    }
}
```

get /book/_doc/3

结果:

```
{
    "_index": "book",
    "_id": "3",
    "_version": 1,
    "_seq_no": 3,
    "_primary_term": 1,
    "found": true,
    "_source": {
        "name": "spring开发基础",
        "description": "spring 在java领域非常流行，java程序员都在用。",
        "studymodel": "201001",
        "price": 88.6,
        "timestamp": "2019-08-24 19:11:35",
        "pic": "group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg",
        "tags": [
            "spring",
            "java"
        ]
    }
}
```

修改文档

方法一：整体覆盖，要带上所有信息

PUT /book/_doc/3

```
{
"name": "spring开发基础",
"description": "spring 在java领域非常流行，java程序员都在用。",
"studymodel": "201001",
"price":88.6,
"timestamp":"2019-08-24 19:21:35",
"pic":"group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg",
"tags": [ "spring", "java"]
}
```

结果:

```
{
    "_index": "book",
    "_id": "3",
    "_version": 2,
    "result": "updated",
    "_shards": {
        "total": 2,
        "successful": 1,
        "failed": 0
    },
    "_seq_no": 4,
    "_primary_term": 1
}
```

方法二：局部替换

语法：POST /{index}/type /{id}/_update

或者POST /{index}/_update/{id}

POST /book/_update/3

```
{
  "doc": {
   "price": 78.6
  }
}
```

结果：

```
{
    "_index": "book",
    "_id": "3",
    "_version": 3,
    "result": "updated",
    "_shards": {
        "total": 2,
        "successful": 1,
        "failed": 0
    },
    "_seq_no": 5,
    "_primary_term": 1
}
```

删除文档

语法：DELETE /book/_doc/{id}

```
DELETE /book/_doc/1
```

结果：

```
{
    "_index": "book",
    "_id": "1",
    "_version": 3,
    "result": "deleted",
    "_shards": {
        "total": 2,
        "successful": 1,
        "failed": 0
    },
    "_seq_no": 6,
    "_primary_term": 1
}
```

再次查询：

```
{
    "_index": "book",
    "_id": "1",
    "found": false
}
```

批量查询

单条查询 GET /test_index/_doc/1，如果查询多个id的文档一条一条查询，网络开销太大。

```
GET /_mget
{
   "docs" :
    [
      {
         "_index" : "book",
         "_id" :    2
      },
      {
         "_index" : "book",
         "_id" :    3
      }
   ]
}
```

结果：

```
{
    "docs": [
        {
            "_index": "book",
            "_id": "2",
            "_version": 1,
```

```
            "_seq_no": 2,
            "_primary_term": 1,
            "found": true,
            "_source": {
                "name": "java编程思想",
                "description": "java语言是世界第一编程语言，在软件开发领域使用人数最
多。",
                "studymodel": "201001",
                "price": 68.6,
                "timestamp": "2019-08-25 19:11:35",
                "pic": "group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg",
                "tags": [
                    "java",
                    "dev"
                ]
            }
        },
        {
            "_index": "book",
            "_id": "3",
            "_version": 3,
            "_seq_no": 5,
            "_primary_term": 1,
            "found": true,
            "_source": {
                "name": "spring开发基础",
                "description": "spring 在java领域非常流行，java程序员都在用。",
                "studymodel": "201001",
                "price": 78.6,
                "timestamp": "2019-08-24 19:21:35",
                "pic": "group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg",
                "tags": [
                    "spring",
                    "java"
                ]
            }
        }
    ]
}
```


同一索引下批量查询:

```
GET /book/_mget
{
   "docs" :
   [
      {
         "_id" :2
      },
      {
         "_id" :3
      }
   ]
}
```

结果:


```
{
    "docs": [
        {
            "_index": "book",
            "_id": "2",
            "_version": 1,
            "_seq_no": 2,
            "_primary_term": 1,
            "found": true,
            "_source": {
                "name": "java编程思想",
                "description": "java语言是世界第一编程语言，在软件开发领域使用人数最
多。",
                "studymodel": "201001",
                "price": 68.6,
                "timestamp": "2019-08-25 19:11:35",
                "pic": "group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg",
                "tags": [
                    "java",
                    "dev"
                ]
            }
        },
        {
            "_index": "book",
            "_id": "3",
            "_version": 3,
            "_seq_no": 5,
            "_primary_term": 1,
            "found": true,
            "_source": {
                "name": "spring开发基础",
                "description": "spring 在java领域非常流行，java程序员都在用。",
                "studymodel": "201001",
                "price": 78.6,
                "timestamp": "2019-08-24 19:21:35",
                "pic": "group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg",
                "tags": [
                    "spring",
                    "java"
                ]
            }
        }
    ]
}
```


搜索写法:

```
post /book/_doc/_search
{
    "query":
    {
        "ids" :
        {
            "values" : ["2", "3"]
        }
    }
}
```

批量增删改

Bulk 操作解释将文档的增删改查一些列操作，通过一次请求全都做完。减少网络传输次数。

语法：

```
POST /_bulk
{"action": {"metadata"}}
{"data"}
```

如下操作，删除5，新增14，修改2。

```
POST /_bulk
{ "delete": { "_index": "test_index",  "_id": "5" }}
{ "create": { "_index": "test_index",  "_id": "14" }}
{ "test_field": "test14" }
{ "update": { "_index": "test_index",  "_id": "2"} }
{ "doc" : {"test_field" : "bulk test"} }
```

文档document

字段说明

示例：

```
{
    "_index": "book",
    "_id": "3",
    "_version": 3,
    "_seq_no": 5,
    "_primary_term": 1,
    "found": true,
    "_source": {
        "name": "spring开发基础",
        "description": "spring 在java领域非常流行，java程序员都在用。",
        "studymodel": "201001",
        "price": 78.6,
        "timestamp": "2019-08-24 19:21:35",
        "pic": "group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg",
```

```
        "tags": [
            "spring",
            "java"
        ]
    }
}
```

_index

- 含义：此文档属于哪个索引
- 原则：类似数据放在一个索引中。数据库中表的定义规则。如图书信息放在book索引中，员工信息放在employee索引中。各个索引存储和搜索时互不影响。
- 定义规则：英文小写。尽量不要使用特殊字符。order user

_id

含义：文档的唯一标识。就像表的id主键。结合索引可以标识和定义一个文档。

生成：手动（put /index/_doc/id）、自动

_version

版本信息，每次修改，版本+1

_source

含义：插入数据时的所有字段和值。在get获取数据时，在_source字段中原样返回。

GET /book/_doc/1

生成文档id

用法：POST /index/_doc

```
POST /test_index/_doc
{
  "test_field": "test1"
}
```

结果：

```
{
    "_index": "book",
    "_id": "P2Gd9oABt1R14CTMIiwV",
    "_version": 1,
    "result": "created",
    "_shards": {
        "total": 2,
        "successful": 1,
        "failed": 0
    },
    "_seq_no": 7,
    "_primary_term": 1
}
```


定制返回字段

就像sql不要select *,而要select name,price from book ...一样。

GET /book/_doc/1?__source_includes=name,price


```
{
  "_index" : "book",
  "_type" : "_doc",
  "_id" : "1",
  "_version" : 1,
  "_seq_no" : 10,
  "_primary_term" : 1,
  "found" : true,
  "_source" : {
    "price" : 38.6,
    "name" : "Bootstrap开发教程1"
  }
}
```


文档的替换

全量替换

执行两次，返回结果中版本号（_version）在不断上升。此过程为全量替换。

实质：旧文档的内容不会立即删除，只是标记为deleted。适当的时机，集群会将这些文档删除。

强制创建

为防止覆盖原有数据，我们在新增时，设置为强制创建，不会覆盖原有文档。

语法：PUT /index/ *doc/id*/create

```
PUT /test_index/_doc/1/_create
{
  "test_field": "test"
}
```

返回

```
{
  "error": {
    "root_cause": [
      {
        "type": "version_conflict_engine_exception",
        "reason": "[2]: version conflict, document already exists (current
version [1])",
        "index_uuid": "lqzVqxZLQuCnd6LYtZsMkg",
        "shard": "0",
        "index": "test_index"
      }
    ],
    "type": "version_conflict_engine_exception",
    "reason": "[2]: version conflict, document already exists (current version
[1])",
    "index_uuid": "lqzVqxZLQuCnd6LYtZsMkg",
    "shard": "0",
    "index": "test_index"
  },
  "status": 409
}
```

内部与全量替换是一样的，旧文档标记为删除，新建一个文档。

优点：

- 大大减少网络传输次数和流量，提升性能
- 减少并发冲突发生的概率。

悲观锁与乐观锁机制

为控制并发问题，我们通常采用锁机制。分为悲观锁和乐观锁两种机制。

- 悲观锁：很悲观，所有情况都上锁。此时只有一个线程可以操作数据。具体例子为数据库中的行级锁、表级锁、读锁、写锁等。

  特点：优点是方便，直接加锁，对程序透明。缺点是效率低。

- 乐观锁：很乐观，对数据本身不加锁。提交数据时，通过一种机制验证是否存在冲突，如es中通过版本号验证。

  特点：优点是并发能力高。缺点是操作繁琐，在提交数据时，可能反复重试多次。

es对于文档的增删改都是基于版本号。

手动控制版本号

背景：已有数据是在数据库中，有自己手动维护的版本号的情况下，可以使用external version控制。hbase。

要求：修改时external version要大于当前文档的_version

对比：基于_version时，修改的文档version等于当前文档的版本号。

使用?version=1&version_type=external

总结

- delete：删除一个文档，只要1个json串就可以了
- create：相当于强制创建 PUT /index/type/id/_create
- index：普通的put操作，可以是创建文档，也可以是全量替换文档
- update：执行的是局部更新partial update操作

- 每个json不能换行。相邻json必须换行。
- 每个操作互不影响。操作失败的行会返回其失败信息。
- bulk请求一次不要太大，否则一下积压到内存中，性能会下降。所以，一次请求几千个操作、大小在几M正好。

Java API 实现文档管理

java api有两种：

low : 偏向底层。

high：高级封装。足够。

maven依赖

```
<!--spring boot elasticsearch high-level-client-->
<dependency>
    <groupId>org.elasticsearch.client</groupId>
    <artifactId>elasticsearch-rest-high-level-client</artifactId>
</dependency>
```

查询文档

```java
package mao.elasticsearch_query_document_data;

import org.apache.http.HttpHost;
import org.elasticsearch.action.ActionListener;
import org.elasticsearch.action.get.GetRequest;
import org.elasticsearch.action.get.GetResponse;
import org.elasticsearch.client.RequestOptions;
import org.elasticsearch.client.RestClient;
import org.elasticsearch.client.RestHighLevelClient;
import org.elasticsearch.search.fetch.subphase.FetchSourceContext;
import org.junit.jupiter.api.BeforeAll;
import org.junit.jupiter.api.Test;
import org.springframework.boot.test.context.SpringBootTest;

import java.io.IOException;
import java.util.Map;

/**
 * Project name(项目名称)：elasticsearch_query_document_data
 * Package(包名): mao.elasticsearch_query_document_data
 * Class(类名): ElasticSearchTest
 * Author(作者）: mao
 * Author QQ：1296193245
 * GitHub：https://github.com/maomao124/
 * Date(创建日期)： 2022/5/25
 * Time(创建时间)： 13:01
 * Version(版本): 1.0
 * Description(描述)： SpringBootTest
 */

@SpringBootTest
public class ElasticSearchTest
{
    private static RestHighLevelClient client;

    @BeforeAll
    static void beforeAll()
    {
        //创建ES客户端，单例，可以交给spring管理
        client = new RestHighLevelClient(
                RestClient.builder(new HttpHost("localhost", 9200, "http")));
    }

    /**
     * 同步查询
     *
     * @throws IOException IOException
     */
    @Test
    void query() throws IOException
    {
        //创建请求
        GetRequest getRequest = new GetRequest("book", "2");
        //设置参数
        String[] includes = new String[]{};
        String[] excludes = new String[]{};
```

```
        FetchSourceContext fetchSourceContext = new FetchSourceContext(true,
includes, excludes);
        getRequest.fetchSourceContext(fetchSourceContext);
        //设置路由
        //getRequest.routing("routing");
        //发起请求
        GetResponse getResponse = client.get(getRequest,
RequestOptions.DEFAULT);
        //获取结果
        String id = getResponse.getId();
        long version = getResponse.getVersion();
        String sourceAsString = getResponse.getSourceAsString();
        Map<String, Object> sourceAsMap = getResponse.getSourceAsMap();
        System.out.println("id: " + id);
        System.out.println("版本: " + version);
        System.out.println("sourceAsString: " + sourceAsString);
        System.out.println("sourceAsMap:" + sourceAsMap);
    }

    /**
     * 异步查询
     */
    @Test
    void query_async()
    {
        //创建请求
        GetRequest getRequest = new GetRequest("book", "3");
        //设置参数
        String[] includes = new String[]{};
        String[] excludes = new String[]{};
        FetchSourceContext fetchSourceContext = new FetchSourceContext(true,
includes, excludes);
        getRequest.fetchSourceContext(fetchSourceContext);
        //设置路由
        //getRequest.routing("routing");
        //发起异步请求
        client.getAsync(getRequest, RequestOptions.DEFAULT, new ActionListener<>
()
        {
            /**
             * 成功的回调
             *
             * @param documentFields GetResponse对象
             */
            @Override
            public void onResponse(GetResponse documentFields)
            {
                System.out.println("id: " + documentFields.getId());
                System.out.println("版本: " + documentFields.getVersion());
                System.out.println("字段: " + documentFields.getSourceAsMap());
            }

            /**
             * 失败的回调
             *
             * @param e  Exception
             */
            @Override
```

```
            public void onFailure(Exception e)
            {
                e.printStackTrace();
            }
        });
        try
        {
            //保证能收到消息
            Thread.sleep(5000);
        }
        catch (InterruptedException e)
        {
            e.printStackTrace();
        }
    }
}
```

新增文档

```
package mao.elasticsearch_insert_document_data;

import org.apache.http.HttpHost;
import org.elasticsearch.action.ActionListener;
import org.elasticsearch.action.DocWriteResponse;
import org.elasticsearch.action.index.IndexRequest;
import org.elasticsearch.action.index.IndexResponse;
import org.elasticsearch.client.RequestOptions;
import org.elasticsearch.client.RestClient;
import org.elasticsearch.client.RestHighLevelClient;
import org.elasticsearch.core.TimeValue;
import org.elasticsearch.index.VersionType;
import org.elasticsearch.xcontent.XContentBuilder;
import org.elasticsearch.xcontent.XContentFactory;
import org.elasticsearch.xcontent.XContentType;
import org.junit.jupiter.api.BeforeAll;
import org.junit.jupiter.api.Test;
import org.springframework.boot.test.context.SpringBootTest;

import java.io.IOException;
import java.util.HashMap;
import java.util.Map;

/**
 * Project name(项目名称)：elasticsearch_insert_document_data
 * Package(包名): mao.elasticsearch_insert_document_data
 * Class(类名): ElasticSearchTest
 * Author(作者）: mao
 * Author QQ：1296193245
 * GitHub：https://github.com/maomao124/
 * Date(创建日期)： 2022/5/25
 * Time(创建时间)： 20:12
 * Version(版本): 1.0
 * Description(描述)： ElasticSearchTest
 * 请求：
```

```
 * <p>
 * PUT book/_doc/5
 * {
 * "name" : "java编程思想",
 * "description" : "java语言是世界第一编程语言，在软件开发领域使用人数最多。",
 * "studymodel" : "201001",
 * "price" : 68.6,
 * "timestamp" : "2022-5-25 19:11:35",
 * "pic" : "group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg",
 * "tags": [ "bootstrap", "dev"]
 * }
 */

@SpringBootTest
public class ElasticSearchTest
{

    private static RestHighLevelClient client;

    @BeforeAll
    static void beforeAll()
    {
        client = new RestHighLevelClient(
                RestClient.builder(new HttpHost("localhost", 9200, "http")));
    }


    /**
     * 同步插入，方法1
     *
     * @throws IOException IOException
     */
    @Test
    void insert() throws IOException
    {
        //构建请求
        IndexRequest indexRequest = new IndexRequest("book");
        indexRequest.id("5");

        //设置请求体

        //方法1
        String json = "{\n" +
                "   \"name\" : \"java编程思想\",\n" +
                "    \"description\" : \"java语言是世界第一编程语言，在软件开发领域使用
人数最多。\",\n" +
                "    \"studymodel\" : \"201001\",\n" +
                "    \"price\" : 68.6,\n" +
                "    \"timestamp\" : \"2022-5-25 19:11:35\",\n" +
                "    \"pic\" :
\"group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg\",\n" +
                "    \"tags\": [ \"bootstrap\", \"dev\"]\n" +
                "}";

        //填入到IndexRequest里
        indexRequest.source(json, XContentType.JSON);
        //设置可选参数
        //超时时间，3秒超时
```

```
        indexRequest.timeout(TimeValue.timeValueSeconds(3));
        //版本号，可以实现乐观锁
        //indexRequest.versionType(VersionType.EXTERNAL);

        //发起请求
        IndexResponse indexResponse = client.index(indexRequest,
RequestOptions.DEFAULT);
        //获取数据
        //获取插入的类型
        if (indexResponse.getResult() == DocWriteResponse.Result.CREATED)
        {
            DocWriteResponse.Result result = indexResponse.getResult();
            System.out.println("创建:" + result);
        }
        else if (indexResponse.getResult() == DocWriteResponse.Result.UPDATED)
        {
            DocWriteResponse.Result result = indexResponse.getResult();
            System.out.println("更新:" + result);
        }
        else
        {
            System.out.println("其它");
        }
    }

    @Test
    void insert_async()
    {
        //构建请求
        IndexRequest indexRequest = new IndexRequest("book");
        indexRequest.id("5");
        //设置请求体

        //方法1
        String json = "{\n" +
                "   \"name\" : \"java编程思想\",\n" +
                "    \"description\" : \"java语言是世界第一编程语言，在软件开发领域使用
人数最多。\",\n" +
                "    \"studymodel\" : \"201001\",\n" +
                "    \"price\" : 68.6,\n" +
                "    \"timestamp\" : \"2022-5-25 19:11:35\",\n" +
                "    \"pic\" :
\"group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg\",\n" +
                "    \"tags\": [ \"bootstrap\", \"dev\"]\n" +
                "}";

        //填入到IndexRequest里
        indexRequest.source(json, XContentType.JSON);
        //发起异步请求
        client.indexAsync(indexRequest, RequestOptions.DEFAULT, new
ActionListener<IndexResponse>()
        {
            @Override
            public void onResponse(IndexResponse indexResponse)
            {
                if (indexResponse.getResult() ==
DocWriteResponse.Result.CREATED)
                {
```

```
                    DocWriteResponse.Result result = indexResponse.getResult();
                    System.out.println("创建:" + result);
                }
                else if (indexResponse.getResult() ==
DocWriteResponse.Result.UPDATED)
                {
                    DocWriteResponse.Result result = indexResponse.getResult();
                    System.out.println("更新:" + result);
                }
                else
                {
                    System.out.println("其它");
                }
            }

            @Override
            public void onFailure(Exception e)
            {
                e.printStackTrace();
            }
        });

        try
        {
            Thread.sleep(2000);
        }
        catch (InterruptedException e)
        {
            e.printStackTrace();
        }
    }


    /**
     * 方法2
     */
    @Test
    void insert2_async()
    {
        //构建请求
        IndexRequest indexRequest = new IndexRequest("book");
        indexRequest.id("5");
        //设置请求体

        //方法2
        Map<String, Object> map = new HashMap<>();
        map.put("name", "java编程思想");
        map.put("description", "java语言是世界第一编程语言，在软件开发领域使用人数最
多。");
        map.put("studymodel", "201001");
        map.put("price", 68.6);
        map.put("timestamp", "2022-5-25 19:11:35");
        map.put("pic", "group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg");
        map.put("tags", new String[]{"bootstrap", "dev"});

        indexRequest.source(map);

        //发起异步请求
```

```
        client.indexAsync(indexRequest, RequestOptions.DEFAULT, new
ActionListener<IndexResponse>()
        {
            @Override
            public void onResponse(IndexResponse indexResponse)
            {
                System.out.println("成功：" + indexResponse.getResult());
            }

            @Override
            public void onFailure(Exception e)
            {
                e.printStackTrace();
            }
        });

        try
        {
            Thread.sleep(2000);
        }
        catch (InterruptedException e)
        {
            e.printStackTrace();
        }
    }


    /**
     * 方法3
     *
     * @throws IOException IOException
     */
    @Test
    void insert3_async() throws IOException
    {
        //构建请求
        IndexRequest indexRequest = new IndexRequest("book");
        indexRequest.id("5");
        //设置请求体

        //方法3
        XContentBuilder xContentBuilder = XContentFactory.jsonBuilder();
        xContentBuilder.startObject();
        {
            xContentBuilder.field("name", "java编程思想");
            xContentBuilder.field("description", "java语言是世界第一编程语言，在软件开
发领域使用人数最多。");
            xContentBuilder.field("studymodel", "201001");
            xContentBuilder.field("price", 68.6);
            xContentBuilder.field("timestamp", "2022-5-25 19:11:35");
            xContentBuilder.field("pic",
"group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg");
            xContentBuilder.field("tags", new String[]{"bootstrap", "dev"});
        }
        xContentBuilder.endObject();

        //加入到请求里
        indexRequest.source(xContentBuilder);
```

```
        //发起异步请求
        client.indexAsync(indexRequest, RequestOptions.DEFAULT, new
ActionListener<IndexResponse>()
        {
            @Override
            public void onResponse(IndexResponse indexResponse)
            {
                System.out.println("成功：" + indexResponse.getResult());
            }

            @Override
            public void onFailure(Exception e)
            {
                e.printStackTrace();
            }
        });

        try
        {
            Thread.sleep(2000);
        }
        catch (InterruptedException e)
        {
            e.printStackTrace();
        }
    }

    /**
     * 方法4
     *
     * @throws IOException IOException
     */
    @Test
    void insert4_async() throws IOException
    {
        //构建请求
        IndexRequest indexRequest = new IndexRequest("book");
        indexRequest.id("5");
        //设置请求体

        //方法4
        indexRequest.source("name", "java编程思想",
                "description", "java语言是世界第一编程语言，在软件开发领域使用人数最
多。",
                "studymodel", "201001",
                "price", 69.6,
                "timestamp", "2022-5-25 19:11:35",
                "pic", "group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg",
                "tags", new String[]{"bootstrap", "dev"});

        //发起异步请求
        client.indexAsync(indexRequest, RequestOptions.DEFAULT, new
ActionListener<IndexResponse>()
        {
            @Override
            public void onResponse(IndexResponse indexResponse)
            {
```

```
                System.out.println("成功：" + indexResponse.getResult());
            }

            @Override
            public void onFailure(Exception e)
            {
                e.printStackTrace();
            }
        });

        try
        {
            Thread.sleep(2000);
        }
        catch (InterruptedException e)
        {
            e.printStackTrace();
        }
    }
}
```

修改文档

```
package mao.elasticsearch_update_document_data;

import org.apache.http.HttpHost;
import org.elasticsearch.action.ActionListener;
import org.elasticsearch.action.update.UpdateRequest;
import org.elasticsearch.action.update.UpdateResponse;
import org.elasticsearch.client.RequestOptions;
import org.elasticsearch.client.RestClient;
import org.elasticsearch.client.RestHighLevelClient;
import org.elasticsearch.index.get.GetResult;
import org.junit.jupiter.api.BeforeAll;
import org.junit.jupiter.api.Test;
import org.springframework.boot.test.context.SpringBootTest;

import java.io.IOException;

/**
 * Project name(项目名称)：elasticsearch_update_document_data
 * Package(包名): mao.elasticsearch_update_document_data
 * Class(类名): ElasticSearchTest
 * Author(作者）: mao
 * Author QQ：1296193245
 * GitHub：https://github.com/maomao124/
 * Date(创建日期)： 2022/5/25
 * Time(创建时间)： 21:20
 * Version(版本): 1.0
 * Description(描述)： SpringBootTest
 * <p>
 * 请求：
 * POST book/_update/5
 * {
```

```
 *      "doc":
 *      {
 *          "price" : 68.5
 *      }
 * }
 */

@SpringBootTest
public class ElasticSearchTest
{
    private static RestHighLevelClient client;

    @BeforeAll
    static void beforeAll()
    {
        client = new RestHighLevelClient(
                RestClient.builder(new HttpHost("localhost", 9200, "http")));
    }

    /**
     * 同步更新
     *
     * @throws IOException IOException
     */
    @Test
    void update() throws IOException
    {
        UpdateRequest updateRequest = new UpdateRequest("book", "5");
        //设置请求体
        updateRequest.doc("price", 68.5);

        //发起请求
        UpdateResponse updateResponse = client.update(updateRequest,
RequestOptions.DEFAULT);
        //获取数据
        GetResult getResult = updateResponse.getGetResult();
        System.out.println(getResult);
    }

    /**
     * 异步更新
     *
     * @throws IOException IOException
     */
    @Test
    void update_async() throws IOException
    {
        UpdateRequest updateRequest = new UpdateRequest("book", "5");
        //设置请求体
        updateRequest.doc("price", 68.5);

        //发起异步请求
        client.updateAsync(updateRequest, RequestOptions.DEFAULT, new
ActionListener<UpdateResponse>()
        {
            @Override
            public void onResponse(UpdateResponse updateResponse)
            {
```

```
                //获取数据
                GetResult getResult = updateResponse.getGetResult();
                System.out.println(getResult);
            }

            @Override
            public void onFailure(Exception e)
            {
                e.printStackTrace();
            }
        });
        try
        {
            Thread.sleep(2000);
        }
        catch (InterruptedException e)
        {
            e.printStackTrace();
        }
    }
}
```

删除文档

```
package mao.elasticsearch_delete_document_data;

import org.apache.http.HttpHost;
import org.elasticsearch.action.ActionListener;
import org.elasticsearch.action.DocWriteResponse;
import org.elasticsearch.action.delete.DeleteRequest;
import org.elasticsearch.action.delete.DeleteResponse;
import org.elasticsearch.client.RequestOptions;
import org.elasticsearch.client.RestClient;
import org.elasticsearch.client.RestHighLevelClient;
import org.junit.jupiter.api.BeforeAll;
import org.junit.jupiter.api.Test;
import org.springframework.boot.test.context.SpringBootTest;

/**
 * Project name(项目名称)：elasticsearch_delete_document_data
 * Package(包名): mao.elasticsearch_delete_document_data
 * Class(类名): ElasticSearchTest
 * Author(作者）: mao
 * Author QQ：1296193245
 * GitHub：https://github.com/maomao124/
 * Date(创建日期)： 2022/5/25
 * Time(创建时间)： 21:50
 * Version(版本): 1.0
 * Description(描述)： SpringBootTest
 */

@SpringBootTest
public class ElasticSearchTest
{
```

```
    private static RestHighLevelClient client;

    @BeforeAll
    static void beforeAll()
    {
        client = new RestHighLevelClient(
                RestClient.builder(new HttpHost("localhost", 9200, "http")));
    }

    /**
     * 同步删除
     *
     * @throws Exception Exception
     */
    @Test
    void delete() throws Exception
    {
        //构建请求
        DeleteRequest deleteRequest = new DeleteRequest("book", "5");
        //发起请求
        DeleteResponse deleteResponse = client.delete(deleteRequest,
RequestOptions.DEFAULT);
        //获取数据
        DocWriteResponse.Result result = deleteResponse.getResult();
        System.out.println(result);
    }

    @Test
    void delete_async() throws Exception
    {
        //构建请求
        DeleteRequest deleteRequest = new DeleteRequest("book", "5");
        //发起异步请求
        client.deleteAsync(deleteRequest, RequestOptions.DEFAULT, new
ActionListener<DeleteResponse>()
        {
            @Override
            public void onResponse(DeleteResponse deleteResponse)
            {
                //获取数据
                DocWriteResponse.Result result = deleteResponse.getResult();
                System.out.println(result);
            }

            @Override
            public void onFailure(Exception e)
            {
                e.printStackTrace();
            }
        });

        Thread.sleep(2000);
    }
}
```

elasticsearch内部机制

对复杂分布式机制的透明隐藏特性：

- 分布式机制：分布式数据存储及共享。
- 分片机制：数据存储到哪个分片，副本数据写入。
- 集群发现机制：cluster discovery。新启动es实例，自动加入集群。
- shard负载均衡：大量数据写入及查询，es会将数据平均分配。
- shard副本：新增副本数，分片重分配。

Elasticsearch的垂直扩容与水平扩容：

- 垂直扩容：使用更加强大的服务器替代老服务器。但单机存储及运算能力有上线。且成本直线上升。如10t服务器1万。单个10T服务器可能20万。
- 水平扩容：采购更多服务器，加入集群。大数据。

增减或减少节点时的数据rebalance：

- 新增或减少es实例时，es集群会将数据重新分配。

节点对等的分布式架构：

- 节点对等，每个节点都能接收所有的请求
- 自动请求路由
- 响应收集

分片shard、副本replica机制

（1）每个index包含一个或多个shard

（2）每个shard都是一个最小工作单元，承载部分数据，lucene实例，完整的建立索引和处理请求的能力

（3）增减节点时，shard会自动在nodes中负载均衡

（4）primary shard和replica shard，每个document肯定只存在于某一个primary shard以及其对应的replica shard中，不可能存在于多个primary shard

（5）replica shard是primary shard的副本，负责容错，以及承担读请求负载

（6）primary shard的数量在创建索引的时候就固定了，replica shard的数量可以随时修改

（7）primary shard的默认数量是1，replica默认是1，默认共有2个shard，1个primary shard，1个replica shard

注意：es7以前primary shard的默认数量是5，replica默认是1，默认有10个shard，5个primary shard，5个replica shard

（8）primary shard不能和自己的replica shard放在同一个节点上（否则节点宕机，primary shard和副本都丢失，起不到容错的作用），但是可以和其他primary shard的replica shard放在同一个节点上

横向扩容

- 分片自动负载均衡，分片向空闲机器转移。
- 每个节点存储更少分片，系统资源给与每个分片的资源更多，整体集群性能提高。
- 扩容极限：节点数大于整体分片数，则必有空闲机器。
- 超出扩容极限时，可以增加副本数，如设置副本数为2，总共3*3=9个分片。9台机器同时运行，存储和搜索性能更强。容错性更好。
- 容错性：只要一个索引的所有主分片在，集群就就可以运行。

容错机制

- master node宕机，自动master选举，集群为red
- replica容错：新master将replica提升为primary shard，yellow
- 重启宕机node，master copy replica到该node，使用原有的shard并同步宕机后的修改，green

文档存储机制

数据路由

一个文档，最终会落在主分片的一个分片上，到底应该在哪一个分片？这就是数据路由。

路由算法

shard = hash(routing) % number_of_primary_shards

取哈希值对主分片数取模。

- number_of_primary_shards：主分片的数量

手动指定 routing number：

```
PUT /test_index/_doc/2?routing=num
{
  "num": 0,
  "tags": []
}
```

好处：

- 可以定制一类文档数据存储到一个分片中

坏处：

- 会造成数据倾斜

主分片数量不可变

涉及到以往数据的查询搜索，所以一旦建立索引，主分片数不可变。

因为根据路由算法shard = hash(routing) % number_of_primary_shards

假设主分片数量可变，更改前某一个routing算到的值到第1个分片上，更改分片数量后，再次计算的值不一定是第一个分片上

增删改内部机制

增删改可以看做update,都是对数据的改动。一个改动请求发送到es集群，经历以下四个步骤：

1. 客户端选择一个node发送请求过去，这个node就是coordinating node（协调节点）
2. coordinating node，对document进行路由，将请求转发给对应的node（有primary shard）
3. 实际的node上的primary shard处理请求，然后将数据同步到replica node。
4. coordinating node，如果发现primary node和所有replica node都搞定之后，就返回响应结果给客户端。

查询内部机制

1、客户端发送请求到任意一个node，成为coordinate node

2、coordinate node对document进行路由，将请求转发到对应的node，此时会使用round-robin随机轮询算法，在primary shard以及其所有replica中随机选择一个，让读请求负载均衡

3、接收请求的node返回document给coordinate node

4、coordinate node返回document给客户端

5、特殊情况：document如果还在建立索引过程中，可能只有primary shard有，任何一个replica shard都没有，此时可能会导致无法读取到document，但是document完成索引建立之后，primary shard和replica shard就都有了。

Mapping映射

自动或手动为index中的_doc建立的一种数据结构和相关配置，简称为mapping映射。

动态映射

动态映射：dynamic mapping，自动为我们建立index，以及对应的mapping，mapping中包含了每个field对应的数据类型，以及如何分词等设置。

```
{
  "book" : {
    "aliases" : { },
    "mappings" : {
      "properties" : {
        "description" : {
          "type" : "text",
          "fields" : {
            "keyword" : {
              "type" : "keyword",
              "ignore_above" : 256
```

```
        }
      }
    },
    "name" : {
      "type" : "text",
      "fields" : {
        "keyword" : {
          "type" : "keyword",
          "ignore_above" : 256
        }
      }
    },
    "pic" : {
      "type" : "text",
      "fields" : {
        "keyword" : {
          "type" : "keyword",
          "ignore_above" : 256
        }
      }
    },
    "price" : {
      "type" : "float"
    },
    "query" : {
      "type" : "text",
      "fields" : {
        "keyword" : {
          "type" : "keyword",
          "ignore_above" : 256
        }
      }
    },
    "studymodel" : {
      "type" : "text",
      "fields" : {
        "keyword" : {
          "type" : "keyword",
          "ignore_above" : 256
        }
      }
    },
    "tags" : {
      "type" : "text",
      "fields" : {
        "keyword" : {
          "type" : "keyword",
          "ignore_above" : 256
        }
      }
    },
    "test_field" : {
      "type" : "text",
      "fields" : {
        "keyword" : {
          "type" : "keyword",
          "ignore_above" : 256
        }
```

```
          }
        },
        "timestamp" : {
          "type" : "text",
          "fields" : {
            "keyword" : {
              "type" : "keyword",
              "ignore_above" : 256
            }
          }
        }
      }
    },
    "settings" : {
      "index" : {
        "routing" : {
          "allocation" : {
            "include" : {
              "_tier_preference" : "data_content"
            }
          }
        },
        "number_of_shards" : "1",
        "provided_name" : "book",
        "creation_date" : "1653403212942",
        "number_of_replicas" : "1",
        "uuid" : "NtcbU6zVTVixiHkWcEuOFw",
        "version" : {
          "created" : "8010399"
        }
      }
    }
  }
}
```


精确匹配

2019-01-01，exact value，搜索的时候，必须输入2019-01-01，才能搜索出来

如果你输入一个01，是搜索不出来的

相当于数据库：select * from book where name= 'java'

full text 全文检索

（1）缩写 vs. 全称：cn vs. china

（2）格式转化：like liked likes

（3）大小写：Tom vs tom

（4）同义词：like vs love

相当于数据库：select * from book where name like '%java%'

china，搜索cn，也可以将china搜索出来

likes，搜索like，也可以将likes搜索出来

Tom，搜索tom，也可以将Tom搜索出来

like，搜索love，同义词，也可以将like搜索出来

全文检索下倒排索引核心原理

1. 分词，初步的倒排索引的建立
2. 重建倒排索引

normalization正规化，建立倒排索引的时候，会执行一个操作，也就是说对拆分出的各个单词进行相应的处理，以提升后面搜索的时候能够搜索到相关联的文 档的概率。时态的转换，单复数的转换，同义词的转换，大小写的转换

分词器 analyzer

作用：切分词语，normalization（提升recall召回率）

给你一段句子，然后将这段句子拆分成一个一个的单个的单词，同时对每个单词进行normalization（时态转换，单复数转换）

内置分词器

例句：Set the shape to semi-transparent by calling set_trans(5)

standard analyzer标准分词器：set, the, shape, to, semi, transparent, by, calling, set_trans, 5（默认的是standard）

simple analyzer简单分词器：set, the, shape, to, semi, transparent, by, calling, set, trans

whitespace analyzer：Set, the, shape, to, semi-transparent, by, calling, set_trans(5)

language analyzer（特定的语言的分词器，比如说，english，英语分词器）：set, shape, semi, transpar, call, set_tran, 5

测试分词器

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

结果：

```
{
  "tokens" : [
    {
      "token" : "set",
      "start_offset" : 0,
      "end_offset" : 3,
      "type" : "<ALPHANUM>",
```

```
      "position" : 0
    },
    {
      "token" : "the",
      "start_offset" : 4,
      "end_offset" : 7,
      "type" : "<ALPHANUM>",
      "position" : 1
    },
    {
      "token" : "shape",
      "start_offset" : 8,
      "end_offset" : 13,
      "type" : "<ALPHANUM>",
      "position" : 2
    },
    {
      "token" : "to",
      "start_offset" : 14,
      "end_offset" : 16,
      "type" : "<ALPHANUM>",
      "position" : 3
    },
    {
      "token" : "semi",
      "start_offset" : 17,
      "end_offset" : 21,
      "type" : "<ALPHANUM>",
      "position" : 4
    },
    {
      "token" : "transparent",
      "start_offset" : 22,
      "end_offset" : 33,
      "type" : "<ALPHANUM>",
      "position" : 5
    },
    {
      "token" : "by",
      "start_offset" : 34,
      "end_offset" : 36,
      "type" : "<ALPHANUM>",
      "position" : 6
    },
    {
      "token" : "calling",
      "start_offset" : 37,
      "end_offset" : 44,
      "type" : "<ALPHANUM>",
      "position" : 7
    },
    {
      "token" : "set_trans",
      "start_offset" : 45,
      "end_offset" : 54,
      "type" : "<ALPHANUM>",
      "position" : 8
    },
```

```
    {
      "token" : "5",
      "start_offset" : 55,
      "end_offset" : 56,
      "type" : "<NUM>",
      "position" : 9
    }
  ]
}
```

- token ： 实际存储的term 关键字
- position ： 在此词条在原文本中的位置
- start_offset/end_offset：字符在原始字符串中的位置

创建映射

Text：文本类型

1）analyzer

通过analyzer属性指定分词器。

上边指定了analyzer是指在索引和搜索都使用english，如果单独想定义搜索时使用的分词器则可以通过search_analyzer属性。

2）index

index属性指定是否索引。

默认为index=true，即要进行索引，只有进行索引才可以从索引库搜索到。

但是也有一些内容不需要索引，比如：商品图片地址只被用来展示图片，不进行搜索图片，此时可以将index设置为false。

删除索引，重新创建映射，将pic的index设置为false，尝试根据pic去搜索，结果搜索不到数据。

3）store

是否在source之外存储，每个文档索引后会在 ES中保存一份原始文档，存放在"*source*"中，一般情况下不需要设置*store*为*true*，因为在source中已经有一份原始文档了。

创建映射：

```
PUT book/_mapping
{
        "properties": {
           "name": {
                  "type": "text"
            },
           "description":
            {
              "type": "text",
              "analyzer":"english",
              "search_analyzer":"english"
```

```
            },
            "pic":{
              "type":"text",
              "index":false
            },
            "studymodel":{
              "type":"text"
            }
    }
}
```

keyword关键字字段：

keyword字段的索引时是不进行分词的，比如：邮政编码、手机号码、身份证等。keyword字段通常用于过虑、排序、聚合等。

日期类型不用设置分词器。

通常日期类型的字段用于排序。

```
{

   "properties": {

       "timestamp": {

         "type":   "date",

         "format": "yyyy-MM-dd HH:mm:ss||yyyy-MM-dd"

        }

     }

}
```

修改映射

只能创建index时手动建立mapping，或者新增field mapping，但是不能update field mapping。

因为已有数据按照映射早已分词存储好。

删除映射

通过删除索引来删除映射。

DELETE /book

复杂数据类型

- multivalue field：

{ "tags": ["tag1", "tag2"]}

- empty field：

null，[]，[null]

- object field：

```
{
  "address": {
    "country": "china",
    "province": "guangdong",
    "city": "guangzhou"
  },
  "name": "jack",
  "age": 27,
  "join_date": "2019-01-01"
}
```

address：object类型

dynamic mapping

true：遇到陌生字段，就进行dynamic mapping

false：新检测到的字段将被忽略。这些字段将不会被索引，因此将无法搜索，但仍将出现在返回点击的源字段中。这些字段不会添加到映射中，必须显式添加新字段。

strict：遇到陌生字段，就报错

```
PUT /my_index
{
    "mappings": {
      "dynamic": "strict",
       "properties": {
        "title": {
          "type": "text"
        },
        "address": {
          "type": "object",
          "dynamic": "true"
        }
        }
    }
}
```

索引Index

创建索引：

```
PUT /index
{
    "settings": {},
    "mappings": {
       "properties" : {
            "field1" : { "type" : "text" }
        }
    },
    "aliases": {
        "default_index": {}
  }
}
```

修改副本数：

```
PUT /my_index/_settings
{
    "index" : {
        "number_of_replicas" : 2
    }
}
```

删除索引

DELETE /my_index

DELETE /index_one,index_two

DELETE /index_*

DELETE /_all

中文分词器 IK分词器

下载地址：https://github.com/medcl/elasticsearch-analysis-ik/releases

ik分词器的使用

存储时，使用ik_max_word，搜索时，使用ik_smart

```
PUT /my_index
{
  "mappings": {
      "properties": {
        "text": {
          "type": "text",
          "analyzer": "ik_max_word",
          "search_analyzer": "ik_smart"
        }
      }
  }
}
```

ik配置文件

ik配置文件地址：es/plugins/ik/config目录

IKAnalyzer.cfg.xml：用来配置自定义词库

main.dic：ik原生内置的中文词库，总共有27万多条，只要是这些单词，都会被分在一起

preposition.dic: 介词

quantifier.dic：放了一些单位相关的词，量词

suffix.dic：放了一些后缀

surname.dic：中国的姓氏

stopword.dic：英文停用词

自定义词库

自己建立词库：每年都会涌现一些特殊的流行词，网红，蓝瘦香菇，喊麦，鬼畜，一般不会在ik的原生词典里

自己补充自己的最新的词语，到ik的词库里面

IKAnalyzer.cfg.xml：ext_dict，创建mydict.dic。

补充自己的词语，然后需要重启es，才能生效

java api 实现索引管理

创建索引

```
package mao.elasticsearch_create_index;
```

```java
import org.apache.http.HttpHost;
import org.elasticsearch.action.ActionListener;
import org.elasticsearch.action.admin.indices.alias.Alias;
import org.elasticsearch.client.IndicesClient;
import org.elasticsearch.client.RequestOptions;
import org.elasticsearch.client.RestClient;
import org.elasticsearch.client.RestHighLevelClient;
import org.elasticsearch.client.indices.CreateIndexRequest;
import org.elasticsearch.client.indices.CreateIndexResponse;
import org.elasticsearch.xcontent.XContentBuilder;
import org.elasticsearch.xcontent.XContentFactory;
import org.elasticsearch.xcontent.XContentType;
import org.junit.jupiter.api.BeforeAll;
import org.junit.jupiter.api.Test;
import org.springframework.boot.test.context.SpringBootTest;

import java.io.IOException;
import java.util.HashMap;
import java.util.Map;

/**
 * Project name(项目名称)：elasticsearch_create_Index
 * Package(包名): mao.elasticsearch_create_index
 * Class(类名): ElasticSearchTest
 * Author(作者）: mao
 * Author QQ：1296193245
 * GitHub：https://github.com/maomao124/
 * Date(创建日期)： 2022/5/27
 * Time(创建时间)： 9:59
 * Version(版本): 1.0
 * Description(描述)： SpringBootTest
 */

@SpringBootTest
public class ElasticSearchTest
{
    private static RestHighLevelClient client;

    @BeforeAll
    static void beforeAll()
    {
        client = new RestHighLevelClient(RestClient.builder(
                new HttpHost("localhost", 9200, "http")));
    }

    //创建索引：
    //PUT /index
    //{
    //    "settings": {},
    //    "mappings": {
    //       "properties" : {
    //            "field1" : { "type" : "text" }
    //        }
    //    },
    //    "aliases": {
    //      "default_index": {}
    //  }
    //}
```

```
    /**
     * 创建索引
     * 方法1
     *
     * @throws IOException IOException
     */
    @Test
    void create_index() throws IOException
    {
        //构建请求
        CreateIndexRequest createIndexRequest = new
CreateIndexRequest("my_index");
        //设置参数settings
        createIndexRequest.settings();
        //设置映射mappings

        //方式1
        createIndexRequest.mapping("{\n" +
                "                \"dynamic\": \"strict\"," +
                "               \"properties\" : {\n" +
                "                    \"name\" : { \"type\" : \"text\" },\n" +
                "                    \"name2\" : { \"type\" : \"text\" }\n" +
                "               }}", XContentType.JSON);

        //设置别名aliases
        //createIndexRequest.alias(new Alias("my_index2"));

        //操作索引的客户端
        IndicesClient indices = client.indices();
        //发起请求
        CreateIndexResponse createIndexResponse =
indices.create(createIndexRequest, RequestOptions.DEFAULT);
        //获得数据
        boolean acknowledged = createIndexResponse.isAcknowledged();
        System.out.println(acknowledged);
    }

    /**
     * 创建索引
     * 方法2
     *
     * @throws IOException IOException
     */
    @Test
    void create_index2() throws IOException
    {
        //构建请求
        CreateIndexRequest createIndexRequest = new
CreateIndexRequest("my_index");
        //设置参数settings
        createIndexRequest.settings();
        //设置映射mappings

        //方式2
        Map<String, Object> name = new HashMap<>();
        name.put("type", "text");
        Map<String, Object> name2 = new HashMap<>();
```

```
        name2.put("type", "text");
        Map<String, Object> properties = new HashMap<>();
        properties.put("name", name);
        properties.put("name2", name2);
        Map<String, Object> map = new HashMap<>();
        map.put("dynamic", "strict");
        map.put("properties", properties);
        createIndexRequest.mapping(map);


        //操作索引的客户端
        IndicesClient indices = client.indices();
        //发起请求
        CreateIndexResponse createIndexResponse =
indices.create(createIndexRequest, RequestOptions.DEFAULT);
        //获得数据
        boolean acknowledged = createIndexResponse.isAcknowledged();
        System.out.println(acknowledged);
    }

    /**
     * 创建索引
     * 方法3
     *
     * @throws IOException IOException
     */
    @Test
    void create_index3() throws IOException
    {
        //构建请求
        CreateIndexRequest createIndexRequest = new
CreateIndexRequest("my_index");
        //设置参数settings
        createIndexRequest.settings();
        //设置映射mappings

        //方式3
        XContentBuilder xContentBuilder = XContentFactory.jsonBuilder();
        xContentBuilder.startObject();
        {
            xContentBuilder.field("dynamic", "strict");
            xContentBuilder.startObject("properties");
            {
                xContentBuilder.startObject("name");
                {
                    xContentBuilder.field("type", "text");
                }
                xContentBuilder.endObject();

                xContentBuilder.startObject("name2");
                {
                    xContentBuilder.field("type", "text");
                }
                xContentBuilder.endObject();
            }
            xContentBuilder.endObject();
        }
        xContentBuilder.endObject();
```

```
        createIndexRequest.mapping(xContentBuilder);


        //操作索引的客户端
        IndicesClient indices = client.indices();
        //发起请求
        CreateIndexResponse createIndexResponse =
indices.create(createIndexRequest, RequestOptions.DEFAULT);
        //获得数据
        boolean acknowledged = createIndexResponse.isAcknowledged();
        System.out.println(acknowledged);
    }

    @Test
    void create_index_async() throws IOException
    {
        //构建请求
        CreateIndexRequest createIndexRequest = new
CreateIndexRequest("my_index");
        //设置参数settings
        createIndexRequest.settings();
        //设置映射mappings

        //方式3
        XContentBuilder xContentBuilder = XContentFactory.jsonBuilder();
        xContentBuilder.startObject();
        {
            xContentBuilder.field("dynamic", "strict");
            xContentBuilder.startObject("properties");
            {
                xContentBuilder.startObject("name");
                {
                    xContentBuilder.field("type", "text");
                }
                xContentBuilder.endObject();

                xContentBuilder.startObject("name2");
                {
                    xContentBuilder.field("type", "text");
                }
                xContentBuilder.endObject();
            }
            xContentBuilder.endObject();
        }
        xContentBuilder.endObject();

        createIndexRequest.mapping(xContentBuilder);

        IndicesClient indices = client.indices();
        //发起异步请求
        indices.createAsync(createIndexRequest, RequestOptions.DEFAULT, new
ActionListener<CreateIndexResponse>()
        {
            @Override
            public void onResponse(CreateIndexResponse createIndexResponse)
            {
                //获得数据
```

```
                boolean acknowledged = createIndexResponse.isAcknowledged();
                System.out.println(acknowledged);
            }

            @Override
            public void onFailure(Exception e)
            {
                e.printStackTrace();
            }
        });
        //休眠
        try
        {
            Thread.sleep(2000);
        }
        catch (InterruptedException e)
        {
            e.printStackTrace();
        }
    }
}
```

删除索引

```
package mao.elasticsearch_delete_index;

import org.apache.http.HttpHost;
import org.elasticsearch.action.ActionListener;
import org.elasticsearch.action.admin.indices.delete.DeleteIndexRequest;
import org.elasticsearch.action.support.master.AcknowledgedResponse;
import org.elasticsearch.client.IndicesClient;
import org.elasticsearch.client.RequestOptions;
import org.elasticsearch.client.RestClient;
import org.elasticsearch.client.RestHighLevelClient;
import org.junit.jupiter.api.AfterAll;
import org.junit.jupiter.api.BeforeAll;
import org.junit.jupiter.api.Test;
import org.springframework.boot.test.autoconfigure.web.client.RestClientTest;
import org.springframework.boot.test.context.SpringBootTest;

import java.io.IOException;

/**
 * Project name(项目名称)：elasticsearch_delete_Index
 * Package(包名): mao.elasticsearch_delete_index
 * Class(类名): ElasticSearchTest
 * Author(作者）: mao
 * Author QQ：1296193245
 * GitHub：https://github.com/maomao124/
 * Date(创建日期)： 2022/5/27
 * Time(创建时间)： 11:44
 * Version(版本): 1.0
 * Description(描述)： SpringBootTest
 */
```

```
@SpringBootTest
public class ElasticSearchTest
{

    private static RestHighLevelClient client;


    @BeforeAll
    static void beforeAll()
    {
        client = new RestHighLevelClient(RestClient.builder(
                new HttpHost("localhost", 9200, "http")));
    }

    /**
     * 删除索引
     *
     * @throws IOException IOException
     */
    @Test
    void delete() throws IOException
    {
        //构建请求
        DeleteIndexRequest deleteIndexRequest = new
DeleteIndexRequest("my_index");
        //获得操作索引的客户端
        IndicesClient indices = client.indices();
        //发起请求
        AcknowledgedResponse acknowledgedResponse =
indices.delete(deleteIndexRequest, RequestOptions.DEFAULT);
        //获得结果
        boolean acknowledged = acknowledgedResponse.isAcknowledged();
        System.out.println(acknowledged);
    }

    /**
     * 删除索引
     * 异步请求
     */
    @Test
    void delete_async()
    {
        //构建请求
        DeleteIndexRequest deleteIndexRequest = new
DeleteIndexRequest("my_index");
        //获得操作索引的客户端
        IndicesClient indices = client.indices();
        //发起异步请求
        indices.deleteAsync(deleteIndexRequest, RequestOptions.DEFAULT, new
ActionListener<AcknowledgedResponse>()
        {
            /**
             * 成功的回调
             * @param acknowledgedResponse AcknowledgedResponse
             */
            @Override
            public void onResponse(AcknowledgedResponse acknowledgedResponse)
```

```
            {
                //获得结果
                boolean acknowledged = acknowledgedResponse.isAcknowledged();
                System.out.println(acknowledged);
            }

            /**
             * 失败的回调
             * @param e Exception
             */
            @Override
            public void onFailure(Exception e)
            {
                System.out.println(e.getMessage());
            }
        });

        try
        {
            Thread.sleep(2000);
        }
        catch (InterruptedException e)
        {
            e.printStackTrace();
        }
    }

    @AfterAll
    static void afterAll() throws IOException
    {
        client.close();
    }
}
```

search搜索

无条件搜索所有

```
GET /{index}/_search
```

搜索book索引:

```
GET /{index}/_search
```

结果:

```
{
  "took" : 1,
```

```
  "timed_out" : false,
  "_shards" : {
    "total" : 1,
    "successful" : 1,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : {
      "value" : 2,
      "relation" : "eq"
    },
    "max_score" : 1.0,
    "hits" : [
      {
        "_index" : "book",
        "_id" : "2",
        "_score" : 1.0,
        "_source" : {
          "name" : "java编程思想",
          "description" : "java语言是世界第一编程语言，在软件开发领域使用人数最多。",
          "studymodel" : "201001",
          "price" : 68.6,
          "timestamp" : "2019-08-25 19:11:35",
          "pic" : "group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg",
          "tags" : [
            "java",
            "dev"
          ]
        }
      },
      {
        "_index" : "book",
        "_id" : "3",
        "_score" : 1.0,
        "_source" : {
          "name" : "spring开发基础",
          "description" : "spring 在java领域非常流行，java程序员都在用。",
          "studymodel" : "201001",
          "price" : 78.6,
          "timestamp" : "2019-08-24 19:21:35",
          "pic" : "group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg",
          "tags" : [
            "spring",
            "java"
          ]
        }
      }
    ]
  }
}
```


- took：耗费了几毫秒
- timed_out：是否超时

- _shards：到几个分片搜索，成功几个，跳过几个，失败几个。
- hits.total：查询结果的数量
- hits.max_score：score的含义，就是document对于一个search的相关度的匹配分数，越相关，就越匹配，分数也高
- hits.hits：包含了匹配搜索的document的所有详细数据

带参数搜索

```
GET /{index}/_search?q=字段名:值&sort=要按什么排序的字段名:desc
```

类似于sql：select * from {index} where 字段名 like ' %值%' order by 要按什么排序的字段名 desc

```
get /book/_search?q=description:java
```

结果：


```
{
  "took" : 11,
  "timed_out" : false,
  "_shards" : {
    "total" : 1,
    "successful" : 1,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : {
      "value" : 2,
      "relation" : "eq"
    },
    "max_score" : 0.26718774,
    "hits" : [
      {
        "_index" : "book",
        "_id" : "3",
        "_score" : 0.26718774,
        "_source" : {
          "name" : "spring开发基础",
          "description" : "spring 在java领域非常流行，java程序员都在用。",
          "studymodel" : "201001",
          "price" : 78.6,
          "timestamp" : "2019-08-24 19:21:35",
          "pic" : "group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg",
          "tags" : [
            "spring",
            "java"
          ]
        }
      },
      {
        "_index" : "book",
        "_id" : "2",
```

```
        "_score" : 0.16729812,
        "_source" : {
          "name" : "java编程思想",
          "description" : "java语言是世界第一编程语言，在软件开发领域使用人数最多。",
          "studymodel" : "201001",
          "price" : 68.6,
          "timestamp" : "2019-08-25 19:11:35",
          "pic" : "group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg",
          "tags" : [
            "java",
            "dev"
          ]
        }
      }
    ]
  }
}
```


_all metadata搜索

直接可以搜索所有的field，任意一个field包含指定的关键字就可以搜索出来

```
GET /{index}/_search?q=搜索关键字
```

```
get /book/_search?q=java
```

搜索所有text字段包含java的信息

多索引搜索

- /_search：所有索引下的所有数据都搜索出来
- /index1/_search：指定一个index，搜索其下所有的数据
- /index1,index2/_search：同时搜索两个index下的数据
- /index*/_search：按照通配符去匹配多个索引

分页搜索

GET /{index}/_search?size=10

GET /{index}/_search?size=10&from=0

GET /{index}/_search?size=10&from=20

不分页:


```
{
  "took" : 0,
  "timed_out" : false,
  "_shards" : {
    "total" : 1,
    "successful" : 1,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : {
      "value" : 11,
      "relation" : "eq"
    },
    "max_score" : 1.0,
    "hits" : [
      {
        "_index" : "book",
        "_id" : "2",
        "_score" : 1.0,
        "_source" : {
          "name" : "java编程思想",
          "description" : "java语言是世界第一编程语言，在软件开发领域使用人数最多。",
          "studymodel" : "201001",
          "price" : 68.6,
          "timestamp" : "2019-08-25 19:11:35",
          "pic" : "group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg",
          "tags" : [
            "java",
            "dev"
          ]
        }
      },
      {
        "_index" : "book",
        "_id" : "3",
        "_score" : 1.0,
        "_source" : {
          "name" : "spring开发基础",
          "description" : "spring 在java领域非常流行，java程序员都在用。",
          "studymodel" : "201001",
          "price" : 78.6,
          "timestamp" : "2019-08-24 19:21:35",
          "pic" : "group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg",
          "tags" : [
            "spring",
            "java"
          ]
        }
      },
      {
        "_index" : "book",
        "_id" : "5",
        "_score" : 1.0,
        "_source" : {
```

```
        "name" : "java编程思想",
        "description" : "java语言是世界第一编程语言，在软件开发领域使用人数最多。",
        "studymodel" : "201001",
        "price" : 68.6,
        "timestamp" : "2022-5-25 19:11:35",
        "pic" : "group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg",
        "tags" : [
          "bootstrap",
          "dev"
        ]
      }
    },
    {
      "_index" : "book",
      "_id" : "6",
      "_score" : 1.0,
      "_source" : {
        "name" : "java编程思想",
        "description" : "java语言是世界第一编程语言，在软件开发领域使用人数最多。",
        "studymodel" : "201001",
        "price" : 68.6,
        "timestamp" : "2022-5-25 19:11:35",
        "pic" : "group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg",
        "tags" : [
          "bootstrap",
          "dev"
        ]
      }
    },
    {
      "_index" : "book",
      "_id" : "7",
      "_score" : 1.0,
      "_source" : {
        "name" : "java编程思想",
        "description" : "java语言是世界第一编程语言，在软件开发领域使用人数最多。",
        "studymodel" : "201001",
        "price" : 68.6,
        "timestamp" : "2022-5-25 19:11:35",
        "pic" : "group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg",
        "tags" : [
          "bootstrap",
          "dev"
        ]
      }
    },
    {
      "_index" : "book",
      "_id" : "8",
      "_score" : 1.0,
      "_source" : {
        "name" : "java编程思想",
        "description" : "java语言是世界第一编程语言，在软件开发领域使用人数最多。",
        "studymodel" : "201001",
        "price" : 68.6,
        "timestamp" : "2022-5-25 19:11:35",
        "pic" : "group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg",
        "tags" : [
```

```
            "bootstrap",
            "dev"
          ]
        }
      },
      {
        "_index" : "book",
        "_id" : "9",
        "_score" : 1.0,
        "_source" : {
          "name" : "java编程思想",
          "description" : "java语言是世界第一编程语言，在软件开发领域使用人数最多。",
          "studymodel" : "201001",
          "price" : 68.6,
          "timestamp" : "2022-5-25 19:11:35",
          "pic" : "group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg",
          "tags" : [
            "bootstrap",
            "dev"
          ]
        }
      },
      {
        "_index" : "book",
        "_id" : "10",
        "_score" : 1.0,
        "_source" : {
          "name" : "java编程思想",
          "description" : "java语言是世界第一编程语言，在软件开发领域使用人数最多。",
          "studymodel" : "201001",
          "price" : 68.6,
          "timestamp" : "2022-5-25 19:11:35",
          "pic" : "group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg",
          "tags" : [
            "bootstrap",
            "dev"
          ]
        }
      },
      {
        "_index" : "book",
        "_id" : "11",
        "_score" : 1.0,
        "_source" : {
          "name" : "java编程思想",
          "description" : "java语言是世界第一编程语言，在软件开发领域使用人数最多。",
          "studymodel" : "201001",
          "price" : 68.6,
          "timestamp" : "2022-5-25 19:11:35",
          "pic" : "group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg",
          "tags" : [
            "bootstrap",
            "dev"
          ]
        }
      },
      {
        "_index" : "book",
```

```
        "_id" : "12",
        "_score" : 1.0,
        "_source" : {
          "name" : "java编程思想",
          "description" : "java语言是世界第一编程语言，在软件开发领域使用人数最多。",
          "studymodel" : "201001",
          "price" : 68.6,
          "timestamp" : "2022-5-25 19:11:35",
          "pic" : "group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg",
          "tags" : [
            "bootstrap",
            "dev"
          ]
        }
      }
    ]
  }
}
```


```
get /book/_search?size=5&from=0
```

结果:


```
{
  "took" : 0,
  "timed_out" : false,
  "_shards" : {
    "total" : 1,
    "successful" : 1,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : {
      "value" : 11,
      "relation" : "eq"
    },
    "max_score" : 1.0,
    "hits" : [
      {
        "_index" : "book",
        "_id" : "2",
        "_score" : 1.0,
        "_source" : {
          "name" : "java编程思想",
          "description" : "java语言是世界第一编程语言，在软件开发领域使用人数最多。",
          "studymodel" : "201001",
          "price" : 68.6,
          "timestamp" : "2019-08-25 19:11:35",
          "pic" : "group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg",
          "tags" : [
            "java",
            "dev"
```

```
        ]
      }
    },
    {
      "_index" : "book",
      "_id" : "3",
      "_score" : 1.0,
      "_source" : {
        "name" : "spring开发基础",
        "description" : "spring 在java领域非常流行，java程序员都在用。",
        "studymodel" : "201001",
        "price" : 78.6,
        "timestamp" : "2019-08-24 19:21:35",
        "pic" : "group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg",
        "tags" : [
          "spring",
          "java"
        ]
      }
    },
    {
      "_index" : "book",
      "_id" : "5",
      "_score" : 1.0,
      "_source" : {
        "name" : "java编程思想",
        "description" : "java语言是世界第一编程语言，在软件开发领域使用人数最多。",
        "studymodel" : "201001",
        "price" : 68.6,
        "timestamp" : "2022-5-25 19:11:35",
        "pic" : "group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg",
        "tags" : [
          "bootstrap",
          "dev"
        ]
      }
    },
    {
      "_index" : "book",
      "_id" : "6",
      "_score" : 1.0,
      "_source" : {
        "name" : "java编程思想",
        "description" : "java语言是世界第一编程语言，在软件开发领域使用人数最多。",
        "studymodel" : "201001",
        "price" : 68.6,
        "timestamp" : "2022-5-25 19:11:35",
        "pic" : "group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg",
        "tags" : [
          "bootstrap",
          "dev"
        ]
      }
    },
    {
      "_index" : "book",
      "_id" : "7",
      "_score" : 1.0,
```

```
        "_source" : {
          "name" : "java编程思想",
          "description" : "java语言是世界第一编程语言，在软件开发领域使用人数最多。",
          "studymodel" : "201001",
          "price" : 68.6,
          "timestamp" : "2022-5-25 19:11:35",
          "pic" : "group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg",
          "tags" : [
            "bootstrap",
            "dev"
          ]
        }
      }
    ]
  }
}
```


```
get /book/_search?size=5&from=3
```

结果:


```
{
  "took" : 0,
  "timed_out" : false,
  "_shards" : {
    "total" : 1,
    "successful" : 1,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : {
      "value" : 11,
      "relation" : "eq"
    },
    "max_score" : 1.0,
    "hits" : [
      {
        "_index" : "book",
        "_id" : "6",
        "_score" : 1.0,
        "_source" : {
          "name" : "java编程思想",
          "description" : "java语言是世界第一编程语言，在软件开发领域使用人数最多。",
          "studymodel" : "201001",
          "price" : 68.6,
          "timestamp" : "2022-5-25 19:11:35",
          "pic" : "group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg",
          "tags" : [
            "bootstrap",
            "dev"
          ]
        }
```

```
    },
    {
      "_index" : "book",
      "_id" : "7",
      "_score" : 1.0,
      "_source" : {
        "name" : "java编程思想",
        "description" : "java语言是世界第一编程语言，在软件开发领域使用人数最多。",
        "studymodel" : "201001",
        "price" : 68.6,
        "timestamp" : "2022-5-25 19:11:35",
        "pic" : "group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg",
        "tags" : [
          "bootstrap",
          "dev"
        ]
      }
    },
    {
      "_index" : "book",
      "_id" : "8",
      "_score" : 1.0,
      "_source" : {
        "name" : "java编程思想",
        "description" : "java语言是世界第一编程语言，在软件开发领域使用人数最多。",
        "studymodel" : "201001",
        "price" : 68.6,
        "timestamp" : "2022-5-25 19:11:35",
        "pic" : "group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg",
        "tags" : [
          "bootstrap",
          "dev"
        ]
      }
    },
    {
      "_index" : "book",
      "_id" : "9",
      "_score" : 1.0,
      "_source" : {
        "name" : "java编程思想",
        "description" : "java语言是世界第一编程语言，在软件开发领域使用人数最多。",
        "studymodel" : "201001",
        "price" : 68.6,
        "timestamp" : "2022-5-25 19:11:35",
        "pic" : "group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg",
        "tags" : [
          "bootstrap",
          "dev"
        ]
      }
    },
    {
      "_index" : "book",
      "_id" : "10",
      "_score" : 1.0,
      "_source" : {
        "name" : "java编程思想",
```

```
          "description" : "java语言是世界第一编程语言，在软件开发领域使用人数最多。",
          "studymodel" : "201001",
          "price" : 68.6,
          "timestamp" : "2022-5-25 19:11:35",
          "pic" : "group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg",
          "tags" : [
            "bootstrap",
            "dev"
          ]
        }
      }
    ]
  }
}
```

deep paging

deep paging是什么?

简单来说，就是搜索特别深，比如总共有60000条数据，每个shard分了20000条数据，每页10条，要搜索到第1000页，所以每个shard都要将第0~10010条返回给coordinate node，然后coordinate node收到总共30030条数据，然后在这些数据中排序，_score，相关度分数，然后取到排位最高的前10条，也就是我们最后要的第1000页的10条数据。

deep paging带来的问题:

- 消耗网络带宽，因为所搜过深的话，各 shard 要把数据传递给 coordinate node，这个过程是有大量数据传递的，消耗网络。
- 消耗内存资源，各 shard 要把数据传送给 coordinate node，这个传递回来的数据，是被 coordinate node 保存在内存中的，这样会大量消耗内存。
- 消耗CPU资源，coordinate node 要把传回来的数据进行排序，这个排序过程很消耗CPU。
  所以：鉴于deep paging的性能问题，所有应尽量减少使用。

query DSL

DSL:Domain Specified Language，特定领域的语言

es特有的搜索语言，可在请求体中携带搜索条件，功能强大。

查询全部

```
GET /{index}/_search
{
  "query": { "match_all": {} }
}
```

```
GET /book/_search
{
  "query": { "match_all": {} }
}
```

结果:


```
{
  "took" : 0,
  "timed_out" : false,
  "_shards" : {
    "total" : 1,
    "successful" : 1,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : {
      "value" : 11,
      "relation" : "eq"
    },
    "max_score" : 1.0,
    "hits" : [
      {
        "_index" : "book",
        "_id" : "2",
        "_score" : 1.0,
        "_source" : {
          "name" : "java编程思想",
          "description" : "java语言是世界第一编程语言，在软件开发领域使用人数最多。",
          "studymodel" : "201001",
          "price" : 68.6,
          "timestamp" : "2019-08-25 19:11:35",
          "pic" : "group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg",
          "tags" : [
            "java",
            "dev"
          ]
        }
      },
      {
        "_index" : "book",
        "_id" : "3",
        "_score" : 1.0,
        "_source" : {
          "name" : "spring开发基础",
          "description" : "spring 在java领域非常流行，java程序员都在用。",
          "studymodel" : "201001",
          "price" : 78.6,
          "timestamp" : "2019-08-24 19:21:35",
          "pic" : "group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg",
          "tags" : [
            "spring",
            "java"
          ]
        }
```

```
    },
    {
      "_index" : "book",
      "_id" : "5",
      "_score" : 1.0,
      "_source" : {
        "name" : "java编程思想",
        "description" : "java语言是世界第一编程语言，在软件开发领域使用人数最多。",
        "studymodel" : "201001",
        "price" : 68.6,
        "timestamp" : "2022-5-25 19:11:35",
        "pic" : "group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg",
        "tags" : [
          "bootstrap",
          "dev"
        ]
      }
    },
    {
      "_index" : "book",
      "_id" : "6",
      "_score" : 1.0,
      "_source" : {
        "name" : "java编程思想",
        "description" : "java语言是世界第一编程语言，在软件开发领域使用人数最多。",
        "studymodel" : "201001",
        "price" : 68.6,
        "timestamp" : "2022-5-25 19:11:35",
        "pic" : "group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg",
        "tags" : [
          "bootstrap",
          "dev"
        ]
      }
    },
    {
      "_index" : "book",
      "_id" : "7",
      "_score" : 1.0,
      "_source" : {
        "name" : "java编程思想",
        "description" : "java语言是世界第一编程语言，在软件开发领域使用人数最多。",
        "studymodel" : "201001",
        "price" : 68.6,
        "timestamp" : "2022-5-25 19:11:35",
        "pic" : "group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg",
        "tags" : [
          "bootstrap",
          "dev"
        ]
      }
    },
    {
      "_index" : "book",
      "_id" : "8",
      "_score" : 1.0,
      "_source" : {
        "name" : "java编程思想",
```

```
      "description" : "java语言是世界第一编程语言，在软件开发领域使用人数最多。",
      "studymodel" : "201001",
      "price" : 68.6,
      "timestamp" : "2022-5-25 19:11:35",
      "pic" : "group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg",
      "tags" : [
        "bootstrap",
        "dev"
      ]
    }
  },
  {
    "_index" : "book",
    "_id" : "9",
    "_score" : 1.0,
    "_source" : {
      "name" : "java编程思想",
      "description" : "java语言是世界第一编程语言，在软件开发领域使用人数最多。",
      "studymodel" : "201001",
      "price" : 68.6,
      "timestamp" : "2022-5-25 19:11:35",
      "pic" : "group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg",
      "tags" : [
        "bootstrap",
        "dev"
      ]
    }
  },
  {
    "_index" : "book",
    "_id" : "10",
    "_score" : 1.0,
    "_source" : {
      "name" : "java编程思想",
      "description" : "java语言是世界第一编程语言，在软件开发领域使用人数最多。",
      "studymodel" : "201001",
      "price" : 68.6,
      "timestamp" : "2022-5-25 19:11:35",
      "pic" : "group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg",
      "tags" : [
        "bootstrap",
        "dev"
      ]
    }
  },
  {
    "_index" : "book",
    "_id" : "11",
    "_score" : 1.0,
    "_source" : {
      "name" : "java编程思想",
      "description" : "java语言是世界第一编程语言，在软件开发领域使用人数最多。",
      "studymodel" : "201001",
      "price" : 68.6,
      "timestamp" : "2022-5-25 19:11:35",
      "pic" : "group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg",
      "tags" : [
        "bootstrap",
```

```
            "dev"
          ]
        }
      },
      {
        "_index" : "book",
        "_id" : "12",
        "_score" : 1.0,
        "_source" : {
          "name" : "java编程思想",
          "description" : "java语言是世界第一编程语言，在软件开发领域使用人数最多。",
          "studymodel" : "201001",
          "price" : 68.6,
          "timestamp" : "2022-5-25 19:11:35",
          "pic" : "group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg",
          "tags" : [
            "bootstrap",
            "dev"
          ]
        }
      }
    ]
  }
}
```


查询某字段

语法:

```
GET /{index}/_search
{
  "query": {
    "match": {
      "字段名": "值"
    }
  }
}
```

```
GET /book/_search
{
  "query": {
    "match": {
      "description": "java"
    }
  }
}
```

结果:


```
{
  "took" : 0,
```

```
"timed_out" : false,
"_shards" : {
  "total" : 1,
  "successful" : 1,
  "skipped" : 0,
  "failed" : 0
},
"hits" : {
  "total" : {
    "value" : 11,
    "relation" : "eq"
  },
  "max_score" : 0.06467382,
  "hits" : [
    {
      "_index" : "book",
      "_id" : "3",
      "_score" : 0.06467382,
      "_source" : {
        "name" : "spring开发基础",
        "description" : "spring 在java领域非常流行，java程序员都在用。",
        "studymodel" : "201001",
        "price" : 78.6,
        "timestamp" : "2019-08-24 19:21:35",
        "pic" : "group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg",
        "tags" : [
          "spring",
          "java"
        ]
      }
    },
    {
      "_index" : "book",
      "_id" : "2",
      "_score" : 0.04197857,
      "_source" : {
        "name" : "java编程思想",
        "description" : "java语言是世界第一编程语言，在软件开发领域使用人数最多。",
        "studymodel" : "201001",
        "price" : 68.6,
        "timestamp" : "2019-08-25 19:11:35",
        "pic" : "group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg",
        "tags" : [
          "java",
          "dev"
        ]
      }
    },
    {
      "_index" : "book",
      "_id" : "5",
      "_score" : 0.04197857,
      "_source" : {
        "name" : "java编程思想",
        "description" : "java语言是世界第一编程语言，在软件开发领域使用人数最多。",
        "studymodel" : "201001",
        "price" : 68.6,
        "timestamp" : "2022-5-25 19:11:35",
```

```
      "pic" : "group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg",
      "tags" : [
        "bootstrap",
        "dev"
      ]
    }
  },
  {
    "_index" : "book",
    "_id" : "6",
    "_score" : 0.04197857,
    "_source" : {
      "name" : "java编程思想",
      "description" : "java语言是世界第一编程语言，在软件开发领域使用人数最多。",
      "studymodel" : "201001",
      "price" : 68.6,
      "timestamp" : "2022-5-25 19:11:35",
      "pic" : "group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg",
      "tags" : [
        "bootstrap",
        "dev"
      ]
    }
  },
  {
    "_index" : "book",
    "_id" : "7",
    "_score" : 0.04197857,
    "_source" : {
      "name" : "java编程思想",
      "description" : "java语言是世界第一编程语言，在软件开发领域使用人数最多。",
      "studymodel" : "201001",
      "price" : 68.6,
      "timestamp" : "2022-5-25 19:11:35",
      "pic" : "group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg",
      "tags" : [
        "bootstrap",
        "dev"
      ]
    }
  },
  {
    "_index" : "book",
    "_id" : "8",
    "_score" : 0.04197857,
    "_source" : {
      "name" : "java编程思想",
      "description" : "java语言是世界第一编程语言，在软件开发领域使用人数最多。",
      "studymodel" : "201001",
      "price" : 68.6,
      "timestamp" : "2022-5-25 19:11:35",
      "pic" : "group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg",
      "tags" : [
        "bootstrap",
        "dev"
      ]
    }
  },
```

```
    {
      "_index" : "book",
      "_id" : "9",
      "_score" : 0.04197857,
      "_source" : {
        "name" : "java编程思想",
        "description" : "java语言是世界第一编程语言，在软件开发领域使用人数最多。",
        "studymodel" : "201001",
        "price" : 68.6,
        "timestamp" : "2022-5-25 19:11:35",
        "pic" : "group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg",
        "tags" : [
          "bootstrap",
          "dev"
        ]
      }
    },
    {
      "_index" : "book",
      "_id" : "10",
      "_score" : 0.04197857,
      "_source" : {
        "name" : "java编程思想",
        "description" : "java语言是世界第一编程语言，在软件开发领域使用人数最多。",
        "studymodel" : "201001",
        "price" : 68.6,
        "timestamp" : "2022-5-25 19:11:35",
        "pic" : "group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg",
        "tags" : [
          "bootstrap",
          "dev"
        ]
      }
    },
    {
      "_index" : "book",
      "_id" : "11",
      "_score" : 0.04197857,
      "_source" : {
        "name" : "java编程思想",
        "description" : "java语言是世界第一编程语言，在软件开发领域使用人数最多。",
        "studymodel" : "201001",
        "price" : 68.6,
        "timestamp" : "2022-5-25 19:11:35",
        "pic" : "group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg",
        "tags" : [
          "bootstrap",
          "dev"
        ]
      }
    },
    {
      "_index" : "book",
      "_id" : "12",
      "_score" : 0.04197857,
      "_source" : {
        "name" : "java编程思想",
        "description" : "java语言是世界第一编程语言，在软件开发领域使用人数最多。",
```

```
          "studymodel" : "201001",
          "price" : 68.6,
          "timestamp" : "2022-5-25 19:11:35",
          "pic" : "group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg",
          "tags" : [
            "bootstrap",
            "dev"
          ]
        }
      }
    ]
  }
}
```


排序

语法:

```
GET /{index}/_search
{
  "query": {
    "match": {
      "字段名": "值"
    }
  },
  "sort": [
    {
      "字段名": {
        "order": "升序或者降序（asc，desc）"
      }
    }
  ]
}
```

```
GET /book/_search
{
  "query": {
    "match": {
      "description": "java"
    }
  },
  "sort": [
    {
      "price": {
        "order": "desc"
      }
    }
  ]
}
```

结果:

```
{
  "took" : 3,
  "timed_out" : false,
  "_shards" : {
    "total" : 1,
    "successful" : 1,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : {
      "value" : 11,
      "relation" : "eq"
    },
    "max_score" : null,
    "hits" : [
      {
        "_index" : "book",
        "_id" : "3",
        "_score" : null,
        "_source" : {
          "name" : "spring开发基础",
          "description" : "spring 在java领域非常流行，java程序员都在用。",
          "studymodel" : "201001",
          "price" : 78.6,
          "timestamp" : "2019-08-24 19:21:35",
          "pic" : "group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg",
          "tags" : [
            "spring",
            "java"
          ]
        },
        "sort" : [
          78.6
        ]
      },
      {
        "_index" : "book",
        "_id" : "2",
        "_score" : null,
        "_source" : {
          "name" : "java编程思想",
          "description" : "java语言是世界第一编程语言，在软件开发领域使用人数最多。",
          "studymodel" : "201001",
          "price" : 68.6,
          "timestamp" : "2019-08-25 19:11:35",
          "pic" : "group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg",
          "tags" : [
            "java",
            "dev"
          ]
        },
        "sort" : [
          68.6
        ]
      },
      {
```

```
      "_index" : "book",
      "_id" : "5",
      "_score" : null,
      "_source" : {
        "name" : "java编程思想",
        "description" : "java语言是世界第一编程语言，在软件开发领域使用人数最多。",
        "studymodel" : "201001",
        "price" : 68.6,
        "timestamp" : "2022-5-25 19:11:35",
        "pic" : "group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg",
        "tags" : [
          "bootstrap",
          "dev"
        ]
      },
      "sort" : [
        68.6
      ]
    },
    {
      "_index" : "book",
      "_id" : "6",
      "_score" : null,
      "_source" : {
        "name" : "java编程思想",
        "description" : "java语言是世界第一编程语言，在软件开发领域使用人数最多。",
        "studymodel" : "201001",
        "price" : 68.6,
        "timestamp" : "2022-5-25 19:11:35",
        "pic" : "group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg",
        "tags" : [
          "bootstrap",
          "dev"
        ]
      },
      "sort" : [
        68.6
      ]
    },
    {
      "_index" : "book",
      "_id" : "7",
      "_score" : null,
      "_source" : {
        "name" : "java编程思想",
        "description" : "java语言是世界第一编程语言，在软件开发领域使用人数最多。",
        "studymodel" : "201001",
        "price" : 68.6,
        "timestamp" : "2022-5-25 19:11:35",
        "pic" : "group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg",
        "tags" : [
          "bootstrap",
          "dev"
        ]
      },
      "sort" : [
        68.6
      ]
```

```
    },
    {
      "_index" : "book",
      "_id" : "8",
      "_score" : null,
      "_source" : {
        "name" : "java编程思想",
        "description" : "java语言是世界第一编程语言，在软件开发领域使用人数最多。",
        "studymodel" : "201001",
        "price" : 68.6,
        "timestamp" : "2022-5-25 19:11:35",
        "pic" : "group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg",
        "tags" : [
          "bootstrap",
          "dev"
        ]
      },
      "sort" : [
        68.6
      ]
    },
    {
      "_index" : "book",
      "_id" : "9",
      "_score" : null,
      "_source" : {
        "name" : "java编程思想",
        "description" : "java语言是世界第一编程语言，在软件开发领域使用人数最多。",
        "studymodel" : "201001",
        "price" : 68.6,
        "timestamp" : "2022-5-25 19:11:35",
        "pic" : "group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg",
        "tags" : [
          "bootstrap",
          "dev"
        ]
      },
      "sort" : [
        68.6
      ]
    },
    {
      "_index" : "book",
      "_id" : "10",
      "_score" : null,
      "_source" : {
        "name" : "java编程思想",
        "description" : "java语言是世界第一编程语言，在软件开发领域使用人数最多。",
        "studymodel" : "201001",
        "price" : 68.6,
        "timestamp" : "2022-5-25 19:11:35",
        "pic" : "group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg",
        "tags" : [
          "bootstrap",
          "dev"
        ]
      },
      "sort" : [
```

```
          68.6
        ]
      },
      {
        "_index" : "book",
        "_id" : "11",
        "_score" : null,
        "_source" : {
          "name" : "java编程思想",
          "description" : "java语言是世界第一编程语言，在软件开发领域使用人数最多。",
          "studymodel" : "201001",
          "price" : 68.6,
          "timestamp" : "2022-5-25 19:11:35",
          "pic" : "group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg",
          "tags" : [
            "bootstrap",
            "dev"
          ]
        },
        "sort" : [
          68.6
        ]
      },
      {
        "_index" : "book",
        "_id" : "12",
        "_score" : null,
        "_source" : {
          "name" : "java编程思想",
          "description" : "java语言是世界第一编程语言，在软件开发领域使用人数最多。",
          "studymodel" : "201001",
          "price" : 68.6,
          "timestamp" : "2022-5-25 19:11:35",
          "pic" : "group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg",
          "tags" : [
            "bootstrap",
            "dev"
          ]
        },
        "sort" : [
          68.6
        ]
      }
    ]
  }
}
```


分页查询

语法:

```
GET /{index}/_search
{
  "query": {
    "match_all": {}
  },
  "from": 从哪一条开始分页,
  "size": 页大小
}
```

```
GET /book/_search
{
  "query": {
    "match_all": {}
  },
  "from": 1,
  "size": 2
}
```

结果:


```
{
  "took" : 1,
  "timed_out" : false,
  "_shards" : {
    "total" : 1,
    "successful" : 1,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : {
      "value" : 4,
      "relation" : "eq"
    },
    "max_score" : 1.0,
    "hits" : [
      {
        "_index" : "book",
        "_id" : "3",
        "_score" : 1.0,
        "_source" : {
          "name" : "spring开发基础",
          "description" : "spring 在java领域非常流行，java程序员都在用。",
          "studymodel" : "201001",
          "price" : 78.6,
          "timestamp" : "2019-08-24 19:21:35",
          "pic" : "group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg",
          "tags" : [
            "spring",
            "java"
          ]
        }
      },
      {
```

```
        "_index" : "book",
        "_id" : "5",
        "_score" : 1.0,
        "_source" : {
          "name" : "java编程思想",
          "description" : "java语言是世界第一编程语言，在软件开发领域使用人数最多。",
          "studymodel" : "201001",
          "price" : 68.6,
          "timestamp" : "2022-5-25 19:11:35",
          "pic" : "group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg",
          "tags" : [
            "bootstrap",
            "dev"
          ]
        }
      }
    ]
  }
}
```


指定返回字段

语法:

```
GET /{index}/_search
{
  "query": {
    "match_all": {}
  },
  "_source": ["字段名","字段名"...]
}
```

```
GET /book/_search
{
  "query": {
    "match_all": {}
  },
  "_source": ["name","price"]
}
```

结果:


```
{
  "took" : 1,
  "timed_out" : false,
  "_shards" : {
    "total" : 1,
    "successful" : 1,
    "skipped" : 0,
    "failed" : 0
  },
```

```
  "hits" : {
    "total" : {
      "value" : 4,
      "relation" : "eq"
    },
    "max_score" : 1.0,
    "hits" : [
      {
        "_index" : "book",
        "_id" : "2",
        "_score" : 1.0,
        "_source" : {
          "price" : 68.6,
          "name" : "java编程思想"
        }
      },
      {
        "_index" : "book",
        "_id" : "3",
        "_score" : 1.0,
        "_source" : {
          "price" : 78.6,
          "name" : "spring开发基础"
        }
      },
      {
        "_index" : "book",
        "_id" : "5",
        "_score" : 1.0,
        "_source" : {
          "price" : 68.6,
          "name" : "java编程思想"
        }
      },
      {
        "_index" : "book",
        "_id" : "6",
        "_score" : 1.0,
        "_source" : {
          "price" : 68.6,
          "name" : "java编程思想"
        }
      }
    ]
  }
}
```

多搜索条件

语法:

```
GET /{index}/_search
{
  "query": {
```

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

```
GET /book/_search
{
  "query": {
    "bool": {
      "must": [
        {
          "match": {
            "name": "编程"
          }
        }
      ],
      "should": [
        {
          "match": {
            "name": "spring开发基础"
          }
        }
      ]
    }
  }
}
```

结果:

```
{
  "took" : 1,
  "timed_out" : false,
  "_shards" : {
```

```
    "total" : 1,
    "successful" : 1,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : {
      "value" : 3,
      "relation" : "eq"
    },
    "max_score" : 0.7133499,
    "hits" : [
      {
        "_index" : "book",
        "_id" : "2",
        "_score" : 0.7133499,
        "_source" : {
          "name" : "java编程思想",
          "description" : "java语言是世界第一编程语言，在软件开发领域使用人数最多。",
          "studymodel" : "201001",
          "price" : 68.6,
          "timestamp" : "2019-08-25 19:11:35",
          "pic" : "group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg",
          "tags" : [
            "java",
            "dev"
          ]
        }
      },
      {
        "_index" : "book",
        "_id" : "5",
        "_score" : 0.7133499,
        "_source" : {
          "name" : "java编程思想",
          "description" : "java语言是世界第一编程语言，在软件开发领域使用人数最多。",
          "studymodel" : "201001",
          "price" : 68.6,
          "timestamp" : "2022-5-25 19:11:35",
          "pic" : "group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg",
          "tags" : [
            "bootstrap",
            "dev"
          ]
        }
      },
      {
        "_index" : "book",
        "_id" : "6",
        "_score" : 0.7133499,
        "_source" : {
          "name" : "java编程思想",
          "description" : "java语言是世界第一编程语言，在软件开发领域使用人数最多。",
          "studymodel" : "201001",
          "price" : 68.6,
          "timestamp" : "2022-5-25 19:11:35",
          "pic" : "group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg",
          "tags" : [
```

```
            "bootstrap",
            "dev"
          ]
        }
      }
    ]
  }
}
```

match_all

获得全部数据

```
GET /book/_search
{
    "query": {
        "match_all": {}
    }
}
```

结果:


```
{
  "took" : 0,
  "timed_out" : false,
  "_shards" : {
    "total" : 1,
    "successful" : 1,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : {
      "value" : 5,
      "relation" : "eq"
    },
    "max_score" : 1.0,
    "hits" : [
      {
        "_index" : "book",
        "_id" : "1",
        "_score" : 1.0,
        "_source" : {
          "name" : "Bootstrap开发",
          "description" : "Bootstrap是由Twitter推出的一个前台页面开发css框架，是一个非常流行的开发框架，此框架集成了多种页面效果。此开发框架包含了大量的CSS、JS程序代码，可以帮助开发者（尤其是不擅长css页面开发的程序人员）轻松的实现一个css，不受浏览器限制的精美界面css效果。",
          "studymodel" : "201002",
          "price" : 38.6,
          "timestamp" : "2019-08-25 19:11:35",
          "pic" : "group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg",
          "tags" : [
```

```
            "bootstrap",
            "dev"
          ]
        }
      },
      {
        "_index" : "book",
        "_id" : "2",
        "_score" : 1.0,
        "_source" : {
          "name" : "java编程思想",
          "description" : "java语言是世界第一编程语言，在软件开发领域使用人数最多。",
          "studymodel" : "201001",
          "price" : 68.6,
          "timestamp" : "2019-08-25 19:11:35",
          "pic" : "group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg",
          "tags" : [
            "java",
            "dev"
          ]
        }
      },
      {
        "_index" : "book",
        "_id" : "3",
        "_score" : 1.0,
        "_source" : {
          "name" : "spring开发基础",
          "description" : "spring 在java领域非常流行，java程序员都在用。",
          "studymodel" : "201001",
          "price" : 78.6,
          "timestamp" : "2019-08-24 19:21:35",
          "pic" : "group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg",
          "tags" : [
            "spring",
            "java"
          ]
        }
      },
      {
        "_index" : "book",
        "_id" : "5",
        "_score" : 1.0,
        "_source" : {
          "name" : "java编程思想",
          "description" : "java语言是世界第一编程语言，在软件开发领域使用人数最多。",
          "studymodel" : "201001",
          "price" : 68.6,
          "timestamp" : "2022-5-25 19:11:35",
          "pic" : "group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg",
          "tags" : [
            "bootstrap",
            "dev"
          ]
        }
      },
      {
        "_index" : "book",
```

```
        "_id" : "6",
        "_score" : 1.0,
        "_source" : {
          "name" : "java编程思想",
          "description" : "java语言是世界第一编程语言，在软件开发领域使用人数最多。",
          "studymodel" : "201001",
          "price" : 68.6,
          "timestamp" : "2022-5-25 19:11:35",
          "pic" : "group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg",
          "tags" : [
            "bootstrap",
            "dev"
          ]
        }
      }
    ]
  }
}
```


match

搜索某字段是否包含某关键字

```
GET /book/_search
{
    "query": {
      "match": {
        "name": "java"
      }
    }
}
```

结果:


```
{
  "took" : 0,
  "timed_out" : false,
  "_shards" : {
    "total" : 1,
    "successful" : 1,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : {
      "value" : 3,
      "relation" : "eq"
    },
    "max_score" : 0.52048135,
    "hits" : [
      {
        "_index" : "book",
        "_id" : "2",
```

```
        "_score" : 0.52048135,
        "_source" : {
          "name" : "java编程思想",
          "description" : "java语言是世界第一编程语言，在软件开发领域使用人数最多。",
          "studymodel" : "201001",
          "price" : 68.6,
          "timestamp" : "2019-08-25 19:11:35",
          "pic" : "group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg",
          "tags" : [
            "java",
            "dev"
          ]
        }
      },
      {
        "_index" : "book",
        "_id" : "5",
        "_score" : 0.52048135,
        "_source" : {
          "name" : "java编程思想",
          "description" : "java语言是世界第一编程语言，在软件开发领域使用人数最多。",
          "studymodel" : "201001",
          "price" : 68.6,
          "timestamp" : "2022-5-25 19:11:35",
          "pic" : "group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg",
          "tags" : [
            "bootstrap",
            "dev"
          ]
        }
      },
      {
        "_index" : "book",
        "_id" : "6",
        "_score" : 0.52048135,
        "_source" : {
          "name" : "java编程思想",
          "description" : "java语言是世界第一编程语言，在软件开发领域使用人数最多。",
          "studymodel" : "201001",
          "price" : 68.6,
          "timestamp" : "2022-5-25 19:11:35",
          "pic" : "group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg",
          "tags" : [
            "bootstrap",
            "dev"
          ]
        }
      }
    ]
  }
}
```

multi_match

搜索在多个字段下是否包含某关键字

```
GET /book/_search
{
    "query": {
      "multi_match": {
        "query": "语言",
        "fields": ["name","description"]
      }
    }
}
```

结果:


```
{
  "took" : 3,
  "timed_out" : false,
  "_shards" : {
    "total" : 1,
    "successful" : 1,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : {
      "value" : 3,
      "relation" : "eq"
    },
    "max_score" : 1.6503837,
    "hits" : [
      {
        "_index" : "book",
        "_id" : "2",
        "_score" : 1.6503837,
        "_source" : {
          "name" : "java编程思想",
          "description" : "java语言是世界第一编程语言，在软件开发领域使用人数最多。",
          "studymodel" : "201001",
          "price" : 68.6,
          "timestamp" : "2019-08-25 19:11:35",
          "pic" : "group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg",
          "tags" : [
            "java",
            "dev"
          ]
        }
      },
      {
        "_index" : "book",
        "_id" : "5",
        "_score" : 1.6503837,
        "_source" : {
          "name" : "java编程思想",
          "description" : "java语言是世界第一编程语言，在软件开发领域使用人数最多。",
```

```
          "studymodel" : "201001",
          "price" : 68.6,
          "timestamp" : "2022-5-25 19:11:35",
          "pic" : "group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg",
          "tags" : [
            "bootstrap",
            "dev"
          ]
        }
      },
      {
        "_index" : "book",
        "_id" : "6",
        "_score" : 1.6503837,
        "_source" : {
          "name" : "java编程思想",
          "description" : "java语言是世界第一编程语言，在软件开发领域使用人数最多。",
          "studymodel" : "201001",
          "price" : 68.6,
          "timestamp" : "2022-5-25 19:11:35",
          "pic" : "group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg",
          "tags" : [
            "bootstrap",
            "dev"
          ]
        }
      }
    ]
  }
}
```


range query

范围查询

```
GET /book/_search
{
    "query": {
      "range": {
        "price": {
          "gte": 69.2,
          "lte": 80
        }
      }
    }
}
```

结果:


```
{
  "took" : 0,
  "timed_out" : false,
  "_shards" : {
```

```
    "total" : 1,
    "successful" : 1,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : {
      "value" : 1,
      "relation" : "eq"
    },
    "max_score" : 1.0,
    "hits" : [
      {
        "_index" : "book",
        "_id" : "3",
        "_score" : 1.0,
        "_source" : {
          "name" : "spring开发基础",
          "description" : "spring 在java领域非常流行，java程序员都在用。",
          "studymodel" : "201001",
          "price" : 78.6,
          "timestamp" : "2019-08-24 19:21:35",
          "pic" : "group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg",
          "tags" : [
            "spring",
            "java"
          ]
        }
      }
    ]
  }
}
```


term query

字段为keyword时，存储和搜索都不分词

```
GET /book/_search
{
    "query": {
      "term": {
        "description": {
          "value": "java程序员"
        }
      }
    }
}
```

结果：


```
{
  "took" : 0,
  "timed_out" : false,
```

```
  "_shards" : {
    "total" : 1,
    "successful" : 1,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : {
      "value" : 0,
      "relation" : "eq"
    },
    "max_score" : null,
    "hits" : [ ]
  }
}
```

terms query

```
GET /book/_search
{
    "query": {
      "terms": {
        "FIELD": [
          "VALUE1",
          "VALUE2"
        ]
      }
    }
}
```

exist query

查询有某些字段值的文档

```
GET /book/_search
{
    "query": {
      "exists": {
        "field": "tags"
      }
    }
}
```

结果:

```
{
  "took" : 0,
  "timed_out" : false,
  "_shards" : {
    "total" : 1,
```

```
    "successful" : 1,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : {
      "value" : 5,
      "relation" : "eq"
    },
    "max_score" : 1.0,
    "hits" : [
      {
        "_index" : "book",
        "_id" : "1",
        "_score" : 1.0,
        "_source" : {
          "name" : "Bootstrap开发",
          "description" : "Bootstrap是由Twitter推出的一个前台页面开发css框架，是一个非常流行的开发框架，此框架集成了多种页面效果。此开发框架包含了大量的CSS、JS程序代码，可以帮助开发者（尤其是不擅长css页面开发的程序人员）轻松的实现一个css，不受浏览器限制的精美界面css效果。",
          "studymodel" : "201002",
          "price" : 38.6,
          "timestamp" : "2019-08-25 19:11:35",
          "pic" : "group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg",
          "tags" : [
            "bootstrap",
            "dev"
          ]
        }
      },
      {
        "_index" : "book",
        "_id" : "2",
        "_score" : 1.0,
        "_source" : {
          "name" : "java编程思想",
          "description" : "java语言是世界第一编程语言，在软件开发领域使用人数最多。",
          "studymodel" : "201001",
          "price" : 68.6,
          "timestamp" : "2019-08-25 19:11:35",
          "pic" : "group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg",
          "tags" : [
            "java",
            "dev"
          ]
        }
      },
      {
        "_index" : "book",
        "_id" : "3",
        "_score" : 1.0,
        "_source" : {
          "name" : "spring开发基础",
          "description" : "spring 在java领域非常流行，java程序员都在用。",
          "studymodel" : "201001",
          "price" : 78.6,
          "timestamp" : "2019-08-24 19:21:35",
          "pic" : "group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg",
```

```
          "tags" : [
            "spring",
            "java"
          ]
        }
      },
      {
        "_index" : "book",
        "_id" : "5",
        "_score" : 1.0,
        "_source" : {
          "name" : "java编程思想",
          "description" : "java语言是世界第一编程语言，在软件开发领域使用人数最多。",
          "studymodel" : "201001",
          "price" : 68.6,
          "timestamp" : "2022-5-25 19:11:35",
          "pic" : "group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg",
          "tags" : [
            "bootstrap",
            "dev"
          ]
        }
      },
      {
        "_index" : "book",
        "_id" : "6",
        "_score" : 1.0,
        "_source" : {
          "name" : "java编程思想",
          "description" : "java语言是世界第一编程语言，在软件开发领域使用人数最多。",
          "studymodel" : "201001",
          "price" : 68.6,
          "timestamp" : "2022-5-25 19:11:35",
          "pic" : "group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg",
          "tags" : [
            "bootstrap",
            "dev"
          ]
        }
      }
    ]
  }
}
```


Fuzzy query

返回包含与搜索词类似的词的文档，该词由Levenshtein编辑距离度量。

包括以下几种情况：

- 更改角色（box→fox）
- 删除字符（aple→apple）
- 插入字符（sick→sic）
- 调换两个相邻字符（ACT→CAT）

```
GET /book/_search
{
    "query": {
      "fuzzy": {
        "name": {
          "value": "jaav"
        }
      }
    }
}
```

结果:


```
{
  "took" : 0,
  "timed_out" : false,
  "_shards" : {
    "total" : 1,
    "successful" : 1,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : {
      "value" : 3,
      "relation" : "eq"
    },
    "max_score" : 0.39036104,
    "hits" : [
      {
        "_index" : "book",
        "_id" : "2",
        "_score" : 0.39036104,
        "_source" : {
          "name" : "java编程思想",
          "description" : "java语言是世界第一编程语言，在软件开发领域使用人数最多。",
          "studymodel" : "201001",
          "price" : 68.6,
          "timestamp" : "2019-08-25 19:11:35",
          "pic" : "group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg",
          "tags" : [
            "java",
            "dev"
          ]
        }
      },
      {
        "_index" : "book",
        "_id" : "5",
        "_score" : 0.39036104,
        "_source" : {
          "name" : "java编程思想",
          "description" : "java语言是世界第一编程语言，在软件开发领域使用人数最多。",
          "studymodel" : "201001",
          "price" : 68.6,
```

```
          "timestamp" : "2022-5-25 19:11:35",
          "pic" : "group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg",
          "tags" : [
            "bootstrap",
            "dev"
          ]
        }
      },
      {
        "_index" : "book",
        "_id" : "6",
        "_score" : 0.39036104,
        "_source" : {
          "name" : "java编程思想",
          "description" : "java语言是世界第一编程语言，在软件开发领域使用人数最多。",
          "studymodel" : "201001",
          "price" : 68.6,
          "timestamp" : "2022-5-25 19:11:35",
          "pic" : "group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg",
          "tags" : [
            "bootstrap",
            "dev"
          ]
        }
      }
    ]
  }
}
```


IDs

查询多个id为某个数的结果

```
GET /book/_search
{
    "query": {
      "ids": {
        "values": ["1","5","3","100"]
      }
    }
}
```

结果:


```
{
  "took" : 0,
  "timed_out" : false,
  "_shards" : {
    "total" : 1,
    "successful" : 1,
    "skipped" : 0,
    "failed" : 0
  },
```

```
  "hits" : {
    "total" : {
      "value" : 3,
      "relation" : "eq"
    },
    "max_score" : 1.0,
    "hits" : [
      {
        "_index" : "book",
        "_id" : "1",
        "_score" : 1.0,
        "_source" : {
          "name" : "Bootstrap开发",
          "description" : "Bootstrap是由Twitter推出的一个前台页面开发css框架，是一个非常流行的开发框架，此框架集成了多种页面效果。此开发框架包含了大量的CSS、JS程序代码，可以帮助开发者（尤其是不擅长css页面开发的程序人员）轻松的实现一个css，不受浏览器限制的精美界面css效果。",
          "studymodel" : "201002",
          "price" : 38.6,
          "timestamp" : "2019-08-25 19:11:35",
          "pic" : "group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg",
          "tags" : [
            "bootstrap",
            "dev"
          ]
        }
      },
      {
        "_index" : "book",
        "_id" : "3",
        "_score" : 1.0,
        "_source" : {
          "name" : "spring开发基础",
          "description" : "spring 在java领域非常流行，java程序员都在用。",
          "studymodel" : "201001",
          "price" : 78.6,
          "timestamp" : "2019-08-24 19:21:35",
          "pic" : "group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg",
          "tags" : [
            "spring",
            "java"
          ]
        }
      },
      {
        "_index" : "book",
        "_id" : "5",
        "_score" : 1.0,
        "_source" : {
          "name" : "java编程思想",
          "description" : "java语言是世界第一编程语言，在软件开发领域使用人数最多。",
          "studymodel" : "201001",
          "price" : 68.6,
          "timestamp" : "2022-5-25 19:11:35",
          "pic" : "group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg",
          "tags" : [
            "bootstrap",
            "dev"
          ]
```

```
        }
      }
    ]
  }
}
```

prefix 前缀查询

查询某字段满足某前缀的所有数据

```
GET /book/_search
{
    "query": {
      "prefix": {
        "description": {
          "value": "sprin"
        }
      }
    }
}
```

结果:

```
{
  "took" : 0,
  "timed_out" : false,
  "_shards" : {
    "total" : 1,
    "successful" : 1,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : {
      "value" : 1,
      "relation" : "eq"
    },
    "max_score" : 1.0,
    "hits" : [
      {
        "_index" : "book",
        "_id" : "3",
        "_score" : 1.0,
        "_source" : {
          "name" : "spring开发基础",
          "description" : "spring 在java领域非常流行，java程序员都在用。",
          "studymodel" : "201001",
          "price" : 78.6,
          "timestamp" : "2019-08-24 19:21:35",
          "pic" : "group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg",
          "tags" : [
            "spring",
            "java"
```

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

regexp query

正则查询

查询某字段满足某正则表达式的所有数据

```
GET /book/_search
{
    "query": {
      "regexp": {
        "description": {
          "value": "j.*a",
          "flags": "ALL"
        }
      }
    }
}
```

结果:

```
{
  "took" : 0,
  "timed_out" : false,
  "_shards" : {
    "total" : 1,
    "successful" : 1,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : {
      "value" : 4,
      "relation" : "eq"
    },
    "max_score" : 1.0,
    "hits" : [
      {
        "_index" : "book",
        "_id" : "2",
        "_score" : 1.0,
        "_source" : {
          "name" : "java编程思想",
          "description" : "java语言是世界第一编程语言，在软件开发领域使用人数最多。",
          "studymodel" : "201001",
          "price" : 68.6,
          "timestamp" : "2019-08-25 19:11:35",
          "pic" : "group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg",
```

```
        "tags" : [
          "java",
          "dev"
        ]
      }
    },
    {
      "_index" : "book",
      "_id" : "3",
      "_score" : 1.0,
      "_source" : {
        "name" : "spring开发基础",
        "description" : "spring 在java领域非常流行，java程序员都在用。",
        "studymodel" : "201001",
        "price" : 78.6,
        "timestamp" : "2019-08-24 19:21:35",
        "pic" : "group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg",
        "tags" : [
          "spring",
          "java"
        ]
      }
    },
    {
      "_index" : "book",
      "_id" : "5",
      "_score" : 1.0,
      "_source" : {
        "name" : "java编程思想",
        "description" : "java语言是世界第一编程语言，在软件开发领域使用人数最多。",
        "studymodel" : "201001",
        "price" : 68.6,
        "timestamp" : "2022-5-25 19:11:35",
        "pic" : "group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg",
        "tags" : [
          "bootstrap",
          "dev"
        ]
      }
    },
    {
      "_index" : "book",
      "_id" : "6",
      "_score" : 1.0,
      "_source" : {
        "name" : "java编程思想",
        "description" : "java语言是世界第一编程语言，在软件开发领域使用人数最多。",
        "studymodel" : "201001",
        "price" : 68.6,
        "timestamp" : "2022-5-25 19:11:35",
        "pic" : "group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg",
        "tags" : [
          "bootstrap",
          "dev"
        ]
      }
    }
  ]
```

```
  }
}
```

Filter

用户查询description中有"java程序员"，并且价格大于60小于70的数据。

```
GET /book/_search
{
    "query":
    {
      "bool":
      {
        "must":
        [
        {
          "match":
          {
            "description": "java程序员"
          }
        },
        {
          "range":
          {
            "price":
            {
              "gte": 60,
              "lte": 70
            }
          }
        }
      ]
      }
    }
}
```

结果：

```
{
  "took" : 0,
  "timed_out" : false,
  "_shards" : {
    "total" : 1,
    "successful" : 1,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : {
      "value" : 3,
```

```
      "relation" : "eq"
    },
    "max_score" : 1.4398797,
    "hits" : [
      {
        "_index" : "book",
        "_id" : "2",
        "_score" : 1.4398797,
        "_source" : {
          "name" : "java编程思想",
          "description" : "java语言是世界第一编程语言，在软件开发领域使用人数最多。",
          "studymodel" : "201001",
          "price" : 68.6,
          "timestamp" : "2019-08-25 19:11:35",
          "pic" : "group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg",
          "tags" : [
            "java",
            "dev"
          ]
        }
      },
      {
        "_index" : "book",
        "_id" : "5",
        "_score" : 1.4398797,
        "_source" : {
          "name" : "java编程思想",
          "description" : "java语言是世界第一编程语言，在软件开发领域使用人数最多。",
          "studymodel" : "201001",
          "price" : 68.6,
          "timestamp" : "2022-5-25 19:11:35",
          "pic" : "group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg",
          "tags" : [
            "bootstrap",
            "dev"
          ]
        }
      },
      {
        "_index" : "book",
        "_id" : "6",
        "_score" : 1.4398797,
        "_source" : {
          "name" : "java编程思想",
          "description" : "java语言是世界第一编程语言，在软件开发领域使用人数最多。",
          "studymodel" : "201001",
          "price" : 68.6,
          "timestamp" : "2022-5-25 19:11:35",
          "pic" : "group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg",
          "tags" : [
            "bootstrap",
            "dev"
          ]
        }
      }
    ]
  }
}
```

使用filter：

```
GET /book/_search
{
    "query":
    {
      "bool":
      {
        "must":
        [
        {
          "match":
          {
            "description": "java程序员"
          }
        }
      ],
      "filter":
      [
        {
          "range": {
            "price": {
              "gte": 60,
              "lte": 70
            }
          }
        }
      ]
      }
    }
}
```

结果：

```
{
  "took" : 1,
  "timed_out" : false,
  "_shards" : {
    "total" : 1,
    "successful" : 1,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : {
      "value" : 3,
      "relation" : "eq"
    },
    "max_score" : 0.4398797,
    "hits" : [
      {
        "_index" : "book",
        "_id" : "2",
```

```
        "_score" : 0.4398797,
        "_source" : {
          "name" : "java编程思想",
          "description" : "java语言是世界第一编程语言，在软件开发领域使用人数最多。",
          "studymodel" : "201001",
          "price" : 68.6,
          "timestamp" : "2019-08-25 19:11:35",
          "pic" : "group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg",
          "tags" : [
            "java",
            "dev"
          ]
        }
      },
      {
        "_index" : "book",
        "_id" : "5",
        "_score" : 0.4398797,
        "_source" : {
          "name" : "java编程思想",
          "description" : "java语言是世界第一编程语言，在软件开发领域使用人数最多。",
          "studymodel" : "201001",
          "price" : 68.6,
          "timestamp" : "2022-5-25 19:11:35",
          "pic" : "group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg",
          "tags" : [
            "bootstrap",
            "dev"
          ]
        }
      },
      {
        "_index" : "book",
        "_id" : "6",
        "_score" : 0.4398797,
        "_source" : {
          "name" : "java编程思想",
          "description" : "java语言是世界第一编程语言，在软件开发领域使用人数最多。",
          "studymodel" : "201001",
          "price" : 68.6,
          "timestamp" : "2022-5-25 19:11:35",
          "pic" : "group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg",
          "tags" : [
            "bootstrap",
            "dev"
          ]
        }
      }
    ]
  }
}
```


分数较低

对比

- filter：仅仅只是按照搜索条件过滤出需要的数据而已，不计算任何相关度分数，对相关度没有任何影响。
- query：会去计算每个document相对于搜索条件的相关度，并按照相关度进行排序。

一般来说，如果你是在进行搜索，需要将最匹配搜索条件的数据先返回，那么用query　　如果你只是要根据一些条件筛选出一部分数据，不关注其排序，那么用filter

性能：

- filter：不需要计算相关度分数，不需要按照相关度分数进行排序，同时还有内置的自动cache最常使用filter的数据
- query：相反，要计算相关度分数，按照分数进行排序，而且无法cache结果

explain

explain就像mysql的执行计划，可以看到搜索的目标等信息。

也可以定位错误语句

```
GET /book/_search?explain=true
{
    "query":
    {
      "bool":
      {
        "must":
        [
        {
          "match":
          {
            "description": "java程序员"
          }
        }
      ],
      "filter":
      [
        {
          "range": {
            "price": {
              "gte": 60,
              "lte": 70
            }
          }
        }
      ]
      }
    }
```

```
}
```

结果:


```
{
  "took" : 3,
  "timed_out" : false,
  "_shards" : {
    "total" : 1,
    "successful" : 1,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : {
      "value" : 3,
      "relation" : "eq"
    },
    "max_score" : 0.4398797,
    "hits" : [
      {
        "_shard" : "[book][0]",
        "_node" : "QrII3Cg2Sh-RnBrmVzmV9Q",
        "_index" : "book",
        "_id" : "2",
        "_score" : 0.4398797,
        "_source" : {
          "name" : "java编程思想",
          "description" : "java语言是世界第一编程语言，在软件开发领域使用人数最多。",
          "studymodel" : "201001",
          "price" : 68.6,
          "timestamp" : "2019-08-25 19:11:35",
          "pic" : "group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg",
          "tags" : [
            "java",
            "dev"
          ]
        },
        "_explanation" : {
          "value" : 0.4398797,
          "description" : "sum of:",
          "details" : [
            {
              "value" : 0.4398797,
              "description" : "sum of:",
              "details" : [
                {
                  "value" : 0.33773077,
                  "description" : "weight(description:java in 1)
[PerFieldSimilarity], result of:",
                  "details" : [
                    {
                      "value" : 0.33773077,
                      "description" : "score(freq=1.0), computed as boost * idf
* tf from:",
                      "details" : [
                        {
```

```
                        "value" : 2.2,
                        "description" : "boost",
                        "details" : [ ]
                      },
                      {
                        "value" : 0.2876821,
                        "description" : "idf, computed as log(1 + (N - n +
0.5) / (n + 0.5)) from:",
                        "details" : [
                          {
                            "value" : 4,
                            "description" : "n, number of documents containing
term",
                            "details" : [ ]
                          },
                          {
                            "value" : 5,
                            "description" : "N, total number of documents with
field",
                            "details" : [ ]
                          }
                        ]
                      },
                      {
                        "value" : 0.5336237,
                        "description" : "tf, computed as freq / (freq + k1 *
(1 - b + b * dl / avgdl)) from:",
                        "details" : [
                          {
                            "value" : 1.0,
                            "description" : "freq, occurrences of term within
document",
                            "details" : [ ]
                          },
                          {
                            "value" : 1.2,
                            "description" : "k1, term saturation parameter",
                            "details" : [ ]
                          },
                          {
                            "value" : 0.75,
                            "description" : "b, length normalization
parameter",
                            "details" : [ ]
                          },
                          {
                            "value" : 25.0,
                            "description" : "dl, length of field",
                            "details" : [ ]
                          },
                          {
                            "value" : 39.2,
                            "description" : "avgdl, average length of field",
                            "details" : [ ]
                          }
                        ]
                      }
                    ]
```

```
                    }
                  ]
                },
                {
                  "value" : 0.10214893,
                  "description" : "weight(description:程 in 1)
[PerFieldSimilarity], result of:",
                  "details" : [
                    {
                      "value" : 0.10214893,
                      "description" : "score(freq=1.0), computed as boost * idf
* tf from:",
                      "details" : [
                        {
                          "value" : 2.2,
                          "description" : "boost",
                          "details" : [ ]
                        },
                        {
                          "value" : 0.087011375,
                          "description" : "idf, computed as log(1 + (N - n +
0.5) / (n + 0.5)) from:",
                          "details" : [
                            {
                              "value" : 5,
                              "description" : "n, number of documents containing
term",
                              "details" : [ ]
                            },
                            {
                              "value" : 5,
                              "description" : "N, total number of documents with
field",
                              "details" : [ ]
                            }
                          ]
                        },
                        {
                          "value" : 0.5336237,
                          "description" : "tf, computed as freq / (freq + k1 *
(1 - b + b * dl / avgdl)) from:",
                          "details" : [
                            {
                              "value" : 1.0,
                              "description" : "freq, occurrences of term within
document",
                              "details" : [ ]
                            },
                            {
                              "value" : 1.2,
                              "description" : "k1, term saturation parameter",
                              "details" : [ ]
                            },
                            {
                              "value" : 0.75,
                              "description" : "b, length normalization
parameter",
                              "details" : [ ]
```

```
                        },
                        {
                          "value" : 25.0,
                          "description" : "dl, length of field",
                          "details" : [ ]
                        },
                        {
                          "value" : 39.2,
                          "description" : "avgdl, average length of field",
                          "details" : [ ]
                        }
                      ]
                    }
                  ]
                }
              ]
            }
          ]
        },
        {
          "value" : 0.0,
          "description" : "match on required clause, product of:",
          "details" : [
            {
              "value" : 0.0,
              "description" : "# clause",
              "details" : [ ]
            },
            {
              "value" : 1.0,
              "description" : "price:[60.0 TO 70.0]",
              "details" : [ ]
            }
          ]
        }
      ]
    }
  },
  {
    "_shard" : "[book][0]",
    "_node" : "QrII3Cg2Sh-RnBrmVzmV9Q",
    "_index" : "book",
    "_id" : "5",
    "_score" : 0.4398797,
    "_source" : {
      "name" : "java编程思想",
      "description" : "java语言是世界第一编程语言，在软件开发领域使用人数最多。",
      "studymodel" : "201001",
      "price" : 68.6,
      "timestamp" : "2022-5-25 19:11:35",
      "pic" : "group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg",
      "tags" : [
        "bootstrap",
        "dev"
      ]
    },
    "_explanation" : {
      "value" : 0.4398797,
```

```
          "description" : "sum of:",
          "details" : [
            {
              "value" : 0.4398797,
              "description" : "sum of:",
              "details" : [
                {
                  "value" : 0.33773077,
                  "description" : "weight(description:java in 3)
[PerFieldSimilarity], result of:",
                  "details" : [
                    {
                      "value" : 0.33773077,
                      "description" : "score(freq=1.0), computed as boost * idf
* tf from:",
                      "details" : [
                        {
                          "value" : 2.2,
                          "description" : "boost",
                          "details" : [ ]
                        },
                        {
                          "value" : 0.2876821,
                          "description" : "idf, computed as log(1 + (N - n +
0.5) / (n + 0.5)) from:",
                          "details" : [
                            {
                              "value" : 4,
                              "description" : "n, number of documents containing
term",
                              "details" : [ ]
                            },
                            {
                              "value" : 5,
                              "description" : "N, total number of documents with
field",
                              "details" : [ ]
                            }
                          ]
                        },
                        {
                          "value" : 0.5336237,
                          "description" : "tf, computed as freq / (freq + k1 *
(1 - b + b * dl / avgdl)) from:",
                          "details" : [
                            {
                              "value" : 1.0,
                              "description" : "freq, occurrences of term within
document",
                              "details" : [ ]
                            },
                            {
                              "value" : 1.2,
                              "description" : "k1, term saturation parameter",
                              "details" : [ ]
                            },
                            {
                              "value" : 0.75,
```

```
                                "description" : "b, length normalization
parameter",
                                "details" : [ ]
                              },
                              {
                                "value" : 25.0,
                                "description" : "dl, length of field",
                                "details" : [ ]
                              },
                              {
                                "value" : 39.2,
                                "description" : "avgdl, average length of field",
                                "details" : [ ]
                              }
                            ]
                          }
                        ]
                      }
                    ]
                  },
                  {
                    "value" : 0.10214893,
                    "description" : "weight(description:程 in 3)
[PerFieldSimilarity], result of:",
                    "details" : [
                      {
                        "value" : 0.10214893,
                        "description" : "score(freq=1.0), computed as boost * idf
* tf from:",
                        "details" : [
                          {
                            "value" : 2.2,
                            "description" : "boost",
                            "details" : [ ]
                          },
                          {
                            "value" : 0.087011375,
                            "description" : "idf, computed as log(1 + (N - n +
0.5) / (n + 0.5)) from:",
                            "details" : [
                              {
                                "value" : 5,
                                "description" : "n, number of documents containing
term",
                                "details" : [ ]
                              },
                              {
                                "value" : 5,
                                "description" : "N, total number of documents with
field",
                                "details" : [ ]
                              }
                            ]
                          },
                          {
                            "value" : 0.5336237,
                            "description" : "tf, computed as freq / (freq + k1 *
(1 - b + b * dl / avgdl)) from:",
```

```
                        "details" : [
                          {
                            "value" : 1.0,
                            "description" : "freq, occurrences of term within
document",
                            "details" : [ ]
                          },
                          {
                            "value" : 1.2,
                            "description" : "k1, term saturation parameter",
                            "details" : [ ]
                          },
                          {
                            "value" : 0.75,
                            "description" : "b, length normalization
parameter",
                            "details" : [ ]
                          },
                          {
                            "value" : 25.0,
                            "description" : "dl, length of field",
                            "details" : [ ]
                          },
                          {
                            "value" : 39.2,
                            "description" : "avgdl, average length of field",
                            "details" : [ ]
                          }
                        ]
                      }
                    ]
                  }
                ]
              }
            ]
          },
          {
            "value" : 0.0,
            "description" : "match on required clause, product of:",
            "details" : [
              {
                "value" : 0.0,
                "description" : "# clause",
                "details" : [ ]
              },
              {
                "value" : 1.0,
                "description" : "price:[60.0 TO 70.0]",
                "details" : [ ]
              }
            ]
          }
        ]
      }
    },
    {
      "_shard" : "[book][0]",
      "_node" : "QrII3Cg2Sh-RnBrmVzmV9Q",
```

```
        "_index" : "book",
        "_id" : "6",
        "_score" : 0.4398797,
        "_source" : {
          "name" : "java编程思想",
          "description" : "java语言是世界第一编程语言，在软件开发领域使用人数最多。",
          "studymodel" : "201001",
          "price" : 68.6,
          "timestamp" : "2022-5-25 19:11:35",
          "pic" : "group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg",
          "tags" : [
            "bootstrap",
            "dev"
          ]
        },
        "_explanation" : {
          "value" : 0.4398797,
          "description" : "sum of:",
          "details" : [
            {
              "value" : 0.4398797,
              "description" : "sum of:",
              "details" : [
                {
                  "value" : 0.33773077,
                  "description" : "weight(description:java in 4)
[PerFieldSimilarity], result of:",
                  "details" : [
                    {
                      "value" : 0.33773077,
                      "description" : "score(freq=1.0), computed as boost * idf
* tf from:",
                      "details" : [
                        {
                          "value" : 2.2,
                          "description" : "boost",
                          "details" : [ ]
                        },
                        {
                          "value" : 0.2876821,
                          "description" : "idf, computed as log(1 + (N - n +
0.5) / (n + 0.5)) from:",
                          "details" : [
                            {
                              "value" : 4,
                              "description" : "n, number of documents containing
term",
                              "details" : [ ]
                            },
                            {
                              "value" : 5,
                              "description" : "N, total number of documents with
field",
                              "details" : [ ]
                            }
                          ]
                        },
                        {
```

```
                        "value" : 0.5336237,
                        "description" : "tf, computed as freq / (freq + k1 *
(1 - b + b * dl / avgdl)) from:",
                        "details" : [
                          {
                            "value" : 1.0,
                            "description" : "freq, occurrences of term within
document",
                            "details" : [ ]
                          },
                          {
                            "value" : 1.2,
                            "description" : "k1, term saturation parameter",
                            "details" : [ ]
                          },
                          {
                            "value" : 0.75,
                            "description" : "b, length normalization
parameter",
                            "details" : [ ]
                          },
                          {
                            "value" : 25.0,
                            "description" : "dl, length of field",
                            "details" : [ ]
                          },
                          {
                            "value" : 39.2,
                            "description" : "avgdl, average length of field",
                            "details" : [ ]
                          }
                        ]
                      }
                    ]
                  }
                ]
              },
              {
                "value" : 0.10214893,
                "description" : "weight(description:程 in 4)
[PerFieldSimilarity], result of:",
                "details" : [
                  {
                    "value" : 0.10214893,
                    "description" : "score(freq=1.0), computed as boost * idf
* tf from:",
                    "details" : [
                      {
                        "value" : 2.2,
                        "description" : "boost",
                        "details" : [ ]
                      },
                      {
                        "value" : 0.087011375,
                        "description" : "idf, computed as log(1 + (N - n +
0.5) / (n + 0.5)) from:",
                        "details" : [
                          {
```

```
                                "value" : 5,
                                "description" : "n, number of documents containing
term",
                                "details" : [ ]
                              },
                              {
                                "value" : 5,
                                "description" : "N, total number of documents with
field",
                                "details" : [ ]
                              }
                            ]
                          },
                          {
                            "value" : 0.5336237,
                            "description" : "tf, computed as freq / (freq + k1 *
(1 - b + b * dl / avgdl)) from:",
                            "details" : [
                              {
                                "value" : 1.0,
                                "description" : "freq, occurrences of term within
document",
                                "details" : [ ]
                              },
                              {
                                "value" : 1.2,
                                "description" : "k1, term saturation parameter",
                                "details" : [ ]
                              },
                              {
                                "value" : 0.75,
                                "description" : "b, length normalization
parameter",
                                "details" : [ ]
                              },
                              {
                                "value" : 25.0,
                                "description" : "dl, length of field",
                                "details" : [ ]
                              },
                              {
                                "value" : 39.2,
                                "description" : "avgdl, average length of field",
                                "details" : [ ]
                              }
                            ]
                          }
                        ]
                      }
                    ]
                  }
                ]
              },
              {
                "value" : 0.0,
                "description" : "match on required clause, product of:",
                "details" : [
                  {
```

```
                "value" : 0.0,
                "description" : "# clause",
                "details" : [ ]
              },
              {
                "value" : 1.0,
                "description" : "price:[60.0 TO 70.0]",
                "details" : [ ]
              }
            ]
          }
        ]
      }
    ]
  }
}
```

排序规则

默认情况下，是按照_score降序排序的

然而，某些情况下，可能没有有用的_score，比如说filter

所以，要对字段进行排序

如果对一个text field进行排序，结果往往不准确，因为分词后是多个单词，再排序就不是我们想要的结果了。

通常解决方案是，将一个text field建立两次索引，一个分词，用来进行搜索；一个不分词，用来进行排序。

1. 创建索引时设置fielddate:true
2. 创建索引时加入以下信息

```
"字段名": {
      "type": "text",
      "fields": {
        "keyword": {
          "type": "keyword"
        }
      }
    },
```

排序:

```
GET /book/_search
{
    "query":
```

```
    {
      "bool":
      {
        "must":
        [
        {
          "match":
          {
            "description": "java程序员"
          }
        }
      ]
    }
    }
  ,
  "sort": [
    {
      "price": {
        "order": "desc"
      }
    }
  ]
}
```

结果:

```
{
  "took" : 9,
  "timed_out" : false,
  "_shards" : {
    "total" : 1,
    "successful" : 1,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : {
      "value" : 5,
      "relation" : "eq"
    },
    "max_score" : null,
    "hits" : [
      {
        "_index" : "book",
        "_id" : "3",
        "_score" : null,
        "_source" : {
          "name" : "spring开发基础",
          "description" : "spring 在java领域非常流行，java程序员都在用。",
          "studymodel" : "201001",
          "price" : 78.6,
          "timestamp" : "2019-08-24 19:21:35",
          "pic" : "group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg",
          "tags" : [
            "spring",
            "java"
          ]
```

```
      },
      "sort" : [
        78.6
      ]
    },
    {
      "_index" : "book",
      "_id" : "2",
      "_score" : null,
      "_source" : {
        "name" : "java编程思想",
        "description" : "java语言是世界第一编程语言，在软件开发领域使用人数最多。",
        "studymodel" : "201001",
        "price" : 68.6,
        "timestamp" : "2019-08-25 19:11:35",
        "pic" : "group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg",
        "tags" : [
          "java",
          "dev"
        ]
      },
      "sort" : [
        68.6
      ]
    },
    {
      "_index" : "book",
      "_id" : "5",
      "_score" : null,
      "_source" : {
        "name" : "java编程思想",
        "description" : "java语言是世界第一编程语言，在软件开发领域使用人数最多。",
        "studymodel" : "201001",
        "price" : 68.6,
        "timestamp" : "2022-5-25 19:11:35",
        "pic" : "group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg",
        "tags" : [
          "bootstrap",
          "dev"
        ]
      },
      "sort" : [
        68.6
      ]
    },
    {
      "_index" : "book",
      "_id" : "6",
      "_score" : null,
      "_source" : {
        "name" : "java编程思想",
        "description" : "java语言是世界第一编程语言，在软件开发领域使用人数最多。",
        "studymodel" : "201001",
        "price" : 68.6,
        "timestamp" : "2022-5-25 19:11:35",
        "pic" : "group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg",
        "tags" : [
          "bootstrap",
```

```
            "dev"
          ]
        },
        "sort" : [
          68.6
        ]
      },
      {
        "_index" : "book",
        "_id" : "1",
        "_score" : null,
        "_source" : {
          "name" : "Bootstrap开发",
          "description" : "Bootstrap是由Twitter推出的一个前台页面开发css框架，是一个非常流行的开发框架，此框架集成了多种页面效果。此开发框架包含了大量的CSS、JS程序代码，可以帮助开发者（尤其是不擅长css页面开发的程序人员）轻松的实现一个css，不受浏览器限制的精美界面css效果。",
          "studymodel" : "201002",
          "price" : 38.6,
          "timestamp" : "2019-08-25 19:11:35",
          "pic" : "group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg",
          "tags" : [
            "bootstrap",
            "dev"
          ]
        },
        "sort" : [
          38.6
        ]
      }
    ]
  }
}
```


Scroll分批查询

场景：下载某一个索引中1亿条数据，到文件或是数据库。

不能一下全查出来，系统内存溢出。所以使用scoll滚动搜索技术，一批一批查询。

scoll搜索会在第一次搜索的时候，保存一个当时的视图快照，之后只会基于该旧的视图快照提供数据搜索，如果这个期间数据变更，是不会让用户看到的

每次发送scroll请求，我们还需要指定一个scoll参数，指定一个时间窗口，每次搜索请求只要在这个时间窗口内能完成就可以了。

搜索：

```
GET /book/_search?scroll=1m
{
  "query": {
    "match_all": {}
  },
  "size": 3
}
```

结果:


```
{
  "_scroll_id" :
"FGluY2x1ZGVfY29udGV4dF91dWlkDXF1ZXJ5QW5kRmVOY2gBFlZZWjZjb1daUVpxZUlYdHUzbDVadGc
AAAAAAAAhYRZRcklJMONnMlNoLVJuQnJtVnptVjlR",
  "took" : 0,
  "timed_out" : false,
  "_shards" : {
    "total" : 1,
    "successful" : 1,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : {
      "value" : 5,
      "relation" : "eq"
    },
    "max_score" : 1.0,
    "hits" : [
      {
        "_index" : "book",
        "_id" : "1",
        "_score" : 1.0,
        "_source" : {
          "name" : "Bootstrap开发",
          "description" : "Bootstrap是由Twitter推出的一个前台页面开发css框架，是一个非常
流行的开发框架，此框架集成了多种页面效果。此开发框架包含了大量的CSS、JS程序代码，可以帮助开发者
（尤其是不擅长css页面开发的程序人员）轻松的实现一个css，不受浏览器限制的精美界面css效果。",
          "studymodel" : "201002",
          "price" : 38.6,
          "timestamp" : "2019-08-25 19:11:35",
          "pic" : "group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg",
          "tags" : [
            "bootstrap",
            "dev"
          ]
        }
      },
      {
        "_index" : "book",
        "_id" : "2",
        "_score" : 1.0,
        "_source" : {
          "name" : "java编程思想",
          "description" : "java语言是世界第一编程语言，在软件开发领域使用人数最多。",
          "studymodel" : "201001",
```

```
          "price" : 68.6,
          "timestamp" : "2019-08-25 19:11:35",
          "pic" : "group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg",
          "tags" : [
            "java",
            "dev"
          ]
        }
      },
      {
        "_index" : "book",
        "_id" : "3",
        "_score" : 1.0,
        "_source" : {
          "name" : "spring开发基础",
          "description" : "spring 在java领域非常流行，java程序员都在用。",
          "studymodel" : "201001",
          "price" : 78.6,
          "timestamp" : "2019-08-24 19:21:35",
          "pic" : "group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg",
          "tags" : [
            "spring",
            "java"
          ]
        }
      }
    ]
  }
}
```


获得的结果会有一个scoll_id，下一次再发送scoll请求的时候，必须带上这个scoll_id

再次发送：

```
GET /_search/scroll
{
    "scroll": "10m",
    "scroll_id" :
"FGluY2x1ZGVfY29udGV4dF91dWlkDXF1ZXJ5QW5kRmVOY2gBFlZZWjZjb1daUVpxZUlYdHUzbDVadGc
AAAAAAAAAhYRZRcklJMONnMlNoLVJuQnJtVnptVjlR"
}
```

结果：


```
{
  "_scroll_id" :
"FGluY2x1ZGVfY29udGV4dF91dWlkDXF1ZXJ5QW5kRmVOY2gBFlZZWjZjb1daUVpxZUlYdHUzbDVadGc
AAAAAAAAAhYRZRcklJMONnMlNoLVJuQnJtVnptVjlR",
  "took" : 3,
  "timed_out" : false,
  "_shards" : {
    "total" : 1,
    "successful" : 1,
    "skipped" : 0,
```

```
    "failed" : 0
  },
  "hits" : {
    "total" : {
      "value" : 5,
      "relation" : "eq"
    },
    "max_score" : 1.0,
    "hits" : [
      {
        "_index" : "book",
        "_id" : "5",
        "_score" : 1.0,
        "_source" : {
          "name" : "java编程思想",
          "description" : "java语言是世界第一编程语言，在软件开发领域使用人数最多。",
          "studymodel" : "201001",
          "price" : 68.6,
          "timestamp" : "2022-5-25 19:11:35",
          "pic" : "group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg",
          "tags" : [
            "bootstrap",
            "dev"
          ]
        }
      },
      {
        "_index" : "book",
        "_id" : "6",
        "_score" : 1.0,
        "_source" : {
          "name" : "java编程思想",
          "description" : "java语言是世界第一编程语言，在软件开发领域使用人数最多。",
          "studymodel" : "201001",
          "price" : 68.6,
          "timestamp" : "2022-5-25 19:11:35",
          "pic" : "group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg",
          "tags" : [
            "bootstrap",
            "dev"
          ]
        }
      }
    ]
  }
}
```


再次发送：

```
GET /_search/scroll
{
    "scroll": "10m",
    "scroll_id" :
"FGluY2x1ZGVfY29udGV4dF91dWlkDXF1ZXJ5QW5kRmVOY2gBFlZZWjZjb1daUVpxZUlYdHUzbDVadGc
AAAAAAAAhYRZRcklJMONnMlNoLVJuQnJtVnptVjlR"
}
```

结果：


```
{
  "_scroll_id" :
"FGluY2x1ZGVfY29udGV4dF91dWlkDXF1ZXJ5QW5kRmVOY2gBFlZZWjZjb1daUVpxZUlYdHUzbDVadGc
AAAAAAAAhYRZRcklJMONnMlNoLVJuQnJtVnptVjlR",
  "took" : 1,
  "timed_out" : false,
  "_shards" : {
    "total" : 1,
    "successful" : 1,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : {
      "value" : 5,
      "relation" : "eq"
    },
    "max_score" : 1.0,
    "hits" : [ ]
  }
}
```


java API实现搜索

查询全部

```
    /**
     * 搜索全部
     * 请求：
     *
     * <pre>
     *
     * GET /book/_search
     * {
     *   "query": { "match_all": {} }
     * }
     *
     * </pre>
     * <p>
     * 结果：
     * <pre>
     *
     * 总数量：5
     *
     *
     * id:1
     * score:1.0
     * 内容：
```

```
     * ---- price：38.6
     * ---- studymodel：201002
     * ---- name：Bootstrap开发
     * ---- description：Bootstrap是由Twitter推出的一个前台页面开发css框架，是一个非常流行的开发框架，此框架集成了多种页面效果。此开发框架包含了大量的CSS、JS程序代码，可以帮助开发者（尤其是不擅长css页面开发的程序人员）轻松的实现一个css，不受浏览器限制的精美界面css效果。
     * ---- pic：group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
     * ---- timestamp：2019-08-25 19:11:35
     * ---- tags：[bootstrap, dev]
     * ------------------------------------
     *
     * id:2
     * score:1.0
     * 内容：
     * ---- price：68.6
     * ---- studymodel：201001
     * ---- name：java编程思想
     * ---- description：java语言是世界第一编程语言，在软件开发领域使用人数最多。
     * ---- pic：group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
     * ---- timestamp：2019-08-25 19:11:35
     * ---- tags：[java, dev]
     * ------------------------------------
     *
     * id:3
     * score:1.0
     * 内容：
     * ---- price：78.6
     * ---- studymodel：201001
     * ---- name：spring开发基础
     * ---- description：spring 在java领域非常流行，java程序员都在用。
     * ---- pic：group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
     * ---- timestamp：2019-08-24 19:21:35
     * ---- tags：[spring, java]
     * ------------------------------------
     *
     * id:5
     * score:1.0
     * 内容：
     * ---- price：68.6
     * ---- studymodel：201001
     * ---- name：java编程思想
     * ---- description：java语言是世界第一编程语言，在软件开发领域使用人数最多。
     * ---- pic：group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
     * ---- timestamp：2022-5-25 19:11:35
     * ---- tags：[bootstrap, dev]
     * ------------------------------------
     *
     * id:6
     * score:1.0
     * 内容：
     * ---- price：68.6
     * ---- studymodel：201001
     * ---- name：java编程思想
     * ---- description：java语言是世界第一编程语言，在软件开发领域使用人数最多。
     * ---- pic：group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
     * ---- timestamp：2022-5-25 19:11:35
     * ---- tags：[bootstrap, dev]
     * ------------------------------------
```

```
     *
     * </pre>
     *
     * @throws IOException IOException
     */
    @Test
    void searchAll() throws IOException
    {
        //构建搜索请求
        SearchRequest searchRequest = new SearchRequest("book");
        //构建请求体
        SearchSourceBuilder searchSourceBuilder = new SearchSourceBuilder();
        searchSourceBuilder.query(QueryBuilders.matchAllQuery());
        //放入请求中
        searchRequest.source(searchSourceBuilder);
        //发起请求
        SearchResponse searchResponse = client.search(searchRequest,
RequestOptions.DEFAULT);
        //获取数据
        SearchHits hits = searchResponse.getHits();
        //总数
        long value = hits.getTotalHits().value;
        System.out.println("总数量：" + value);
        System.out.println();
        SearchHit[] hitsHits = hits.getHits();
        //遍历数据
        for (SearchHit hitsHit : hitsHits)
        {
            System.out.println();
            //获得id
            String id = hitsHit.getId();
            //获得得分
            float score = hitsHit.getScore();
            //获得内容
            Map<String, Object> sourceAsMap = hitsHit.getSourceAsMap();

            //打印内容
            System.out.println("id:" + id);
            System.out.println("score:" + score);
            System.out.println("内容：");
            for (String key : sourceAsMap.keySet())
            {
                System.out.println("---- " + key + ": " + sourceAsMap.get(key));
            }
            System.out.println("-----------------------------------");
        }

    }


    /**
     * 搜索全部，异步请求。
     * 请求内容：
     * <pre>
     *
     * GET /book/_search
     * {
     *   "query": { "match_all": {} }
```

```
     * }
     *
     * </pre>
     *
     * @throws Exception Exception
     */
    @Test
    void searchAll_async() throws Exception
    {
        //构建搜索请求
        SearchRequest searchRequest = new SearchRequest("book");
        //构建请求体
        SearchSourceBuilder searchSourceBuilder = new SearchSourceBuilder();
        searchSourceBuilder.query(QueryBuilders.matchAllQuery());
        //放入请求中
        searchRequest.source(searchSourceBuilder);
        //发起异步请求
        client.searchAsync(searchRequest, RequestOptions.DEFAULT, new
ActionListener<SearchResponse>()
        {
            /**
             * 成功的回调
             *
             * @param searchResponse SearchResponse
             */
            @Override
            public void onResponse(SearchResponse searchResponse)
            {
                //获取数据
                SearchHits hits = searchResponse.getHits();
                //总数
                long value = hits.getTotalHits().value;
                System.out.println("总数量: " + value);
                System.out.println();
                SearchHit[] hitsHits = hits.getHits();
                //遍历数据
                for (SearchHit hitsHit : hitsHits)
                {
                    System.out.println();
                    //获得id
                    String id = hitsHit.getId();
                    //获得得分
                    float score = hitsHit.getScore();
                    //获得内容
                    Map<String, Object> sourceAsMap = hitsHit.getSourceAsMap();

                    //打印内容
                    System.out.println("id:" + id);
                    System.out.println("score:" + score);
                    System.out.println("内容: ");
                    for (String key : sourceAsMap.keySet())
                    {
                        System.out.println("---- " + key + ": " +
sourceAsMap.get(key));
                    }
                    System.out.println("-----------------------------------");
                }
            }
```

```
            /**
             * 失败的回调
             *
             * @param e Exception
             */
            @Override
            public void onFailure(Exception e)
            {
                e.printStackTrace();
            }
        });
        Thread.sleep(2000);
    }
```

查询某字段

```
    /**
     * 查询某字段
     * 请求内容：
     *
     * <pre>
     *
     * GET /book/_search
     * {
     *   "query":
     *   {
     *     "match":
     *     {
     *       "description": "java"
     *     }
     *   }
     * }
     *
     * </pre>
     * <p>
     * 结果：
     * <pre>
     *
     * 总数量：4
     *
     *
     * id:3
     * score:0.4745544
     * 内容：
     * ---- price：78.6
     * ---- studymodel：201001
     * ---- name：spring开发基础
     * ---- description：spring 在java领域非常流行，java程序员都在用。
     * ---- pic：group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
     * ---- timestamp：2019-08-24 19:21:35
     * ---- tags：[spring, java]
     * ----------------------------------
     *
```

```
     * id:2
     * score:0.33773077
     * 内容：
     * ---- price：68.6
     * ---- studymodel：201001
     * ---- name：java编程思想
     * ---- description：java语言是世界第一编程语言，在软件开发领域使用人数最多。
     * ---- pic：group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
     * ---- timestamp：2019-08-25 19:11:35
     * ---- tags：[java, dev]
     * ----------------------------------
     *
     * id:5
     * score:0.33773077
     * 内容：
     * ---- price：68.6
     * ---- studymodel：201001
     * ---- name：java编程思想
     * ---- description：java语言是世界第一编程语言，在软件开发领域使用人数最多。
     * ---- pic：group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
     * ---- timestamp：2022-5-25 19:11:35
     * ---- tags：[bootstrap, dev]
     * ----------------------------------
     *
     * id:6
     * score:0.33773077
     * 内容：
     * ---- price：68.6
     * ---- studymodel：201001
     * ---- name：java编程思想
     * ---- description：java语言是世界第一编程语言，在软件开发领域使用人数最多。
     * ---- pic：group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
     * ---- timestamp：2022-5-25 19:11:35
     * ---- tags：[bootstrap, dev]
     * ----------------------------------
     *
     * </pre>
     *
     * @throws Exception Exception
     */
    @Test
    void search() throws Exception
    {
        //构建搜索请求
        SearchRequest searchRequest = new SearchRequest("book");
        //构建请求体
        SearchSourceBuilder searchSourceBuilder = new SearchSourceBuilder();
        //查询某字段
        searchSourceBuilder.query(QueryBuilders.matchQuery("description",
"java"));
        //放入请求中
        searchRequest.source(searchSourceBuilder);
        //发起请求
        SearchResponse searchResponse = client.search(searchRequest,
RequestOptions.DEFAULT);
        //获取数据
        SearchHits hits = searchResponse.getHits();
        //总数
```

```
        long value = hits.getTotalHits().value;
        System.out.println("总数量：" + value);
        System.out.println();
        SearchHit[] hitsHits = hits.getHits();
        //遍历数据
        for (SearchHit hitsHit : hitsHits)
        {
            System.out.println();
            //获得id
            String id = hitsHit.getId();
            //获得得分
            float score = hitsHit.getScore();
            //获得内容
            Map<String, Object> sourceAsMap = hitsHit.getSourceAsMap();

            //打印内容
            System.out.println("id:" + id);
            System.out.println("score:" + score);
            System.out.println("内容：");
            for (String key : sourceAsMap.keySet())
            {
                System.out.println("---- " + key + "：" + sourceAsMap.get(key));
            }
            System.out.println("-----------------------------------");
        }
    }

    /**
     * 查询某字段，异步请求
     * 请求内容：
     *
     * <pre>
     *
     * GET /book/_search
     * {
     *   "query":
     *   {
     *     "match":
     *     {
     *       "description": "java"
     *     }
     *   }
     * }
     *
     * </pre>
     * <p>
     * 结果：
     * <pre>
     *
     * 总数量：4
     *
     *
     * id:3
     * score:0.4745544
     * 内容：
     * ---- price：78.6
     * ---- studymodel：201001
     * ---- name：spring开发基础
```

```
     * ---- description：spring 在java领域非常流行，java程序员都在用。
     * ---- pic：group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
     * ---- timestamp：2019-08-24 19:21:35
     * ---- tags：[spring, java]
     * ----------------------------------
     *
     * id:2
     * score:0.33773077
     * 内容：
     * ---- price：68.6
     * ---- studymodel：201001
     * ---- name：java编程思想
     * ---- description：java语言是世界第一编程语言，在软件开发领域使用人数最多。
     * ---- pic：group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
     * ---- timestamp：2019-08-25 19:11:35
     * ---- tags：[java, dev]
     * ----------------------------------
     *
     * id:5
     * score:0.33773077
     * 内容：
     * ---- price：68.6
     * ---- studymodel：201001
     * ---- name：java编程思想
     * ---- description：java语言是世界第一编程语言，在软件开发领域使用人数最多。
     * ---- pic：group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
     * ---- timestamp：2022-5-25 19:11:35
     * ---- tags：[bootstrap, dev]
     * ----------------------------------
     *
     * id:6
     * score:0.33773077
     * 内容：
     * ---- price：68.6
     * ---- studymodel：201001
     * ---- name：java编程思想
     * ---- description：java语言是世界第一编程语言，在软件开发领域使用人数最多。
     * ---- pic：group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
     * ---- timestamp：2022-5-25 19:11:35
     * ---- tags：[bootstrap, dev]
     * ----------------------------------
     *
     * </pre>
     *
     * @throws Exception Exception
     */
    @Test
    void search_async() throws Exception
    {
        //构建搜索请求
        SearchRequest searchRequest = new SearchRequest("book");
        //构建请求体
        SearchSourceBuilder searchSourceBuilder = new SearchSourceBuilder();
        //查询某字段
        searchSourceBuilder.query(QueryBuilders.matchQuery("description",
"java"));
        //放入请求中
        searchRequest.source(searchSourceBuilder);
```

```
        //发起异步请求
        client.searchAsync(searchRequest, RequestOptions.DEFAULT, new
ActionListener<SearchResponse>()
        {
            /**
             * 成功的回调
             *
             * @param searchResponse SearchResponse
             */
            @Override
            public void onResponse(SearchResponse searchResponse)
            {
                //获取数据
                SearchHits hits = searchResponse.getHits();
                //总数
                long value = hits.getTotalHits().value;
                System.out.println("总数量: " + value);
                System.out.println();
                SearchHit[] hitsHits = hits.getHits();
                //遍历数据
                for (SearchHit hitsHit : hitsHits)
                {
                    System.out.println();
                    //获得id
                    String id = hitsHit.getId();
                    //获得得分
                    float score = hitsHit.getScore();
                    //获得内容
                    Map<String, Object> sourceAsMap = hitsHit.getSourceAsMap();

                    //打印内容
                    System.out.println("id:" + id);
                    System.out.println("score:" + score);
                    System.out.println("内容: ");
                    for (String key : sourceAsMap.keySet())
                    {
                        System.out.println("---- " + key + ": " +
sourceAsMap.get(key));
                    }
                    System.out.println("-----------------------------------");
                }
            }

            /**
             * 失败的回调
             *
             * @param e Exception
             */
            @Override
            public void onFailure(Exception e)
            {
                e.printStackTrace();
            }
        });
        Thread.sleep(2000);
    }
```

排序

```
/**
 * 搜索并排序
 * 请求内容：
 * <pre>
 *
 * GET /book/_search
 * {
 *   "query":
 *   {
 *     "match":
 *     {
 *       "description": "java"
 *     }
 *   },
 *   "sort":
 *   [
 *     {
 *       "price":
 *       {
 *         "order": "desc"
 *       }
 *     }
 *   ]
 * }
 *
 * </pre>
 * <p>
 * 结果：
 * <pre>
 *
 * 总数量：4
 *
 *
 * id:3
 * score:NaN
 * 内容：
 * ---- price：78.6
 * ---- studymodel：201001
 * ---- name：spring开发基础
 * ---- description：spring 在java领域非常流行，java程序员都在用。
 * ---- pic：group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
 * ---- timestamp：2019-08-24 19:21:35
 * ---- tags：[spring, java]
 * ----------------------------------
 *
 * id:2
 * score:NaN
 * 内容：
 * ---- price：68.6
 * ---- studymodel：201001
 * ---- name：java编程思想
 * ---- description：java语言是世界第一编程语言，在软件开发领域使用人数最多。
```

```
     * ---- pic：group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
     * ---- timestamp：2019-08-25 19:11:35
     * ---- tags：[java, dev]
     * ----------------------------------
     *
     * id:5
     * score:NaN
     * 内容：
     * ---- price：68.6
     * ---- studymodel：201001
     * ---- name：java编程思想
     * ---- description：java语言是世界第一编程语言，在软件开发领域使用人数最多。
     * ---- pic：group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
     * ---- timestamp：2022-5-25 19:11:35
     * ---- tags：[bootstrap, dev]
     * ----------------------------------
     *
     * id:6
     * score:NaN
     * 内容：
     * ---- price：68.6
     * ---- studymodel：201001
     * ---- name：java编程思想
     * ---- description：java语言是世界第一编程语言，在软件开发领域使用人数最多。
     * ---- pic：group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
     * ---- timestamp：2022-5-25 19:11:35
     * ---- tags：[bootstrap, dev]
     * ----------------------------------
     *
     * </pre>
     *
     * @throws Exception Exception
     */
    @Test
    void search_sort() throws Exception
    {
        //构建搜索请求
        SearchRequest searchRequest = new SearchRequest("book");
        //构建请求体
        SearchSourceBuilder searchSourceBuilder = new SearchSourceBuilder();
        //查询某字段
        searchSourceBuilder.query(QueryBuilders.matchQuery("description",
"java"));
        //排序
        searchSourceBuilder.sort("price", SortOrder.DESC);
        //放入请求中
        searchRequest.source(searchSourceBuilder);
        //发起请求
        SearchResponse searchResponse = client.search(searchRequest,
RequestOptions.DEFAULT);
        //获取数据
        SearchHits hits = searchResponse.getHits();
        //总数
        long value = hits.getTotalHits().value;
        System.out.println("总数量：" + value);
        System.out.println();
        SearchHit[] hitsHits = hits.getHits();
        //遍历数据
```

```
        for (SearchHit hitsHit : hitsHits)
        {
            System.out.println();
            //获得id
            String id = hitsHit.getId();
            //获得得分
            float score = hitsHit.getScore();
            //获得内容
            Map<String, Object> sourceAsMap = hitsHit.getSourceAsMap();

            //打印内容
            System.out.println("id:" + id);
            System.out.println("score:" + score);
            System.out.println("内容: ");
            for (String key : sourceAsMap.keySet())
            {
                System.out.println("---- " + key + ": " + sourceAsMap.get(key));
            }
            System.out.println("-----------------------------------");
        }
    }

    /**
     * 搜索并排序，异步请求
     * 请求内容:
     * <pre>
     *
     * GET /book/_search
     * {
     *   "query":
     *   {
     *     "match":
     *     {
     *       "description": "java"
     *     }
     *   },
     *   "sort":
     *   [
     *     {
     *       "price":
     *       {
     *         "order": "desc"
     *       }
     *     }
     *   ]
     * }
     *
     * </pre>
     * <p>
     * 结果:
     * <pre>
     *
     * 总数量: 4
     *
     *
     * id:3
     * score:NaN
     * 内容:
```

```
     * ---- price：78.6
     * ---- studymodel：201001
     * ---- name：spring开发基础
     * ---- description：spring 在java领域非常流行，java程序员都在用。
     * ---- pic：group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
     * ---- timestamp：2019-08-24 19:21:35
     * ---- tags：[spring, java]
     * ----------------------------------
     *
     * id:2
     * score:NaN
     * 内容：
     * ---- price：68.6
     * ---- studymodel：201001
     * ---- name：java编程思想
     * ---- description：java语言是世界第一编程语言，在软件开发领域使用人数最多。
     * ---- pic：group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
     * ---- timestamp：2019-08-25 19:11:35
     * ---- tags：[java, dev]
     * ----------------------------------
     *
     * id:5
     * score:NaN
     * 内容：
     * ---- price：68.6
     * ---- studymodel：201001
     * ---- name：java编程思想
     * ---- description：java语言是世界第一编程语言，在软件开发领域使用人数最多。
     * ---- pic：group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
     * ---- timestamp：2022-5-25 19:11:35
     * ---- tags：[bootstrap, dev]
     * ----------------------------------
     *
     * id:6
     * score:NaN
     * 内容：
     * ---- price：68.6
     * ---- studymodel：201001
     * ---- name：java编程思想
     * ---- description：java语言是世界第一编程语言，在软件开发领域使用人数最多。
     * ---- pic：group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
     * ---- timestamp：2022-5-25 19:11:35
     * ---- tags：[bootstrap, dev]
     * ----------------------------------
     *
     * </pre>
     *
     * @throws Exception Exception
     */
    @Test
    void search_sort_async() throws Exception
    {
        //构建搜索请求
        SearchRequest searchRequest = new SearchRequest("book");
        //构建请求体
        SearchSourceBuilder searchSourceBuilder = new SearchSourceBuilder();
        //查询某字段
```

```
        searchSourceBuilder.query(QueryBuilders.matchQuery("description",
"java"));
        //排序
        searchSourceBuilder.sort("price", SortOrder.DESC);
        //放入请求中
        searchRequest.source(searchSourceBuilder);
        //发起异步请求
        client.searchAsync(searchRequest, RequestOptions.DEFAULT, new
ActionListener<SearchResponse>()
        {
            /**
             * 成功的回调
             *
             * @param searchResponse SearchResponse
             */
            @Override
            public void onResponse(SearchResponse searchResponse)
            {
                //获取数据
                SearchHits hits = searchResponse.getHits();
                //总数
                long value = hits.getTotalHits().value;
                System.out.println("总数量: " + value);
                System.out.println();
                SearchHit[] hitsHits = hits.getHits();
                //遍历数据
                for (SearchHit hitsHit : hitsHits)
                {
                    System.out.println();
                    //获得id
                    String id = hitsHit.getId();
                    //获得得分
                    float score = hitsHit.getScore();
                    //获得内容
                    Map<String, Object> sourceAsMap = hitsHit.getSourceAsMap();

                    //打印内容
                    System.out.println("id:" + id);
                    System.out.println("score:" + score);
                    System.out.println("内容: ");
                    for (String key : sourceAsMap.keySet())
                    {
                        System.out.println("---- " + key + ": " +
sourceAsMap.get(key));
                    }
                    System.out.println("-----------------------------------");
                }
            }

            /**
             * 失败的回调
             *
             * @param e Exception
             */
            @Override
            public void onFailure(Exception e)
            {
                e.printStackTrace();
```

```
            }
        });
        //休眠2秒
        Thread.sleep(2000);
    }
```

分页查询

```
    /**
     * 分页查询
     * 请求内容:
     * <pre>
     *
     * GET /book/_search
     * {
     *   "query":
     *   {
     *     "match_all": {}
     *   },
     *   "from": 1,
     *   "size": 2
     * }
     *
     * </pre>
     * <p>
     * 结果:
     * <pre>
     *
     * 总数量: 5
     *
     *
     * id:2
     * score:1.0
     * 内容:
     * ---- price: 68.6
     * ---- studymodel: 201001
     * ---- name: java编程思想
     * ---- description: java语言是世界第一编程语言，在软件开发领域使用人数最多。
     * ---- pic: group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
     * ---- timestamp: 2019-08-25 19:11:35
     * ---- tags: [java, dev]
     * ----------------------------------
     *
     * id:3
     * score:1.0
     * 内容:
     * ---- price: 78.6
     * ---- studymodel: 201001
     * ---- name: spring开发基础
     * ---- description: spring 在java领域非常流行，java程序员都在用。
     * ---- pic: group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
     * ---- timestamp: 2019-08-24 19:21:35
     * ---- tags: [spring, java]
```

```
     * ----------------------------------
     *
     * </pre>
     *
     * @throws Exception Exception
     */
    @Test
    void search_page() throws Exception
    {
        //构建搜索请求
        SearchRequest searchRequest = new SearchRequest("book");
        //构建请求体
        SearchSourceBuilder searchSourceBuilder = new SearchSourceBuilder();
        //查询
        searchSourceBuilder.query(QueryBuilders.matchAllQuery());
        //分页
        searchSourceBuilder.from(1);
        searchSourceBuilder.size(2);
        //放入请求中
        searchRequest.source(searchSourceBuilder);
        //发起请求
        SearchResponse searchResponse = client.search(searchRequest,
RequestOptions.DEFAULT);
        //获取数据
        SearchHits hits = searchResponse.getHits();
        //总数
        long value = hits.getTotalHits().value;
        System.out.println("总数量: " + value);
        System.out.println();
        SearchHit[] hitsHits = hits.getHits();
        //遍历数据
        for (SearchHit hitsHit : hitsHits)
        {
            System.out.println();
            //获得id
            String id = hitsHit.getId();
            //获得得分
            float score = hitsHit.getScore();
            //获得内容
            Map<String, Object> sourceAsMap = hitsHit.getSourceAsMap();

            //打印内容
            System.out.println("id:" + id);
            System.out.println("score:" + score);
            System.out.println("内容: ");
            for (String key : sourceAsMap.keySet())
            {
                System.out.println("---- " + key + ": " + sourceAsMap.get(key));
            }
            System.out.println("----------------------------------");
        }
    }


    /**
     * 分页查询，异步
     * 请求内容:
     * <pre>
```

```
     *
     * GET /book/_search
     * {
     *   "query":
     *   {
     *     "match_all": {}
     *   },
     *   "from": 1,
     *   "size": 2
     * }
     *
     * </pre>
     * <p>
     * 结果:
     * <pre>
     *
     * 总数量: 5
     *
     *
     * id:2
     * score:1.0
     * 内容:
     * ---- price: 68.6
     * ---- studymodel: 201001
     * ---- name: java编程思想
     * ---- description: java语言是世界第一编程语言，在软件开发领域使用人数最多。
     * ---- pic: group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
     * ---- timestamp: 2019-08-25 19:11:35
     * ---- tags: [java, dev]
     * ----------------------------------
     *
     * id:3
     * score:1.0
     * 内容:
     * ---- price: 78.6
     * ---- studymodel: 201001
     * ---- name: spring开发基础
     * ---- description: spring 在java领域非常流行，java程序员都在用。
     * ---- pic: group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
     * ---- timestamp: 2019-08-24 19:21:35
     * ---- tags: [spring, java]
     * ----------------------------------
     *
     * </pre>
     *
     * @throws Exception Exception
     */
    @Test
    void search_page_async() throws Exception
    {
        //构建搜索请求
        SearchRequest searchRequest = new SearchRequest("book");
        //构建请求体
        SearchSourceBuilder searchSourceBuilder = new SearchSourceBuilder();
        //查询
        searchSourceBuilder.query(QueryBuilders.matchAllQuery());
        //分页
        searchSourceBuilder.from(1);
```

```
        searchSourceBuilder.size(2);
        //放入请求中
        searchRequest.source(searchSourceBuilder);
        //发起异步请求
        client.searchAsync(searchRequest, RequestOptions.DEFAULT, new
ActionListener<SearchResponse>()
        {
            /**
             * 成功的回调
             *
             * @param searchResponse SearchResponse
             */
            @Override
            public void onResponse(SearchResponse searchResponse)
            {
                //获取数据
                SearchHits hits = searchResponse.getHits();
                //总数
                long value = hits.getTotalHits().value;
                System.out.println("总数量: " + value);
                System.out.println();
                SearchHit[] hitsHits = hits.getHits();
                //遍历数据
                for (SearchHit hitsHit : hitsHits)
                {
                    System.out.println();
                    //获得id
                    String id = hitsHit.getId();
                    //获得得分
                    float score = hitsHit.getScore();
                    //获得内容
                    Map<String, Object> sourceAsMap = hitsHit.getSourceAsMap();

                    //打印内容
                    System.out.println("id:" + id);
                    System.out.println("score:" + score);
                    System.out.println("内容: ");
                    for (String key : sourceAsMap.keySet())
                    {
                        System.out.println("---- " + key + ": " +
sourceAsMap.get(key));
                    }
                    System.out.println("-----------------------------------");
                }
            }

            /**
             * 失败的回调
             *
             * @param e Exception
             */
            @Override
            public void onFailure(Exception e)
            {
                e.printStackTrace();
            }
        });
        //休眠2秒
```

```
        Thread.sleep(2000);
    }
```

指定返回字段

```
    /**
     * 查询，返回指定的字段
     * 请求内容：
     * <pre>
     *
     * GET /book/_search
     * {
     *   "query":
     *   {
     *     "match_all": {}
     *   },
     *   "_source": ["name","price"]
     * }
     *
     * </pre>
     *
     * 结果：
     * <pre>
     *
     * 总数量：5
     *
     *
     * id:1
     * score:1.0
     * 内容：
     * ---- price：38.6
     * ---- name：Bootstrap开发
     * ------------------------------------
     *
     * id:2
     * score:1.0
     * 内容：
     * ---- price：68.6
     * ---- name：java编程思想
     * ------------------------------------
     *
     * id:3
     * score:1.0
     * 内容：
     * ---- price：78.6
     * ---- name：spring开发基础
     * ------------------------------------
     *
     * id:5
     * score:1.0
     * 内容：
     * ---- price：68.6
     * ---- name：java编程思想
```

```
     * ----------------------------------
     *
     * id:6
     * score:1.0
     * 内容:
     * ---- price: 68.6
     * ---- name: java编程思想
     * ----------------------------------
     *
     * </pre>
     *
     * @throws Exception Exception
     */
    @Test
    void search_field() throws Exception
    {
        //构建搜索请求
        SearchRequest searchRequest = new SearchRequest("book");
        //构建请求体
        SearchSourceBuilder searchSourceBuilder = new SearchSourceBuilder();
        //查询
        searchSourceBuilder.query(QueryBuilders.matchAllQuery());
        //指定某些字段
        searchSourceBuilder.fetchSource(new String[]{"name", "price"}, new
String[]{});
        //放入请求中
        searchRequest.source(searchSourceBuilder);
        //发起请求
        SearchResponse searchResponse = client.search(searchRequest,
RequestOptions.DEFAULT);
        //获取数据
        SearchHits hits = searchResponse.getHits();
        //总数
        long value = hits.getTotalHits().value;
        System.out.println("总数量: " + value);
        System.out.println();
        SearchHit[] hitsHits = hits.getHits();
        //遍历数据
        for (SearchHit hitsHit : hitsHits)
        {
            System.out.println();
            //获得id
            String id = hitsHit.getId();
            //获得得分
            float score = hitsHit.getScore();
            //获得内容
            Map<String, Object> sourceAsMap = hitsHit.getSourceAsMap();

            //打印内容
            System.out.println("id:" + id);
            System.out.println("score:" + score);
            System.out.println("内容: ");
            for (String key : sourceAsMap.keySet())
            {
                System.out.println("---- " + key + ": " + sourceAsMap.get(key));
            }
            System.out.println("----------------------------------");
        }
```

```
    }


    /**
     * 查询，返回指定的字段，异步请求
     * 请求内容：
     * <pre>
     *
     * GET /book/_search
     * {
     *   "query":
     *   {
     *     "match_all": {}
     *   },
     *   "_source": ["name","price"]
     * }
     *
     * </pre>
     *
     * 结果：
     * <pre>
     *
     * 总数量：5
     *
     *
     * id:1
     * score:1.0
     * 内容：
     * ---- price：38.6
     * ---- name：Bootstrap开发
     * ------------------------------------
     *
     * id:2
     * score:1.0
     * 内容：
     * ---- price：68.6
     * ---- name：java编程思想
     * ------------------------------------
     *
     * id:3
     * score:1.0
     * 内容：
     * ---- price：78.6
     * ---- name：spring开发基础
     * ------------------------------------
     *
     * id:5
     * score:1.0
     * 内容：
     * ---- price：68.6
     * ---- name：java编程思想
     * ------------------------------------
     *
     * id:6
     * score:1.0
     * 内容：
     * ---- price：68.6
     * ---- name：java编程思想
```

```
     * ----------------------------------
     *
     * </pre>
     *
     * @throws Exception Exception
     */
    @Test
    void search_field_async() throws Exception
    {
        //构建搜索请求
        SearchRequest searchRequest = new SearchRequest("book");
        //构建请求体
        SearchSourceBuilder searchSourceBuilder = new SearchSourceBuilder();
        //查询
        searchSourceBuilder.query(QueryBuilders.matchAllQuery());
        //指定某些字段
        searchSourceBuilder.fetchSource(new String[]{"name", "price"}, new
String[]{});
        //放入请求中
        searchRequest.source(searchSourceBuilder);
        //发起异步请求
        client.searchAsync(searchRequest, RequestOptions.DEFAULT, new
ActionListener<SearchResponse>()
        {
            /**
             * 成功的回调
             *
             * @param searchResponse SearchResponse
             */
            @Override
            public void onResponse(SearchResponse searchResponse)
            {
                //获取数据
                SearchHits hits = searchResponse.getHits();
                //总数
                long value = hits.getTotalHits().value;
                System.out.println("总数量: " + value);
                System.out.println();
                SearchHit[] hitsHits = hits.getHits();
                //遍历数据
                for (SearchHit hitsHit : hitsHits)
                {
                    System.out.println();
                    //获得id
                    String id = hitsHit.getId();
                    //获得得分
                    float score = hitsHit.getScore();
                    //获得内容
                    Map<String, Object> sourceAsMap = hitsHit.getSourceAsMap();

                    //打印内容
                    System.out.println("id:" + id);
                    System.out.println("score:" + score);
                    System.out.println("内容: ");
                    for (String key : sourceAsMap.keySet())
                    {
                        System.out.println("---- " + key + ": " +
sourceAsMap.get(key));
```

```
                    }
                    System.out.println("----------------------------------");
                }
            }

            /**
             * 失败的回调
             *
             * @param e Exception
             */
            @Override
            public void onFailure(Exception e)
            {
                e.printStackTrace();
            }
        });
        //休眠2秒
        Thread.sleep(2000);
    }
```

多搜索条件

```
/**
     * 多搜索条件
     * 请求内容:
     * <pre>
     *
     * GET /book/_search
     * {
     *   "query":
     *   {
     *     "bool":
     *     {
     *       "must":
     *       [
     *         {
     *           "match":
     *           {
     *             "name": "编程"
     *           }
     *         }
     *       ],
     *       "should":
     *       [
     *         {
     *           "match":
     *           {
     *             "name": "spring开发基础"
     *           }
     *         }
     *       ]
     *     }
     *   }
     * }
     *
```

```
     * </pre>
     * <p>
     * 结果:
     * <pre>
     *
     * 总数量: 3
     *
     *
     * id:2
     * score:1.0409627
     * 内容:
     * ---- price: 68.6
     * ---- studymodel: 201001
     * ---- name: java编程思想
     * ---- description: java语言是世界第一编程语言，在软件开发领域使用人数最多。
     * ---- pic: group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
     * ---- timestamp: 2019-08-25 19:11:35
     * ---- tags: [java, dev]
     * ----------------------------------
     *
     * id:5
     * score:1.0409627
     * 内容:
     * ---- price: 68.6
     * ---- studymodel: 201001
     * ---- name: java编程思想
     * ---- description: java语言是世界第一编程语言，在软件开发领域使用人数最多。
     * ---- pic: group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
     * ---- timestamp: 2022-5-25 19:11:35
     * ---- tags: [bootstrap, dev]
     * ----------------------------------
     *
     * id:6
     * score:1.0409627
     * 内容:
     * ---- price: 68.6
     * ---- studymodel: 201001
     * ---- name: java编程思想
     * ---- description: java语言是世界第一编程语言，在软件开发领域使用人数最多。
     * ---- pic: group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
     * ---- timestamp: 2022-5-25 19:11:35
     * ---- tags: [bootstrap, dev]
     * ----------------------------------
     *
     * </pre>
     *
     * @throws Exception Exception
     */
    @Test
    void search_multipleConditions() throws Exception
    {
        //构建搜索请求
        SearchRequest searchRequest = new SearchRequest("book");
        //构建请求体
        SearchSourceBuilder searchSourceBuilder = new SearchSourceBuilder();
        //查询
        searchSourceBuilder.query(QueryBuilders.boolQuery().
                must(QueryBuilders.matchQuery("name", "编程")).
```

```
                should(QueryBuilders.matchQuery("name", "spring开发基础")));

        //放入请求中
        searchRequest.source(searchSourceBuilder);
        //发起请求
        SearchResponse searchResponse = client.search(searchRequest,
RequestOptions.DEFAULT);
        //获取数据
        SearchHits hits = searchResponse.getHits();
        //总数
        long value = hits.getTotalHits().value;
        System.out.println("总数量：" + value);
        float maxScore = hits.getMaxScore();
        System.out.println("最大分数：" + maxScore);
        System.out.println();
        SearchHit[] hitsHits = hits.getHits();
        //遍历数据
        for (SearchHit hitsHit : hitsHits)
        {
            System.out.println();
            //获得id
            String id = hitsHit.getId();
            //获得得分
            float score = hitsHit.getScore();
            //获得内容
            Map<String, Object> sourceAsMap = hitsHit.getSourceAsMap();

            //打印内容
            System.out.println("id:" + id);
            System.out.println("score:" + score);
            System.out.println("内容：");
            for (String key : sourceAsMap.keySet())
            {
                System.out.println("---- " + key + ": " + sourceAsMap.get(key));
            }
            System.out.println("-----------------------------------");
        }
    }


    /**
     * 多搜索条件，异步请求
     * 请求内容：
     * <pre>
     *
     * GET /book/_search
     * {
     *   "query":
     *   {
     *     "bool":
     *     {
     *       "must":
     *       [
     *         {
     *           "match":
     *           {
     *             "name": "编程"
     *           }
```

```
*          }
*        ],
*        "should":
*        [
*          {
*            "match":
*            {
*              "name": "spring开发基础"
*            }
*          }
*        ]
*      }
*    }
* }
*
* </pre>
* <p>
* 结果：
* <pre>
*
* 总数量：3
*
*
* id:2
* score:1.0409627
* 内容：
* ---- price：68.6
* ---- studymodel：201001
* ---- name：java编程思想
* ---- description：java语言是世界第一编程语言，在软件开发领域使用人数最多。
* ---- pic：group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
* ---- timestamp：2019-08-25 19:11:35
* ---- tags：[java, dev]
* ------------------------------------
*
* id:5
* score:1.0409627
* 内容：
* ---- price：68.6
* ---- studymodel：201001
* ---- name：java编程思想
* ---- description：java语言是世界第一编程语言，在软件开发领域使用人数最多。
* ---- pic：group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
* ---- timestamp：2022-5-25 19:11:35
* ---- tags：[bootstrap, dev]
* ------------------------------------
*
* id:6
* score:1.0409627
* 内容：
* ---- price：68.6
* ---- studymodel：201001
* ---- name：java编程思想
* ---- description：java语言是世界第一编程语言，在软件开发领域使用人数最多。
* ---- pic：group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
* ---- timestamp：2022-5-25 19:11:35
* ---- tags：[bootstrap, dev]
* ------------------------------------
```

```
     *
     * </pre>
     *
     * @throws Exception Exception
     */
    @Test
    void search_multipleConditions_async() throws Exception
    {
        //构建搜索请求
        SearchRequest searchRequest = new SearchRequest("book");
        //构建请求体
        SearchSourceBuilder searchSourceBuilder = new SearchSourceBuilder();
        //查询
        searchSourceBuilder.query(QueryBuilders.boolQuery().
                must(QueryBuilders.matchQuery("name", "编程")).
                should(QueryBuilders.matchQuery("name", "spring开发基础")));

        //放入请求中
        searchRequest.source(searchSourceBuilder);
        //发起异步请求
        client.searchAsync(searchRequest, RequestOptions.DEFAULT, new
ActionListener<SearchResponse>()
        {
            /**
             * 成功的回调
             *
             * @param searchResponse SearchResponse
             */
            @Override
            public void onResponse(SearchResponse searchResponse)
            {
                //获取数据
                SearchHits hits = searchResponse.getHits();
                //总数
                long value = hits.getTotalHits().value;
                System.out.println("总数量: " + value);
                float maxScore = hits.getMaxScore();
                System.out.println("最大分数: " + maxScore);
                System.out.println();
                SearchHit[] hitsHits = hits.getHits();
                //遍历数据
                for (SearchHit hitsHit : hitsHits)
                {
                    System.out.println();
                    //获得id
                    String id = hitsHit.getId();
                    //获得得分
                    float score = hitsHit.getScore();
                    //获得内容
                    Map<String, Object> sourceAsMap = hitsHit.getSourceAsMap();

                    //打印内容
                    System.out.println("id:" + id);
                    System.out.println("score:" + score);
                    System.out.println("内容: ");
                    for (String key : sourceAsMap.keySet())
                    {
```

```
                        System.out.println("---- " + key + ": " +
sourceAsMap.get(key));
                    }
                    System.out.println("-----------------------------------");
                }
            }

            /**
             * 失败的回调
             *
             * @param e Exception
             */
            @Override
            public void onFailure(Exception e)
            {
                e.printStackTrace();
            }
        });
        //休眠2秒
        Thread.sleep(2000);
    }
```

multi_match

```
/**
     * multi_match
     * 搜索在多个字段下是否包含某关键字
     * <p>
     * 请求内容:
     * <pre>
     *
     * GET /book/_search
     * {
     *     "query":
     *     {
     *       "multi_match":
     *       {
     *         "query": "语言",
     *         "fields": ["name","description"]
     *       }
     *     }
     * }
     *
     * </pre>
     * <p>
     * 结果:
     * <pre>
     *
     * 总数量: 3
     * 最大分数: 1.6503837
     *
     *
     * id:2
     * score:1.6503837
     * 内容:
```

```
     * ---- price：68.6
     * ---- studymodel：201001
     * ---- name：java编程思想
     * ---- description：java语言是世界第一编程语言，在软件开发领域使用人数最多。
     * ---- pic：group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
     * ---- timestamp：2019-08-25 19:11:35
     * ---- tags：[java, dev]
     * ------------------------------------
     *
     * id:5
     * score:1.6503837
     * 内容：
     * ---- price：68.6
     * ---- studymodel：201001
     * ---- name：java编程思想
     * ---- description：java语言是世界第一编程语言，在软件开发领域使用人数最多。
     * ---- pic：group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
     * ---- timestamp：2022-5-25 19:11:35
     * ---- tags：[bootstrap, dev]
     * ------------------------------------
     *
     * id:6
     * score:1.6503837
     * 内容：
     * ---- price：68.6
     * ---- studymodel：201001
     * ---- name：java编程思想
     * ---- description：java语言是世界第一编程语言，在软件开发领域使用人数最多。
     * ---- pic：group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
     * ---- timestamp：2022-5-25 19:11:35
     * ---- tags：[bootstrap, dev]
     * ------------------------------------
     *
     * </pre>
     *
     * @throws Exception Exception
     */
    @Test
    void search_multi_match() throws Exception
    {
        //构建搜索请求
        SearchRequest searchRequest = new SearchRequest("book");
        //构建请求体
        SearchSourceBuilder searchSourceBuilder = new SearchSourceBuilder();
        //查询
        searchSourceBuilder.query(QueryBuilders.multiMatchQuery("语言", "name",
"description"));
        //放入请求中
        searchRequest.source(searchSourceBuilder);
        //发起请求
        SearchResponse searchResponse = client.search(searchRequest,
RequestOptions.DEFAULT);
        //获取数据
        SearchHits hits = searchResponse.getHits();
        //总数
        long value = hits.getTotalHits().value;
        System.out.println("总数量：" + value);
        float maxScore = hits.getMaxScore();
```

```
        System.out.println("最大分数: " + maxScore);
        System.out.println();
        SearchHit[] hitsHits = hits.getHits();
        //遍历数据
        for (SearchHit hitsHit : hitsHits)
        {
            System.out.println();
            //获得id
            String id = hitsHit.getId();
            //获得得分
            float score = hitsHit.getScore();
            //获得内容
            Map<String, Object> sourceAsMap = hitsHit.getSourceAsMap();

            //打印内容
            System.out.println("id:" + id);
            System.out.println("score:" + score);
            System.out.println("内容: ");
            for (String key : sourceAsMap.keySet())
            {
                System.out.println("---- " + key + ": " + sourceAsMap.get(key));
            }
            System.out.println("-----------------------------------");
        }
    }


    /**
     * multi_match ,异步请求
     * 搜索在多个字段下是否包含某关键字
     * <p>
     * 请求内容:
     * <pre>
     *
     * GET /book/_search
     * {
     *     "query":
     *     {
     *       "multi_match":
     *       {
     *         "query": "语言",
     *         "fields": ["name","description"]
     *       }
     *     }
     * }
     *
     * </pre>
     * <p>
     * 结果:
     * <pre>
     *
     * 总数量: 3
     * 最大分数: 1.6503837
     *
     *
     * id:2
     * score:1.6503837
     * 内容:
```

```
     * ---- price：68.6
     * ---- studymodel：201001
     * ---- name：java编程思想
     * ---- description：java语言是世界第一编程语言，在软件开发领域使用人数最多。
     * ---- pic：group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
     * ---- timestamp：2019-08-25 19:11:35
     * ---- tags：[java, dev]
     * ------------------------------------
     *
     * id:5
     * score:1.6503837
     * 内容：
     * ---- price：68.6
     * ---- studymodel：201001
     * ---- name：java编程思想
     * ---- description：java语言是世界第一编程语言，在软件开发领域使用人数最多。
     * ---- pic：group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
     * ---- timestamp：2022-5-25 19:11:35
     * ---- tags：[bootstrap, dev]
     * ------------------------------------
     *
     * id:6
     * score:1.6503837
     * 内容：
     * ---- price：68.6
     * ---- studymodel：201001
     * ---- name：java编程思想
     * ---- description：java语言是世界第一编程语言，在软件开发领域使用人数最多。
     * ---- pic：group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
     * ---- timestamp：2022-5-25 19:11:35
     * ---- tags：[bootstrap, dev]
     * ------------------------------------
     *
     * </pre>
     *
     * @throws Exception Exception
     */
    @Test
    void search_multi_match_async() throws Exception
    {
        //构建搜索请求
        SearchRequest searchRequest = new SearchRequest("book");
        //构建请求体
        SearchSourceBuilder searchSourceBuilder = new SearchSourceBuilder();
        //查询
        searchSourceBuilder.query(QueryBuilders.multiMatchQuery("语言", "name",
"description"));
        //放入请求中
        searchRequest.source(searchSourceBuilder);
        //发起异步请求
        client.searchAsync(searchRequest, RequestOptions.DEFAULT, new
ActionListener<SearchResponse>()
        {
            /**
             * 成功的回调
             *
             * @param searchResponse SearchResponse
             */
```

```
            @Override
            public void onResponse(SearchResponse searchResponse)
            {
                //获取数据
                SearchHits hits = searchResponse.getHits();
                //总数
                long value = hits.getTotalHits().value;
                System.out.println("总数量：" + value);
                float maxScore = hits.getMaxScore();
                System.out.println("最大分数：" + maxScore);
                System.out.println();
                SearchHit[] hitsHits = hits.getHits();
                //遍历数据
                for (SearchHit hitsHit : hitsHits)
                {
                    System.out.println();
                    //获得id
                    String id = hitsHit.getId();
                    //获得得分
                    float score = hitsHit.getScore();
                    //获得内容
                    Map<String, Object> sourceAsMap = hitsHit.getSourceAsMap();

                    //打印内容
                    System.out.println("id:" + id);
                    System.out.println("score:" + score);
                    System.out.println("内容：");
                    for (String key : sourceAsMap.keySet())
                    {
                        System.out.println("---- " + key + ": " +
sourceAsMap.get(key));
                    }
                    System.out.println("-----------------------------------");
                }
            }

            /**
             * 失败的回调
             *
             * @param e Exception
             */
            @Override
            public void onFailure(Exception e)
            {
                e.printStackTrace();
            }
        });
        //休眠2秒
        Thread.sleep(2000);
    }
```

range query

```
/**
     * range query
```

```
     * 范围查询
     * <p>
     * 请求内容：
     * <pre>
     *
     * GET /book/_search
     * {
     *     "query":
     *     {
     *       "range":
     *       {
     *         "price":
     *         {
     *           "gte": 69.2,
     *           "lte": 80
     *         }
     *       }
     *     }
     * }
     *
     * </pre>
     * <p>
     * 结果：
     * <pre>
     *
     * 总数量：1
     * 最大分数：1.0
     *
     *
     * id:3
     * score:1.0
     * 内容：
     * ---- price：78.6
     * ---- studymodel：201001
     * ---- name：spring开发基础
     * ---- description：spring 在java领域非常流行，java程序员都在用。
     * ---- pic：group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
     * ---- timestamp：2019-08-24 19:21:35
     * ---- tags：[spring, java]
     * ----------------------------------
     *
     * </pre>
     *
     * @throws Exception Exception
     */
    @Test
    void search_range_query() throws Exception
    {
        //构建搜索请求
        SearchRequest searchRequest = new SearchRequest("book");
        //构建请求体
        SearchSourceBuilder searchSourceBuilder = new SearchSourceBuilder();
        //查询

searchSourceBuilder.query(QueryBuilders.rangeQuery("price").gte(69.2).lte(80));
        //放入请求中
        searchRequest.source(searchSourceBuilder);
        //发起请求
```

```
        SearchResponse searchResponse = client.search(searchRequest,
RequestOptions.DEFAULT);
        //获取数据
        SearchHits hits = searchResponse.getHits();
        //总数
        long value = hits.getTotalHits().value;
        System.out.println("总数量：" + value);
        float maxScore = hits.getMaxScore();
        System.out.println("最大分数：" + maxScore);
        System.out.println();
        SearchHit[] hitsHits = hits.getHits();
        //遍历数据
        for (SearchHit hitsHit : hitsHits)
        {
            System.out.println();
            //获得id
            String id = hitsHit.getId();
            //获得得分
            float score = hitsHit.getScore();
            //获得内容
            Map<String, Object> sourceAsMap = hitsHit.getSourceAsMap();

            //打印内容
            System.out.println("id:" + id);
            System.out.println("score:" + score);
            System.out.println("内容：");
            for (String key : sourceAsMap.keySet())
            {
                System.out.println("---- " + key + "： " + sourceAsMap.get(key));
            }
            System.out.println("-----------------------------------");
        }
    }


    /**
     * range query ,异步请求
     * 范围查询
     * <p>
     * 请求内容：
     * <pre>
     *
     * GET /book/_search
     * {
     *     "query":
     *     {
     *       "range":
     *       {
     *         "price":
     *         {
     *           "gte": 69.2,
     *           "lte": 80
     *         }
     *       }
     *     }
     * }
     *
     * </pre>
```

```
     * <p>
     * 结果：
     * <pre>
     *
     * 总数量：1
     * 最大分数：1.0
     *
     *
     * id:3
     * score:1.0
     * 内容：
     * ---- price：78.6
     * ---- studymodel：201001
     * ---- name：spring开发基础
     * ---- description：spring 在java领域非常流行，java程序员都在用。
     * ---- pic：group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
     * ---- timestamp：2019-08-24 19:21:35
     * ---- tags：[spring, java]
     * ----------------------------------
     *
     * </pre>
     *
     * @throws Exception Exception
     */
    @Test
    void search_range_query_async() throws Exception
    {
        //构建搜索请求
        SearchRequest searchRequest = new SearchRequest("book");
        //构建请求体
        SearchSourceBuilder searchSourceBuilder = new SearchSourceBuilder();
        //查询

 searchSourceBuilder.query(QueryBuilders.rangeQuery("price").gte(69.2).lte(80));
        //放入请求中
        searchRequest.source(searchSourceBuilder);
        //发起异步请求
        client.searchAsync(searchRequest, RequestOptions.DEFAULT, new
ActionListener<SearchResponse>()
        {
            /**
             * 成功的回调
             *
             * @param searchResponse SearchResponse
             */
            @Override
            public void onResponse(SearchResponse searchResponse)
            {
                //获取数据
                SearchHits hits = searchResponse.getHits();
                //总数
                long value = hits.getTotalHits().value;
                System.out.println("总数量：" + value);
                float maxScore = hits.getMaxScore();
                System.out.println("最大分数：" + maxScore);
                System.out.println();
                SearchHit[] hitsHits = hits.getHits();
                //遍历数据
```

```
                for (SearchHit hitsHit : hitsHits)
                {
                    System.out.println();
                    //获得id
                    String id = hitsHit.getId();
                    //获得得分
                    float score = hitsHit.getScore();
                    //获得内容
                    Map<String, Object> sourceAsMap = hitsHit.getSourceAsMap();

                    //打印内容
                    System.out.println("id:" + id);
                    System.out.println("score:" + score);
                    System.out.println("内容: ");
                    for (String key : sourceAsMap.keySet())
                    {
                        System.out.println("---- " + key + ": " +
sourceAsMap.get(key));
                    }
                    System.out.println("----------------------------------");
                }
            }

            /**
             * 失败的回调
             *
             * @param e Exception
             */
            @Override
            public void onFailure(Exception e)
            {
                e.printStackTrace();
            }
        });
        //休眠2秒
        Thread.sleep(2000);
    }
```

term query

```
/**
     * term query
     * 字段为keyword时，存储和搜索都不分词
     * 请求内容:
     * <pre>
     *
     * GET /book/_search
     * {
     *     "query":
     *     {
     *       "term":
     *       {
     *         "description":
     *         {
     *           "value": "java程序员"
```

```
     *          }
     *        }
     *      }
     * }
     *
     * </pre>
     * <p>
     * 结果:
     * <pre>
     *
     * 总数量: 0
     * 最大分数: NaN
     *
     * </pre>
     *
     * @throws Exception Exception
     */
    @Test
    void search_term_query() throws Exception
    {
        //构建搜索请求
        SearchRequest searchRequest = new SearchRequest("book");
        //构建请求体
        SearchSourceBuilder searchSourceBuilder = new SearchSourceBuilder();
        //查询
        searchSourceBuilder.query(QueryBuilders.termQuery("description", "java程
序员"));
        //放入请求中
        searchRequest.source(searchSourceBuilder);
        //发起请求
        SearchResponse searchResponse = client.search(searchRequest,
RequestOptions.DEFAULT);
        //获取数据
        SearchHits hits = searchResponse.getHits();
        //总数
        long value = hits.getTotalHits().value;
        System.out.println("总数量: " + value);
        float maxScore = hits.getMaxScore();
        System.out.println("最大分数: " + maxScore);
        System.out.println();
        SearchHit[] hitsHits = hits.getHits();
        //遍历数据
        for (SearchHit hitsHit : hitsHits)
        {
            System.out.println();
            //获得id
            String id = hitsHit.getId();
            //获得得分
            float score = hitsHit.getScore();
            //获得内容
            Map<String, Object> sourceAsMap = hitsHit.getSourceAsMap();

            //打印内容
            System.out.println("id:" + id);
            System.out.println("score:" + score);
            System.out.println("内容: ");
            for (String key : sourceAsMap.keySet())
            {
```

```
                System.out.println("---- " + key + ": " + sourceAsMap.get(key));
            }
            System.out.println("-----------------------------------");
        }
    }


    /**
     * term query ,异步请求
     * 字段为keyword时，存储和搜索都不分词
     * 请求内容：
     * <pre>
     *
     * GET /book/_search
     * {
     *     "query":
     *     {
     *       "term":
     *       {
     *         "description":
     *         {
     *           "value": "java程序员"
     *         }
     *       }
     *     }
     * }
     *
     * </pre>
     * <p>
     * 结果：
     * <pre>
     *
     * 总数量：0
     * 最大分数：NaN
     *
     * </pre>
     *
     * @throws Exception Exception
     */
    @Test
    void search_term_query_async() throws Exception
    {
        //构建搜索请求
        SearchRequest searchRequest = new SearchRequest("book");
        //构建请求体
        SearchSourceBuilder searchSourceBuilder = new SearchSourceBuilder();
        //查询
        searchSourceBuilder.query(QueryBuilders.termQuery("description", "java程
序员"));
        //放入请求中
        searchRequest.source(searchSourceBuilder);
        //发起异步请求
        client.searchAsync(searchRequest, RequestOptions.DEFAULT, new
ActionListener<SearchResponse>()
        {
            /**
             * 成功的回调
             *
```

```
             * @param searchResponse SearchResponse
             */
            @Override
            public void onResponse(SearchResponse searchResponse)
            {
                //获取数据
                SearchHits hits = searchResponse.getHits();
                //总数
                long value = hits.getTotalHits().value;
                System.out.println("总数量: " + value);
                float maxScore = hits.getMaxScore();
                System.out.println("最大分数: " + maxScore);
                System.out.println();
                SearchHit[] hitsHits = hits.getHits();
                //遍历数据
                for (SearchHit hitsHit : hitsHits)
                {
                    System.out.println();
                    //获得id
                    String id = hitsHit.getId();
                    //获得得分
                    float score = hitsHit.getScore();
                    //获得内容
                    Map<String, Object> sourceAsMap = hitsHit.getSourceAsMap();

                    //打印内容
                    System.out.println("id:" + id);
                    System.out.println("score:" + score);
                    System.out.println("内容: ");
                    for (String key : sourceAsMap.keySet())
                    {
                        System.out.println("---- " + key + ": " +
sourceAsMap.get(key));
                    }
                    System.out.println("-----------------------------------");
                }
            }

            /**
             * 失败的回调
             *
             * @param e Exception
             */
            @Override
            public void onFailure(Exception e)
            {
                e.printStackTrace();
            }
        });
        //休眠2秒
        Thread.sleep(2000);
    }
```

terms query

```
 /**
     * terms query
     * <p>
     * 请求内容：
     * <pre>
     *
     * GET /book/_search
     * {
     *     "query":
     *     {
     *       "terms":
     *       {
     *         "description":
     *         [
     *           "spring",
     *           "第一编程语言"
     *         ]
     *       }
     *     }
     * }
     *
     * </pre>
     * <p>
     * 结果：
     * <pre>
     *
     * 总数量：1
     * 最大分数：1.0
     *
     *
     * id:3
     * score:1.0
     * 内容：
     * ---- price：78.6
     * ---- studymodel：201001
     * ---- name：spring开发基础
     * ---- description：spring 在java领域非常流行，java程序员都在用。
     * ---- pic：group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
     * ---- timestamp：2019-08-24 19:21:35
     * ---- tags：[spring, java]
     * ----------------------------------
     *
     * </pre>
     *
     * @throws Exception Exception
     */
    @Test
    void search_terms_query() throws Exception
    {
        //构建搜索请求
        SearchRequest searchRequest = new SearchRequest("book");
        //构建请求体
        SearchSourceBuilder searchSourceBuilder = new SearchSourceBuilder();
        //查询
        searchSourceBuilder.query(QueryBuilders.termsQuery("description",
"spring", "第一编程语言"));
        //放入请求中
        searchRequest.source(searchSourceBuilder);
```

```
        //发起请求
        SearchResponse searchResponse = client.search(searchRequest,
RequestOptions.DEFAULT);
        //获取数据
        SearchHits hits = searchResponse.getHits();
        //总数
        long value = hits.getTotalHits().value;
        System.out.println("总数量：" + value);
        float maxScore = hits.getMaxScore();
        System.out.println("最大分数：" + maxScore);
        System.out.println();
        SearchHit[] hitsHits = hits.getHits();
        //遍历数据
        for (SearchHit hitsHit : hitsHits)
        {
            System.out.println();
            //获得id
            String id = hitsHit.getId();
            //获得得分
            float score = hitsHit.getScore();
            //获得内容
            Map<String, Object> sourceAsMap = hitsHit.getSourceAsMap();

            //打印内容
            System.out.println("id:" + id);
            System.out.println("score:" + score);
            System.out.println("内容：");
            for (String key : sourceAsMap.keySet())
            {
                System.out.println("---- " + key + ": " + sourceAsMap.get(key));
            }
            System.out.println("-----------------------------------");
        }
    }


    /**
     * terms query ,异步请求
     * <p>
     * 请求内容:
     * <pre>
     *
     * GET /book/_search
     * {
     *     "query":
     *     {
     *       "terms":
     *       {
     *         "description":
     *         [
     *           "spring",
     *           "第一编程语言"
     *         ]
     *       }
     *     }
     * }
     *
     * </pre>
```

```
     * <p>
     * 结果：
     * <pre>
     *
     * 总数量：1
     * 最大分数：1.0
     *
     *
     * id:3
     * score:1.0
     * 内容：
     * ---- price：78.6
     * ---- studymodel：201001
     * ---- name：spring开发基础
     * ---- description：spring 在java领域非常流行，java程序员都在用。
     * ---- pic：group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
     * ---- timestamp：2019-08-24 19:21:35
     * ---- tags：[spring, java]
     * ----------------------------------
     *
     * </pre>
     *
     * @throws Exception Exception
     */
    @Test
    void search_terms_query_async() throws Exception
    {
        //构建搜索请求
        SearchRequest searchRequest = new SearchRequest("book");
        //构建请求体
        SearchSourceBuilder searchSourceBuilder = new SearchSourceBuilder();
        //查询
        searchSourceBuilder.query(QueryBuilders.termsQuery("description",
"spring", "第一编程语言"));
        //放入请求中
        searchRequest.source(searchSourceBuilder);
        //发起异步请求
        client.searchAsync(searchRequest, RequestOptions.DEFAULT, new
ActionListener<SearchResponse>()
        {
            /**
             * 成功的回调
             *
             * @param searchResponse SearchResponse
             */
            @Override
            public void onResponse(SearchResponse searchResponse)
            {
                //获取数据
                SearchHits hits = searchResponse.getHits();
                //总数
                long value = hits.getTotalHits().value;
                System.out.println("总数量：" + value);
                float maxScore = hits.getMaxScore();
                System.out.println("最大分数：" + maxScore);
                System.out.println();
                SearchHit[] hitsHits = hits.getHits();
                //遍历数据
```

```
                for (SearchHit hitsHit : hitsHits)
                {
                    System.out.println();
                    //获得id
                    String id = hitsHit.getId();
                    //获得得分
                    float score = hitsHit.getScore();
                    //获得内容
                    Map<String, Object> sourceAsMap = hitsHit.getSourceAsMap();

                    //打印内容
                    System.out.println("id:" + id);
                    System.out.println("score:" + score);
                    System.out.println("内容: ");
                    for (String key : sourceAsMap.keySet())
                    {
                        System.out.println("---- " + key + ": " +
sourceAsMap.get(key));
                    }
                    System.out.println("----------------------------------");
                }
            }

            /**
             * 失败的回调
             *
             * @param e Exception
             */
            @Override
            public void onFailure(Exception e)
            {
                e.printStackTrace();
            }
        });
        //休眠2秒
        Thread.sleep(2000);
    }
```

exist query

```
/**
     * exist query
     * 查询有某些字段值的文档
     * <p>
     * 请求内容:
     * <pre>
     *
     * GET /book/_search
     * {
     *     "query":
     *     {
     *       "exists":
     *       {
     *         "field": "tags"
     *       }
```

```
     *     }
     * }
     *
     * </pre>
     * <p>
     * 结果：
     * <pre>
     *
     * 总数量：5
     * 最大分数：1.0
     *
     *
     * id:1
     * score:1.0
     * 内容：
     * ---- price：38.6
     * ---- studymodel：201002
     * ---- name：Bootstrap开发
     * ---- description：Bootstrap是由Twitter推出的一个前台页面开发css框架，是一个非常流行的开发框架，此框架集成了多种页面效果。此开发框架包含了大量的CSS、JS程序代码，可以帮助开发者（尤其是不擅长css页面开发的程序人员）轻松的实现一个css，不受浏览器限制的精美界面css效果。
     * ---- pic：group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
     * ---- timestamp：2019-08-25 19:11:35
     * ---- tags：[bootstrap, dev]
     * ----------------------------------
     *
     * id:2
     * score:1.0
     * 内容：
     * ---- price：68.6
     * ---- studymodel：201001
     * ---- name：java编程思想
     * ---- description：java语言是世界第一编程语言，在软件开发领域使用人数最多。
     * ---- pic：group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
     * ---- timestamp：2019-08-25 19:11:35
     * ---- tags：[java, dev]
     * ----------------------------------
     *
     * id:3
     * score:1.0
     * 内容：
     * ---- price：78.6
     * ---- studymodel：201001
     * ---- name：spring开发基础
     * ---- description：spring 在java领域非常流行，java程序员都在用。
     * ---- pic：group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
     * ---- timestamp：2019-08-24 19:21:35
     * ---- tags：[spring, java]
     * ----------------------------------
     *
     * id:5
     * score:1.0
     * 内容：
     * ---- price：68.6
     * ---- studymodel：201001
     * ---- name：java编程思想
     * ---- description：java语言是世界第一编程语言，在软件开发领域使用人数最多。
     * ---- pic：group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
```

```
     * ---- timestamp：2022-5-25 19:11:35
     * ---- tags：[bootstrap, dev]
     * ----------------------------------
     *
     * id:6
     * score:1.0
     * 内容：
     * ---- price：68.6
     * ---- studymodel：201001
     * ---- name：java编程思想
     * ---- description：java语言是世界第一编程语言，在软件开发领域使用人数最多。
     * ---- pic：group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
     * ---- timestamp：2022-5-25 19:11:35
     * ---- tags：[bootstrap, dev]
     * ----------------------------------
     *
     * </pre>
     *
     * @throws Exception Exception
     */
    @Test
    void search_exist_query() throws Exception
    {
        //构建搜索请求
        SearchRequest searchRequest = new SearchRequest("book");
        //构建请求体
        SearchSourceBuilder searchSourceBuilder = new SearchSourceBuilder();
        //查询
        searchSourceBuilder.query(QueryBuilders.existsQuery("tags"));
        //放入请求中
        searchRequest.source(searchSourceBuilder);
        //发起请求
        SearchResponse searchResponse = client.search(searchRequest,
RequestOptions.DEFAULT);
        //获取数据
        SearchHits hits = searchResponse.getHits();
        //总数
        long value = hits.getTotalHits().value;
        System.out.println("总数量：" + value);
        float maxScore = hits.getMaxScore();
        System.out.println("最大分数：" + maxScore);
        System.out.println();
        SearchHit[] hitsHits = hits.getHits();
        //遍历数据
        for (SearchHit hitsHit : hitsHits)
        {
            System.out.println();
            //获得id
            String id = hitsHit.getId();
            //获得得分
            float score = hitsHit.getScore();
            //获得内容
            Map<String, Object> sourceAsMap = hitsHit.getSourceAsMap();

            //打印内容
            System.out.println("id:" + id);
            System.out.println("score:" + score);
            System.out.println("内容：");
```

```
            for (String key : sourceAsMap.keySet())
            {
                System.out.println("---- " + key + ": " + sourceAsMap.get(key));
            }
            System.out.println("-----------------------------------");
        }
    }


    /**
     * exist query ，异步请求
     * 查询有某些字段值的文档
     * <p>
     * 请求内容：
     * <pre>
     *
     * GET /book/_search
     * {
     *     "query":
     *     {
     *       "exists":
     *       {
     *         "field": "tags"
     *       }
     *     }
     * }
     *
     * </pre>
     * <p>
     * 结果：
     * <pre>
     *
     * 总数量：5
     * 最大分数：1.0
     *
     *
     * id:1
     * score:1.0
     * 内容：
     * ---- price：38.6
     * ---- studymodel：201002
     * ---- name：Bootstrap开发
     * ---- description：Bootstrap是由Twitter推出的一个前台页面开发css框架，是一个非常流行的开发框架，此框架集成了多种页面效果。此开发框架包含了大量的CSS、JS程序代码，可以帮助开发者（尤其是不擅长css页面开发的程序人员）轻松的实现一个css，不受浏览器限制的精美界面css效果。
     * ---- pic：group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
     * ---- timestamp：2019-08-25 19:11:35
     * ---- tags：[bootstrap, dev]
     * ----------------------------------
     *
     * id:2
     * score:1.0
     * 内容：
     * ---- price：68.6
     * ---- studymodel：201001
     * ---- name：java编程思想
     * ---- description：java语言是世界第一编程语言，在软件开发领域使用人数最多。
     * ---- pic：group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
```

```
     * ---- timestamp：2019-08-25 19:11:35
     * ---- tags：[java, dev]
     * ----------------------------------
     *
     * id:3
     * score:1.0
     * 内容：
     * ---- price：78.6
     * ---- studymodel：201001
     * ---- name：spring开发基础
     * ---- description：spring 在java领域非常流行，java程序员都在用。
     * ---- pic：group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
     * ---- timestamp：2019-08-24 19:21:35
     * ---- tags：[spring, java]
     * ----------------------------------
     *
     * id:5
     * score:1.0
     * 内容：
     * ---- price：68.6
     * ---- studymodel：201001
     * ---- name：java编程思想
     * ---- description：java语言是世界第一编程语言，在软件开发领域使用人数最多。
     * ---- pic：group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
     * ---- timestamp：2022-5-25 19:11:35
     * ---- tags：[bootstrap, dev]
     * ----------------------------------
     *
     * id:6
     * score:1.0
     * 内容：
     * ---- price：68.6
     * ---- studymodel：201001
     * ---- name：java编程思想
     * ---- description：java语言是世界第一编程语言，在软件开发领域使用人数最多。
     * ---- pic：group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
     * ---- timestamp：2022-5-25 19:11:35
     * ---- tags：[bootstrap, dev]
     * ----------------------------------
     *
     * </pre>
     *
     * @throws Exception Exception
     */
    @Test
    void search_exist_query_async() throws Exception
    {
        //构建搜索请求
        SearchRequest searchRequest = new SearchRequest("book");
        //构建请求体
        SearchSourceBuilder searchSourceBuilder = new SearchSourceBuilder();
        //查询
        searchSourceBuilder.query(QueryBuilders.existsQuery("tags"));
        //放入请求中
        searchRequest.source(searchSourceBuilder);
        //发起异步请求
        client.searchAsync(searchRequest, RequestOptions.DEFAULT, new
ActionListener<SearchResponse>()
```

```
        {
            /**
             * 成功的回调
             *
             * @param searchResponse SearchResponse
             */
            @Override
            public void onResponse(SearchResponse searchResponse)
            {
                //获取数据
                SearchHits hits = searchResponse.getHits();
                //总数
                long value = hits.getTotalHits().value;
                System.out.println("总数量: " + value);
                float maxScore = hits.getMaxScore();
                System.out.println("最大分数: " + maxScore);
                System.out.println();
                SearchHit[] hitsHits = hits.getHits();
                //遍历数据
                for (SearchHit hitsHit : hitsHits)
                {
                    System.out.println();
                    //获得id
                    String id = hitsHit.getId();
                    //获得得分
                    float score = hitsHit.getScore();
                    //获得内容
                    Map<String, Object> sourceAsMap = hitsHit.getSourceAsMap();

                    //打印内容
                    System.out.println("id:" + id);
                    System.out.println("score:" + score);
                    System.out.println("内容: ");
                    for (String key : sourceAsMap.keySet())
                    {
                        System.out.println("---- " + key + ": " +
sourceAsMap.get(key));
                    }
                    System.out.println("------------------------------------");
                }
            }

            /**
             * 失败的回调
             *
             * @param e Exception
             */
            @Override
            public void onFailure(Exception e)
            {
                e.printStackTrace();
            }
        });
        //休眠2秒
        Thread.sleep(2000);
    }
```

Fuzzy query

```
/**
     * Fuzzy query
     * 返回包含与搜索词类似的词的文档，该词由Levenshtein编辑距离度量。
     * <p>
     * 请求内容：
     * <pre>
     *
     * GET /book/_search
     * {
     *     "query":
     *     {
     *       "fuzzy":
     *       {
     *         "name":
     *         {
     *           "value": "jaav"
     *         }
     *       }
     *     }
     * }
     *
     * </pre>
     * <p>
     * 结果：
     * <pre>
     *
     * 总数量：3
     * 最大分数：0.39036104
     *
     *
     * id:2
     * score:0.39036104
     * 内容：
     * ---- price：68.6
     * ---- studymodel：201001
     * ---- name：java编程思想
     * ---- description：java语言是世界第一编程语言，在软件开发领域使用人数最多。
     * ---- pic：group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
     * ---- timestamp：2019-08-25 19:11:35
     * ---- tags：[java, dev]
     * ------------------------------------
     *
     * id:5
     * score:0.39036104
     * 内容：
     * ---- price：68.6
     * ---- studymodel：201001
     * ---- name：java编程思想
     * ---- description：java语言是世界第一编程语言，在软件开发领域使用人数最多。
     * ---- pic：group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
     * ---- timestamp：2022-5-25 19:11:35
     * ---- tags：[bootstrap, dev]
     * ------------------------------------
```

```
     *
     * id:6
     * score:0.39036104
     * 内容:
     * ---- price: 68.6
     * ---- studymodel: 201001
     * ---- name: java编程思想
     * ---- description: java语言是世界第一编程语言，在软件开发领域使用人数最多。
     * ---- pic: group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
     * ---- timestamp: 2022-5-25 19:11:35
     * ---- tags: [bootstrap, dev]
     * ----------------------------------
     *
     * </pre>
     *
     * @throws Exception Exception
     */
    @Test
    void search_Fuzzy_query() throws Exception
    {
        //构建搜索请求
        SearchRequest searchRequest = new SearchRequest("book");
        //构建请求体
        SearchSourceBuilder searchSourceBuilder = new SearchSourceBuilder();
        //查询
        searchSourceBuilder.query(QueryBuilders.fuzzyQuery("name", "jaav"));
        //放入请求中
        searchRequest.source(searchSourceBuilder);
        //发起请求
        SearchResponse searchResponse = client.search(searchRequest,
RequestOptions.DEFAULT);
        //获取数据
        SearchHits hits = searchResponse.getHits();
        //总数
        long value = hits.getTotalHits().value;
        System.out.println("总数量: " + value);
        float maxScore = hits.getMaxScore();
        System.out.println("最大分数: " + maxScore);
        System.out.println();
        SearchHit[] hitsHits = hits.getHits();
        //遍历数据
        for (SearchHit hitsHit : hitsHits)
        {
            System.out.println();
            //获得id
            String id = hitsHit.getId();
            //获得得分
            float score = hitsHit.getScore();
            //获得内容
            Map<String, Object> sourceAsMap = hitsHit.getSourceAsMap();

            //打印内容
            System.out.println("id:" + id);
            System.out.println("score:" + score);
            System.out.println("内容: ");
            for (String key : sourceAsMap.keySet())
            {
                System.out.println("---- " + key + ": " + sourceAsMap.get(key));
```

```
            }
            System.out.println("----------------------------------");
        }
    }


    /**
     * Fuzzy query ，异步请求
     * 返回包含与搜索词类似的词的文档，该词由Levenshtein编辑距离度量。
     * <p>
     * 请求内容：
     * <pre>
     *
     * GET /book/_search
     * {
     *     "query":
     *     {
     *       "fuzzy":
     *       {
     *         "name":
     *         {
     *           "value": "jaav"
     *         }
     *       }
     *     }
     * }
     *
     * </pre>
     * <p>
     * 结果：
     * <pre>
     *
     * 总数量：3
     * 最大分数：0.39036104
     *
     *
     * id:2
     * score:0.39036104
     * 内容：
     * ---- price：68.6
     * ---- studymodel：201001
     * ---- name：java编程思想
     * ---- description：java语言是世界第一编程语言，在软件开发领域使用人数最多。
     * ---- pic：group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
     * ---- timestamp：2019-08-25 19:11:35
     * ---- tags：[java, dev]
     * ----------------------------------
     *
     * id:5
     * score:0.39036104
     * 内容：
     * ---- price：68.6
     * ---- studymodel：201001
     * ---- name：java编程思想
     * ---- description：java语言是世界第一编程语言，在软件开发领域使用人数最多。
     * ---- pic：group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
     * ---- timestamp：2022-5-25 19:11:35
     * ---- tags：[bootstrap, dev]
```

```
     * ----------------------------------
     *
     * id:6
     * score:0.39036104
     * 内容:
     * ---- price: 68.6
     * ---- studymodel: 201001
     * ---- name: java编程思想
     * ---- description: java语言是世界第一编程语言，在软件开发领域使用人数最多。
     * ---- pic: group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
     * ---- timestamp: 2022-5-25 19:11:35
     * ---- tags: [bootstrap, dev]
     * ----------------------------------
     *
     * </pre>
     *
     * @throws Exception Exception
     */
    @Test
    void search_Fuzzy_query_async() throws Exception
    {
        //构建搜索请求
        SearchRequest searchRequest = new SearchRequest("book");
        //构建请求体
        SearchSourceBuilder searchSourceBuilder = new SearchSourceBuilder();
        //查询
        searchSourceBuilder.query(QueryBuilders.fuzzyQuery("name", "jaav"));
        //放入请求中
        searchRequest.source(searchSourceBuilder);
        //发起异步请求
        client.searchAsync(searchRequest, RequestOptions.DEFAULT, new
ActionListener<SearchResponse>()
        {
            /**
             * 成功的回调
             *
             * @param searchResponse SearchResponse
             */
            @Override
            public void onResponse(SearchResponse searchResponse)
            {
                //获取数据
                SearchHits hits = searchResponse.getHits();
                //总数
                long value = hits.getTotalHits().value;
                System.out.println("总数量: " + value);
                float maxScore = hits.getMaxScore();
                System.out.println("最大分数: " + maxScore);
                System.out.println();
                SearchHit[] hitsHits = hits.getHits();
                //遍历数据
                for (SearchHit hitsHit : hitsHits)
                {
                    System.out.println();
                    //获得id
                    String id = hitsHit.getId();
                    //获得得分
                    float score = hitsHit.getScore();
```

```
                    //获得内容
                    Map<String, Object> sourceAsMap = hitsHit.getSourceAsMap();

                    //打印内容
                    System.out.println("id:" + id);
                    System.out.println("score:" + score);
                    System.out.println("内容: ");
                    for (String key : sourceAsMap.keySet())
                    {
                        System.out.println("---- " + key + ": " +
sourceAsMap.get(key));
                    }
                    System.out.println("-----------------------------------");
                }
            }

            /**
             * 失败的回调
             *
             * @param e Exception
             */
            @Override
            public void onFailure(Exception e)
            {
                e.printStackTrace();
            }
        });
        //休眠2秒
        Thread.sleep(2000);
    }
```

IDs

```
/**
     * IDs
     * 查询多个id为某个数的结果
     * <p>
     * 请求内容:
     * <pre>
     *
     * GET /book/_search
     * {
     *     "query":
     *     {
     *       "ids":
     *       {
     *         "values": ["1","5","3","100"]
     *       }
     *     }
     * }
     *
     * </pre>
     * <p>
     * 结果:
     * <pre>
```

```
     *
     * 总数量：3
     * 最大分数：1.0
     *
     *
     * id:1
     * score:1.0
     * 内容：
     * ---- price：38.6
     * ---- studymodel：201002
     * ---- name：Bootstrap开发
     * ---- description：Bootstrap是由Twitter推出的一个前台页面开发css框架，是一个非常流行的开发框架，此框架集成了多种页面效果。此开发框架包含了大量的CSS、JS程序代码，可以帮助开发者（尤其是不擅长css页面开发的程序人员）轻松的实现一个css，不受浏览器限制的精美界面css效果。
     * ---- pic：group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
     * ---- timestamp：2019-08-25 19:11:35
     * ---- tags：[bootstrap, dev]
     * ----------------------------------
     *
     * id:3
     * score:1.0
     * 内容：
     * ---- price：78.6
     * ---- studymodel：201001
     * ---- name：spring开发基础
     * ---- description：spring 在java领域非常流行，java程序员都在用。
     * ---- pic：group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
     * ---- timestamp：2019-08-24 19:21:35
     * ---- tags：[spring, java]
     * ----------------------------------
     *
     * id:5
     * score:1.0
     * 内容：
     * ---- price：68.6
     * ---- studymodel：201001
     * ---- name：java编程思想
     * ---- description：java语言是世界第一编程语言，在软件开发领域使用人数最多。
     * ---- pic：group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
     * ---- timestamp：2022-5-25 19:11:35
     * ---- tags：[bootstrap, dev]
     * ----------------------------------
     *
     * </pre>
     *
     * @throws Exception Exception
     */
    @Test
    void search_IDs() throws Exception
    {
        //构建搜索请求
        SearchRequest searchRequest = new SearchRequest("book");
        //构建请求体
        SearchSourceBuilder searchSourceBuilder = new SearchSourceBuilder();
        //查询
        searchSourceBuilder.query(QueryBuilders.idsQuery().addIds("1", "5", "3",
"100"));
        //放入请求中
```

```
        searchRequest.source(searchSourceBuilder);
        //发起请求
        SearchResponse searchResponse = client.search(searchRequest,
RequestOptions.DEFAULT);
        //获取数据
        SearchHits hits = searchResponse.getHits();
        //总数
        long value = hits.getTotalHits().value;
        System.out.println("总数量：" + value);
        float maxScore = hits.getMaxScore();
        System.out.println("最大分数：" + maxScore);
        System.out.println();
        SearchHit[] hitsHits = hits.getHits();
        //遍历数据
        for (SearchHit hitsHit : hitsHits)
        {
            System.out.println();
            //获得id
            String id = hitsHit.getId();
            //获得得分
            float score = hitsHit.getScore();
            //获得内容
            Map<String, Object> sourceAsMap = hitsHit.getSourceAsMap();

            //打印内容
            System.out.println("id:" + id);
            System.out.println("score:" + score);
            System.out.println("内容：");
            for (String key : sourceAsMap.keySet())
            {
                System.out.println("---- " + key + ": " + sourceAsMap.get(key));
            }
            System.out.println("-----------------------------------");
        }
    }


    /**
     * IDs , 异步请求
     * 查询多个id为某个数的结果
     * <p>
     * 请求内容：
     * <pre>
     *
     * GET /book/_search
     * {
     *     "query":
     *     {
     *       "ids":
     *       {
     *         "values": ["1","5","3","100"]
     *       }
     *     }
     * }
     *
     * </pre>
     * <p>
     * 结果：
```

```
     * <pre>
     *
     * 总数量：3
     * 最大分数：1.0
     *
     *
     * id:1
     * score:1.0
     * 内容：
     * ---- price：38.6
     * ---- studymodel：201002
     * ---- name：Bootstrap开发
     * ---- description：Bootstrap是由Twitter推出的一个前台页面开发css框架，是一个非常流行的开发框架，此框架集成了多种页面效果。此开发框架包含了大量的CSS、JS程序代码，可以帮助开发者（尤其是不擅长css页面开发的程序人员）轻松的实现一个css，不受浏览器限制的精美界面css效果。
     * ---- pic：group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
     * ---- timestamp：2019-08-25 19:11:35
     * ---- tags：[bootstrap, dev]
     * ----------------------------------
     *
     * id:3
     * score:1.0
     * 内容：
     * ---- price：78.6
     * ---- studymodel：201001
     * ---- name：spring开发基础
     * ---- description：spring 在java领域非常流行，java程序员都在用。
     * ---- pic：group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
     * ---- timestamp：2019-08-24 19:21:35
     * ---- tags：[spring, java]
     * ----------------------------------
     *
     * id:5
     * score:1.0
     * 内容：
     * ---- price：68.6
     * ---- studymodel：201001
     * ---- name：java编程思想
     * ---- description：java语言是世界第一编程语言，在软件开发领域使用人数最多。
     * ---- pic：group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
     * ---- timestamp：2022-5-25 19:11:35
     * ---- tags：[bootstrap, dev]
     * ----------------------------------
     *
     * </pre>
     *
     * @throws Exception Exception
     */
    @Test
    void search_IDs_async() throws Exception
    {
        //构建搜索请求
        SearchRequest searchRequest = new SearchRequest("book");
        //构建请求体
        SearchSourceBuilder searchSourceBuilder = new SearchSourceBuilder();
        //查询
        searchSourceBuilder.query(QueryBuilders.idsQuery().addIds("1", "5", "3",
"100"));
```

```
        //放入请求中
        searchRequest.source(searchSourceBuilder);
        //发起异步请求
        client.searchAsync(searchRequest, RequestOptions.DEFAULT, new
ActionListener<SearchResponse>()
        {
            /**
             * 成功的回调
             *
             * @param searchResponse SearchResponse
             */
            @Override
            public void onResponse(SearchResponse searchResponse)
            {
                //获取数据
                SearchHits hits = searchResponse.getHits();
                //总数
                long value = hits.getTotalHits().value;
                System.out.println("总数量: " + value);
                float maxScore = hits.getMaxScore();
                System.out.println("最大分数: " + maxScore);
                System.out.println();
                SearchHit[] hitsHits = hits.getHits();
                //遍历数据
                for (SearchHit hitsHit : hitsHits)
                {
                    System.out.println();
                    //获得id
                    String id = hitsHit.getId();
                    //获得得分
                    float score = hitsHit.getScore();
                    //获得内容
                    Map<String, Object> sourceAsMap = hitsHit.getSourceAsMap();

                    //打印内容
                    System.out.println("id:" + id);
                    System.out.println("score:" + score);
                    System.out.println("内容: ");
                    for (String key : sourceAsMap.keySet())
                    {
                        System.out.println("---- " + key + ": " +
sourceAsMap.get(key));
                    }
                    System.out.println("-----------------------------------");
                }
            }

            /**
             * 失败的回调
             *
             * @param e Exception
             */
            @Override
            public void onFailure(Exception e)
            {
                e.printStackTrace();
            }
        });
```

```
        //休眠2秒
        Thread.sleep(2000);
    }
```

prefix 前缀查询

```
/**
     * prefix
     * 查询某字段满足某前缀的所有数据
     * <p>
     * 请求内容：
     * <pre>
     *
     * GET /book/_search
     * {
     *     "query":
     *     {
     *       "prefix":
     *       {
     *         "description":
     *         {
     *           "value": "sprin"
     *         }
     *       }
     *     }
     * }
     *
     * </pre>
     * <p>
     * 结果：
     * <pre>
     *
     * 总数量：1
     * 最大分数：1.0
     *
     *
     * id:3
     * score:1.0
     * 内容：
     * ---- price：78.6
     * ---- studymodel：201001
     * ---- name：spring开发基础
     * ---- description：spring 在java领域非常流行，java程序员都在用。
     * ---- pic：group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
     * ---- timestamp：2019-08-24 19:21:35
     * ---- tags：[spring, java]
     * ----------------------------------
     *
     * </pre>
     *
     * @throws Exception
     */
    @Test
    void search_prefix() throws Exception
    {
```

```
        //构建搜索请求
        SearchRequest searchRequest = new SearchRequest("book");
        //构建请求体
        SearchSourceBuilder searchSourceBuilder = new SearchSourceBuilder();
        //查询
        searchSourceBuilder.query(QueryBuilders.prefixQuery("description",
"sprin"));
        //放入请求中
        searchRequest.source(searchSourceBuilder);
        //发起请求
        SearchResponse searchResponse = client.search(searchRequest,
RequestOptions.DEFAULT);
        //获取数据
        SearchHits hits = searchResponse.getHits();
        //总数
        long value = hits.getTotalHits().value;
        System.out.println("总数量: " + value);
        float maxScore = hits.getMaxScore();
        System.out.println("最大分数: " + maxScore);
        System.out.println();
        SearchHit[] hitsHits = hits.getHits();
        //遍历数据
        for (SearchHit hitsHit : hitsHits)
        {
            System.out.println();
            //获得id
            String id = hitsHit.getId();
            //获得得分
            float score = hitsHit.getScore();
            //获得内容
            Map<String, Object> sourceAsMap = hitsHit.getSourceAsMap();

            //打印内容
            System.out.println("id:" + id);
            System.out.println("score:" + score);
            System.out.println("内容: ");
            for (String key : sourceAsMap.keySet())
            {
                System.out.println("---- " + key + ": " + sourceAsMap.get(key));
            }
            System.out.println("-----------------------------------");
        }
    }


    /**
     * prefix , 异步请求
     * 查询某字段满足某前缀的所有数据
     * <p>
     * 请求内容:
     * <pre>
     *
     * GET /book/_search
     * {
     *     "query":
     *     {
     *       "prefix":
     *       {
```

```
     *          "description":
     *          {
     *            "value": "sprin"
     *          }
     *        }
     *     }
     * }
     *
     * </pre>
     * <p>
     * 结果：
     * <pre>
     *
     * 总数量：1
     * 最大分数：1.0
     *
     *
     * id:3
     * score:1.0
     * 内容：
     * ---- price：78.6
     * ---- studymodel：201001
     * ---- name：spring开发基础
     * ---- description：spring 在java领域非常流行，java程序员都在用。
     * ---- pic：group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
     * ---- timestamp：2019-08-24 19:21:35
     * ---- tags：[spring, java]
     * ----------------------------------
     *
     * </pre>
     *
     * @throws Exception
     */
    @Test
    void search_prefix_async() throws Exception
    {
        //构建搜索请求
        SearchRequest searchRequest = new SearchRequest("book");
        //构建请求体
        SearchSourceBuilder searchSourceBuilder = new SearchSourceBuilder();
        //查询
        searchSourceBuilder.query(QueryBuilders.prefixQuery("description",
"sprin"));
        //放入请求中
        searchRequest.source(searchSourceBuilder);
        //发起异步请求
        client.searchAsync(searchRequest, RequestOptions.DEFAULT, new
ActionListener<SearchResponse>()
        {
            /**
             * 成功的回调
             *
             * @param searchResponse SearchResponse
             */
            @Override
            public void onResponse(SearchResponse searchResponse)
            {
                //获取数据
```

```
                SearchHits hits = searchResponse.getHits();
                //总数
                long value = hits.getTotalHits().value;
                System.out.println("总数量：" + value);
                float maxScore = hits.getMaxScore();
                System.out.println("最大分数：" + maxScore);
                System.out.println();
                SearchHit[] hitsHits = hits.getHits();
                //遍历数据
                for (SearchHit hitsHit : hitsHits)
                {
                    System.out.println();
                    //获得id
                    String id = hitsHit.getId();
                    //获得得分
                    float score = hitsHit.getScore();
                    //获得内容
                    Map<String, Object> sourceAsMap = hitsHit.getSourceAsMap();

                    //打印内容
                    System.out.println("id:" + id);
                    System.out.println("score:" + score);
                    System.out.println("内容：");
                    for (String key : sourceAsMap.keySet())
                    {
                        System.out.println("---- " + key + ": " +
sourceAsMap.get(key));
                    }
                    System.out.println("-----------------------------------");
                }
            }

            /**
             * 失败的回调
             *
             * @param e Exception
             */
            @Override
            public void onFailure(Exception e)
            {
                e.printStackTrace();
            }
        });
        //休眠2秒
        Thread.sleep(2000);
    }
```

regexp query

```
/**
     * regexp query
     * 正则查询
     * 查询某字段满足某正则表达式的所有数据
     * <p>
     * 请求内容：
```

```
     * <pre>
     *
     * GET /book/_search
     * {
     *     "query":
     *     {
     *       "regexp":
     *       {
     *         "description":
     *         {
     *           "value": "j.*a",
     *           "flags": "ALL"
     *         }
     *       }
     *     }
     * }
     *
     * 结果：
     * <pre>
     *
     * 总数量：4
     * 最大分数：1.0
     *
     *
     * id:2
     * score:1.0
     * 内容：
     * ---- price：68.6
     * ---- studymodel：201001
     * ---- name：java编程思想
     * ---- description：java语言是世界第一编程语言，在软件开发领域使用人数最多。
     * ---- pic：group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
     * ---- timestamp：2019-08-25 19:11:35
     * ---- tags：[java, dev]
     * ----------------------------------
     *
     * id:3
     * score:1.0
     * 内容：
     * ---- price：78.6
     * ---- studymodel：201001
     * ---- name：spring开发基础
     * ---- description：spring 在java领域非常流行，java程序员都在用。
     * ---- pic：group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
     * ---- timestamp：2019-08-24 19:21:35
     * ---- tags：[spring, java]
     * ----------------------------------
     *
     * id:5
     * score:1.0
     * 内容：
     * ---- price：68.6
     * ---- studymodel：201001
     * ---- name：java编程思想
     * ---- description：java语言是世界第一编程语言，在软件开发领域使用人数最多。
     * ---- pic：group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
     * ---- timestamp：2022-5-25 19:11:35
     * ---- tags：[bootstrap, dev]
```

```
     * ----------------------------------
     *
     * id:6
     * score:1.0
     * 内容:
     * ---- price: 68.6
     * ---- studymodel: 201001
     * ---- name: java编程思想
     * ---- description: java语言是世界第一编程语言，在软件开发领域使用人数最多。
     * ---- pic: group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
     * ---- timestamp: 2022-5-25 19:11:35
     * ---- tags: [bootstrap, dev]
     * ----------------------------------
     *
     * </pre>
     *
     * </pre>
     *
     * @throws Exception
     */
    @Test
    void search_regexp_query() throws Exception
    {
        //构建搜索请求
        SearchRequest searchRequest = new SearchRequest("book");
        //构建请求体
        SearchSourceBuilder searchSourceBuilder = new SearchSourceBuilder();
        //查询
        searchSourceBuilder.query(QueryBuilders.regexpQuery("description",
"j.*a"));
        //放入请求中
        searchRequest.source(searchSourceBuilder);
        //发起请求
        SearchResponse searchResponse = client.search(searchRequest,
RequestOptions.DEFAULT);
        //获取数据
        SearchHits hits = searchResponse.getHits();
        //总数
        long value = hits.getTotalHits().value;
        System.out.println("总数量: " + value);
        float maxScore = hits.getMaxScore();
        System.out.println("最大分数: " + maxScore);
        System.out.println();
        SearchHit[] hitsHits = hits.getHits();
        //遍历数据
        for (SearchHit hitsHit : hitsHits)
        {
            System.out.println();
            //获得id
            String id = hitsHit.getId();
            //获得得分
            float score = hitsHit.getScore();
            //获得内容
            Map<String, Object> sourceAsMap = hitsHit.getSourceAsMap();

            //打印内容
            System.out.println("id:" + id);
            System.out.println("score:" + score);
```

```
            System.out.println("内容：");
            for (String key : sourceAsMap.keySet())
            {
                System.out.println("---- " + key + "：" + sourceAsMap.get(key));
            }
            System.out.println("-----------------------------------");
        }
    }


    /**
     * regexp query ， 异步请求
     * 正则查询
     * 查询某字段满足某正则表达式的所有数据
     * <p>
     * 请求内容：
     * <pre>
     *
     * GET /book/_search
     * {
     *     "query":
     *     {
     *       "regexp":
     *       {
     *         "description":
     *         {
     *           "value": "j.*a",
     *           "flags": "ALL"
     *         }
     *       }
     *     }
     * }
     *
     * </pre>
     * <p>
     * 结果：
     * <pre>
     *
     * 总数量：4
     * 最大分数：1.0
     *
     *
     * id:2
     * score:1.0
     * 内容：
     * ---- price：68.6
     * ---- studymodel：201001
     * ---- name：java编程思想
     * ---- description：java语言是世界第一编程语言，在软件开发领域使用人数最多。
     * ---- pic：group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
     * ---- timestamp：2019-08-25 19:11:35
     * ---- tags：[java, dev]
     * -----------------------------------
     *
     * id:3
     * score:1.0
     * 内容：
     * ---- price：78.6
```

```
     * ---- studymodel：201001
     * ---- name：spring开发基础
     * ---- description：spring 在java领域非常流行，java程序员都在用。
     * ---- pic：group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
     * ---- timestamp：2019-08-24 19:21:35
     * ---- tags：[spring, java]
     * ----------------------------------
     *
     * id:5
     * score:1.0
     * 内容：
     * ---- price：68.6
     * ---- studymodel：201001
     * ---- name：java编程思想
     * ---- description：java语言是世界第一编程语言，在软件开发领域使用人数最多。
     * ---- pic：group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
     * ---- timestamp：2022-5-25 19:11:35
     * ---- tags：[bootstrap, dev]
     * ----------------------------------
     *
     * id:6
     * score:1.0
     * 内容：
     * ---- price：68.6
     * ---- studymodel：201001
     * ---- name：java编程思想
     * ---- description：java语言是世界第一编程语言，在软件开发领域使用人数最多。
     * ---- pic：group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
     * ---- timestamp：2022-5-25 19:11:35
     * ---- tags：[bootstrap, dev]
     * ----------------------------------
     *
     * </pre>
     *
     * @throws Exception
     */
    @Test
    void search_regexp_query_async() throws Exception
    {
        //构建搜索请求
        SearchRequest searchRequest = new SearchRequest("book");
        //构建请求体
        SearchSourceBuilder searchSourceBuilder = new SearchSourceBuilder();
        //查询
        searchSourceBuilder.query(QueryBuilders.regexpQuery("description",
"j.*a"));
        //放入请求中
        searchRequest.source(searchSourceBuilder);
        //发起异步请求
        client.searchAsync(searchRequest, RequestOptions.DEFAULT, new
ActionListener<SearchResponse>()
        {
            /**
             * 成功的回调
             *
             * @param searchResponse SearchResponse
             */
            @Override
```

```
            public void onResponse(SearchResponse searchResponse)
            {
                //获取数据
                SearchHits hits = searchResponse.getHits();
                //总数
                long value = hits.getTotalHits().value;
                System.out.println("总数量：" + value);
                float maxScore = hits.getMaxScore();
                System.out.println("最大分数：" + maxScore);
                System.out.println();
                SearchHit[] hitsHits = hits.getHits();
                //遍历数据
                for (SearchHit hitsHit : hitsHits)
                {
                    System.out.println();
                    //获得id
                    String id = hitsHit.getId();
                    //获得得分
                    float score = hitsHit.getScore();
                    //获得内容
                    Map<String, Object> sourceAsMap = hitsHit.getSourceAsMap();

                    //打印内容
                    System.out.println("id:" + id);
                    System.out.println("score:" + score);
                    System.out.println("内容：");
                    for (String key : sourceAsMap.keySet())
                    {
                        System.out.println("---- " + key + ": " +
sourceAsMap.get(key));
                    }
                    System.out.println("-----------------------------------");
                }
            }

            /**
             * 失败的回调
             *
             * @param e Exception
             */
            @Override
            public void onFailure(Exception e)
            {
                e.printStackTrace();
            }
        });
        //休眠2秒
        Thread.sleep(2000);
    }
```

Filter

```
/**
```

```
     * Filter
     * <p>
     * 请求内容:
     * <pre>
     *
     * GET /book/_search
     * {
     *     "query":
     *     {
     *       "bool":
     *       {
     *         "must":
     *         [
     *         {
     *           "match":
     *           {
     *             "description": "java程序员"
     *           }
     *         }
     *       ],
     *       "filter":
     *       [
     *         {
     *           "range":
     *           {
     *             "price":
     *             {
     *               "gte": 60,
     *               "lte": 70
     *             }
     *           }
     *         }
     *       ]
     *       }
     *     }
     * }
     *
     * </pre>
     * <p>
     * 结果:
     * <pre>
     *
     * 总数量:3
     * 最大分数:0.4398797
     *
     *
     * id:2
     * score:0.4398797
     * 内容:
     * ---- price:68.6
     * ---- studymodel:201001
     * ---- name:java编程思想
     * ---- description:java语言是世界第一编程语言,在软件开发领域使用人数最多。
     * ---- pic:group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
     * ---- timestamp:2019-08-25 19:11:35
     * ---- tags:[java, dev]
     * ----------------------------------
     *
```

```
     * id:5
     * score:0.4398797
     * 内容：
     * ---- price：68.6
     * ---- studymodel：201001
     * ---- name：java编程思想
     * ---- description：java语言是世界第一编程语言，在软件开发领域使用人数最多。
     * ---- pic：group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
     * ---- timestamp：2022-5-25 19:11:35
     * ---- tags：[bootstrap, dev]
     * ----------------------------------
     *
     * id:6
     * score:0.4398797
     * 内容：
     * ---- price：68.6
     * ---- studymodel：201001
     * ---- name：java编程思想
     * ---- description：java语言是世界第一编程语言，在软件开发领域使用人数最多。
     * ---- pic：group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
     * ---- timestamp：2022-5-25 19:11:35
     * ---- tags：[bootstrap, dev]
     * ----------------------------------
     *
     * </pre>
     *
     * @throws Exception Exception
     */
    @Test
    void search_Filter() throws Exception
    {
        //构建搜索请求
        SearchRequest searchRequest = new SearchRequest("book");
        //构建请求体
        SearchSourceBuilder searchSourceBuilder = new SearchSourceBuilder();
        //查询
        searchSourceBuilder.query(QueryBuilders.boolQuery()
                .must(QueryBuilders.matchQuery("description", "java程序员"))
                .filter(QueryBuilders.rangeQuery("price").gte(60).lte(70)));
        //放入请求中
        searchRequest.source(searchSourceBuilder);
        //发起请求
        SearchResponse searchResponse = client.search(searchRequest,
RequestOptions.DEFAULT);
        //获取数据
        SearchHits hits = searchResponse.getHits();
        //总数
        long value = hits.getTotalHits().value;
        System.out.println("总数量：" + value);
        float maxScore = hits.getMaxScore();
        System.out.println("最大分数：" + maxScore);
        System.out.println();
        SearchHit[] hitsHits = hits.getHits();
        //遍历数据
        for (SearchHit hitsHit : hitsHits)
        {
            System.out.println();
            //获得id
```

```
            String id = hitsHit.getId();
            //获得得分
            float score = hitsHit.getScore();
            //获得内容
            Map<String, Object> sourceAsMap = hitsHit.getSourceAsMap();

            //打印内容
            System.out.println("id:" + id);
            System.out.println("score:" + score);
            System.out.println("内容：");
            for (String key : sourceAsMap.keySet())
            {
                System.out.println("---- " + key + ": " + sourceAsMap.get(key));
            }
            System.out.println("-----------------------------------");
        }
    }


    /**
     * Filter ，异步请求
     * <p>
     * 请求内容：
     * <pre>
     *
     * GET /book/_search
     * {
     *     "query":
     *     {
     *       "bool":
     *       {
     *         "must":
     *         [
     *         {
     *           "match":
     *           {
     *             "description": "java程序员"
     *           }
     *         }
     *       ],
     *       "filter":
     *       [
     *         {
     *           "range":
     *           {
     *             "price":
     *             {
     *               "gte": 60,
     *               "lte": 70
     *             }
     *           }
     *         }
     *       ]
     *       }
     *     }
     * }
     *
     * </pre>
```

```
     * <p>
     * 结果：
     * <pre>
     *
     * 总数量：3
     * 最大分数：0.4398797
     *
     *
     * id:2
     * score:0.4398797
     * 内容：
     * ---- price：68.6
     * ---- studymodel：201001
     * ---- name：java编程思想
     * ---- description：java语言是世界第一编程语言，在软件开发领域使用人数最多。
     * ---- pic：group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
     * ---- timestamp：2019-08-25 19:11:35
     * ---- tags：[java, dev]
     * ----------------------------------
     *
     * id:5
     * score:0.4398797
     * 内容：
     * ---- price：68.6
     * ---- studymodel：201001
     * ---- name：java编程思想
     * ---- description：java语言是世界第一编程语言，在软件开发领域使用人数最多。
     * ---- pic：group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
     * ---- timestamp：2022-5-25 19:11:35
     * ---- tags：[bootstrap, dev]
     * ----------------------------------
     *
     * id:6
     * score:0.4398797
     * 内容：
     * ---- price：68.6
     * ---- studymodel：201001
     * ---- name：java编程思想
     * ---- description：java语言是世界第一编程语言，在软件开发领域使用人数最多。
     * ---- pic：group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
     * ---- timestamp：2022-5-25 19:11:35
     * ---- tags：[bootstrap, dev]
     * ----------------------------------
     *
     * </pre>
     *
     * @throws Exception Exception
     */
    @Test
    void search_Filter_async() throws Exception
    {
        //构建搜索请求
        SearchRequest searchRequest = new SearchRequest("book");
        //构建请求体
        SearchSourceBuilder searchSourceBuilder = new SearchSourceBuilder();
        //查询
        searchSourceBuilder.query(QueryBuilders.boolQuery()
                .must(QueryBuilders.matchQuery("description", "java程序员"))
```

```
                .filter(QueryBuilders.rangeQuery("price").gte(60).lte(70)));
        //放入请求中
        searchRequest.source(searchSourceBuilder);
        //发起异步请求
        client.searchAsync(searchRequest, RequestOptions.DEFAULT, new
ActionListener<SearchResponse>()
        {
            /**
             * 成功的回调
             *
             * @param searchResponse SearchResponse
             */
            @Override
            public void onResponse(SearchResponse searchResponse)
            {
                //获取数据
                SearchHits hits = searchResponse.getHits();
                //总数
                long value = hits.getTotalHits().value;
                System.out.println("总数量: " + value);
                float maxScore = hits.getMaxScore();
                System.out.println("最大分数: " + maxScore);
                System.out.println();
                SearchHit[] hitsHits = hits.getHits();
                //遍历数据
                for (SearchHit hitsHit : hitsHits)
                {
                    System.out.println();
                    //获得id
                    String id = hitsHit.getId();
                    //获得得分
                    float score = hitsHit.getScore();
                    //获得内容
                    Map<String, Object> sourceAsMap = hitsHit.getSourceAsMap();

                    //打印内容
                    System.out.println("id:" + id);
                    System.out.println("score:" + score);
                    System.out.println("内容: ");
                    for (String key : sourceAsMap.keySet())
                    {
                        System.out.println("---- " + key + ": " +
sourceAsMap.get(key));
                    }
                    System.out.println("-----------------------------------");
                }
            }

            /**
             * 失败的回调
             *
             * @param e Exception
             */
            @Override
            public void onFailure(Exception e)
            {
                e.printStackTrace();
            }
```

```
        });
        //休眠2秒
        Thread.sleep(2000);
    }
```

完整代码

https://github.com/maomao124/elasticsearch_Implement_search.git

```
package mao.elasticsearch_implement_search;

import org.apache.http.HttpHost;
import org.elasticsearch.action.ActionListener;
import org.elasticsearch.action.search.SearchRequest;
import org.elasticsearch.action.search.SearchResponse;
import org.elasticsearch.client.RequestOptions;
import org.elasticsearch.client.RestClient;
import org.elasticsearch.client.RestHighLevelClient;
import org.elasticsearch.index.query.QueryBuilder;
import org.elasticsearch.index.query.QueryBuilders;
import org.elasticsearch.search.SearchHit;
import org.elasticsearch.search.SearchHits;
import org.elasticsearch.search.builder.SearchSourceBuilder;
import org.elasticsearch.search.sort.SortOrder;
import org.junit.jupiter.api.AfterAll;
import org.junit.jupiter.api.BeforeAll;
import org.junit.jupiter.api.Test;
import org.springframework.boot.test.context.SpringBootTest;

import java.io.IOException;
import java.util.Map;


/**
 * Project name(项目名称)：elasticsearch_Implement_search
 * Package(包名): mao.elasticsearch_implement_search
 * Class(类名): ElasticSearchTest
 * Author(作者）: mao
 * Author QQ：1296193245
 * GitHub：https://github.com/maomao124/
 * Date(创建日期)： 2022/5/28
 * Time(创建时间)： 20:12
 * Version(版本): 1.0
 * Description(描述)： SpringBootTest
 * <p>
 * 映射：
 * get /book/_mapping
 *
 * <pre>
 *
 * {
```

```
*   "book" :
*   {
*     "mappings" :
*     {
*       "properties" :
*       {
*         "description" :
*         {
*           "type" : "text",
*           "fields" :
*           {
*             "keyword" :
*             {
*               "type" : "keyword",
*               "ignore_above" : 256
*             }
*           }
*         },
*         "name" :
*         {
*           "type" : "text",
*           "fields" :
*           {
*             "keyword" :
*             {
*               "type" : "keyword",
*               "ignore_above" : 256
*             }
*           }
*         },
*         "pic" :
*         {
*           "type" : "text",
*           "fields" :
*           {
*             "keyword" :
*             {
*               "type" : "keyword",
*               "ignore_above" : 256
*             }
*           }
*         },
*         "price" :
*         {
*           "type" : "float"
*         },
*         "query" :
*         {
*           "type" : "text",
*           "fields" :
*           {
*             "keyword" :
*             {
*               "type" : "keyword",
*               "ignore_above" : 256
*             }
*           }
*         },
```

```
*         "studymodel" :
*         {
*           "type" : "text",
*           "fields" :
*           {
*             "keyword" :
*             {
*               "type" : "keyword",
*               "ignore_above" : 256
*             }
*           }
*         },
*         "tags" :
*         {
*           "type" : "text",
*           "fields" :
*           {
*             "keyword" :
*             {
*               "type" : "keyword",
*               "ignore_above" : 256
*             }
*           }
*         },
*         "timestamp" :
*         {
*           "type" : "text",
*           "fields" :
*           {
*             "keyword" :
*             {
*               "type" : "keyword",
*               "ignore_above" : 256
*             }
*           }
*         }
*       }
*     }
*   }
* }
*
*
* </pre>
* <p>
* <p>
* 数据:
* get book/_search
*
* <pre>
*
* {
*   "took" : 0,
*   "timed_out" : false,
*   "_shards" :
*   {
*     "total" : 1,
*     "successful" : 1,
*     "skipped" : 0,
```

```
 *     "failed" : 0
 *   },
 *   "hits" :
 *   {
 *     "total" :
 *     {
 *       "value" : 5,
 *       "relation" : "eq"
 *     },
 *     "max_score" : 1.0,
 *     "hits" :
 *     [
 *       {
 *         "_index" : "book",
 *         "_id" : "1",
 *         "_score" : 1.0,
 *         "_source" :
 *         {
 *           "name" : "Bootstrap开发",
 *           "description" : "Bootstrap是由Twitter推出的一个前台页面开发css框架，是一个非常流行的开发框架，此框架集成了多种页面效果。此开发框架包含了大量的CSS、JS程序代码，可以帮助开发者（尤其是不擅长css页面开发的程序人员）轻松的实现一个css，不受浏览器限制的精美界面css效果。",
 *           "studymodel" : "201002",
 *           "price" : 38.6,
 *           "timestamp" : "2019-08-25 19:11:35",
 *           "pic" : "group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg",
 *           "tags" :
 *           [
 *             "bootstrap",
 *             "dev"
 *           ]
 *         }
 *       },
 *       {
 *         "_index" : "book",
 *         "_id" : "2",
 *         "_score" : 1.0,
 *         "_source" :
 *         {
 *           "name" : "java编程思想",
 *           "description" : "java语言是世界第一编程语言，在软件开发领域使用人数最多。",
 *           "studymodel" : "201001",
 *           "price" : 68.6,
 *           "timestamp" : "2019-08-25 19:11:35",
 *           "pic" : "group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg",
 *           "tags" :
 *           [
 *             "java",
 *             "dev"
 *           ]
 *         }
 *       },
 *       {
 *         "_index" : "book",
 *         "_id" : "3",
 *         "_score" : 1.0,
 *         "_source" :
 *         {
```

```
 *           "name" : "spring开发基础",
 *           "description" : "spring 在java领域非常流行，java程序员都在用。",
 *           "studymodel" : "201001",
 *           "price" : 78.6,
 *           "timestamp" : "2019-08-24 19:21:35",
 *           "pic" : "group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg",
 *           "tags" :
 *           [
 *             "spring",
 *             "java"
 *           ]
 *         }
 *       },
 *       {
 *         "_index" : "book",
 *         "_id" : "5",
 *         "_score" : 1.0,
 *         "_source" :
 *         {
 *           "name" : "java编程思想",
 *           "description" : "java语言是世界第一编程语言，在软件开发领域使用人数最多。",
 *           "studymodel" : "201001",
 *           "price" : 68.6,
 *           "timestamp" : "2022-5-25 19:11:35",
 *           "pic" : "group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg",
 *           "tags" :
 *           [
 *             "bootstrap",
 *             "dev"
 *           ]
 *         }
 *       },
 *       {
 *         "_index" : "book",
 *         "_id" : "6",
 *         "_score" : 1.0,
 *         "_source" :
 *         {
 *           "name" : "java编程思想",
 *           "description" : "java语言是世界第一编程语言，在软件开发领域使用人数最多。",
 *           "studymodel" : "201001",
 *           "price" : 68.6,
 *           "timestamp" : "2022-5-25 19:11:35",
 *           "pic" : "group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg",
 *           "tags" :
 *           [
 *             "bootstrap",
 *             "dev"
 *           ]
 *         }
 *       }
 *     ]
 *   }
 * }
 *
 * </pre>
 */
@SpringBootTest
```

```
public class ElasticSearchTest
{

    private static RestHighLevelClient client;

    /**
     * Before all.
     */
    @BeforeAll
    static void beforeAll()
    {
        client = new RestHighLevelClient(RestClient.builder(
                new HttpHost("localhost", 9200, "http")));
    }

    /**
     * After all.
     *
     * @throws IOException the io exception
     */
    @AfterAll
    static void afterAll() throws IOException
    {
        client.close();
    }


    /**
     * 搜索全部
     * 请求：
     *
     * <pre>
     *
     * GET /book/_search
     * {
     *   "query": { "match_all": {} }
     * }
     *
     * </pre>
     * <p>
     * 结果：
     * <pre>
     *
     * 总数量：5
     *
     *
     * id:1
     * score:1.0
     * 内容：
     * ---- price：38.6
     * ---- studymodel：201002
     * ---- name：Bootstrap开发
     * ---- description：Bootstrap是由Twitter推出的一个前台页面开发css框架，是一个非常流行的开发框架，此框架集成了多种页面效果。此开发框架包含了大量的CSS、JS程序代码，可以帮助开发者（尤其是不擅长css页面开发的程序人员）轻松的实现一个css，不受浏览器限制的精美界面css效果。
     * ---- pic：group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
     * ---- timestamp：2019-08-25 19:11:35
     * ---- tags：[bootstrap, dev]
```

```
     * ----------------------------------
     *
     * id:2
     * score:1.0
     * 内容:
     * ---- price: 68.6
     * ---- studymodel: 201001
     * ---- name: java编程思想
     * ---- description: java语言是世界第一编程语言，在软件开发领域使用人数最多。
     * ---- pic: group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
     * ---- timestamp: 2019-08-25 19:11:35
     * ---- tags: [java, dev]
     * ----------------------------------
     *
     * id:3
     * score:1.0
     * 内容:
     * ---- price: 78.6
     * ---- studymodel: 201001
     * ---- name: spring开发基础
     * ---- description: spring 在java领域非常流行，java程序员都在用。
     * ---- pic: group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
     * ---- timestamp: 2019-08-24 19:21:35
     * ---- tags: [spring, java]
     * ----------------------------------
     *
     * id:5
     * score:1.0
     * 内容:
     * ---- price: 68.6
     * ---- studymodel: 201001
     * ---- name: java编程思想
     * ---- description: java语言是世界第一编程语言，在软件开发领域使用人数最多。
     * ---- pic: group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
     * ---- timestamp: 2022-5-25 19:11:35
     * ---- tags: [bootstrap, dev]
     * ----------------------------------
     *
     * id:6
     * score:1.0
     * 内容:
     * ---- price: 68.6
     * ---- studymodel: 201001
     * ---- name: java编程思想
     * ---- description: java语言是世界第一编程语言，在软件开发领域使用人数最多。
     * ---- pic: group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
     * ---- timestamp: 2022-5-25 19:11:35
     * ---- tags: [bootstrap, dev]
     * ----------------------------------
     *
     * </pre>
     *
     * @throws IOException IOException
     */
    @Test
    void searchAll() throws IOException
    {
        //构建搜索请求
```

```
        SearchRequest searchRequest = new SearchRequest("book");
        //构建请求体
        SearchSourceBuilder searchSourceBuilder = new SearchSourceBuilder();
        searchSourceBuilder.query(QueryBuilders.matchAllQuery());
        //放入请求中
        searchRequest.source(searchSourceBuilder);
        //发起请求
        SearchResponse searchResponse = client.search(searchRequest,
RequestOptions.DEFAULT);
        //获取数据
        SearchHits hits = searchResponse.getHits();
        //总数
        long value = hits.getTotalHits().value;
        System.out.println("总数量：" + value);
        System.out.println();
        SearchHit[] hitsHits = hits.getHits();
        //遍历数据
        for (SearchHit hitsHit : hitsHits)
        {
            System.out.println();
            //获得id
            String id = hitsHit.getId();
            //获得得分
            float score = hitsHit.getScore();
            //获得内容
            Map<String, Object> sourceAsMap = hitsHit.getSourceAsMap();

            //打印内容
            System.out.println("id:" + id);
            System.out.println("score:" + score);
            System.out.println("内容：");
            for (String key : sourceAsMap.keySet())
            {
                System.out.println("---- " + key + ": " + sourceAsMap.get(key));
            }
            System.out.println("-----------------------------------");
        }

    }


    /**
     * 搜索全部，异步请求。
     * 请求内容：
     * <pre>
     *
     * GET /book/_search
     * {
     *   "query": { "match_all": {} }
     * }
     *
     * </pre>
     *
     * @throws Exception Exception
     */
    @Test
    void searchAll_async() throws Exception
    {
```

```
        //构建搜索请求
        SearchRequest searchRequest = new SearchRequest("book");
        //构建请求体
        SearchSourceBuilder searchSourceBuilder = new SearchSourceBuilder();
        searchSourceBuilder.query(QueryBuilders.matchAllQuery());
        //放入请求中
        searchRequest.source(searchSourceBuilder);
        //发起异步请求
        client.searchAsync(searchRequest, RequestOptions.DEFAULT, new
ActionListener<SearchResponse>()
        {
            /**
             * 成功的回调
             *
             * @param searchResponse SearchResponse
             */
            @Override
            public void onResponse(SearchResponse searchResponse)
            {
                //获取数据
                SearchHits hits = searchResponse.getHits();
                //总数
                long value = hits.getTotalHits().value;
                System.out.println("总数量: " + value);
                System.out.println();
                SearchHit[] hitsHits = hits.getHits();
                //遍历数据
                for (SearchHit hitsHit : hitsHits)
                {
                    System.out.println();
                    //获得id
                    String id = hitsHit.getId();
                    //获得得分
                    float score = hitsHit.getScore();
                    //获得内容
                    Map<String, Object> sourceAsMap = hitsHit.getSourceAsMap();

                    //打印内容
                    System.out.println("id:" + id);
                    System.out.println("score:" + score);
                    System.out.println("内容: ");
                    for (String key : sourceAsMap.keySet())
                    {
                        System.out.println("---- " + key + ": " +
sourceAsMap.get(key));
                    }
                    System.out.println("-----------------------------------");
                }
            }

            /**
             * 失败的回调
             *
             * @param e Exception
             */
            @Override
            public void onFailure(Exception e)
            {
```

```
                e.printStackTrace();
            }
        });
        Thread.sleep(2000);
    }

    /**
     * 查询某字段
     * 请求内容：
     *
     * <pre>
     *
     * GET /book/_search
     * {
     *   "query":
     *   {
     *     "match":
     *     {
     *       "description": "java"
     *     }
     *   }
     * }
     *
     * </pre>
     * <p>
     * 结果：
     * <pre>
     *
     * 总数量：4
     *
     *
     * id:3
     * score:0.4745544
     * 内容：
     * ---- price：78.6
     * ---- studymodel：201001
     * ---- name：spring开发基础
     * ---- description：spring 在java领域非常流行，java程序员都在用。
     * ---- pic：group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
     * ---- timestamp：2019-08-24 19:21:35
     * ---- tags：[spring, java]
     * ----------------------------------
     *
     * id:2
     * score:0.33773077
     * 内容：
     * ---- price：68.6
     * ---- studymodel：201001
     * ---- name：java编程思想
     * ---- description：java语言是世界第一编程语言，在软件开发领域使用人数最多。
     * ---- pic：group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
     * ---- timestamp：2019-08-25 19:11:35
     * ---- tags：[java, dev]
     * ----------------------------------
     *
     * id:5
     * score:0.33773077
     * 内容：
```

```
     * ---- price：68.6
     * ---- studymodel：201001
     * ---- name：java编程思想
     * ---- description：java语言是世界第一编程语言，在软件开发领域使用人数最多。
     * ---- pic：group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
     * ---- timestamp：2022-5-25 19:11:35
     * ---- tags：[bootstrap, dev]
     * ------------------------------------
     *
     * id:6
     * score:0.33773077
     * 内容：
     * ---- price：68.6
     * ---- studymodel：201001
     * ---- name：java编程思想
     * ---- description：java语言是世界第一编程语言，在软件开发领域使用人数最多。
     * ---- pic：group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
     * ---- timestamp：2022-5-25 19:11:35
     * ---- tags：[bootstrap, dev]
     * ------------------------------------
     *
     * </pre>
     *
     * @throws Exception Exception
     */
    @Test
    void search() throws Exception
    {
        //构建搜索请求
        SearchRequest searchRequest = new SearchRequest("book");
        //构建请求体
        SearchSourceBuilder searchSourceBuilder = new SearchSourceBuilder();
        //查询某字段
        searchSourceBuilder.query(QueryBuilders.matchQuery("description",
"java"));
        //放入请求中
        searchRequest.source(searchSourceBuilder);
        //发起请求
        SearchResponse searchResponse = client.search(searchRequest,
RequestOptions.DEFAULT);
        //获取数据
        SearchHits hits = searchResponse.getHits();
        //总数
        long value = hits.getTotalHits().value;
        System.out.println("总数量：" + value);
        System.out.println();
        SearchHit[] hitsHits = hits.getHits();
        //遍历数据
        for (SearchHit hitsHit : hitsHits)
        {
            System.out.println();
            //获得id
            String id = hitsHit.getId();
            //获得得分
            float score = hitsHit.getScore();
            //获得内容
            Map<String, Object> sourceAsMap = hitsHit.getSourceAsMap();
```

```
            //打印内容
            System.out.println("id:" + id);
            System.out.println("score:" + score);
            System.out.println("内容：");
            for (String key : sourceAsMap.keySet())
            {
                System.out.println("---- " + key + ": " + sourceAsMap.get(key));
            }
            System.out.println("----------------------------------");
        }
    }

    /**
     * 查询某字段，异步请求
     * 请求内容：
     *
     * <pre>
     *
     * GET /book/_search
     * {
     *   "query":
     *   {
     *     "match":
     *     {
     *       "description": "java"
     *     }
     *   }
     * }
     *
     * </pre>
     * <p>
     * 结果：
     * <pre>
     *
     * 总数量：4
     *
     *
     * id:3
     * score:0.4745544
     * 内容：
     * ---- price：78.6
     * ---- studymodel：201001
     * ---- name：spring开发基础
     * ---- description：spring 在java领域非常流行，java程序员都在用。
     * ---- pic：group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
     * ---- timestamp：2019-08-24 19:21:35
     * ---- tags：[spring, java]
     * ----------------------------------
     *
     * id:2
     * score:0.33773077
     * 内容：
     * ---- price：68.6
     * ---- studymodel：201001
     * ---- name：java编程思想
     * ---- description：java语言是世界第一编程语言，在软件开发领域使用人数最多。
     * ---- pic：group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
     * ---- timestamp：2019-08-25 19:11:35
```

```
     * ---- tags：[java, dev]
     * ----------------------------------
     *
     * id:5
     * score:0.33773077
     * 内容：
     * ---- price：68.6
     * ---- studymodel：201001
     * ---- name：java编程思想
     * ---- description：java语言是世界第一编程语言，在软件开发领域使用人数最多。
     * ---- pic：group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
     * ---- timestamp：2022-5-25 19:11:35
     * ---- tags：[bootstrap, dev]
     * ----------------------------------
     *
     * id:6
     * score:0.33773077
     * 内容：
     * ---- price：68.6
     * ---- studymodel：201001
     * ---- name：java编程思想
     * ---- description：java语言是世界第一编程语言，在软件开发领域使用人数最多。
     * ---- pic：group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
     * ---- timestamp：2022-5-25 19:11:35
     * ---- tags：[bootstrap, dev]
     * ----------------------------------
     *
     * </pre>
     *
     * @throws Exception Exception
     */
    @Test
    void search_async() throws Exception
    {
        //构建搜索请求
        SearchRequest searchRequest = new SearchRequest("book");
        //构建请求体
        SearchSourceBuilder searchSourceBuilder = new SearchSourceBuilder();
        //查询某字段
        searchSourceBuilder.query(QueryBuilders.matchQuery("description",
"java"));
        //放入请求中
        searchRequest.source(searchSourceBuilder);
        //发起异步请求
        client.searchAsync(searchRequest, RequestOptions.DEFAULT, new
ActionListener<SearchResponse>()
        {
            /**
             * 成功的回调
             *
             * @param searchResponse SearchResponse
             */
            @Override
            public void onResponse(SearchResponse searchResponse)
            {
                //获取数据
                SearchHits hits = searchResponse.getHits();
                //总数
```

```
                long value = hits.getTotalHits().value;
                System.out.println("总数量：" + value);
                System.out.println();
                SearchHit[] hitsHits = hits.getHits();
                //遍历数据
                for (SearchHit hitsHit : hitsHits)
                {
                    System.out.println();
                    //获得id
                    String id = hitsHit.getId();
                    //获得得分
                    float score = hitsHit.getScore();
                    //获得内容
                    Map<String, Object> sourceAsMap = hitsHit.getSourceAsMap();

                    //打印内容
                    System.out.println("id:" + id);
                    System.out.println("score:" + score);
                    System.out.println("内容：");
                    for (String key : sourceAsMap.keySet())
                    {
                        System.out.println("---- " + key + ": " +
sourceAsMap.get(key));
                    }
                    System.out.println("-----------------------------------");
                }
            }

            /**
             * 失败的回调
             *
             * @param e Exception
             */
            @Override
            public void onFailure(Exception e)
            {
                e.printStackTrace();
            }
        });
        Thread.sleep(2000);
    }


    /**
     * 搜索并排序
     * 请求内容：
     * <pre>
     *
     * GET /book/_search
     * {
     *   "query":
     *   {
     *     "match":
     *     {
     *       "description": "java"
     *     }
     *   },
     *   "sort":
```

```
*   [
*     {
*       "price":
*       {
*         "order": "desc"
*       }
*     }
*   ]
* }
*
* </pre>
* <p>
* 结果：
* <pre>
*
* 总数量：4
*
*
* id:3
* score:NaN
* 内容：
* ---- price：78.6
* ---- studymodel：201001
* ---- name：spring开发基础
* ---- description：spring 在java领域非常流行，java程序员都在用。
* ---- pic：group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
* ---- timestamp：2019-08-24 19:21:35
* ---- tags：[spring, java]
* ------------------------------------
*
* id:2
* score:NaN
* 内容：
* ---- price：68.6
* ---- studymodel：201001
* ---- name：java编程思想
* ---- description：java语言是世界第一编程语言，在软件开发领域使用人数最多。
* ---- pic：group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
* ---- timestamp：2019-08-25 19:11:35
* ---- tags：[java, dev]
* ------------------------------------
*
* id:5
* score:NaN
* 内容：
* ---- price：68.6
* ---- studymodel：201001
* ---- name：java编程思想
* ---- description：java语言是世界第一编程语言，在软件开发领域使用人数最多。
* ---- pic：group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
* ---- timestamp：2022-5-25 19:11:35
* ---- tags：[bootstrap, dev]
* ------------------------------------
*
* id:6
* score:NaN
* 内容：
* ---- price：68.6
```

```
     * ---- studymodel：201001
     * ---- name：java编程思想
     * ---- description：java语言是世界第一编程语言，在软件开发领域使用人数最多。
     * ---- pic：group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
     * ---- timestamp：2022-5-25 19:11:35
     * ---- tags：[bootstrap, dev]
     * ----------------------------------
     *
     * </pre>
     *
     * @throws Exception Exception
     */
    @Test
    void search_sort() throws Exception
    {
        //构建搜索请求
        SearchRequest searchRequest = new SearchRequest("book");
        //构建请求体
        SearchSourceBuilder searchSourceBuilder = new SearchSourceBuilder();
        //查询某字段
        searchSourceBuilder.query(QueryBuilders.matchQuery("description",
"java"));
        //排序
        searchSourceBuilder.sort("price", SortOrder.DESC);
        //放入请求中
        searchRequest.source(searchSourceBuilder);
        //发起请求
        SearchResponse searchResponse = client.search(searchRequest,
RequestOptions.DEFAULT);
        //获取数据
        SearchHits hits = searchResponse.getHits();
        //总数
        long value = hits.getTotalHits().value;
        System.out.println("总数量：" + value);
        System.out.println();
        SearchHit[] hitsHits = hits.getHits();
        //遍历数据
        for (SearchHit hitsHit : hitsHits)
        {
            System.out.println();
            //获得id
            String id = hitsHit.getId();
            //获得得分
            float score = hitsHit.getScore();
            //获得内容
            Map<String, Object> sourceAsMap = hitsHit.getSourceAsMap();

            //打印内容
            System.out.println("id:" + id);
            System.out.println("score:" + score);
            System.out.println("内容：");
            for (String key : sourceAsMap.keySet())
            {
                System.out.println("---- " + key + ": " + sourceAsMap.get(key));
            }
            System.out.println("----------------------------------");
        }
    }
```

```
/**
 * 搜索并排序，异步请求
 * 请求内容：
 * <pre>
 *
 * GET /book/_search
 * {
 *   "query":
 *   {
 *     "match":
 *     {
 *       "description": "java"
 *     }
 *   },
 *   "sort":
 *   [
 *     {
 *       "price":
 *       {
 *         "order": "desc"
 *       }
 *     }
 *   ]
 * }
 *
 * </pre>
 * <p>
 * 结果：
 * <pre>
 *
 * 总数量：4
 *
 *
 * id:3
 * score:NaN
 * 内容：
 * ---- price：78.6
 * ---- studymodel：201001
 * ---- name：spring开发基础
 * ---- description：spring 在java领域非常流行，java程序员都在用。
 * ---- pic：group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
 * ---- timestamp：2019-08-24 19:21:35
 * ---- tags：[spring, java]
 * ----------------------------------
 *
 * id:2
 * score:NaN
 * 内容：
 * ---- price：68.6
 * ---- studymodel：201001
 * ---- name：java编程思想
 * ---- description：java语言是世界第一编程语言，在软件开发领域使用人数最多。
 * ---- pic：group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
 * ---- timestamp：2019-08-25 19:11:35
 * ---- tags：[java, dev]
 * ----------------------------------
 *
```

```
     * id:5
     * score:NaN
     * 内容：
     * ---- price：68.6
     * ---- studymodel：201001
     * ---- name：java编程思想
     * ---- description：java语言是世界第一编程语言，在软件开发领域使用人数最多。
     * ---- pic：group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
     * ---- timestamp：2022-5-25 19:11:35
     * ---- tags：[bootstrap, dev]
     * ----------------------------------
     *
     * id:6
     * score:NaN
     * 内容：
     * ---- price：68.6
     * ---- studymodel：201001
     * ---- name：java编程思想
     * ---- description：java语言是世界第一编程语言，在软件开发领域使用人数最多。
     * ---- pic：group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
     * ---- timestamp：2022-5-25 19:11:35
     * ---- tags：[bootstrap, dev]
     * ----------------------------------
     *
     * </pre>
     *
     * @throws Exception Exception
     */
    @Test
    void search_sort_async() throws Exception
    {
        //构建搜索请求
        SearchRequest searchRequest = new SearchRequest("book");
        //构建请求体
        SearchSourceBuilder searchSourceBuilder = new SearchSourceBuilder();
        //查询某字段
        searchSourceBuilder.query(QueryBuilders.matchQuery("description",
"java"));
        //排序
        searchSourceBuilder.sort("price", SortOrder.DESC);
        //放入请求中
        searchRequest.source(searchSourceBuilder);
        //发起异步请求
        client.searchAsync(searchRequest, RequestOptions.DEFAULT, new
ActionListener<SearchResponse>()
        {
            /**
             * 成功的回调
             *
             * @param searchResponse SearchResponse
             */
            @Override
            public void onResponse(SearchResponse searchResponse)
            {
                //获取数据
                SearchHits hits = searchResponse.getHits();
                //总数
                long value = hits.getTotalHits().value;
```

```
                System.out.println("总数量: " + value);
                System.out.println();
                SearchHit[] hitsHits = hits.getHits();
                //遍历数据
                for (SearchHit hitsHit : hitsHits)
                {
                    System.out.println();
                    //获得id
                    String id = hitsHit.getId();
                    //获得得分
                    float score = hitsHit.getScore();
                    //获得内容
                    Map<String, Object> sourceAsMap = hitsHit.getSourceAsMap();

                    //打印内容
                    System.out.println("id:" + id);
                    System.out.println("score:" + score);
                    System.out.println("内容: ");
                    for (String key : sourceAsMap.keySet())
                    {
                        System.out.println("---- " + key + ": " +
sourceAsMap.get(key));
                    }
                    System.out.println("----------------------------------");
                }
            }

            /**
             * 失败的回调
             *
             * @param e Exception
             */
            @Override
            public void onFailure(Exception e)
            {
                e.printStackTrace();
            }
        });
        //休眠2秒
        Thread.sleep(2000);
    }

    /**
     * 分页查询
     * 请求内容:
     * <pre>
     *
     * GET /book/_search
     * {
     *   "query":
     *   {
     *     "match_all": {}
     *   },
     *   "from": 1,
     *   "size": 2
     * }
     *
     * </pre>
```

```
     * <p>
     * 结果：
     * <pre>
     *
     * 总数量：5
     *
     *
     * id:2
     * score:1.0
     * 内容：
     * ---- price：68.6
     * ---- studymodel：201001
     * ---- name：java编程思想
     * ---- description：java语言是世界第一编程语言，在软件开发领域使用人数最多。
     * ---- pic：group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
     * ---- timestamp：2019-08-25 19:11:35
     * ---- tags：[java, dev]
     * ----------------------------------
     *
     * id:3
     * score:1.0
     * 内容：
     * ---- price：78.6
     * ---- studymodel：201001
     * ---- name：spring开发基础
     * ---- description：spring 在java领域非常流行，java程序员都在用。
     * ---- pic：group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
     * ---- timestamp：2019-08-24 19:21:35
     * ---- tags：[spring, java]
     * ----------------------------------
     *
     * </pre>
     *
     * @throws Exception Exception
     */
    @Test
    void search_page() throws Exception
    {
        //构建搜索请求
        SearchRequest searchRequest = new SearchRequest("book");
        //构建请求体
        SearchSourceBuilder searchSourceBuilder = new SearchSourceBuilder();
        //查询
        searchSourceBuilder.query(QueryBuilders.matchAllQuery());
        //分页
        searchSourceBuilder.from(1);
        searchSourceBuilder.size(2);
        //放入请求中
        searchRequest.source(searchSourceBuilder);
        //发起请求
        SearchResponse searchResponse = client.search(searchRequest,
RequestOptions.DEFAULT);
        //获取数据
        SearchHits hits = searchResponse.getHits();
        //总数
        long value = hits.getTotalHits().value;
        System.out.println("总数量：" + value);
        System.out.println();
```

```
        SearchHit[] hitsHits = hits.getHits();
        //遍历数据
        for (SearchHit hitsHit : hitsHits)
        {
            System.out.println();
            //获得id
            String id = hitsHit.getId();
            //获得得分
            float score = hitsHit.getScore();
            //获得内容
            Map<String, Object> sourceAsMap = hitsHit.getSourceAsMap();

            //打印内容
            System.out.println("id:" + id);
            System.out.println("score:" + score);
            System.out.println("内容：");
            for (String key : sourceAsMap.keySet())
            {
                System.out.println("---- " + key + ": " + sourceAsMap.get(key));
            }
            System.out.println("-----------------------------------");
        }
    }


    /**
     * 分页查询，异步
     * 请求内容：
     * <pre>
     *
     * GET /book/_search
     * {
     *   "query":
     *   {
     *     "match_all": {}
     *   },
     *   "from": 1,
     *   "size": 2
     * }
     *
     * </pre>
     * <p>
     * 结果：
     * <pre>
     *
     * 总数量：5
     *
     *
     * id:2
     * score:1.0
     * 内容：
     * ---- price: 68.6
     * ---- studymodel: 201001
     * ---- name: java编程思想
     * ---- description: java语言是世界第一编程语言，在软件开发领域使用人数最多。
     * ---- pic: group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
     * ---- timestamp: 2019-08-25 19:11:35
     * ---- tags: [java, dev]
```

```
     * ----------------------------------
     *
     * id:3
     * score:1.0
     * 内容:
     * ---- price: 78.6
     * ---- studymodel: 201001
     * ---- name: spring开发基础
     * ---- description: spring 在java领域非常流行，java程序员都在用。
     * ---- pic: group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
     * ---- timestamp: 2019-08-24 19:21:35
     * ---- tags: [spring, java]
     * ----------------------------------
     *
     * </pre>
     *
     * @throws Exception Exception
     */
    @Test
    void search_page_async() throws Exception
    {
        //构建搜索请求
        SearchRequest searchRequest = new SearchRequest("book");
        //构建请求体
        SearchSourceBuilder searchSourceBuilder = new SearchSourceBuilder();
        //查询
        searchSourceBuilder.query(QueryBuilders.matchAllQuery());
        //分页
        searchSourceBuilder.from(1);
        searchSourceBuilder.size(2);
        //放入请求中
        searchRequest.source(searchSourceBuilder);
        //发起异步请求
        client.searchAsync(searchRequest, RequestOptions.DEFAULT, new
ActionListener<SearchResponse>()
        {
            /**
             * 成功的回调
             *
             * @param searchResponse SearchResponse
             */
            @Override
            public void onResponse(SearchResponse searchResponse)
            {
                //获取数据
                SearchHits hits = searchResponse.getHits();
                //总数
                long value = hits.getTotalHits().value;
                System.out.println("总数量: " + value);
                System.out.println();
                SearchHit[] hitsHits = hits.getHits();
                //遍历数据
                for (SearchHit hitsHit : hitsHits)
                {
                    System.out.println();
                    //获得id
                    String id = hitsHit.getId();
                    //获得得分
```

```
                    float score = hitsHit.getScore();
                    //获得内容
                    Map<String, Object> sourceAsMap = hitsHit.getSourceAsMap();

                    //打印内容
                    System.out.println("id:" + id);
                    System.out.println("score:" + score);
                    System.out.println("内容：");
                    for (String key : sourceAsMap.keySet())
                    {
                        System.out.println("---- " + key + ": " +
sourceAsMap.get(key));
                    }
                    System.out.println("-----------------------------------");
                }
            }

            /**
             * 失败的回调
             *
             * @param e Exception
             */
            @Override
            public void onFailure(Exception e)
            {
                e.printStackTrace();
            }
        });
        //休眠2秒
        Thread.sleep(2000);
    }

    /**
     * 查询，返回指定的字段
     * 请求内容：
     * <pre>
     *
     * GET /book/_search
     * {
     *   "query":
     *   {
     *     "match_all": {}
     *   },
     *   "_source": ["name","price"]
     * }
     *
     * </pre>
     * <p>
     * 结果：
     * <pre>
     *
     * 总数量：5
     *
     *
     * id:1
     * score:1.0
     * 内容：
     * ---- price：38.6
```

```
     * ---- name：Bootstrap开发
     * ----------------------------------
     *
     * id:2
     * score:1.0
     * 内容：
     * ---- price：68.6
     * ---- name：java编程思想
     * ----------------------------------
     *
     * id:3
     * score:1.0
     * 内容：
     * ---- price：78.6
     * ---- name：spring开发基础
     * ----------------------------------
     *
     * id:5
     * score:1.0
     * 内容：
     * ---- price：68.6
     * ---- name：java编程思想
     * ----------------------------------
     *
     * id:6
     * score:1.0
     * 内容：
     * ---- price：68.6
     * ---- name：java编程思想
     * ----------------------------------
     *
     * </pre>
     *
     * @throws Exception Exception
     */
    @Test
    void search_field() throws Exception
    {
        //构建搜索请求
        SearchRequest searchRequest = new SearchRequest("book");
        //构建请求体
        SearchSourceBuilder searchSourceBuilder = new SearchSourceBuilder();
        //查询
        searchSourceBuilder.query(QueryBuilders.matchAllQuery());
        //指定某些字段
        searchSourceBuilder.fetchSource(new String[]{"name", "price"}, new
String[]{});
        //放入请求中
        searchRequest.source(searchSourceBuilder);
        //发起请求
        SearchResponse searchResponse = client.search(searchRequest,
RequestOptions.DEFAULT);
        //获取数据
        SearchHits hits = searchResponse.getHits();
        //总数
        long value = hits.getTotalHits().value;
        System.out.println("总数量：" + value);
        System.out.println();
```

```
        SearchHit[] hitsHits = hits.getHits();
        //遍历数据
        for (SearchHit hitsHit : hitsHits)
        {
            System.out.println();
            //获得id
            String id = hitsHit.getId();
            //获得得分
            float score = hitsHit.getScore();
            //获得内容
            Map<String, Object> sourceAsMap = hitsHit.getSourceAsMap();

            //打印内容
            System.out.println("id:" + id);
            System.out.println("score:" + score);
            System.out.println("内容：");
            for (String key : sourceAsMap.keySet())
            {
                System.out.println("---- " + key + ": " + sourceAsMap.get(key));
            }
            System.out.println("-----------------------------------");
        }
    }


    /**
     * 查询，返回指定的字段，异步请求
     * 请求内容：
     * <pre>
     *
     * GET /book/_search
     * {
     *   "query":
     *   {
     *     "match_all": {}
     *   },
     *   "_source": ["name","price"]
     * }
     *
     * </pre>
     * <p>
     * 结果：
     * <pre>
     *
     * 总数量：5
     *
     *
     * id:1
     * score:1.0
     * 内容：
     * ---- price: 38.6
     * ---- name: Bootstrap开发
     * -----------------------------------
     *
     * id:2
     * score:1.0
     * 内容：
     * ---- price: 68.6
```

```
     * ---- name：java编程思想
     * ----------------------------------
     *
     * id:3
     * score:1.0
     * 内容：
     * ---- price：78.6
     * ---- name：spring开发基础
     * ----------------------------------
     *
     * id:5
     * score:1.0
     * 内容：
     * ---- price：68.6
     * ---- name：java编程思想
     * ----------------------------------
     *
     * id:6
     * score:1.0
     * 内容：
     * ---- price：68.6
     * ---- name：java编程思想
     * ----------------------------------
     *
     * </pre>
     *
     * @throws Exception Exception
     */
    @Test
    void search_field_async() throws Exception
    {
        //构建搜索请求
        SearchRequest searchRequest = new SearchRequest("book");
        //构建请求体
        SearchSourceBuilder searchSourceBuilder = new SearchSourceBuilder();
        //查询
        searchSourceBuilder.query(QueryBuilders.matchAllQuery());
        //指定某些字段
        searchSourceBuilder.fetchSource(new String[]{"name", "price"}, new
String[]{});
        //放入请求中
        searchRequest.source(searchSourceBuilder);
        //发起异步请求
        client.searchAsync(searchRequest, RequestOptions.DEFAULT, new
ActionListener<SearchResponse>()
        {
            /**
             * 成功的回调
             *
             * @param searchResponse SearchResponse
             */
            @Override
            public void onResponse(SearchResponse searchResponse)
            {
                //获取数据
                SearchHits hits = searchResponse.getHits();
                //总数
                long value = hits.getTotalHits().value;
```

```
                System.out.println("总数量: " + value);
                System.out.println();
                SearchHit[] hitsHits = hits.getHits();
                //遍历数据
                for (SearchHit hitsHit : hitsHits)
                {
                    System.out.println();
                    //获得id
                    String id = hitsHit.getId();
                    //获得得分
                    float score = hitsHit.getScore();
                    //获得内容
                    Map<String, Object> sourceAsMap = hitsHit.getSourceAsMap();

                    //打印内容
                    System.out.println("id:" + id);
                    System.out.println("score:" + score);
                    System.out.println("内容: ");
                    for (String key : sourceAsMap.keySet())
                    {
                        System.out.println("---- " + key + ": " +
sourceAsMap.get(key));
                    }
                    System.out.println("----------------------------------");
                }
            }

            /**
             * 失败的回调
             *
             * @param e Exception
             */
            @Override
            public void onFailure(Exception e)
            {
                e.printStackTrace();
            }
        });
        //休眠2秒
        Thread.sleep(2000);
    }


    /**
     * 多搜索条件
     * 请求内容:
     * <pre>
     *
     * GET /book/_search
     * {
     *   "query":
     *   {
     *     "bool":
     *     {
     *       "must":
     *       [
     *         {
     *           "match":
```

```
*            {
*              "name": "编程"
*            }
*          }
*       ],
*       "should":
*       [
*         {
*           "match":
*           {
*             "name": "spring开发基础"
*           }
*         }
*       ]
*     }
*   }
* }
*
* </pre>
* <p>
* 结果：
* <pre>
*
* 总数量：3
*
*
* id:2
* score:1.0409627
* 内容：
* ---- price：68.6
* ---- studymodel：201001
* ---- name：java编程思想
* ---- description：java语言是世界第一编程语言，在软件开发领域使用人数最多。
* ---- pic：group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
* ---- timestamp：2019-08-25 19:11:35
* ---- tags：[java, dev]
* ----------------------------------
*
* id:5
* score:1.0409627
* 内容：
* ---- price：68.6
* ---- studymodel：201001
* ---- name：java编程思想
* ---- description：java语言是世界第一编程语言，在软件开发领域使用人数最多。
* ---- pic：group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
* ---- timestamp：2022-5-25 19:11:35
* ---- tags：[bootstrap, dev]
* ----------------------------------
*
* id:6
* score:1.0409627
* 内容：
* ---- price：68.6
* ---- studymodel：201001
* ---- name：java编程思想
* ---- description：java语言是世界第一编程语言，在软件开发领域使用人数最多。
* ---- pic：group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
```

```
     * ---- timestamp: 2022-5-25 19:11:35
     * ---- tags: [bootstrap, dev]
     * ----------------------------------
     *
     * </pre>
     *
     * @throws Exception Exception
     */
    @Test
    void search_multipleConditions() throws Exception
    {
        //构建搜索请求
        SearchRequest searchRequest = new SearchRequest("book");
        //构建请求体
        SearchSourceBuilder searchSourceBuilder = new SearchSourceBuilder();
        //查询
        searchSourceBuilder.query(QueryBuilders.boolQuery().
                must(QueryBuilders.matchQuery("name", "编程")).
                should(QueryBuilders.matchQuery("name", "spring开发基础")));

        //放入请求中
        searchRequest.source(searchSourceBuilder);
        //发起请求
        SearchResponse searchResponse = client.search(searchRequest,
RequestOptions.DEFAULT);
        //获取数据
        SearchHits hits = searchResponse.getHits();
        //总数
        long value = hits.getTotalHits().value;
        System.out.println("总数量: " + value);
        float maxScore = hits.getMaxScore();
        System.out.println("最大分数: " + maxScore);
        System.out.println();
        SearchHit[] hitsHits = hits.getHits();
        //遍历数据
        for (SearchHit hitsHit : hitsHits)
        {
            System.out.println();
            //获得id
            String id = hitsHit.getId();
            //获得得分
            float score = hitsHit.getScore();
            //获得内容
            Map<String, Object> sourceAsMap = hitsHit.getSourceAsMap();

            //打印内容
            System.out.println("id:" + id);
            System.out.println("score:" + score);
            System.out.println("内容: ");
            for (String key : sourceAsMap.keySet())
            {
                System.out.println("---- " + key + ": " + sourceAsMap.get(key));
            }
            System.out.println("----------------------------------");
        }
    }
```

```
/**
 * 多搜索条件，异步请求
 * 请求内容：
 * <pre>
 *
 * GET /book/_search
 * {
 *   "query":
 *   {
 *     "bool":
 *     {
 *       "must":
 *       [
 *         {
 *           "match":
 *           {
 *             "name": "编程"
 *           }
 *         }
 *       ],
 *       "should":
 *       [
 *         {
 *           "match":
 *           {
 *             "name": "spring开发基础"
 *           }
 *         }
 *       ]
 *     }
 *   }
 * }
 *
 * </pre>
 * <p>
 * 结果：
 * <pre>
 *
 * 总数量：3
 *
 *
 * id:2
 * score:1.0409627
 * 内容：
 * ---- price：68.6
 * ---- studymodel：201001
 * ---- name：java编程思想
 * ---- description：java语言是世界第一编程语言，在软件开发领域使用人数最多。
 * ---- pic：group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
 * ---- timestamp：2019-08-25 19:11:35
 * ---- tags：[java, dev]
 * ------------------------------------
 *
 * id:5
 * score:1.0409627
 * 内容：
 * ---- price：68.6
 * ---- studymodel：201001
```

```
     * ---- name：java编程思想
     * ---- description：java语言是世界第一编程语言，在软件开发领域使用人数最多。
     * ---- pic：group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
     * ---- timestamp：2022-5-25 19:11:35
     * ---- tags：[bootstrap, dev]
     * ----------------------------------
     *
     * id:6
     * score:1.0409627
     * 内容：
     * ---- price：68.6
     * ---- studymodel：201001
     * ---- name：java编程思想
     * ---- description：java语言是世界第一编程语言，在软件开发领域使用人数最多。
     * ---- pic：group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
     * ---- timestamp：2022-5-25 19:11:35
     * ---- tags：[bootstrap, dev]
     * ----------------------------------
     *
     * </pre>
     *
     * @throws Exception Exception
     */
    @Test
    void search_multipleConditions_async() throws Exception
    {
        //构建搜索请求
        SearchRequest searchRequest = new SearchRequest("book");
        //构建请求体
        SearchSourceBuilder searchSourceBuilder = new SearchSourceBuilder();
        //查询
        searchSourceBuilder.query(QueryBuilders.boolQuery().
                must(QueryBuilders.matchQuery("name", "编程")).
                should(QueryBuilders.matchQuery("name", "spring开发基础")));

        //放入请求中
        searchRequest.source(searchSourceBuilder);
        //发起异步请求
        client.searchAsync(searchRequest, RequestOptions.DEFAULT, new
ActionListener<SearchResponse>()
        {
            /**
             * 成功的回调
             *
             * @param searchResponse SearchResponse
             */
            @Override
            public void onResponse(SearchResponse searchResponse)
            {
                //获取数据
                SearchHits hits = searchResponse.getHits();
                //总数
                long value = hits.getTotalHits().value;
                System.out.println("总数量：" + value);
                float maxScore = hits.getMaxScore();
                System.out.println("最大分数：" + maxScore);
                System.out.println();
                SearchHit[] hitsHits = hits.getHits();
```

```
                //遍历数据
                for (SearchHit hitsHit : hitsHits)
                {
                    System.out.println();
                    //获得id
                    String id = hitsHit.getId();
                    //获得得分
                    float score = hitsHit.getScore();
                    //获得内容
                    Map<String, Object> sourceAsMap = hitsHit.getSourceAsMap();

                    //打印内容
                    System.out.println("id:" + id);
                    System.out.println("score:" + score);
                    System.out.println("内容: ");
                    for (String key : sourceAsMap.keySet())
                    {
                        System.out.println("---- " + key + ": " +
sourceAsMap.get(key));
                    }
                    System.out.println("-----------------------------------");
                }
            }

            /**
             * 失败的回调
             *
             * @param e Exception
             */
            @Override
            public void onFailure(Exception e)
            {
                e.printStackTrace();
            }
        });
        //休眠2秒
        Thread.sleep(2000);
    }


    /**
     * multi_match
     * 搜索在多个字段下是否包含某关键字
     * <p>
     * 请求内容:
     * <pre>
     *
     * GET /book/_search
     * {
     *     "query":
     *     {
     *       "multi_match":
     *       {
     *         "query": "语言",
     *         "fields": ["name","description"]
     *       }
     *     }
     * }
```

```
     *
     * </pre>
     * <p>
     * 结果：
     * <pre>
     *
     * 总数量：3
     * 最大分数：1.6503837
     *
     *
     * id:2
     * score:1.6503837
     * 内容：
     * ---- price：68.6
     * ---- studymodel：201001
     * ---- name：java编程思想
     * ---- description：java语言是世界第一编程语言，在软件开发领域使用人数最多。
     * ---- pic：group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
     * ---- timestamp：2019-08-25 19:11:35
     * ---- tags：[java, dev]
     * ----------------------------------
     *
     * id:5
     * score:1.6503837
     * 内容：
     * ---- price：68.6
     * ---- studymodel：201001
     * ---- name：java编程思想
     * ---- description：java语言是世界第一编程语言，在软件开发领域使用人数最多。
     * ---- pic：group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
     * ---- timestamp：2022-5-25 19:11:35
     * ---- tags：[bootstrap, dev]
     * ----------------------------------
     *
     * id:6
     * score:1.6503837
     * 内容：
     * ---- price：68.6
     * ---- studymodel：201001
     * ---- name：java编程思想
     * ---- description：java语言是世界第一编程语言，在软件开发领域使用人数最多。
     * ---- pic：group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
     * ---- timestamp：2022-5-25 19:11:35
     * ---- tags：[bootstrap, dev]
     * ----------------------------------
     *
     * </pre>
     *
     * @throws Exception Exception
     */
    @Test
    void search_multi_match() throws Exception
    {
        //构建搜索请求
        SearchRequest searchRequest = new SearchRequest("book");
        //构建请求体
        SearchSourceBuilder searchSourceBuilder = new SearchSourceBuilder();
        //查询
```

```
        searchSourceBuilder.query(QueryBuilders.multiMatchQuery("语言", "name",
"description"));
        //放入请求中
        searchRequest.source(searchSourceBuilder);
        //发起请求
        SearchResponse searchResponse = client.search(searchRequest,
RequestOptions.DEFAULT);
        //获取数据
        SearchHits hits = searchResponse.getHits();
        //总数
        long value = hits.getTotalHits().value;
        System.out.println("总数量：" + value);
        float maxScore = hits.getMaxScore();
        System.out.println("最大分数：" + maxScore);
        System.out.println();
        SearchHit[] hitsHits = hits.getHits();
        //遍历数据
        for (SearchHit hitsHit : hitsHits)
        {
            System.out.println();
            //获得id
            String id = hitsHit.getId();
            //获得得分
            float score = hitsHit.getScore();
            //获得内容
            Map<String, Object> sourceAsMap = hitsHit.getSourceAsMap();

            //打印内容
            System.out.println("id:" + id);
            System.out.println("score:" + score);
            System.out.println("内容：");
            for (String key : sourceAsMap.keySet())
            {
                System.out.println("---- " + key + ": " + sourceAsMap.get(key));
            }
            System.out.println("-----------------------------------");
        }
    }


    /**
     * multi_match ,异步请求
     * 搜索在多个字段下是否包含某关键字
     * <p>
     * 请求内容：
     * <pre>
     *
     * GET /book/_search
     * {
     *     "query":
     *     {
     *       "multi_match":
     *       {
     *         "query": "语言",
     *         "fields": ["name","description"]
     *       }
     *     }
     * }
```

```
     *
     * </pre>
     * <p>
     * 结果：
     * <pre>
     *
     * 总数量：3
     * 最大分数：1.6503837
     *
     *
     * id:2
     * score:1.6503837
     * 内容：
     * ---- price：68.6
     * ---- studymodel：201001
     * ---- name：java编程思想
     * ---- description：java语言是世界第一编程语言，在软件开发领域使用人数最多。
     * ---- pic：group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
     * ---- timestamp：2019-08-25 19:11:35
     * ---- tags：[java, dev]
     * ----------------------------------
     *
     * id:5
     * score:1.6503837
     * 内容：
     * ---- price：68.6
     * ---- studymodel：201001
     * ---- name：java编程思想
     * ---- description：java语言是世界第一编程语言，在软件开发领域使用人数最多。
     * ---- pic：group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
     * ---- timestamp：2022-5-25 19:11:35
     * ---- tags：[bootstrap, dev]
     * ----------------------------------
     *
     * id:6
     * score:1.6503837
     * 内容：
     * ---- price：68.6
     * ---- studymodel：201001
     * ---- name：java编程思想
     * ---- description：java语言是世界第一编程语言，在软件开发领域使用人数最多。
     * ---- pic：group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
     * ---- timestamp：2022-5-25 19:11:35
     * ---- tags：[bootstrap, dev]
     * ----------------------------------
     *
     * </pre>
     *
     * @throws Exception Exception
     */
    @Test
    void search_multi_match_async() throws Exception
    {
        //构建搜索请求
        SearchRequest searchRequest = new SearchRequest("book");
        //构建请求体
        SearchSourceBuilder searchSourceBuilder = new SearchSourceBuilder();
        //查询
```

```
        searchSourceBuilder.query(QueryBuilders.multiMatchQuery("语言", "name",
"description"));
        //放入请求中
        searchRequest.source(searchSourceBuilder);
        //发起异步请求
        client.searchAsync(searchRequest, RequestOptions.DEFAULT, new
ActionListener<SearchResponse>()
        {
            /**
             * 成功的回调
             *
             * @param searchResponse SearchResponse
             */
            @Override
            public void onResponse(SearchResponse searchResponse)
            {
                //获取数据
                SearchHits hits = searchResponse.getHits();
                //总数
                long value = hits.getTotalHits().value;
                System.out.println("总数量: " + value);
                float maxScore = hits.getMaxScore();
                System.out.println("最大分数: " + maxScore);
                System.out.println();
                SearchHit[] hitsHits = hits.getHits();
                //遍历数据
                for (SearchHit hitsHit : hitsHits)
                {
                    System.out.println();
                    //获得id
                    String id = hitsHit.getId();
                    //获得得分
                    float score = hitsHit.getScore();
                    //获得内容
                    Map<String, Object> sourceAsMap = hitsHit.getSourceAsMap();

                    //打印内容
                    System.out.println("id:" + id);
                    System.out.println("score:" + score);
                    System.out.println("内容: ");
                    for (String key : sourceAsMap.keySet())
                    {
                        System.out.println("---- " + key + ": " +
sourceAsMap.get(key));
                    }
                    System.out.println("-----------------------------------");
                }
            }

            /**
             * 失败的回调
             *
             * @param e Exception
             */
            @Override
            public void onFailure(Exception e)
            {
                e.printStackTrace();
```

```
            }
        });
        //休眠2秒
        Thread.sleep(2000);
    }


    /**
     * range query
     * 范围查询
     * <p>
     * 请求内容：
     * <pre>
     *
     * GET /book/_search
     * {
     *     "query":
     *     {
     *       "range":
     *       {
     *         "price":
     *         {
     *           "gte": 69.2,
     *           "lte": 80
     *         }
     *       }
     *     }
     * }
     *
     * </pre>
     * <p>
     * 结果：
     * <pre>
     *
     * 总数量：1
     * 最大分数：1.0
     *
     *
     * id:3
     * score:1.0
     * 内容：
     * ---- price：78.6
     * ---- studymodel：201001
     * ---- name：spring开发基础
     * ---- description：spring 在java领域非常流行，java程序员都在用。
     * ---- pic：group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
     * ---- timestamp：2019-08-24 19:21:35
     * ---- tags：[spring, java]
     * ----------------------------------
     *
     * </pre>
     *
     * @throws Exception Exception
     */
    @Test
    void search_range_query() throws Exception
    {
        //构建搜索请求
```

```
        SearchRequest searchRequest = new SearchRequest("book");
        //构建请求体
        SearchSourceBuilder searchSourceBuilder = new SearchSourceBuilder();
        //查询

 searchSourceBuilder.query(QueryBuilders.rangeQuery("price").gte(69.2).lte(80));
        //放入请求中
        searchRequest.source(searchSourceBuilder);
        //发起请求
        SearchResponse searchResponse = client.search(searchRequest,
RequestOptions.DEFAULT);
        //获取数据
        SearchHits hits = searchResponse.getHits();
        //总数
        long value = hits.getTotalHits().value;
        System.out.println("总数量：" + value);
        float maxScore = hits.getMaxScore();
        System.out.println("最大分数：" + maxScore);
        System.out.println();
        SearchHit[] hitsHits = hits.getHits();
        //遍历数据
        for (SearchHit hitsHit : hitsHits)
        {
            System.out.println();
            //获得id
            String id = hitsHit.getId();
            //获得得分
            float score = hitsHit.getScore();
            //获得内容
            Map<String, Object> sourceAsMap = hitsHit.getSourceAsMap();

            //打印内容
            System.out.println("id:" + id);
            System.out.println("score:" + score);
            System.out.println("内容：");
            for (String key : sourceAsMap.keySet())
            {
                System.out.println("---- " + key + ": " + sourceAsMap.get(key));
            }
            System.out.println("----------------------------------");
        }
    }


    /**
     * range query ,异步请求
     * 范围查询
     * <p>
     * 请求内容：
     * <pre>
     *
     * GET /book/_search
     * {
     *     "query":
     *     {
     *       "range":
     *       {
     *         "price":
```

```
     *          {
     *            "gte": 69.2,
     *            "lte": 80
     *          }
     *        }
     *     }
     * }
     *
     * </pre>
     * <p>
     * 结果：
     * <pre>
     *
     * 总数量：1
     * 最大分数：1.0
     *
     *
     * id:3
     * score:1.0
     * 内容：
     * ---- price：78.6
     * ---- studymodel：201001
     * ---- name：spring开发基础
     * ---- description：spring 在java领域非常流行，java程序员都在用。
     * ---- pic：group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
     * ---- timestamp：2019-08-24 19:21:35
     * ---- tags：[spring, java]
     * ----------------------------------
     *
     * </pre>
     *
     * @throws Exception Exception
     */
    @Test
    void search_range_query_async() throws Exception
    {
        //构建搜索请求
        SearchRequest searchRequest = new SearchRequest("book");
        //构建请求体
        SearchSourceBuilder searchSourceBuilder = new SearchSourceBuilder();
        //查询

 searchSourceBuilder.query(QueryBuilders.rangeQuery("price").gte(69.2).lte(80));
        //放入请求中
        searchRequest.source(searchSourceBuilder);
        //发起异步请求
        client.searchAsync(searchRequest, RequestOptions.DEFAULT, new
ActionListener<SearchResponse>()
        {
            /**
             * 成功的回调
             *
             * @param searchResponse SearchResponse
             */
            @Override
            public void onResponse(SearchResponse searchResponse)
            {
                //获取数据
```

```
                SearchHits hits = searchResponse.getHits();
                //总数
                long value = hits.getTotalHits().value;
                System.out.println("总数量：" + value);
                float maxScore = hits.getMaxScore();
                System.out.println("最大分数：" + maxScore);
                System.out.println();
                SearchHit[] hitsHits = hits.getHits();
                //遍历数据
                for (SearchHit hitsHit : hitsHits)
                {
                    System.out.println();
                    //获得id
                    String id = hitsHit.getId();
                    //获得得分
                    float score = hitsHit.getScore();
                    //获得内容
                    Map<String, Object> sourceAsMap = hitsHit.getSourceAsMap();

                    //打印内容
                    System.out.println("id:" + id);
                    System.out.println("score:" + score);
                    System.out.println("内容：");
                    for (String key : sourceAsMap.keySet())
                    {
                        System.out.println("---- " + key + ": " +
sourceAsMap.get(key));
                    }
                    System.out.println("-----------------------------------");
                }
            }

            /**
             * 失败的回调
             *
             * @param e Exception
             */
            @Override
            public void onFailure(Exception e)
            {
                e.printStackTrace();
            }
        });
        //休眠2秒
        Thread.sleep(2000);
    }


    /**
     * term query
     * 字段为keyword时，存储和搜索都不分词
     * 请求内容：
     * <pre>
     *
     * GET /book/_search
     * {
     *     "query":
     *     {
```

```
     *       "term":
     *       {
     *         "description":
     *         {
     *           "value": "java程序员"
     *         }
     *       }
     *     }
     * }
     *
     * </pre>
     * <p>
     * 结果：
     * <pre>
     *
     * 总数量：0
     * 最大分数：NaN
     *
     * </pre>
     *
     * @throws Exception Exception
     */
    @Test
    void search_term_query() throws Exception
    {
        //构建搜索请求
        SearchRequest searchRequest = new SearchRequest("book");
        //构建请求体
        SearchSourceBuilder searchSourceBuilder = new SearchSourceBuilder();
        //查询
        searchSourceBuilder.query(QueryBuilders.termQuery("description", "java程
序员"));
        //放入请求中
        searchRequest.source(searchSourceBuilder);
        //发起请求
        SearchResponse searchResponse = client.search(searchRequest,
RequestOptions.DEFAULT);
        //获取数据
        SearchHits hits = searchResponse.getHits();
        //总数
        long value = hits.getTotalHits().value;
        System.out.println("总数量：" + value);
        float maxScore = hits.getMaxScore();
        System.out.println("最大分数：" + maxScore);
        System.out.println();
        SearchHit[] hitsHits = hits.getHits();
        //遍历数据
        for (SearchHit hitsHit : hitsHits)
        {
            System.out.println();
            //获得id
            String id = hitsHit.getId();
            //获得得分
            float score = hitsHit.getScore();
            //获得内容
            Map<String, Object> sourceAsMap = hitsHit.getSourceAsMap();

            //打印内容
```

```
            System.out.println("id:" + id);
            System.out.println("score:" + score);
            System.out.println("内容：");
            for (String key : sourceAsMap.keySet())
            {
                System.out.println("---- " + key + ": " + sourceAsMap.get(key));
            }
            System.out.println("------------------------------------");
        }
    }


    /**
     * term query ,异步请求
     * 字段为keyword时，存储和搜索都不分词
     * 请求内容：
     * <pre>
     *
     * GET /book/_search
     * {
     *     "query":
     *     {
     *       "term":
     *       {
     *         "description":
     *         {
     *           "value": "java程序员"
     *         }
     *       }
     *     }
     * }
     *
     * </pre>
     * <p>
     * 结果：
     * <pre>
     *
     * 总数量：0
     * 最大分数：NaN
     *
     * </pre>
     *
     * @throws Exception Exception
     */
    @Test
    void search_term_query_async() throws Exception
    {
        //构建搜索请求
        SearchRequest searchRequest = new SearchRequest("book");
        //构建请求体
        SearchSourceBuilder searchSourceBuilder = new SearchSourceBuilder();
        //查询
        searchSourceBuilder.query(QueryBuilders.termQuery("description", "java程
序员"));
        //放入请求中
        searchRequest.source(searchSourceBuilder);
        //发起异步请求
```

```
        client.searchAsync(searchRequest, RequestOptions.DEFAULT, new
ActionListener<SearchResponse>()
        {
            /**
             * 成功的回调
             *
             * @param searchResponse SearchResponse
             */
            @Override
            public void onResponse(SearchResponse searchResponse)
            {
                //获取数据
                SearchHits hits = searchResponse.getHits();
                //总数
                long value = hits.getTotalHits().value;
                System.out.println("总数量：" + value);
                float maxScore = hits.getMaxScore();
                System.out.println("最大分数：" + maxScore);
                System.out.println();
                SearchHit[] hitsHits = hits.getHits();
                //遍历数据
                for (SearchHit hitsHit : hitsHits)
                {
                    System.out.println();
                    //获得id
                    String id = hitsHit.getId();
                    //获得得分
                    float score = hitsHit.getScore();
                    //获得内容
                    Map<String, Object> sourceAsMap = hitsHit.getSourceAsMap();

                    //打印内容
                    System.out.println("id:" + id);
                    System.out.println("score:" + score);
                    System.out.println("内容：");
                    for (String key : sourceAsMap.keySet())
                    {
                        System.out.println("---- " + key + "： " +
sourceAsMap.get(key));
                    }
                    System.out.println("-----------------------------------");
                }
            }

            /**
             * 失败的回调
             *
             * @param e Exception
             */
            @Override
            public void onFailure(Exception e)
            {
                e.printStackTrace();
            }
        });
        //休眠2秒
        Thread.sleep(2000);
    }
```

```
    /**
     * terms query
     * <p>
     * 请求内容：
     * <pre>
     *
     * GET /book/_search
     * {
     *     "query":
     *     {
     *       "terms":
     *       {
     *         "description":
     *         [
     *           "spring",
     *           "第一编程语言"
     *         ]
     *       }
     *     }
     * }
     *
     * </pre>
     * <p>
     * 结果：
     * <pre>
     *
     * 总数量：1
     * 最大分数：1.0
     *
     *
     * id:3
     * score:1.0
     * 内容：
     * ---- price：78.6
     * ---- studymodel：201001
     * ---- name：spring开发基础
     * ---- description：spring 在java领域非常流行，java程序员都在用。
     * ---- pic：group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
     * ---- timestamp：2019-08-24 19:21:35
     * ---- tags：[spring, java]
     * ----------------------------------
     *
     * </pre>
     *
     * @throws Exception Exception
     */
    @Test
    void search_terms_query() throws Exception
    {
        //构建搜索请求
        SearchRequest searchRequest = new SearchRequest("book");
        //构建请求体
        SearchSourceBuilder searchSourceBuilder = new SearchSourceBuilder();
        //查询
        searchSourceBuilder.query(QueryBuilders.termsQuery("description",
"spring", "第一编程语言"));
```

```
        //放入请求中
        searchRequest.source(searchSourceBuilder);
        //发起请求
        SearchResponse searchResponse = client.search(searchRequest,
RequestOptions.DEFAULT);
        //获取数据
        SearchHits hits = searchResponse.getHits();
        //总数
        long value = hits.getTotalHits().value;
        System.out.println("总数量：" + value);
        float maxScore = hits.getMaxScore();
        System.out.println("最大分数：" + maxScore);
        System.out.println();
        SearchHit[] hitsHits = hits.getHits();
        //遍历数据
        for (SearchHit hitsHit : hitsHits)
        {
            System.out.println();
            //获得id
            String id = hitsHit.getId();
            //获得得分
            float score = hitsHit.getScore();
            //获得内容
            Map<String, Object> sourceAsMap = hitsHit.getSourceAsMap();

            //打印内容
            System.out.println("id:" + id);
            System.out.println("score:" + score);
            System.out.println("内容：");
            for (String key : sourceAsMap.keySet())
            {
                System.out.println("---- " + key + ": " + sourceAsMap.get(key));
            }
            System.out.println("-----------------------------------");
        }
    }


    /**
     * terms query ,异步请求
     * <p>
     * 请求内容：
     * <pre>
     *
     * GET /book/_search
     * {
     *     "query":
     *     {
     *       "terms":
     *       {
     *         "description":
     *         [
     *           "spring",
     *           "第一编程语言"
     *         ]
     *       }
     *     }
     * }
```

```
     *
     * </pre>
     * <p>
     * 结果：
     * <pre>
     *
     * 总数量：1
     * 最大分数：1.0
     *
     *
     * id:3
     * score:1.0
     * 内容：
     * ---- price：78.6
     * ---- studymodel：201001
     * ---- name：spring开发基础
     * ---- description：spring 在java领域非常流行，java程序员都在用。
     * ---- pic：group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
     * ---- timestamp：2019-08-24 19:21:35
     * ---- tags：[spring, java]
     * ----------------------------------
     *
     * </pre>
     *
     * @throws Exception Exception
     */
    @Test
    void search_terms_query_async() throws Exception
    {
        //构建搜索请求
        SearchRequest searchRequest = new SearchRequest("book");
        //构建请求体
        SearchSourceBuilder searchSourceBuilder = new SearchSourceBuilder();
        //查询
        searchSourceBuilder.query(QueryBuilders.termsQuery("description",
"spring", "第一编程语言"));
        //放入请求中
        searchRequest.source(searchSourceBuilder);
        //发起异步请求
        client.searchAsync(searchRequest, RequestOptions.DEFAULT, new
ActionListener<SearchResponse>()
        {
            /**
             * 成功的回调
             *
             * @param searchResponse SearchResponse
             */
            @Override
            public void onResponse(SearchResponse searchResponse)
            {
                //获取数据
                SearchHits hits = searchResponse.getHits();
                //总数
                long value = hits.getTotalHits().value;
                System.out.println("总数量：" + value);
                float maxScore = hits.getMaxScore();
                System.out.println("最大分数：" + maxScore);
                System.out.println();
```

```
                SearchHit[] hitsHits = hits.getHits();
                //遍历数据
                for (SearchHit hitsHit : hitsHits)
                {
                    System.out.println();
                    //获得id
                    String id = hitsHit.getId();
                    //获得得分
                    float score = hitsHit.getScore();
                    //获得内容
                    Map<String, Object> sourceAsMap = hitsHit.getSourceAsMap();

                    //打印内容
                    System.out.println("id:" + id);
                    System.out.println("score:" + score);
                    System.out.println("内容：");
                    for (String key : sourceAsMap.keySet())
                    {
                        System.out.println("---- " + key + ": " +
sourceAsMap.get(key));
                    }
                    System.out.println("-----------------------------------");
                }
            }

            /**
             * 失败的回调
             *
             * @param e Exception
             */
            @Override
            public void onFailure(Exception e)
            {
                e.printStackTrace();
            }
        });
        //休眠2秒
        Thread.sleep(2000);
    }


    /**
     * exist query
     * 查询有某些字段值的文档
     * <p>
     * 请求内容：
     * <pre>
     *
     * GET /book/_search
     * {
     *     "query":
     *     {
     *       "exists":
     *       {
     *         "field": "tags"
     *       }
     *     }
     * }
```

```
     *
     * </pre>
     * <p>
     * 结果：
     * <pre>
     *
     * 总数量：5
     * 最大分数：1.0
     *
     *
     * id:1
     * score:1.0
     * 内容：
     * ---- price：38.6
     * ---- studymodel：201002
     * ---- name：Bootstrap开发
     * ---- description：Bootstrap是由Twitter推出的一个前台页面开发css框架，是一个非常流行的开发框架，此框架集成了多种页面效果。此开发框架包含了大量的CSS、JS程序代码，可以帮助开发者（尤其是不擅长css页面开发的程序人员）轻松的实现一个css，不受浏览器限制的精美界面css效果。
     * ---- pic：group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
     * ---- timestamp：2019-08-25 19:11:35
     * ---- tags：[bootstrap, dev]
     * ----------------------------------
     *
     * id:2
     * score:1.0
     * 内容：
     * ---- price：68.6
     * ---- studymodel：201001
     * ---- name：java编程思想
     * ---- description：java语言是世界第一编程语言，在软件开发领域使用人数最多。
     * ---- pic：group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
     * ---- timestamp：2019-08-25 19:11:35
     * ---- tags：[java, dev]
     * ----------------------------------
     *
     * id:3
     * score:1.0
     * 内容：
     * ---- price：78.6
     * ---- studymodel：201001
     * ---- name：spring开发基础
     * ---- description：spring 在java领域非常流行，java程序员都在用。
     * ---- pic：group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
     * ---- timestamp：2019-08-24 19:21:35
     * ---- tags：[spring, java]
     * ----------------------------------
     *
     * id:5
     * score:1.0
     * 内容：
     * ---- price：68.6
     * ---- studymodel：201001
     * ---- name：java编程思想
     * ---- description：java语言是世界第一编程语言，在软件开发领域使用人数最多。
     * ---- pic：group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
     * ---- timestamp：2022-5-25 19:11:35
     * ---- tags：[bootstrap, dev]
```

```
     * ----------------------------------
     *
     * id:6
     * score:1.0
     * 内容:
     * ---- price: 68.6
     * ---- studymodel: 201001
     * ---- name: java编程思想
     * ---- description: java语言是世界第一编程语言，在软件开发领域使用人数最多。
     * ---- pic: group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
     * ---- timestamp: 2022-5-25 19:11:35
     * ---- tags: [bootstrap, dev]
     * ----------------------------------
     *
     * </pre>
     *
     * @throws Exception Exception
     */
    @Test
    void search_exist_query() throws Exception
    {
        //构建搜索请求
        SearchRequest searchRequest = new SearchRequest("book");
        //构建请求体
        SearchSourceBuilder searchSourceBuilder = new SearchSourceBuilder();
        //查询
        searchSourceBuilder.query(QueryBuilders.existsQuery("tags"));
        //放入请求中
        searchRequest.source(searchSourceBuilder);
        //发起请求
        SearchResponse searchResponse = client.search(searchRequest,
RequestOptions.DEFAULT);
        //获取数据
        SearchHits hits = searchResponse.getHits();
        //总数
        long value = hits.getTotalHits().value;
        System.out.println("总数量: " + value);
        float maxScore = hits.getMaxScore();
        System.out.println("最大分数: " + maxScore);
        System.out.println();
        SearchHit[] hitsHits = hits.getHits();
        //遍历数据
        for (SearchHit hitsHit : hitsHits)
        {
            System.out.println();
            //获得id
            String id = hitsHit.getId();
            //获得得分
            float score = hitsHit.getScore();
            //获得内容
            Map<String, Object> sourceAsMap = hitsHit.getSourceAsMap();

            //打印内容
            System.out.println("id:" + id);
            System.out.println("score:" + score);
            System.out.println("内容: ");
            for (String key : sourceAsMap.keySet())
            {
```

```
                System.out.println("---- " + key + ": " + sourceAsMap.get(key));
            }
            System.out.println("----------------------------------");
        }
    }


    /**
     * exist query ，异步请求
     * 查询有某些字段值的文档
     * <p>
     * 请求内容：
     * <pre>
     *
     * GET /book/_search
     * {
     *     "query":
     *     {
     *       "exists":
     *       {
     *         "field": "tags"
     *       }
     *     }
     * }
     *
     * </pre>
     * <p>
     * 结果：
     * <pre>
     *
     * 总数量：5
     * 最大分数：1.0
     *
     *
     * id:1
     * score:1.0
     * 内容：
     * ---- price：38.6
     * ---- studymodel：201002
     * ---- name：Bootstrap开发
     * ---- description：Bootstrap是由Twitter推出的一个前台页面开发css框架，是一个非常流行的开发框架，此框架集成了多种页面效果。此开发框架包含了大量的CSS、JS程序代码，可以帮助开发者（尤其是不擅长css页面开发的程序人员）轻松的实现一个css，不受浏览器限制的精美界面css效果。
     * ---- pic：group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
     * ---- timestamp：2019-08-25 19:11:35
     * ---- tags：[bootstrap, dev]
     * ----------------------------------
     *
     * id:2
     * score:1.0
     * 内容：
     * ---- price：68.6
     * ---- studymodel：201001
     * ---- name：java编程思想
     * ---- description：java语言是世界第一编程语言，在软件开发领域使用人数最多。
     * ---- pic：group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
     * ---- timestamp：2019-08-25 19:11:35
     * ---- tags：[java, dev]
```

```
     * ----------------------------------
     *
     * id:3
     * score:1.0
     * 内容:
     * ---- price: 78.6
     * ---- studymodel: 201001
     * ---- name: spring开发基础
     * ---- description: spring 在java领域非常流行，java程序员都在用。
     * ---- pic: group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
     * ---- timestamp: 2019-08-24 19:21:35
     * ---- tags: [spring, java]
     * ----------------------------------
     *
     * id:5
     * score:1.0
     * 内容:
     * ---- price: 68.6
     * ---- studymodel: 201001
     * ---- name: java编程思想
     * ---- description: java语言是世界第一编程语言，在软件开发领域使用人数最多。
     * ---- pic: group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
     * ---- timestamp: 2022-5-25 19:11:35
     * ---- tags: [bootstrap, dev]
     * ----------------------------------
     *
     * id:6
     * score:1.0
     * 内容:
     * ---- price: 68.6
     * ---- studymodel: 201001
     * ---- name: java编程思想
     * ---- description: java语言是世界第一编程语言，在软件开发领域使用人数最多。
     * ---- pic: group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
     * ---- timestamp: 2022-5-25 19:11:35
     * ---- tags: [bootstrap, dev]
     * ----------------------------------
     *
     * </pre>
     *
     * @throws Exception Exception
     */
    @Test
    void search_exist_query_async() throws Exception
    {
        //构建搜索请求
        SearchRequest searchRequest = new SearchRequest("book");
        //构建请求体
        SearchSourceBuilder searchSourceBuilder = new SearchSourceBuilder();
        //查询
        searchSourceBuilder.query(QueryBuilders.existsQuery("tags"));
        //放入请求中
        searchRequest.source(searchSourceBuilder);
        //发起异步请求
        client.searchAsync(searchRequest, RequestOptions.DEFAULT, new
ActionListener<SearchResponse>()
        {
            /**
```

```
             * 成功的回调
             *
             * @param searchResponse SearchResponse
             */
            @Override
            public void onResponse(SearchResponse searchResponse)
            {
                //获取数据
                SearchHits hits = searchResponse.getHits();
                //总数
                long value = hits.getTotalHits().value;
                System.out.println("总数量: " + value);
                float maxScore = hits.getMaxScore();
                System.out.println("最大分数: " + maxScore);
                System.out.println();
                SearchHit[] hitsHits = hits.getHits();
                //遍历数据
                for (SearchHit hitsHit : hitsHits)
                {
                    System.out.println();
                    //获得id
                    String id = hitsHit.getId();
                    //获得得分
                    float score = hitsHit.getScore();
                    //获得内容
                    Map<String, Object> sourceAsMap = hitsHit.getSourceAsMap();

                    //打印内容
                    System.out.println("id:" + id);
                    System.out.println("score:" + score);
                    System.out.println("内容: ");
                    for (String key : sourceAsMap.keySet())
                    {
                        System.out.println("---- " + key + ": " +
sourceAsMap.get(key));
                    }
                    System.out.println("-----------------------------------");
                }
            }

            /**
             * 失败的回调
             *
             * @param e Exception
             */
            @Override
            public void onFailure(Exception e)
            {
                e.printStackTrace();
            }
        });
        //休眠2秒
        Thread.sleep(2000);
    }


    /**
     * Fuzzy query
```

```
     * 返回包含与搜索词类似的词的文档，该词由Levenshtein编辑距离度量。
     * <p>
     * 请求内容：
     * <pre>
     *
     * GET /book/_search
     * {
     *     "query":
     *     {
     *       "fuzzy":
     *       {
     *         "name":
     *         {
     *           "value": "jaav"
     *         }
     *       }
     *     }
     * }
     *
     * </pre>
     * <p>
     * 结果：
     * <pre>
     *
     * 总数量：3
     * 最大分数：0.39036104
     *
     *
     * id:2
     * score:0.39036104
     * 内容：
     * ---- price：68.6
     * ---- studymodel：201001
     * ---- name：java编程思想
     * ---- description：java语言是世界第一编程语言，在软件开发领域使用人数最多。
     * ---- pic：group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
     * ---- timestamp：2019-08-25 19:11:35
     * ---- tags：[java, dev]
     * ------------------------------------
     *
     * id:5
     * score:0.39036104
     * 内容：
     * ---- price：68.6
     * ---- studymodel：201001
     * ---- name：java编程思想
     * ---- description：java语言是世界第一编程语言，在软件开发领域使用人数最多。
     * ---- pic：group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
     * ---- timestamp：2022-5-25 19:11:35
     * ---- tags：[bootstrap, dev]
     * ------------------------------------
     *
     * id:6
     * score:0.39036104
     * 内容：
     * ---- price：68.6
     * ---- studymodel：201001
     * ---- name：java编程思想
```

```
     * ---- description：java语言是世界第一编程语言，在软件开发领域使用人数最多。
     * ---- pic：group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
     * ---- timestamp：2022-5-25 19:11:35
     * ---- tags：[bootstrap, dev]
     * ----------------------------------
     *
     * </pre>
     *
     * @throws Exception Exception
     */
    @Test
    void search_Fuzzy_query() throws Exception
    {
        //构建搜索请求
        SearchRequest searchRequest = new SearchRequest("book");
        //构建请求体
        SearchSourceBuilder searchSourceBuilder = new SearchSourceBuilder();
        //查询
        searchSourceBuilder.query(QueryBuilders.fuzzyQuery("name", "jaav"));
        //放入请求中
        searchRequest.source(searchSourceBuilder);
        //发起请求
        SearchResponse searchResponse = client.search(searchRequest,
RequestOptions.DEFAULT);
        //获取数据
        SearchHits hits = searchResponse.getHits();
        //总数
        long value = hits.getTotalHits().value;
        System.out.println("总数量：" + value);
        float maxScore = hits.getMaxScore();
        System.out.println("最大分数：" + maxScore);
        System.out.println();
        SearchHit[] hitsHits = hits.getHits();
        //遍历数据
        for (SearchHit hitsHit : hitsHits)
        {
            System.out.println();
            //获得id
            String id = hitsHit.getId();
            //获得得分
            float score = hitsHit.getScore();
            //获得内容
            Map<String, Object> sourceAsMap = hitsHit.getSourceAsMap();

            //打印内容
            System.out.println("id:" + id);
            System.out.println("score:" + score);
            System.out.println("内容：");
            for (String key : sourceAsMap.keySet())
            {
                System.out.println("---- " + key + "： " + sourceAsMap.get(key));
            }
            System.out.println("-----------------------------------");
        }
    }


    /**
```

```
     * Fuzzy query ，异步请求
     * 返回包含与搜索词类似的词的文档，该词由Levenshtein编辑距离度量。
     * <p>
     * 请求内容：
     * <pre>
     *
     * GET /book/_search
     * {
     *     "query":
     *     {
     *       "fuzzy":
     *       {
     *         "name":
     *         {
     *           "value": "jaav"
     *         }
     *       }
     *     }
     * }
     *
     * </pre>
     * <p>
     * 结果：
     * <pre>
     *
     * 总数量：3
     * 最大分数：0.39036104
     *
     *
     * id:2
     * score:0.39036104
     * 内容：
     * ---- price：68.6
     * ---- studymodel：201001
     * ---- name：java编程思想
     * ---- description：java语言是世界第一编程语言，在软件开发领域使用人数最多。
     * ---- pic：group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
     * ---- timestamp：2019-08-25 19:11:35
     * ---- tags：[java, dev]
     * ----------------------------------
     *
     * id:5
     * score:0.39036104
     * 内容：
     * ---- price：68.6
     * ---- studymodel：201001
     * ---- name：java编程思想
     * ---- description：java语言是世界第一编程语言，在软件开发领域使用人数最多。
     * ---- pic：group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
     * ---- timestamp：2022-5-25 19:11:35
     * ---- tags：[bootstrap, dev]
     * ----------------------------------
     *
     * id:6
     * score:0.39036104
     * 内容：
     * ---- price：68.6
     * ---- studymodel：201001
```

```
     * ---- name：java编程思想
     * ---- description：java语言是世界第一编程语言，在软件开发领域使用人数最多。
     * ---- pic：group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
     * ---- timestamp：2022-5-25 19:11:35
     * ---- tags：[bootstrap, dev]
     * ----------------------------------
     *
     * </pre>
     *
     * @throws Exception Exception
     */
    @Test
    void search_Fuzzy_query_async() throws Exception
    {
        //构建搜索请求
        SearchRequest searchRequest = new SearchRequest("book");
        //构建请求体
        SearchSourceBuilder searchSourceBuilder = new SearchSourceBuilder();
        //查询
        searchSourceBuilder.query(QueryBuilders.fuzzyQuery("name", "jaav"));
        //放入请求中
        searchRequest.source(searchSourceBuilder);
        //发起异步请求
        client.searchAsync(searchRequest, RequestOptions.DEFAULT, new
ActionListener<SearchResponse>()
        {
            /**
             * 成功的回调
             *
             * @param searchResponse SearchResponse
             */
            @Override
            public void onResponse(SearchResponse searchResponse)
            {
                //获取数据
                SearchHits hits = searchResponse.getHits();
                //总数
                long value = hits.getTotalHits().value;
                System.out.println("总数量：" + value);
                float maxScore = hits.getMaxScore();
                System.out.println("最大分数：" + maxScore);
                System.out.println();
                SearchHit[] hitsHits = hits.getHits();
                //遍历数据
                for (SearchHit hitsHit : hitsHits)
                {
                    System.out.println();
                    //获得id
                    String id = hitsHit.getId();
                    //获得得分
                    float score = hitsHit.getScore();
                    //获得内容
                    Map<String, Object> sourceAsMap = hitsHit.getSourceAsMap();

                    //打印内容
                    System.out.println("id:" + id);
                    System.out.println("score:" + score);
                    System.out.println("内容：");
```

```
                    for (String key : sourceAsMap.keySet())
                    {
                        System.out.println("---- " + key + ": " +
sourceAsMap.get(key));
                    }
                    System.out.println("----------------------------------");
                }
            }

            /**
             * 失败的回调
             *
             * @param e Exception
             */
            @Override
            public void onFailure(Exception e)
            {
                e.printStackTrace();
            }
        });
        //休眠2秒
        Thread.sleep(2000);
    }


    /**
     * IDs
     * 查询多个id为某个数的结果
     * <p>
     * 请求内容:
     * <pre>
     *
     * GET /book/_search
     * {
     *     "query":
     *     {
     *       "ids":
     *       {
     *         "values": ["1","5","3","100"]
     *       }
     *     }
     * }
     *
     * </pre>
     * <p>
     * 结果:
     * <pre>
     *
     * 总数量: 3
     * 最大分数: 1.0
     *
     *
     * id:1
     * score:1.0
     * 内容:
     * ---- price: 38.6
     * ---- studymodel: 201002
     * ---- name: Bootstrap开发
```

```
     * ---- description：Bootstrap是由Twitter推出的一个前台页面开发css框架，是一个非常流行的开发框架，此框架集成了多种页面效果。此开发框架包含了大量的CSS、JS程序代码，可以帮助开发者（尤其是不擅长css页面开发的程序人员）轻松的实现一个css，不受浏览器限制的精美界面css效果。
     * ---- pic：group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
     * ---- timestamp：2019-08-25 19:11:35
     * ---- tags：[bootstrap, dev]
     * ---------------------------------
     *
     * id:3
     * score:1.0
     * 内容：
     * ---- price：78.6
     * ---- studymodel：201001
     * ---- name：spring开发基础
     * ---- description：spring 在java领域非常流行，java程序员都在用。
     * ---- pic：group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
     * ---- timestamp：2019-08-24 19:21:35
     * ---- tags：[spring, java]
     * ---------------------------------
     *
     * id:5
     * score:1.0
     * 内容：
     * ---- price：68.6
     * ---- studymodel：201001
     * ---- name：java编程思想
     * ---- description：java语言是世界第一编程语言，在软件开发领域使用人数最多。
     * ---- pic：group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
     * ---- timestamp：2022-5-25 19:11:35
     * ---- tags：[bootstrap, dev]
     * ---------------------------------
     *
     * </pre>
     *
     * @throws Exception Exception
     */
    @Test
    void search_IDs() throws Exception
    {
        //构建搜索请求
        SearchRequest searchRequest = new SearchRequest("book");
        //构建请求体
        SearchSourceBuilder searchSourceBuilder = new SearchSourceBuilder();
        //查询
        searchSourceBuilder.query(QueryBuilders.idsQuery().addIds("1", "5", "3",
"100"));
        //放入请求中
        searchRequest.source(searchSourceBuilder);
        //发起请求
        SearchResponse searchResponse = client.search(searchRequest,
RequestOptions.DEFAULT);
        //获取数据
        SearchHits hits = searchResponse.getHits();
        //总数
        long value = hits.getTotalHits().value;
        System.out.println("总数量：" + value);
        float maxScore = hits.getMaxScore();
        System.out.println("最大分数：" + maxScore);
```

```
        System.out.println();
        SearchHit[] hitsHits = hits.getHits();
        //遍历数据
        for (SearchHit hitsHit : hitsHits)
        {
            System.out.println();
            //获得id
            String id = hitsHit.getId();
            //获得得分
            float score = hitsHit.getScore();
            //获得内容
            Map<String, Object> sourceAsMap = hitsHit.getSourceAsMap();

            //打印内容
            System.out.println("id:" + id);
            System.out.println("score:" + score);
            System.out.println("内容：");
            for (String key : sourceAsMap.keySet())
            {
                System.out.println("---- " + key + ": " + sourceAsMap.get(key));
            }
            System.out.println("-----------------------------------");
        }
    }


    /**
     * IDs ， 异步请求
     * 查询多个id为某个数的结果
     * <p>
     * 请求内容：
     * <pre>
     *
     * GET /book/_search
     * {
     *     "query":
     *     {
     *       "ids":
     *       {
     *         "values": ["1","5","3","100"]
     *       }
     *     }
     * }
     *
     * </pre>
     * <p>
     * 结果：
     * <pre>
     *
     * 总数量：3
     * 最大分数：1.0
     *
     *
     * id:1
     * score:1.0
     * 内容：
     * ---- price: 38.6
     * ---- studymodel: 201002
```

```
     * ---- name：Bootstrap开发
     * ---- description：Bootstrap是由Twitter推出的一个前台页面开发css框架，是一个非常流行的开发框架，此框架集成了多种页面效果。此开发框架包含了大量的CSS、JS程序代码，可以帮助开发者（尤其是不擅长css页面开发的程序人员）轻松的实现一个css，不受浏览器限制的精美界面css效果。
     * ---- pic：group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
     * ---- timestamp：2019-08-25 19:11:35
     * ---- tags：[bootstrap, dev]
     * ------------------------------------
     *
     * id:3
     * score:1.0
     * 内容：
     * ---- price：78.6
     * ---- studymodel：201001
     * ---- name：spring开发基础
     * ---- description：spring 在java领域非常流行，java程序员都在用。
     * ---- pic：group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
     * ---- timestamp：2019-08-24 19:21:35
     * ---- tags：[spring, java]
     * ------------------------------------
     *
     * id:5
     * score:1.0
     * 内容：
     * ---- price：68.6
     * ---- studymodel：201001
     * ---- name：java编程思想
     * ---- description：java语言是世界第一编程语言，在软件开发领域使用人数最多。
     * ---- pic：group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
     * ---- timestamp：2022-5-25 19:11:35
     * ---- tags：[bootstrap, dev]
     * ------------------------------------
     *
     * </pre>
     *
     * @throws Exception Exception
     */
    @Test
    void search_IDs_async() throws Exception
    {
        //构建搜索请求
        SearchRequest searchRequest = new SearchRequest("book");
        //构建请求体
        SearchSourceBuilder searchSourceBuilder = new SearchSourceBuilder();
        //查询
        searchSourceBuilder.query(QueryBuilders.idsQuery().addIds("1", "5", "3",
"100"));
        //放入请求中
        searchRequest.source(searchSourceBuilder);
        //发起异步请求
        client.searchAsync(searchRequest, RequestOptions.DEFAULT, new
ActionListener<SearchResponse>()
        {
            /**
             * 成功的回调
             *
             * @param searchResponse SearchResponse
             */
```

```
            @Override
            public void onResponse(SearchResponse searchResponse)
            {
                //获取数据
                SearchHits hits = searchResponse.getHits();
                //总数
                long value = hits.getTotalHits().value;
                System.out.println("总数量：" + value);
                float maxScore = hits.getMaxScore();
                System.out.println("最大分数：" + maxScore);
                System.out.println();
                SearchHit[] hitsHits = hits.getHits();
                //遍历数据
                for (SearchHit hitsHit : hitsHits)
                {
                    System.out.println();
                    //获得id
                    String id = hitsHit.getId();
                    //获得得分
                    float score = hitsHit.getScore();
                    //获得内容
                    Map<String, Object> sourceAsMap = hitsHit.getSourceAsMap();

                    //打印内容
                    System.out.println("id:" + id);
                    System.out.println("score:" + score);
                    System.out.println("内容：");
                    for (String key : sourceAsMap.keySet())
                    {
                        System.out.println("---- " + key + ": " +
sourceAsMap.get(key));
                    }
                    System.out.println("-----------------------------------");
                }
            }

            /**
             * 失败的回调
             *
             * @param e Exception
             */
            @Override
            public void onFailure(Exception e)
            {
                e.printStackTrace();
            }
        });
        //休眠2秒
        Thread.sleep(2000);
    }


    /**
     * prefix
     * 查询某字段满足某前缀的所有数据
     * <p>
     * 请求内容：
     * <pre>
```

```
     *
     * GET /book/_search
     * {
     *     "query":
     *     {
     *       "prefix":
     *       {
     *         "description":
     *         {
     *           "value": "sprin"
     *         }
     *       }
     *     }
     * }
     *
     * </pre>
     * <p>
     * 结果：
     * <pre>
     *
     * 总数量：1
     * 最大分数：1.0
     *
     *
     * id:3
     * score:1.0
     * 内容：
     * ---- price：78.6
     * ---- studymodel：201001
     * ---- name：spring开发基础
     * ---- description：spring 在java领域非常流行，java程序员都在用。
     * ---- pic：group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
     * ---- timestamp：2019-08-24 19:21:35
     * ---- tags：[spring, java]
     * ----------------------------------
     *
     * </pre>
     *
     * @throws Exception
     */
    @Test
    void search_prefix() throws Exception
    {
        //构建搜索请求
        SearchRequest searchRequest = new SearchRequest("book");
        //构建请求体
        SearchSourceBuilder searchSourceBuilder = new SearchSourceBuilder();
        //查询
        searchSourceBuilder.query(QueryBuilders.prefixQuery("description",
"sprin"));
        //放入请求中
        searchRequest.source(searchSourceBuilder);
        //发起请求
        SearchResponse searchResponse = client.search(searchRequest,
RequestOptions.DEFAULT);
        //获取数据
        SearchHits hits = searchResponse.getHits();
        //总数
```

```
        long value = hits.getTotalHits().value;
        System.out.println("总数量：" + value);
        float maxScore = hits.getMaxScore();
        System.out.println("最大分数：" + maxScore);
        System.out.println();
        SearchHit[] hitsHits = hits.getHits();
        //遍历数据
        for (SearchHit hitsHit : hitsHits)
        {
            System.out.println();
            //获得id
            String id = hitsHit.getId();
            //获得得分
            float score = hitsHit.getScore();
            //获得内容
            Map<String, Object> sourceAsMap = hitsHit.getSourceAsMap();

            //打印内容
            System.out.println("id:" + id);
            System.out.println("score:" + score);
            System.out.println("内容：");
            for (String key : sourceAsMap.keySet())
            {
                System.out.println("---- " + key + ": " + sourceAsMap.get(key));
            }
            System.out.println("-----------------------------------");
        }
    }


    /**
     * prefix ， 异步请求
     * 查询某字段满足某前缀的所有数据
     * <p>
     * 请求内容：
     * <pre>
     *
     * GET /book/_search
     * {
     *     "query":
     *     {
     *       "prefix":
     *       {
     *         "description":
     *         {
     *           "value": "sprin"
     *         }
     *       }
     *     }
     * }
     *
     * </pre>
     * <p>
     * 结果：
     * <pre>
     *
     * 总数量：1
     * 最大分数：1.0
```

```
     *
     *
     * id:3
     * score:1.0
     * 内容:
     * ---- price: 78.6
     * ---- studymodel: 201001
     * ---- name: spring开发基础
     * ---- description: spring 在java领域非常流行，java程序员都在用。
     * ---- pic: group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
     * ---- timestamp: 2019-08-24 19:21:35
     * ---- tags: [spring, java]
     * ----------------------------------
     *
     * </pre>
     *
     * @throws Exception
     */
    @Test
    void search_prefix_async() throws Exception
    {
        //构建搜索请求
        SearchRequest searchRequest = new SearchRequest("book");
        //构建请求体
        SearchSourceBuilder searchSourceBuilder = new SearchSourceBuilder();
        //查询
        searchSourceBuilder.query(QueryBuilders.prefixQuery("description",
"sprin"));
        //放入请求中
        searchRequest.source(searchSourceBuilder);
        //发起异步请求
        client.searchAsync(searchRequest, RequestOptions.DEFAULT, new
ActionListener<SearchResponse>()
        {
            /**
             * 成功的回调
             *
             * @param searchResponse SearchResponse
             */
            @Override
            public void onResponse(SearchResponse searchResponse)
            {
                //获取数据
                SearchHits hits = searchResponse.getHits();
                //总数
                long value = hits.getTotalHits().value;
                System.out.println("总数量: " + value);
                float maxScore = hits.getMaxScore();
                System.out.println("最大分数: " + maxScore);
                System.out.println();
                SearchHit[] hitsHits = hits.getHits();
                //遍历数据
                for (SearchHit hitsHit : hitsHits)
                {
                    System.out.println();
                    //获得id
                    String id = hitsHit.getId();
                    //获得得分
```

```
                    float score = hitsHit.getScore();
                    //获得内容
                    Map<String, Object> sourceAsMap = hitsHit.getSourceAsMap();

                    //打印内容
                    System.out.println("id:" + id);
                    System.out.println("score:" + score);
                    System.out.println("内容: ");
                    for (String key : sourceAsMap.keySet())
                    {
                        System.out.println("---- " + key + ": " +
sourceAsMap.get(key));
                    }
                    System.out.println("-----------------------------------");
                }
            }

            /**
             * 失败的回调
             *
             * @param e Exception
             */
            @Override
            public void onFailure(Exception e)
            {
                e.printStackTrace();
            }
        });
        //休眠2秒
        Thread.sleep(2000);
    }


    /**
     * regexp query
     * 正则查询
     * 查询某字段满足某正则表达式的所有数据
     * <p>
     * 请求内容:
     * <pre>
     *
     * GET /book/_search
     * {
     *     "query":
     *     {
     *       "regexp":
     *       {
     *         "description":
     *         {
     *           "value": "j.*a",
     *           "flags": "ALL"
     *         }
     *       }
     *     }
     * }
     *
     * 结果:
     * <pre>
```

```
     *
     * 总数量：4
     * 最大分数：1.0
     *
     *
     * id:2
     * score:1.0
     * 内容：
     * ---- price：68.6
     * ---- studymodel：201001
     * ---- name：java编程思想
     * ---- description：java语言是世界第一编程语言，在软件开发领域使用人数最多。
     * ---- pic：group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
     * ---- timestamp：2019-08-25 19:11:35
     * ---- tags：[java, dev]
     * ----------------------------------
     *
     * id:3
     * score:1.0
     * 内容：
     * ---- price：78.6
     * ---- studymodel：201001
     * ---- name：spring开发基础
     * ---- description：spring 在java领域非常流行，java程序员都在用。
     * ---- pic：group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
     * ---- timestamp：2019-08-24 19:21:35
     * ---- tags：[spring, java]
     * ----------------------------------
     *
     * id:5
     * score:1.0
     * 内容：
     * ---- price：68.6
     * ---- studymodel：201001
     * ---- name：java编程思想
     * ---- description：java语言是世界第一编程语言，在软件开发领域使用人数最多。
     * ---- pic：group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
     * ---- timestamp：2022-5-25 19:11:35
     * ---- tags：[bootstrap, dev]
     * ----------------------------------
     *
     * id:6
     * score:1.0
     * 内容：
     * ---- price：68.6
     * ---- studymodel：201001
     * ---- name：java编程思想
     * ---- description：java语言是世界第一编程语言，在软件开发领域使用人数最多。
     * ---- pic：group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
     * ---- timestamp：2022-5-25 19:11:35
     * ---- tags：[bootstrap, dev]
     * ----------------------------------
     *
     * </pre>
     *
     * </pre>
     *
     * @throws Exception
```

```
     */
    @Test
    void search_regexp_query() throws Exception
    {
        //构建搜索请求
        SearchRequest searchRequest = new SearchRequest("book");
        //构建请求体
        SearchSourceBuilder searchSourceBuilder = new SearchSourceBuilder();
        //查询
        searchSourceBuilder.query(QueryBuilders.regexpQuery("description",
"j.*a"));
        //放入请求中
        searchRequest.source(searchSourceBuilder);
        //发起请求
        SearchResponse searchResponse = client.search(searchRequest,
RequestOptions.DEFAULT);
        //获取数据
        SearchHits hits = searchResponse.getHits();
        //总数
        long value = hits.getTotalHits().value;
        System.out.println("总数量: " + value);
        float maxScore = hits.getMaxScore();
        System.out.println("最大分数: " + maxScore);
        System.out.println();
        SearchHit[] hitsHits = hits.getHits();
        //遍历数据
        for (SearchHit hitsHit : hitsHits)
        {
            System.out.println();
            //获得id
            String id = hitsHit.getId();
            //获得得分
            float score = hitsHit.getScore();
            //获得内容
            Map<String, Object> sourceAsMap = hitsHit.getSourceAsMap();

            //打印内容
            System.out.println("id:" + id);
            System.out.println("score:" + score);
            System.out.println("内容: ");
            for (String key : sourceAsMap.keySet())
            {
                System.out.println("---- " + key + ": " + sourceAsMap.get(key));
            }
            System.out.println("-----------------------------------");
        }
    }


    /**
     * regexp query , 异步请求
     * 正则查询
     * 查询某字段满足某正则表达式的所有数据
     * <p>
     * 请求内容:
     * <pre>
     *
     * GET /book/_search
```

```
* {
*     "query":
*     {
*       "regexp":
*       {
*         "description":
*         {
*           "value": "j.*a",
*           "flags": "ALL"
*         }
*       }
*     }
* }
*
* </pre>
* <p>
* 结果：
* <pre>
*
* 总数量：4
* 最大分数：1.0
*
*
* id:2
* score:1.0
* 内容：
* ---- price：68.6
* ---- studymodel：201001
* ---- name：java编程思想
* ---- description：java语言是世界第一编程语言，在软件开发领域使用人数最多。
* ---- pic：group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
* ---- timestamp：2019-08-25 19:11:35
* ---- tags：[java, dev]
* ------------------------------------
*
* id:3
* score:1.0
* 内容：
* ---- price：78.6
* ---- studymodel：201001
* ---- name：spring开发基础
* ---- description：spring 在java领域非常流行，java程序员都在用。
* ---- pic：group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
* ---- timestamp：2019-08-24 19:21:35
* ---- tags：[spring, java]
* ------------------------------------
*
* id:5
* score:1.0
* 内容：
* ---- price：68.6
* ---- studymodel：201001
* ---- name：java编程思想
* ---- description：java语言是世界第一编程语言，在软件开发领域使用人数最多。
* ---- pic：group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
* ---- timestamp：2022-5-25 19:11:35
* ---- tags：[bootstrap, dev]
* ------------------------------------
```

```
     *
     * id:6
     * score:1.0
     * 内容:
     * ---- price: 68.6
     * ---- studymodel: 201001
     * ---- name: java编程思想
     * ---- description: java语言是世界第一编程语言，在软件开发领域使用人数最多。
     * ---- pic: group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
     * ---- timestamp: 2022-5-25 19:11:35
     * ---- tags: [bootstrap, dev]
     * ---------------------------------
     *
     * </pre>
     *
     * @throws Exception
     */
    @Test
    void search_regexp_query_async() throws Exception
    {
        //构建搜索请求
        SearchRequest searchRequest = new SearchRequest("book");
        //构建请求体
        SearchSourceBuilder searchSourceBuilder = new SearchSourceBuilder();
        //查询
        searchSourceBuilder.query(QueryBuilders.regexpQuery("description",
"j.*a"));
        //放入请求中
        searchRequest.source(searchSourceBuilder);
        //发起异步请求
        client.searchAsync(searchRequest, RequestOptions.DEFAULT, new
ActionListener<SearchResponse>()
        {
            /**
             * 成功的回调
             *
             * @param searchResponse SearchResponse
             */
            @Override
            public void onResponse(SearchResponse searchResponse)
            {
                //获取数据
                SearchHits hits = searchResponse.getHits();
                //总数
                long value = hits.getTotalHits().value;
                System.out.println("总数量: " + value);
                float maxScore = hits.getMaxScore();
                System.out.println("最大分数: " + maxScore);
                System.out.println();
                SearchHit[] hitsHits = hits.getHits();
                //遍历数据
                for (SearchHit hitsHit : hitsHits)
                {
                    System.out.println();
                    //获得id
                    String id = hitsHit.getId();
                    //获得得分
                    float score = hitsHit.getScore();
```

```
                    //获得内容
                    Map<String, Object> sourceAsMap = hitsHit.getSourceAsMap();

                    //打印内容
                    System.out.println("id:" + id);
                    System.out.println("score:" + score);
                    System.out.println("内容: ");
                    for (String key : sourceAsMap.keySet())
                    {
                        System.out.println("---- " + key + ": " +
sourceAsMap.get(key));
                    }
                    System.out.println("-----------------------------------");
                }
            }

            /**
             * 失败的回调
             *
             * @param e Exception
             */
            @Override
            public void onFailure(Exception e)
            {
                e.printStackTrace();
            }
        });
        //休眠2秒
        Thread.sleep(2000);
    }


    /**
     * Filter
     * <p>
     * 请求内容:
     * <pre>
     *
     * GET /book/_search
     * {
     *     "query":
     *     {
     *       "bool":
     *       {
     *         "must":
     *         [
     *         {
     *           "match":
     *           {
     *             "description": "java程序员"
     *           }
     *         }
     *       ],
     *       "filter":
     *       [
     *         {
     *           "range":
     *           {
```

```
*              "price":
*              {
*                "gte": 60,
*                "lte": 70
*              }
*            }
*          }
*        ]
*        }
*     }
* }
*
* </pre>
* <p>
* 结果:
* <pre>
*
* 总数量:3
* 最大分数:0.4398797
*
*
* id:2
* score:0.4398797
* 内容:
* ---- price:68.6
* ---- studymodel:201001
* ---- name:java编程思想
* ---- description:java语言是世界第一编程语言,在软件开发领域使用人数最多。
* ---- pic:group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
* ---- timestamp:2019-08-25 19:11:35
* ---- tags:[java, dev]
* ----------------------------------
*
* id:5
* score:0.4398797
* 内容:
* ---- price:68.6
* ---- studymodel:201001
* ---- name:java编程思想
* ---- description:java语言是世界第一编程语言,在软件开发领域使用人数最多。
* ---- pic:group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
* ---- timestamp:2022-5-25 19:11:35
* ---- tags:[bootstrap, dev]
* ----------------------------------
*
* id:6
* score:0.4398797
* 内容:
* ---- price:68.6
* ---- studymodel:201001
* ---- name:java编程思想
* ---- description:java语言是世界第一编程语言,在软件开发领域使用人数最多。
* ---- pic:group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
* ---- timestamp:2022-5-25 19:11:35
* ---- tags:[bootstrap, dev]
* ----------------------------------
*
* </pre>
```

```
     *
     * @throws Exception Exception
     */
    @Test
    void search_Filter() throws Exception
    {
        //构建搜索请求
        SearchRequest searchRequest = new SearchRequest("book");
        //构建请求体
        SearchSourceBuilder searchSourceBuilder = new SearchSourceBuilder();
        //查询
        searchSourceBuilder.query(QueryBuilders.boolQuery()
                .must(QueryBuilders.matchQuery("description", "java程序员"))
                .filter(QueryBuilders.rangeQuery("price").gte(60).lte(70)));
        //放入请求中
        searchRequest.source(searchSourceBuilder);
        //发起请求
        SearchResponse searchResponse = client.search(searchRequest,
RequestOptions.DEFAULT);
        //获取数据
        SearchHits hits = searchResponse.getHits();
        //总数
        long value = hits.getTotalHits().value;
        System.out.println("总数量: " + value);
        float maxScore = hits.getMaxScore();
        System.out.println("最大分数: " + maxScore);
        System.out.println();
        SearchHit[] hitsHits = hits.getHits();
        //遍历数据
        for (SearchHit hitsHit : hitsHits)
        {
            System.out.println();
            //获得id
            String id = hitsHit.getId();
            //获得得分
            float score = hitsHit.getScore();
            //获得内容
            Map<String, Object> sourceAsMap = hitsHit.getSourceAsMap();

            //打印内容
            System.out.println("id:" + id);
            System.out.println("score:" + score);
            System.out.println("内容: ");
            for (String key : sourceAsMap.keySet())
            {
                System.out.println("---- " + key + ": " + sourceAsMap.get(key));
            }
            System.out.println("-----------------------------------");
        }
    }


    /**
     * Filter , 异步请求
     * <p>
     * 请求内容:
     * <pre>
     *
```

```
* GET /book/_search
* {
*     "query":
*     {
*       "bool":
*       {
*         "must":
*         [
*         {
*           "match":
*           {
*             "description": "java程序员"
*           }
*         }
*       ],
*       "filter":
*       [
*         {
*           "range":
*           {
*             "price":
*             {
*               "gte": 60,
*               "lte": 70
*             }
*           }
*         }
*       ]
*       }
*     }
* }
*
* </pre>
* <p>
* 结果:
* <pre>
*
* 总数量:3
* 最大分数:0.4398797
*
*
* id:2
* score:0.4398797
* 内容:
* ---- price:68.6
* ---- studymodel:201001
* ---- name:java编程思想
* ---- description:java语言是世界第一编程语言,在软件开发领域使用人数最多。
* ---- pic:group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
* ---- timestamp:2019-08-25 19:11:35
* ---- tags:[java, dev]
* ------------------------------------
*
* id:5
* score:0.4398797
* 内容:
* ---- price:68.6
* ---- studymodel:201001
```

```
     * ---- name：java编程思想
     * ---- description：java语言是世界第一编程语言，在软件开发领域使用人数最多。
     * ---- pic：group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
     * ---- timestamp：2022-5-25 19:11:35
     * ---- tags：[bootstrap, dev]
     * ----------------------------------
     *
     * id:6
     * score:0.4398797
     * 内容：
     * ---- price：68.6
     * ---- studymodel：201001
     * ---- name：java编程思想
     * ---- description：java语言是世界第一编程语言，在软件开发领域使用人数最多。
     * ---- pic：group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
     * ---- timestamp：2022-5-25 19:11:35
     * ---- tags：[bootstrap, dev]
     * ----------------------------------
     *
     * </pre>
     *
     * @throws Exception Exception
     */
    @Test
    void search_Filter_async() throws Exception
    {
        //构建搜索请求
        SearchRequest searchRequest = new SearchRequest("book");
        //构建请求体
        SearchSourceBuilder searchSourceBuilder = new SearchSourceBuilder();
        //查询
        searchSourceBuilder.query(QueryBuilders.boolQuery()
                .must(QueryBuilders.matchQuery("description", "java程序员"))
                .filter(QueryBuilders.rangeQuery("price").gte(60).lte(70)));
        //放入请求中
        searchRequest.source(searchSourceBuilder);
        //发起异步请求
        client.searchAsync(searchRequest, RequestOptions.DEFAULT, new
ActionListener<SearchResponse>()
        {
            /**
             * 成功的回调
             *
             * @param searchResponse SearchResponse
             */
            @Override
            public void onResponse(SearchResponse searchResponse)
            {
                //获取数据
                SearchHits hits = searchResponse.getHits();
                //总数
                long value = hits.getTotalHits().value;
                System.out.println("总数量：" + value);
                float maxScore = hits.getMaxScore();
                System.out.println("最大分数：" + maxScore);
                System.out.println();
                SearchHit[] hitsHits = hits.getHits();
                //遍历数据
```

```
                for (SearchHit hitsHit : hitsHits)
                {
                    System.out.println();
                    //获得id
                    String id = hitsHit.getId();
                    //获得得分
                    float score = hitsHit.getScore();
                    //获得内容
                    Map<String, Object> sourceAsMap = hitsHit.getSourceAsMap();

                    //打印内容
                    System.out.println("id:" + id);
                    System.out.println("score:" + score);
                    System.out.println("内容: ");
                    for (String key : sourceAsMap.keySet())
                    {
                        System.out.println("---- " + key + ": " +
sourceAsMap.get(key));
                    }
                    System.out.println("----------------------------------");
                }
            }

            /**
             * 失败的回调
             *
             * @param e Exception
             */
            @Override
            public void onFailure(Exception e)
            {
                e.printStackTrace();
            }
        });
        //休眠2秒
        Thread.sleep(2000);
    }
}
```

评分机制

Elasticsearch使用的是 term frequency/inverse document frequency算法，简称为TF/IDF算法。TF词频(Term Frequency)，IDF逆向文件频率(Inverse Document Frequency)

relevance score算法，简单来说，就是计算出，一个索引中的文本，与搜索文本，他们之间的关联匹配程度。

Term frequency：搜索文本中的各个词条在field文本中出现了多少次，出现次数越多，就越相关。

Inverse document frequency：搜索文本中的各个词条在整个索引的所有文档中出现了多少次，出现的次数越多，就越不相关.

Field-length norm：field长度，field越长，相关度越弱

聚合

bucket和metric

- bucket：一个数据分组

city name
北京 张三
北京 李四
天津 王五
天津 赵六

天津 王麻子

划分出来两个bucket，一个是北京bucket，一个是天津bucket
北京bucket：包含了2个人，张三，李四
上海bucket：包含了3个人，王五，赵六，王麻子

- metric：对一个数据分组执行的统计

metric，就是对一个bucket执行的某种聚合分析的操作，比如说求平均值，求最大值，求最小值

示例

计算每个studymodel下的商品数量

```
GET /book/_search
{
  "size": 0,
  "query": {
    "match_all": {}
  },
  "aggs":
    {
    "group_by_model": {
      "terms": { "field": "studymodel" }
    }
  }
}
```

计算每个tags下的商品数量

```
GET /book/_search
{
  "size": 0,
  "query": {
    "match_all": {}
  },
  "aggs": {
    "group_by_tags": {
      "terms": { "field": "tags" }
    }
  }
}
```

先分组，再算每组的平均值，计算每个tag下的商品的平均价格

```
GET /book/_search
{
  "size": 0,
  "query": {
    "match_all": {}
  },
  "aggs": {
    "group_by_model":
    {
      "terms":
      {
        "field": "tags"
      },
      "aggs": {
        "avg_price":
        {
          "avg":
          {
            "field": "price"
          }
        }
      }
    }
  }
}
```

计算每个tag下的商品的平均价格，并且按照平均价格降序排序

```
GET /book/_search
{
  "size": 0,
  "query": {
    "match_all": {}
```

```
  },
  "aggs": {
    "group_by_model":
    {
      "terms":
      {
        "field": "tags" ,
        "order":
        {
          "price": "desc"
        }
      },
      "aggs": {
        "avg_price":
        {
          "avg":
          {
            "field": "price"
          }
        }
      }
    }
  }
}
```

按照指定的价格范围区间进行分组，然后在每组内再按照tag进行分组，最后再计算每组的平均价格

```
GET /book/_search
{
  "size": 0,
  "query":
  {
    "match_all": {}
  },
  "aggs": {
    "group_by_price":
    {
      "range":
      {
        "field": "price",
        "ranges":
        [
          {
            "from": 0,
            "to": 40
          },
          {
            "from": 40,
            "to": 60
          },
          {
            "from": 60,
```

```
            "to": 80
          },
          {
            "from": 80,
            "to": 100
          }
        ]
      },
      "aggs":
      {
        "group_by_tags":
        {
          "terms":
          {
            "field": "tags"
          },
          "aggs":
          {
            "price_avg":
            {
              "avg":
              {
                "field": "price"
              }
            }
          }
        }
      }
    }
  }
}
```

电视案例

创建索引及映射

```
PUT /tvs
PUT /tvs/_mapping
{
            "properties": {
                "price": {
                    "type": "long"
                },
                "color": {
                    "type": "keyword"
                },
                "brand": {
                    "type": "keyword"
                },
                "sold_date": {
```

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

查看索引

```
{
  "tvs" : {
    "aliases" : { },
    "mappings" : {
      "properties" : {
        "brand" : {
          "type" : "keyword"
        },
        "color" : {
          "type" : "keyword"
        },
        "price" : {
          "type" : "long"
        },
        "sold_date" : {
          "type" : "date"
        }
      }
    },
    "settings" : {
      "index" : {
        "routing" : {
          "allocation" : {
            "include" : {
              "_tier_preference" : "data_content"
            }
          }
        },
        "number_of_shards" : "1",
        "provided_name" : "tvs",
        "creation_date" : "1653799931947",
        "number_of_replicas" : "1",
        "uuid" : "UPq3NuVHTlWq1r9GrCj4fw",
        "version" : {
          "created" : "8010399"
        }
      }
    }
  }
}
```

插入数据

```
POST /tvs/_bulk
{ "index": {}}
{ "price" : 1000, "color" : "红色", "brand" : "长虹", "sold_date" : "2019-10-28" }
{ "index": {}}
{ "price" : 2000, "color" : "红色", "brand" : "长虹", "sold_date" : "2019-11-05" }
{ "index": {}}
{ "price" : 3000, "color" : "绿色", "brand" : "小米", "sold_date" : "2019-05-18" }
{ "index": {}}
{ "price" : 1500, "color" : "蓝色", "brand" : "TCL", "sold_date" : "2019-07-02" }
{ "index": {}}
{ "price" : 1200, "color" : "绿色", "brand" : "TCL", "sold_date" : "2019-08-19" }
{ "index": {}}
{ "price" : 2000, "color" : "红色", "brand" : "长虹", "sold_date" : "2019-11-05" }
{ "index": {}}
{ "price" : 8000, "color" : "红色", "brand" : "三星", "sold_date" : "2020-01-01" }
{ "index": {}}
{ "price" : 2500, "color" : "蓝色", "brand" : "小米", "sold_date" : "2020-02-12" }
{ "index": {}}
{ "price" : 4500, "color" : "绿色", "brand" : "小米", "sold_date" : "2020-04-22" }
{ "index": {}}
{ "price" : 6100, "color" : "蓝色", "brand" : "三星", "sold_date" : "2020-05-16" }
{ "index": {}}
{ "price" : 2100, "color" : "白色", "brand" : "TCL", "sold_date" : "2020-05-17" }
{ "index": {}}
{ "price" : 8500, "color" : "红色", "brand" : "小米", "sold_date" : "2020-05-19" }
{ "index": {}}
{ "price" : 4200, "color" : "蓝色", "brand" : "长虹", "sold_date" : "2020-05-23" }
{ "index": {}}
{ "price" : 4800, "color" : "黑色", "brand" : "小米", "sold_date" : "2020-06-10" }
```

查看数据

```
GET /tvs/_search
```

结果:

```
{
  "took" : 0,
  "timed_out" : false,
  "_shards" : {
    "total" : 1,
    "successful" : 1,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : {
      "value" : 14,
```

```
    "relation" : "eq"
  },
  "max_score" : 1.0,
  "hits" : [
    {
      "_index" : "tvs",
      "_id" : "66ouDoEBEpQthbP41cfj",
      "_score" : 1.0,
      "_source" : {
        "price" : 1000,
        "color" : "红色",
        "brand" : "长虹",
        "sold_date" : "2019-10-28"
      }
    },
    {
      "_index" : "tvs",
      "_id" : "7KouDoEBEpQthbP41cfj",
      "_score" : 1.0,
      "_source" : {
        "price" : 2000,
        "color" : "红色",
        "brand" : "长虹",
        "sold_date" : "2019-11-05"
      }
    },
    {
      "_index" : "tvs",
      "_id" : "7aouDoEBEpQthbP41cfj",
      "_score" : 1.0,
      "_source" : {
        "price" : 3000,
        "color" : "绿色",
        "brand" : "小米",
        "sold_date" : "2019-05-18"
      }
    },
    {
      "_index" : "tvs",
      "_id" : "7qouDoEBEpQthbP41cfj",
      "_score" : 1.0,
      "_source" : {
        "price" : 1500,
        "color" : "蓝色",
        "brand" : "TCL",
        "sold_date" : "2019-07-02"
      }
    },
    {
      "_index" : "tvs",
      "_id" : "76ouDoEBEpQthbP41cfj",
      "_score" : 1.0,
      "_source" : {
        "price" : 1200,
        "color" : "绿色",
        "brand" : "TCL",
        "sold_date" : "2019-08-19"
      }
```

```
      },
      {
        "_index" : "tvs",
        "_id" : "8KouDoEBEpQthbP41cfj",
        "_score" : 1.0,
        "_source" : {
          "price" : 2000,
          "color" : "红色",
          "brand" : "长虹",
          "sold_date" : "2019-11-05"
        }
      },
      {
        "_index" : "tvs",
        "_id" : "8aouDoEBEpQthbP41cfj",
        "_score" : 1.0,
        "_source" : {
          "price" : 8000,
          "color" : "红色",
          "brand" : "三星",
          "sold_date" : "2020-01-01"
        }
      },
      {
        "_index" : "tvs",
        "_id" : "8qouDoEBEpQthbP41cfj",
        "_score" : 1.0,
        "_source" : {
          "price" : 2500,
          "color" : "蓝色",
          "brand" : "小米",
          "sold_date" : "2020-02-12"
        }
      },
      {
        "_index" : "tvs",
        "_id" : "86ouDoEBEpQthbP41cfj",
        "_score" : 1.0,
        "_source" : {
          "price" : 4500,
          "color" : "绿色",
          "brand" : "小米",
          "sold_date" : "2020-04-22"
        }
      },
      {
        "_index" : "tvs",
        "_id" : "9KouDoEBEpQthbP41cfj",
        "_score" : 1.0,
        "_source" : {
          "price" : 6100,
          "color" : "蓝色",
          "brand" : "三星",
          "sold_date" : "2020-05-16"
        }
      }
    ]
  }
```

```
}
```

统计哪种颜色的电视销量最高

```
GET /tvs/_search
{
  "query":
  {
    "match_all": {}
  },
  "size": 0,
  "aggs":
  {
    "popular_colors":
    {
      "terms":
      {
        "field": "color"
      }
    }
  }
}
```

结果:

```
{
  "took" : 12,
  "timed_out" : false,
  "_shards" : {
    "total" : 1,
    "successful" : 1,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : {
      "value" : 14,
      "relation" : "eq"
    },
    "max_score" : null,
    "hits" : [ ]
  },
  "aggregations" : {
    "popular_colors" : {
      "doc_count_error_upper_bound" : 0,
      "sum_other_doc_count" : 0,
      "buckets" : [
        {
          "key" : "红色",
          "doc_count" : 5
        },
```

```
        {
          "key" : "蓝色",
          "doc_count" : 4
        },
        {
          "key" : "绿色",
          "doc_count" : 3
        },
        {
          "key" : "白色",
          "doc_count" : 1
        },
        {
          "key" : "黑色",
          "doc_count" : 1
        }
      ]
    }
  }
}
```

统计每种颜色电视平均价格

```
GET /tvs/_search
{
  "query":
  {
    "match_all": {}
  },
  "size": 0,
  "aggs":
  {
    "group_by_colors":
    {
      "terms":
      {
        "field": "color"
      },
      "aggs": {
        "avg_price":
        {
          "avg":
          {
            "field": "price"
          }
        }
      }
    }
  }
}
```

结果：

```
{
  "took" : 1,
  "timed_out" : false,
  "_shards" : {
    "total" : 1,
    "successful" : 1,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : {
      "value" : 14,
      "relation" : "eq"
    },
    "max_score" : null,
    "hits" : [ ]
  },
  "aggregations" : {
    "group_by_colors" : {
      "doc_count_error_upper_bound" : 0,
      "sum_other_doc_count" : 0,
      "buckets" : [
        {
          "key" : "红色",
          "doc_count" : 5,
          "avg_price" : {
            "value" : 4300.0
          }
        },
        {
          "key" : "蓝色",
          "doc_count" : 4,
          "avg_price" : {
            "value" : 3575.0
          }
        },
        {
          "key" : "绿色",
          "doc_count" : 3,
          "avg_price" : {
            "value" : 2900.0
          }
        },
        {
          "key" : "白色",
          "doc_count" : 1,
          "avg_price" : {
            "value" : 2100.0
          }
        },
        {
          "key" : "黑色",
          "doc_count" : 1,
          "avg_price" : {
            "value" : 4800.0
          }
        }
      ]
```

```
    }
  }
}
```

统计每个颜色下，平均价格及每个颜色下，每个品牌的平均价格

```
GET /tvs/_search
{
  "query":
  {
    "match_all": {}
  },
  "size": 0,
  "aggs":
  {
    "group_by_colors":
    {
      "terms":
      {
        "field": "color"
      },
      "aggs":
      {
        "avg_price":
        {
          "avg":
          {
            "field": "price"
          }
        },
        "group_by_brand":
        {
          "terms":
          {
            "field": "brand"
          },
          "aggs":
          {
            "avg_price":
            {
              "avg":
              {
                "field": "price"
              }
            }
          }
        }
      }
    }
  }
}
```

结果：


```
{
  "took" : 1,
  "timed_out" : false,
  "_shards" : {
    "total" : 1,
    "successful" : 1,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : {
      "value" : 14,
      "relation" : "eq"
    },
    "max_score" : null,
    "hits" : [ ]
  },
  "aggregations" : {
    "group_by_colors" : {
      "doc_count_error_upper_bound" : 0,
      "sum_other_doc_count" : 0,
      "buckets" : [
        {
          "key" : "红色",
          "doc_count" : 5,
          "avg_price" : {
            "value" : 4300.0
          },
          "group_by_brand" : {
            "doc_count_error_upper_bound" : 0,
            "sum_other_doc_count" : 0,
            "buckets" : [
              {
                "key" : "长虹",
                "doc_count" : 3,
                "avg_price" : {
                  "value" : 1666.6666666666667
                }
              },
              {
                "key" : "三星",
                "doc_count" : 1,
                "avg_price" : {
                  "value" : 8000.0
                }
              },
              {
                "key" : "小米",
                "doc_count" : 1,
                "avg_price" : {
                  "value" : 8500.0
                }
              }
            ]
          }
```

```
        },
        {
          "key" : "蓝色",
          "doc_count" : 4,
          "avg_price" : {
            "value" : 3575.0
          },
          "group_by_brand" : {
            "doc_count_error_upper_bound" : 0,
            "sum_other_doc_count" : 0,
            "buckets" : [
              {
                "key" : "TCL",
                "doc_count" : 1,
                "avg_price" : {
                  "value" : 1500.0
                }
              },
              {
                "key" : "三星",
                "doc_count" : 1,
                "avg_price" : {
                  "value" : 6100.0
                }
              },
              {
                "key" : "小米",
                "doc_count" : 1,
                "avg_price" : {
                  "value" : 2500.0
                }
              },
              {
                "key" : "长虹",
                "doc_count" : 1,
                "avg_price" : {
                  "value" : 4200.0
                }
              }
            ]
          }
        },
        {
          "key" : "绿色",
          "doc_count" : 3,
          "avg_price" : {
            "value" : 2900.0
          },
          "group_by_brand" : {
            "doc_count_error_upper_bound" : 0,
            "sum_other_doc_count" : 0,
            "buckets" : [
              {
                "key" : "小米",
                "doc_count" : 2,
                "avg_price" : {
                  "value" : 3750.0
                }
```

```
            },
            {
              "key" : "TCL",
              "doc_count" : 1,
              "avg_price" : {
                "value" : 1200.0
              }
            }
          ]
        }
      },
      {
        "key" : "白色",
        "doc_count" : 1,
        "avg_price" : {
          "value" : 2100.0
        },
        "group_by_brand" : {
          "doc_count_error_upper_bound" : 0,
          "sum_other_doc_count" : 0,
          "buckets" : [
            {
              "key" : "TCL",
              "doc_count" : 1,
              "avg_price" : {
                "value" : 2100.0
              }
            }
          ]
        }
      },
      {
        "key" : "黑色",
        "doc_count" : 1,
        "avg_price" : {
          "value" : 4800.0
        },
        "group_by_brand" : {
          "doc_count_error_upper_bound" : 0,
          "sum_other_doc_count" : 0,
          "buckets" : [
            {
              "key" : "小米",
              "doc_count" : 1,
              "avg_price" : {
                "value" : 4800.0
              }
            }
          ]
        }
      }
    ]
  }
}
```

求每个颜色的最大价格、最小价格、平均价格和总价格

```
GET /tvs/_search
{
  "query":
  {
    "match_all": {}
  },
  "size": 0,
  "aggs":
  {
    "group_by_color":
    {
      "terms":
      {
        "field": "color"
      },
      "aggs":
      {
        "max_price":
        {
          "max":
          {
            "field": "price"
          }
        },
        "min_price":
        {
          "min":
          {
            "field": "price"
          }
        },
        "avg_price":
        {
          "avg":
          {
            "field": "price"
          }
        },
        "sum_price":
        {
          "sum":
          {
            "field": "price"
          }
        }
      }
    }
  }
}
```

结果:

```
{
  "took" : 1,
  "timed_out" : false,
  "_shards" : {
    "total" : 1,
    "successful" : 1,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : {
      "value" : 14,
      "relation" : "eq"
    },
    "max_score" : null,
    "hits" : [ ]
  },
  "aggregations" : {
    "group_by_color" : {
      "doc_count_error_upper_bound" : 0,
      "sum_other_doc_count" : 0,
      "buckets" : [
        {
          "key" : "红色",
          "doc_count" : 5,
          "max_price" : {
            "value" : 8500.0
          },
          "min_price" : {
            "value" : 1000.0
          },
          "avg_price" : {
            "value" : 4300.0
          },
          "sum_price" : {
            "value" : 21500.0
          }
        },
        {
          "key" : "蓝色",
          "doc_count" : 4,
          "max_price" : {
            "value" : 6100.0
          },
          "min_price" : {
            "value" : 1500.0
          },
          "avg_price" : {
            "value" : 3575.0
          },
          "sum_price" : {
            "value" : 14300.0
          }
        },
        {
          "key" : "绿色",
          "doc_count" : 3,
          "max_price" : {
```

```
            "value" : 4500.0
          },
          "min_price" : {
            "value" : 1200.0
          },
          "avg_price" : {
            "value" : 2900.0
          },
          "sum_price" : {
            "value" : 8700.0
          }
        },
        {
          "key" : "白色",
          "doc_count" : 1,
          "max_price" : {
            "value" : 2100.0
          },
          "min_price" : {
            "value" : 2100.0
          },
          "avg_price" : {
            "value" : 2100.0
          },
          "sum_price" : {
            "value" : 2100.0
          }
        },
        {
          "key" : "黑色",
          "doc_count" : 1,
          "max_price" : {
            "value" : 4800.0
          },
          "min_price" : {
            "value" : 4800.0
          },
          "avg_price" : {
            "value" : 4800.0
          },
          "sum_price" : {
            "value" : 4800.0
          }
        }
      ]
    }
  }
}
```

划分范围 histogram

```
GET /tvs/_search
{
  "query":
  {
    "match_all": {}
  },
  "size": 0,
  "aggs":
  {
    "histogram_price":
    {
      "histogram":
      {
        "field": "price",
        "interval": 2000
      }
    }
  }
}
```

结果:

```
{
  "took" : 1,
  "timed_out" : false,
  "_shards" : {
    "total" : 1,
    "successful" : 1,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : {
      "value" : 14,
      "relation" : "eq"
    },
    "max_score" : null,
    "hits" : [ ]
  },
  "aggregations" : {
    "histogram_price" : {
      "buckets" : [
        {
          "key" : 0.0,
          "doc_count" : 3
        },
        {
          "key" : 2000.0,
          "doc_count" : 5
        },
        {
          "key" : 4000.0,
          "doc_count" : 3
        },
```

```
        {
          "key" : 6000.0,
          "doc_count" : 1
        },
        {
          "key" : 8000.0,
          "doc_count" : 2
        }
      ]
    }
  }
}
```

按照日期分组聚合

```
GET /tvs/_search
{
   "size" : 0,
   "aggs": {
      "sales": {
         "date_histogram": {
            "field": "sold_date",
            "interval": "month",
            "format": "yyyy-MM-dd",
            "min_doc_count" : 0,
            "extended_bounds" : {
                "min" : "2019-01-01",
                "max" : "2022-12-31"
            }
         }
      }
   }
}
```

统计每季度每个品牌的销售额

```
GET /tvs/_search
{
  "size": 0,
  "aggs": {
    "group_by_sold_date": {
      "date_histogram": {
        "field": "sold_date",
        "interval": "quarter",
        "format": "yyyy-MM-dd",
        "min_doc_count": 0,
        "extended_bounds": {
          "min": "2019-01-01",
          "max": "2022-12-31"
```

```
        }
      },
      "aggs": {
        "group_by_brand": {
          "terms": {
            "field": "brand"
          },
          "aggs": {
            "sum_price": {
              "sum": {
                "field": "price"
              }
            }
          }
        },
        "total_sum_price": {
          "sum": {
            "field": "price"
          }
        }
      }
    }
  }
}
```

搜索与聚合结合，查询某个品牌按颜色销量

```
GET /tvs/_search
{
  "query":
  {
    "match":
    {
      "brand": "小米"
    }
  },
  "aggs": {
    "group_by_color":
    {
      "terms":
      {
        "field": "color"
      }
    }
  }
}
```

结果:

```
{
  "took" : 1,
```

```
  "timed_out" : false,
  "_shards" : {
    "total" : 1,
    "successful" : 1,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : {
      "value" : 5,
      "relation" : "eq"
    },
    "max_score" : 1.0033021,
    "hits" : [
      {
        "_index" : "tvs",
        "_id" : "7aouDoEBEpQthbP41cfj",
        "_score" : 1.0033021,
        "_source" : {
          "price" : 3000,
          "color" : "绿色",
          "brand" : "小米",
          "sold_date" : "2019-05-18"
        }
      },
      {
        "_index" : "tvs",
        "_id" : "8qouDoEBEpQthbP41cfj",
        "_score" : 1.0033021,
        "_source" : {
          "price" : 2500,
          "color" : "蓝色",
          "brand" : "小米",
          "sold_date" : "2020-02-12"
        }
      },
      {
        "_index" : "tvs",
        "_id" : "86ouDoEBEpQthbP41cfj",
        "_score" : 1.0033021,
        "_source" : {
          "price" : 4500,
          "color" : "绿色",
          "brand" : "小米",
          "sold_date" : "2020-04-22"
        }
      },
      {
        "_index" : "tvs",
        "_id" : "9qouDoEBEpQthbP41cfj",
        "_score" : 1.0033021,
        "_source" : {
          "price" : 8500,
          "color" : "红色",
          "brand" : "小米",
          "sold_date" : "2020-05-19"
        }
      },
```

```
      {
        "_index" : "tvs",
        "_id" : "-KouDoEBEpQthbP41cfj",
        "_score" : 1.0033021,
        "_source" : {
          "price" : 4800,
          "color" : "黑色",
          "brand" : "小米",
          "sold_date" : "2020-06-10"
        }
      }
    ]
  },
  "aggregations" : {
    "group_by_color" : {
      "doc_count_error_upper_bound" : 0,
      "sum_other_doc_count" : 0,
      "buckets" : [
        {
          "key" : "绿色",
          "doc_count" : 2
        },
        {
          "key" : "红色",
          "doc_count" : 1
        },
        {
          "key" : "蓝色",
          "doc_count" : 1
        },
        {
          "key" : "黑色",
          "doc_count" : 1
        }
      ]
    }
  }
}
```


单个品牌与所有品牌销量对比

```
GET /tvs/_search
{
  "query":
  {
    "term":
    {
      "brand":
      {
        "value": "小米"
      }
    }
```

```
  },
  "aggs":
  {
    "single_brand_avg_price":
    {
      "avg":
      {
        "field": "price"
      }
    },
    "all":
    {
      "global":
      {

      },
      "aggs":
      {
        "all_brand_avg_price":
        {
          "avg":
          {
            "field": "price"
          }
        }
      }
    }
  }
}
```

结果:

```
{
  "took" : 2,
  "timed_out" : false,
  "_shards" : {
    "total" : 1,
    "successful" : 1,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : {
      "value" : 5,
      "relation" : "eq"
    },
    "max_score" : 1.0033021,
    "hits" : [
      {
        "_index" : "tvs",
        "_id" : "7aouDoEBEpQthbP41cfj",
        "_score" : 1.0033021,
        "_source" : {
          "price" : 3000,
          "color" : "绿色",
          "brand" : "小米",
          "sold_date" : "2019-05-18"
```

```
        }
      },
      {
        "_index" : "tvs",
        "_id" : "8qouDoEBEpQthbP41cfj",
        "_score" : 1.0033021,
        "_source" : {
          "price" : 2500,
          "color" : "蓝色",
          "brand" : "小米",
          "sold_date" : "2020-02-12"
        }
      },
      {
        "_index" : "tvs",
        "_id" : "86ouDoEBEpQthbP41cfj",
        "_score" : 1.0033021,
        "_source" : {
          "price" : 4500,
          "color" : "绿色",
          "brand" : "小米",
          "sold_date" : "2020-04-22"
        }
      },
      {
        "_index" : "tvs",
        "_id" : "9qouDoEBEpQthbP41cfj",
        "_score" : 1.0033021,
        "_source" : {
          "price" : 8500,
          "color" : "红色",
          "brand" : "小米",
          "sold_date" : "2020-05-19"
        }
      },
      {
        "_index" : "tvs",
        "_id" : "-KouDoEBEpQthbP41cfj",
        "_score" : 1.0033021,
        "_source" : {
          "price" : 4800,
          "color" : "黑色",
          "brand" : "小米",
          "sold_date" : "2020-06-10"
        }
      }
    ]
  },
  "aggregations" : {
    "all" : {
      "doc_count" : 14,
      "all_brand_avg_price" : {
        "value" : 3671.4285714285716
      }
    },
    "single_brand_avg_price" : {
      "value" : 4660.0
    }
```

```
  }
}
```

统计价格大于1200的电视平均价格

```
GET /tvs/_search
{
  "query":
  {
    "constant_score":
    {
      "filter":
      {
        "range":
        {
          "price":
          {
            "gte": 1200
          }
        }
      }
    }
  },
  "aggs":
  {
    "avg_price":
    {
      "avg":
      {
        "field": "price"
      }
    }
  }

}
```

结果:

```
{
  "took" : 0,
  "timed_out" : false,
  "_shards" : {
    "total" : 1,
    "successful" : 1,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : {
      "value" : 13,
      "relation" : "eq"
    },
    "max_score" : 1.0,
```

```
    "hits" : [
      {
        "_index" : "tvs",
        "_id" : "7KouDoEBEpQthbP41cfj",
        "_score" : 1.0,
        "_source" : {
          "price" : 2000,
          "color" : "红色",
          "brand" : "长虹",
          "sold_date" : "2019-11-05"
        }
      },
      {
        "_index" : "tvs",
        "_id" : "7aouDoEBEpQthbP41cfj",
        "_score" : 1.0,
        "_source" : {
          "price" : 3000,
          "color" : "绿色",
          "brand" : "小米",
          "sold_date" : "2019-05-18"
        }
      },
      {
        "_index" : "tvs",
        "_id" : "7qouDoEBEpQthbP41cfj",
        "_score" : 1.0,
        "_source" : {
          "price" : 1500,
          "color" : "蓝色",
          "brand" : "TCL",
          "sold_date" : "2019-07-02"
        }
      },
      {
        "_index" : "tvs",
        "_id" : "76ouDoEBEpQthbP41cfj",
        "_score" : 1.0,
        "_source" : {
          "price" : 1200,
          "color" : "绿色",
          "brand" : "TCL",
          "sold_date" : "2019-08-19"
        }
      },
      {
        "_index" : "tvs",
        "_id" : "8KouDoEBEpQthbP41cfj",
        "_score" : 1.0,
        "_source" : {
          "price" : 2000,
          "color" : "红色",
          "brand" : "长虹",
          "sold_date" : "2019-11-05"
        }
      },
      {
        "_index" : "tvs",
```

```
        "_id" : "8aouDoEBEpQthbP41cfj",
        "_score" : 1.0,
        "_source" : {
          "price" : 8000,
          "color" : "红色",
          "brand" : "三星",
          "sold_date" : "2020-01-01"
        }
      },
      {
        "_index" : "tvs",
        "_id" : "8qouDoEBEpQthbP41cfj",
        "_score" : 1.0,
        "_source" : {
          "price" : 2500,
          "color" : "蓝色",
          "brand" : "小米",
          "sold_date" : "2020-02-12"
        }
      },
      {
        "_index" : "tvs",
        "_id" : "86ouDoEBEpQthbP41cfj",
        "_score" : 1.0,
        "_source" : {
          "price" : 4500,
          "color" : "绿色",
          "brand" : "小米",
          "sold_date" : "2020-04-22"
        }
      },
      {
        "_index" : "tvs",
        "_id" : "9KouDoEBEpQthbP41cfj",
        "_score" : 1.0,
        "_source" : {
          "price" : 6100,
          "color" : "蓝色",
          "brand" : "三星",
          "sold_date" : "2020-05-16"
        }
      },
      {
        "_index" : "tvs",
        "_id" : "9aouDoEBEpQthbP41cfj",
        "_score" : 1.0,
        "_source" : {
          "price" : 2100,
          "color" : "白色",
          "brand" : "TCL",
          "sold_date" : "2020-05-17"
        }
      }
    ]
  },
  "aggregations" : {
    "avg_price" : {
      "value" : 3876.923076923077
```

```
    }
  }
}
```

统计品牌最近一个月的平均价格

```
GET /tvs/_search
{
  "size": 0,
  "query": {
    "term": {
      "brand": {
        "value": "小米"
      }
    }
  },
  "aggs": {
    "recent_150d": {
      "filter": {
        "range": {
          "sold_date": {
            "gte": "now-150d"
          }
        }
      },
      "aggs": {
        "recent_150d_avg_price": {
          "avg": {
            "field": "price"
          }
        }
      }
    },
    "recent_140d": {
      "filter": {
        "range": {
          "sold_date": {
            "gte": "now-140d"
          }
        }
      },
      "aggs": {
        "recent_140d_avg_price": {
          "avg": {
            "field": "price"
          }
        }
      }
    },
    "recent_130d": {
      "filter": {
        "range": {
          "sold_date": {
            "gte": "now-130d"
```

```
          }
        }
      },
      "aggs": {
        "recent_130d_avg_price": {
          "avg": {
            "field": "price"
          }
        }
      }
    },
    "recent_800d": {
      "filter": {
        "range": {
          "sold_date": {
            "gte": "now-800d"
          }
        }
      },
      "aggs": {
        "recent_800d_avg_price": {
          "avg": {
            "field": "price"
          }
        }
      }
    }
  }
}
```

结果:

```
{
  "took" : 0,
  "timed_out" : false,
  "_shards" : {
    "total" : 1,
    "successful" : 1,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : {
      "value" : 5,
      "relation" : "eq"
    },
    "max_score" : null,
    "hits" : [ ]
  },
  "aggregations" : {
    "recent_800d" : {
      "doc_count" : 3,
      "recent_800d_avg_price" : {
        "value" : 5933.333333333333
      }
    },
```

```
    "recent_130d" : {
      "meta" : { },
      "doc_count" : 0,
      "recent_130d_avg_price" : {
        "value" : null
      }
    },
    "recent_140d" : {
      "meta" : { },
      "doc_count" : 0,
      "recent_140d_avg_price" : {
        "value" : null
      }
    },
    "recent_150d" : {
      "meta" : { },
      "doc_count" : 0,
      "recent_150d_avg_price" : {
        "value" : null
      }
    }
  }
}
```

按每种颜色的平均销售额降序排序

```
GET /tvs/_search
{
  "query":
  {
    "match_all": {}
  },
  "size": 0,
  "aggs":
  {
    "group_by_color":
    {
      "terms":
      {
        "field": "color",
        "order":
        {
          "avg_price": "desc"
        }
      },
      "aggs":
      {
        "avg_price":
        {
          "avg":
          {
            "field": "price"
```

```
          }
        }
      }
    }
  }
}
```

结果:


```
{
  "took" : 1,
  "timed_out" : false,
  "_shards" : {
    "total" : 1,
    "successful" : 1,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : {
      "value" : 14,
      "relation" : "eq"
    },
    "max_score" : null,
    "hits" : [ ]
  },
  "aggregations" : {
    "group_by_color" : {
      "doc_count_error_upper_bound" : 0,
      "sum_other_doc_count" : 0,
      "buckets" : [
        {
          "key" : "黑色",
          "doc_count" : 1,
          "avg_price" : {
            "value" : 4800.0
          }
        },
        {
          "key" : "红色",
          "doc_count" : 5,
          "avg_price" : {
            "value" : 4300.0
          }
        },
        {
          "key" : "蓝色",
          "doc_count" : 4,
          "avg_price" : {
            "value" : 3575.0
          }
        },
        {
          "key" : "绿色",
          "doc_count" : 3,
          "avg_price" : {
            "value" : 2900.0
```

```
          }
        },
        {
          "key" : "白色",
          "doc_count" : 1,
          "avg_price" : {
            "value" : 2100.0
          }
        }
      ]
    }
  }
}
```

按每种颜色的每种品牌平均销售额降序排序

```
GET /tvs/_search
{
  "query":
  {
    "match_all": {}
  },
  "size": 0,
  "aggs":
  {
    "group_by_color":
    {
      "terms":
      {
        "field": "color"

      },
      "aggs":
      {
        "group_by_brand":
        {
          "terms":
          {
            "field": "brand",
            "order":
            {
              "avg_price": "desc"
            }
          },
          "aggs":
          {
            "avg_price":
            {
              "avg":
              {
                "field": "price"
              }
            }
          }
```

```
        }
      }
    }
  }
}
```

结果:


```
{
  "took" : 2,
  "timed_out" : false,
  "_shards" : {
    "total" : 1,
    "successful" : 1,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : {
      "value" : 14,
      "relation" : "eq"
    },
    "max_score" : null,
    "hits" : [ ]
  },
  "aggregations" : {
    "group_by_color" : {
      "doc_count_error_upper_bound" : 0,
      "sum_other_doc_count" : 0,
      "buckets" : [
        {
          "key" : "红色",
          "doc_count" : 5,
          "group_by_brand" : {
            "doc_count_error_upper_bound" : 0,
            "sum_other_doc_count" : 0,
            "buckets" : [
              {
                "key" : "小米",
                "doc_count" : 1,
                "avg_price" : {
                  "value" : 8500.0
                }
              },
              {
                "key" : "三星",
                "doc_count" : 1,
                "avg_price" : {
                  "value" : 8000.0
                }
              },
              {
                "key" : "长虹",
                "doc_count" : 3,
                "avg_price" : {
                  "value" : 1666.6666666666667
                }
```

```
        }
      ]
    }
  },
  {
    "key" : "蓝色",
    "doc_count" : 4,
    "group_by_brand" : {
      "doc_count_error_upper_bound" : 0,
      "sum_other_doc_count" : 0,
      "buckets" : [
        {
          "key" : "三星",
          "doc_count" : 1,
          "avg_price" : {
            "value" : 6100.0
          }
        },
        {
          "key" : "长虹",
          "doc_count" : 1,
          "avg_price" : {
            "value" : 4200.0
          }
        },
        {
          "key" : "小米",
          "doc_count" : 1,
          "avg_price" : {
            "value" : 2500.0
          }
        },
        {
          "key" : "TCL",
          "doc_count" : 1,
          "avg_price" : {
            "value" : 1500.0
          }
        }
      ]
    }
  },
  {
    "key" : "绿色",
    "doc_count" : 3,
    "group_by_brand" : {
      "doc_count_error_upper_bound" : 0,
      "sum_other_doc_count" : 0,
      "buckets" : [
        {
          "key" : "小米",
          "doc_count" : 2,
          "avg_price" : {
            "value" : 3750.0
          }
        },
        {
          "key" : "TCL",
```

```
                "doc_count" : 1,
                "avg_price" : {
                  "value" : 1200.0
                }
              }
            ]
          }
        },
        {
          "key" : "白色",
          "doc_count" : 1,
          "group_by_brand" : {
            "doc_count_error_upper_bound" : 0,
            "sum_other_doc_count" : 0,
            "buckets" : [
              {
                "key" : "TCL",
                "doc_count" : 1,
                "avg_price" : {
                  "value" : 2100.0
                }
              }
            ]
          }
        },
        {
          "key" : "黑色",
          "doc_count" : 1,
          "group_by_brand" : {
            "doc_count_error_upper_bound" : 0,
            "sum_other_doc_count" : 0,
            "buckets" : [
              {
                "key" : "小米",
                "doc_count" : 1,
                "avg_price" : {
                  "value" : 4800.0
                }
              }
            ]
          }
        }
      ]
    }
  }
}
```


java API实现聚合

统计哪种颜色的电视销量最高

```
/**
     * 统计哪种颜色的电视销量最高
     * <p>
     * 请求内容：
     * <pre>
     *
     * GET /tvs/_search
     * {
     *   "query":
     *   {
     *     "match_all": {}
     *   },
     *   "size": 0,
     *   "aggs":
     *   {
     *     "popular_colors":
     *     {
     *       "terms":
     *       {
     *         "field": "color"
     *       }
     *     }
     *   }
     * }
     *
     * </pre>
     * <p>
     * 结果：
     * <pre>
     *
     * {
     *   "took" : 12,
     *   "timed_out" : false,
     *   "_shards" : {
     *     "total" : 1,
     *     "successful" : 1,
     *     "skipped" : 0,
     *     "failed" : 0
     *   },
     *   "hits" : {
     *     "total" : {
     *       "value" : 14,
     *       "relation" : "eq"
     *     },
     *     "max_score" : null,
     *     "hits" : [ ]
     *   },
     *   "aggregations" : {
     *     "popular_colors" : {
     *       "doc_count_error_upper_bound" : 0,
     *       "sum_other_doc_count" : 0,
     *       "buckets" : [
     *         {
     *           "key" : "红色",
     *           "doc_count" : 5
```

```
     *          },
     *          {
     *            "key" : "蓝色",
     *            "doc_count" : 4
     *          },
     *          {
     *            "key" : "绿色",
     *            "doc_count" : 3
     *          },
     *          {
     *            "key" : "白色",
     *            "doc_count" : 1
     *          },
     *          {
     *            "key" : "黑色",
     *            "doc_count" : 1
     *          }
     *        ]
     *      }
     *    }
     * }
     *
     * </pre>
     *
     * 程序结果：
     * <pre>
     *
     * ----key：红色
     * ----doc_count：5
     * ----------------------------------------
     * ----key：蓝色
     * ----doc_count：4
     * ----------------------------------------
     * ----key：绿色
     * ----doc_count：3
     * ----------------------------------------
     * ----key：白色
     * ----doc_count：1
     * ----------------------------------------
     * ----key：黑色
     * ----doc_count：1
     * ----------------------------------------
     *
     * </pre>
     *
     * @throws Exception Exception
     */
    @Test
    void aggregation1() throws Exception
    {
        //构建请求
        SearchRequest searchRequest = new SearchRequest("tvs");
        //构建请求体
        SearchSourceBuilder searchSourceBuilder = new SearchSourceBuilder();
        //查询
        searchSourceBuilder.query(QueryBuilders.matchAllQuery());
        //分页
        searchSourceBuilder.size(0);
```

```
        //聚合

 searchSourceBuilder.aggregation(AggregationBuilders.terms("popular_colors").fie
ld("color"));
        //放入到请求中
        searchRequest.source(searchSourceBuilder);
        //发起请求
        SearchResponse searchResponse = client.search(searchRequest,
RequestOptions.DEFAULT);
        //获取数据
        //获取aggregations部分
        Aggregations aggregations = searchResponse.getAggregations();
        //获得popular_colors
        Terms popular_colors = aggregations.get("popular_colors");
        //获取buckets部分
        List<? extends Terms.Bucket> buckets = popular_colors.getBuckets();
        //遍历
        for (Terms.Bucket bucket : buckets)
        {
            //获取数据
            String key = (String) bucket.getKey();
            long docCount = bucket.getDocCount();
            //打印
            System.out.println("----key: " + key);
            System.out.println("----doc_count: " + docCount);
            System.out.println("-----------------------------------------");
        }

    }
```

统计每种颜色电视平均价格

```
/**
     * 统计每种颜色电视平均价格
     * <p>
     * 请求内容:
     * <pre>
     *
     * GET /tvs/_search
     * {
     *   "query":
     *   {
     *     "match_all": {}
     *   },
     *   "size": 0,
     *   "aggs":
     *   {
     *     "group_by_colors":
     *     {
     *       "terms":
     *       {
     *         "field": "color"
     *       },
```

```
*       "aggs": {
*         "avg_price":
*         {
*           "avg":
*           {
*             "field": "price"
*           }
*         }
*       }
*     }
*   }
* }
*
*
*
* </pre>
* <p>
* 结果:
* <pre>
*
* {
*   "took" : 1,
*   "timed_out" : false,
*   "_shards" : {
*     "total" : 1,
*     "successful" : 1,
*     "skipped" : 0,
*     "failed" : 0
*   },
*   "hits" : {
*     "total" : {
*       "value" : 14,
*       "relation" : "eq"
*     },
*     "max_score" : null,
*     "hits" : [ ]
*   },
*   "aggregations" : {
*     "group_by_colors" : {
*       "doc_count_error_upper_bound" : 0,
*       "sum_other_doc_count" : 0,
*       "buckets" : [
*         {
*           "key" : "红色",
*           "doc_count" : 5,
*           "avg_price" : {
*             "value" : 4300.0
*           }
*         },
*         {
*           "key" : "蓝色",
*           "doc_count" : 4,
*           "avg_price" : {
*             "value" : 3575.0
*           }
*         },
*         {
*           "key" : "绿色",
```

```
     *            "doc_count" : 3,
     *            "avg_price" : {
     *              "value" : 2900.0
     *            }
     *          },
     *          {
     *            "key" : "白色",
     *            "doc_count" : 1,
     *            "avg_price" : {
     *              "value" : 2100.0
     *            }
     *          },
     *          {
     *            "key" : "黑色",
     *            "doc_count" : 1,
     *            "avg_price" : {
     *              "value" : 4800.0
     *            }
     *          }
     *        ]
     *      }
     *    }
     * }
     *
     * </pre>
     * <p>
     * 程序结果：
     * <pre>
     *
     * ----key：红色
     * ----doc_count：5
     * ----平均价格：4300.0
     * ----------------------------------------
     * ----key：蓝色
     * ----doc_count：4
     * ----平均价格：3575.0
     * ----------------------------------------
     * ----key：绿色
     * ----doc_count：3
     * ----平均价格：2900.0
     * ----------------------------------------
     * ----key：白色
     * ----doc_count：1
     * ----平均价格：2100.0
     * ----------------------------------------
     * ----key：黑色
     * ----doc_count：1
     * ----平均价格：4800.0
     * ----------------------------------------
     *
     * </pre>
     *
     * @throws Exception Exception
     */
    @Test
    void aggregation2() throws Exception
    {
        //构建请求
```

```
        SearchRequest searchRequest = new SearchRequest("tvs");
        //构建请求体
        SearchSourceBuilder searchSourceBuilder = new SearchSourceBuilder();
        //查询
        searchSourceBuilder.query(QueryBuilders.matchAllQuery());
        //分页
        searchSourceBuilder.size(0);
        //聚合

 searchSourceBuilder.aggregation(AggregationBuilders.terms("group_by_colors").fi
eld("color")

.subAggregation(AggregationBuilders.avg("avg_price").field("price")));
        //放入到请求中
        searchRequest.source(searchSourceBuilder);
        //发起请求
        SearchResponse searchResponse = client.search(searchRequest,
RequestOptions.DEFAULT);
        //获取数据
        //获取aggregations部分
        Aggregations aggregations = searchResponse.getAggregations();
        //获得group_by_colors
        Terms group_by_colors = aggregations.get("group_by_colors");
        //获取buckets部分
        List<? extends Terms.Bucket> buckets = group_by_colors.getBuckets();
        //遍历
        for (Terms.Bucket bucket : buckets)
        {
            //获取数据
            String key = (String) bucket.getKey();
            long docCount = bucket.getDocCount();
            Avg avg_price = bucket.getAggregations().get("avg_price");
            double avgPriceValue = avg_price.getValue();
            //打印
            System.out.println("----key: " + key);
            System.out.println("----doc_count: " + docCount);
            System.out.println("----平均价格: " + avgPriceValue);
            System.out.println("----------------------------------------");
        }
    }
```

统计每个颜色下，平均价格及每个颜色下，每个品牌的平均价格

```
/**
     * 统计每个颜色下，平均价格及每个颜色下，每个品牌的平均价格
     * <p>
     * 请求内容:
     * <pre>
     *
     * GET /tvs/_search
```

```
 * {
 *   "query":
 *   {
 *     "match_all": {}
 *   },
 *   "size": 0,
 *   "aggs":
 *   {
 *     "group_by_colors":
 *     {
 *       "terms":
 *       {
 *         "field": "color"
 *       },
 *       "aggs":
 *       {
 *         "avg_price":
 *         {
 *           "avg":
 *           {
 *             "field": "price"
 *           }
 *         },
 *         "group_by_brand":
 *         {
 *           "terms":
 *           {
 *             "field": "brand"
 *           },
 *           "aggs":
 *           {
 *             "avg_price":
 *             {
 *               "avg":
 *               {
 *                 "field": "price"
 *               }
 *             }
 *           }
 *         }
 *       }
 *     }
 *   }
 * }
 *
 * </pre>
 * <p>
 * 结果:
 * <pre>
 *
 * {
 *   "took" : 1,
 *   "timed_out" : false,
 *   "_shards" : {
 *     "total" : 1,
 *     "successful" : 1,
 *     "skipped" : 0,
 *     "failed" : 0
```

```
*   },
*   "hits" : {
*     "total" : {
*       "value" : 14,
*       "relation" : "eq"
*     },
*     "max_score" : null,
*     "hits" : [ ]
*   },
*   "aggregations" : {
*     "group_by_colors" : {
*       "doc_count_error_upper_bound" : 0,
*       "sum_other_doc_count" : 0,
*       "buckets" : [
*         {
*           "key" : "红色",
*           "doc_count" : 5,
*           "avg_price" : {
*             "value" : 4300.0
*           },
*           "group_by_brand" : {
*             "doc_count_error_upper_bound" : 0,
*             "sum_other_doc_count" : 0,
*             "buckets" : [
*               {
*                 "key" : "长虹",
*                 "doc_count" : 3,
*                 "avg_price" : {
*                   "value" : 1666.6666666666667
*                 }
*               },
*               {
*                 "key" : "三星",
*                 "doc_count" : 1,
*                 "avg_price" : {
*                   "value" : 8000.0
*                 }
*               },
*               {
*                 "key" : "小米",
*                 "doc_count" : 1,
*                 "avg_price" : {
*                   "value" : 8500.0
*                 }
*               }
*             ]
*           }
*         },
*         {
*           "key" : "蓝色",
*           "doc_count" : 4,
*           "avg_price" : {
*             "value" : 3575.0
*           },
*           "group_by_brand" : {
*             "doc_count_error_upper_bound" : 0,
*             "sum_other_doc_count" : 0,
*             "buckets" : [
```

```
*                {
*                  "key" : "TCL",
*                  "doc_count" : 1,
*                  "avg_price" : {
*                    "value" : 1500.0
*                  }
*                },
*                {
*                  "key" : "三星",
*                  "doc_count" : 1,
*                  "avg_price" : {
*                    "value" : 6100.0
*                  }
*                },
*                {
*                  "key" : "小米",
*                  "doc_count" : 1,
*                  "avg_price" : {
*                    "value" : 2500.0
*                  }
*                },
*                {
*                  "key" : "长虹",
*                  "doc_count" : 1,
*                  "avg_price" : {
*                    "value" : 4200.0
*                  }
*                }
*              ]
*            }
*          },
*          {
*            "key" : "绿色",
*            "doc_count" : 3,
*            "avg_price" : {
*              "value" : 2900.0
*            },
*            "group_by_brand" : {
*              "doc_count_error_upper_bound" : 0,
*              "sum_other_doc_count" : 0,
*              "buckets" : [
*                {
*                  "key" : "小米",
*                  "doc_count" : 2,
*                  "avg_price" : {
*                    "value" : 3750.0
*                  }
*                },
*                {
*                  "key" : "TCL",
*                  "doc_count" : 1,
*                  "avg_price" : {
*                    "value" : 1200.0
*                  }
*                }
*              ]
*            }
*          },
```

```
*          {
*            "key" : "白色",
*            "doc_count" : 1,
*            "avg_price" : {
*              "value" : 2100.0
*            },
*            "group_by_brand" : {
*              "doc_count_error_upper_bound" : 0,
*              "sum_other_doc_count" : 0,
*              "buckets" : [
*                {
*                  "key" : "TCL",
*                  "doc_count" : 1,
*                  "avg_price" : {
*                    "value" : 2100.0
*                  }
*                }
*              ]
*            }
*          },
*          {
*            "key" : "黑色",
*            "doc_count" : 1,
*            "avg_price" : {
*              "value" : 4800.0
*            },
*            "group_by_brand" : {
*              "doc_count_error_upper_bound" : 0,
*              "sum_other_doc_count" : 0,
*              "buckets" : [
*                {
*                  "key" : "小米",
*                  "doc_count" : 1,
*                  "avg_price" : {
*                    "value" : 4800.0
*                  }
*                }
*              ]
*            }
*          }
*        ]
*      }
*    }
* }
*
* </pre>
* <p>
* 程序结果：
* <pre>
*
* ----key：红色
* ----doc_count：5
* ----平均价格：4300.0
* ----group_by_brand：
* --------key：长虹
* --------doc_count：3
* --------平均价格：1666.6666666666667
* ----------------
```

```
* --------key：三星
* --------doc_count：1
* --------平均价格：8000.0
* ----------------
* --------key：小米
* --------doc_count：1
* --------平均价格：8500.0
* ----------------
* ----------------------------------------
* ----key：蓝色
* ----doc_count：4
* ----平均价格：3575.0
* ----group_by_brand：
* --------key：TCL
* --------doc_count：1
* --------平均价格：1500.0
* ----------------
* --------key：三星
* --------doc_count：1
* --------平均价格：6100.0
* ----------------
* --------key：小米
* --------doc_count：1
* --------平均价格：2500.0
* ----------------
* --------key：长虹
* --------doc_count：1
* --------平均价格：4200.0
* ----------------
* ----------------------------------------
* ----key：绿色
* ----doc_count：3
* ----平均价格：2900.0
* ----group_by_brand：
* --------key：小米
* --------doc_count：2
* --------平均价格：3750.0
* ----------------
* --------key：TCL
* --------doc_count：1
* --------平均价格：1200.0
* ----------------
* ----------------------------------------
* ----key：白色
* ----doc_count：1
* ----平均价格：2100.0
* ----group_by_brand：
* --------key：TCL
* --------doc_count：1
* --------平均价格：2100.0
* ----------------
* ----------------------------------------
* ----key：黑色
* ----doc_count：1
* ----平均价格：4800.0
* ----group_by_brand：
* --------key：小米
* --------doc_count：1
```

```
     * --------平均价格：4800.0
     * ----------------
     * ----------------------------------------
     *
     * </pre>
     *
     * @throws Exception Exception
     */
    @Test
    void aggregation3() throws Exception
    {
        //构建请求
        SearchRequest searchRequest = new SearchRequest("tvs");
        //构建请求体
        SearchSourceBuilder searchSourceBuilder = new SearchSourceBuilder();
        //查询
        searchSourceBuilder.query(QueryBuilders.matchAllQuery());
        //分页
        searchSourceBuilder.size(0);
        //聚合
        searchSourceBuilder.aggregation(
                AggregationBuilders.terms("group_by_colors").field("color")
                        .subAggregations(AggregatorFactories.builder()

.addAggregator(AggregationBuilders.avg("avg_price").field("price"))

.addAggregator(AggregationBuilders.terms("group_by_brand").field("brand")

.subAggregation(AggregationBuilders.avg("avg_price").field("price")))));
        //放入到请求中
        searchRequest.source(searchSourceBuilder);
        //发起请求
        SearchResponse searchResponse = client.search(searchRequest,
RequestOptions.DEFAULT);
        //获取数据
        //获取aggregations部分
        Aggregations aggregations = searchResponse.getAggregations();
        //获得group_by_colors
        Terms group_by_colors = aggregations.get("group_by_colors");
        //获取buckets部分
        List<? extends Terms.Bucket> buckets = group_by_colors.getBuckets();
        //遍历
        for (Terms.Bucket bucket : buckets)
        {
            //获取数据
            String key = (String) bucket.getKey();
            long docCount = bucket.getDocCount();
            Avg avg_price = bucket.getAggregations().get("avg_price");
            Terms group_by_brand =
bucket.getAggregations().get("group_by_brand");
            List<? extends Terms.Bucket> buckets1 = group_by_brand.getBuckets();
            double avgPriceValue = avg_price.getValue();
            //打印
            System.out.println("----key：" + key);
            System.out.println("----doc_count：" + docCount);
            System.out.println("----平均价格：" + avgPriceValue);
            System.out.println("----group_by_brand：");
            for (Terms.Bucket bucket1 : buckets1)
```

```
            {
                //获取数据
                String key1 = (String) bucket1.getKey();
                long docCount1 = bucket1.getDocCount();
                Avg avg_price1 = bucket1.getAggregations().get("avg_price");
                double avgPrice1Value = avg_price1.getValue();
                //打印
                System.out.println("--------key: " + key1);
                System.out.println("--------doc_count: " + docCount1);
                System.out.println("--------平均价格: " + avgPrice1Value);
                System.out.println("----------------");
            }
            System.out.println("----------------------------------------");
        }
    }
```

求每个颜色的最大价格、最小价格、平均价格和总价格

```
/**
    * 求每个颜色的最大价格、最小价格、平均价格和总价格
    * 请求内容:
    * <pre>
    *
    * GET /tvs/_search
    * {
    *   "query":
    *   {
    *     "match_all": {}
    *   },
    *   "size": 0,
    *   "aggs":
    *   {
    *     "group_by_color":
    *     {
    *       "terms":
    *       {
    *         "field": "color"
    *       },
    *       "aggs":
    *       {
    *         "max_price":
    *         {
    *           "max":
    *           {
    *             "field": "price"
    *           }
    *         },
    *         "min_price":
    *         {
    *           "min":
    *           {
```

```
*             "field": "price"
*           }
*         },
*         "avg_price":
*         {
*           "avg":
*           {
*             "field": "price"
*           }
*         },
*         "sum_price":
*         {
*           "sum":
*           {
*             "field": "price"
*           }
*         }
*       }
*     }
*   }
* }
*
* </pre>
* <p>
* 结果:
* <pre>
*
* {
*   "took" : 1,
*   "timed_out" : false,
*   "_shards" : {
*     "total" : 1,
*     "successful" : 1,
*     "skipped" : 0,
*     "failed" : 0
*   },
*   "hits" : {
*     "total" : {
*       "value" : 14,
*       "relation" : "eq"
*     },
*     "max_score" : null,
*     "hits" : [ ]
*   },
*   "aggregations" : {
*     "group_by_color" : {
*       "doc_count_error_upper_bound" : 0,
*       "sum_other_doc_count" : 0,
*       "buckets" : [
*         {
*           "key" : "红色",
*           "doc_count" : 5,
*           "max_price" : {
*             "value" : 8500.0
*           },
*           "min_price" : {
*             "value" : 1000.0
*           },
```

```
*            "avg_price" : {
*              "value" : 4300.0
*            },
*            "sum_price" : {
*              "value" : 21500.0
*            }
*          },
*          {
*            "key" : "蓝色",
*            "doc_count" : 4,
*            "max_price" : {
*              "value" : 6100.0
*            },
*            "min_price" : {
*              "value" : 1500.0
*            },
*            "avg_price" : {
*              "value" : 3575.0
*            },
*            "sum_price" : {
*              "value" : 14300.0
*            }
*          },
*          {
*            "key" : "绿色",
*            "doc_count" : 3,
*            "max_price" : {
*              "value" : 4500.0
*            },
*            "min_price" : {
*              "value" : 1200.0
*            },
*            "avg_price" : {
*              "value" : 2900.0
*            },
*            "sum_price" : {
*              "value" : 8700.0
*            }
*          },
*          {
*            "key" : "白色",
*            "doc_count" : 1,
*            "max_price" : {
*              "value" : 2100.0
*            },
*            "min_price" : {
*              "value" : 2100.0
*            },
*            "avg_price" : {
*              "value" : 2100.0
*            },
*            "sum_price" : {
*              "value" : 2100.0
*            }
*          },
*          {
*            "key" : "黑色",
*            "doc_count" : 1,
```

```
*            "max_price" : {
*              "value" : 4800.0
*            },
*            "min_price" : {
*              "value" : 4800.0
*            },
*            "avg_price" : {
*              "value" : 4800.0
*            },
*            "sum_price" : {
*              "value" : 4800.0
*            }
*          }
*        ]
*      }
*    }
* }
*
* </pre>
* <p>
* 程序结果：
* <pre>
*
* ----key：红色
* ----doc_count：5
* ----group_by_brand：
* ----max_price：8500.0
* ----min_price：1000.0
* ----avg_price：4300.0
* ----sum_price：21500.0
* ----------------------------------------
* ----key：蓝色
* ----doc_count：4
* ----group_by_brand：
* ----max_price：6100.0
* ----min_price：1500.0
* ----avg_price：3575.0
* ----sum_price：14300.0
* ----------------------------------------
* ----key：绿色
* ----doc_count：3
* ----group_by_brand：
* ----max_price：4500.0
* ----min_price：1200.0
* ----avg_price：2900.0
* ----sum_price：8700.0
* ----------------------------------------
* ----key：白色
* ----doc_count：1
* ----group_by_brand：
* ----max_price：2100.0
* ----min_price：2100.0
* ----avg_price：2100.0
* ----sum_price：2100.0
* ----------------------------------------
* ----key：黑色
* ----doc_count：1
* ----group_by_brand：
```

```
     * ----max_price: 4800.0
     * ----min_price: 4800.0
     * ----avg_price: 4800.0
     * ----sum_price: 4800.0
     * ----------------------------------------
     *
     * </pre>
     *
     * @throws Exception Exception
     */
    @Test
    void aggregation4() throws Exception
    {
        //构建请求
        SearchRequest searchRequest = new SearchRequest("tvs");
        //构建请求体
        SearchSourceBuilder searchSourceBuilder = new SearchSourceBuilder();
        //查询
        searchSourceBuilder.query(QueryBuilders.matchAllQuery());
        //分页
        searchSourceBuilder.size(0);
        //聚合
        searchSourceBuilder.aggregation(
                AggregationBuilders.terms("group_by_colors").field("color")
                        .subAggregations(AggregatorFactories.builder()

.addAggregator(AggregationBuilders.max("max_price").field("price"))

.addAggregator(AggregationBuilders.min("min_price").field("price"))

.addAggregator(AggregationBuilders.avg("avg_price").field("price"))

.addAggregator(AggregationBuilders.sum("sum_price").field("price")))
        );

        //放入到请求中
        searchRequest.source(searchSourceBuilder);
        //发起请求
        SearchResponse searchResponse = client.search(searchRequest,
RequestOptions.DEFAULT);
        //获取数据
        //获取aggregations部分
        Aggregations aggregations = searchResponse.getAggregations();
        //获得group_by_colors
        Terms group_by_colors = aggregations.get("group_by_colors");
        //获取buckets部分
        List<? extends Terms.Bucket> buckets = group_by_colors.getBuckets();
        //遍历
        for (Terms.Bucket bucket : buckets)
        {
            //获取数据
            String key = (String) bucket.getKey();
            long docCount = bucket.getDocCount();
            Max max_price = bucket.getAggregations().get("max_price");
            Min min_price = bucket.getAggregations().get("min_price");
            Avg avg_price = bucket.getAggregations().get("avg_price");
            Sum sum_price = bucket.getAggregations().get("sum_price");
            double maxPriceValue = max_price.getValue();
```

```
            double minPriceValue = min_price.getValue();
            double avgPriceValue = avg_price.getValue();
            double sumPriceValue = sum_price.getValue();

            //打印
            System.out.println("----key: " + key);
            System.out.println("----doc_count: " + docCount);
            System.out.println("----group_by_brand: ");
            System.out.println("----max_price: " + maxPriceValue);
            System.out.println("----min_price: " + minPriceValue);
            System.out.println("----avg_price: " + avgPriceValue);
            System.out.println("----sum_price: " + sumPriceValue);


            System.out.println("-----------------------------------------");
        }
    }
```

划分范围 histogram

```
/**
     * 划分范围 histogram
     * <p>
     * 请求内容:
     * <pre>
     *
     * GET /tvs/_search
     * {
     *   "query":
     *   {
     *     "match_all": {}
     *   },
     *   "size": 0,
     *   "aggs":
     *   {
     *     "histogram_price":
     *     {
     *       "histogram":
     *       {
     *         "field": "price",
     *         "interval": 2000
     *       }
     *     }
     *   }
     * }
     *
     * </pre>
     * <p>
     * 结果:
     * <pre>
     *
```

```
     * {
     *   "took" : 1,
     *   "timed_out" : false,
     *   "_shards" : {
     *     "total" : 1,
     *     "successful" : 1,
     *     "skipped" : 0,
     *     "failed" : 0
     *   },
     *   "hits" : {
     *     "total" : {
     *       "value" : 14,
     *       "relation" : "eq"
     *     },
     *     "max_score" : null,
     *     "hits" : [ ]
     *   },
     *   "aggregations" : {
     *     "histogram_price" : {
     *       "buckets" : [
     *         {
     *           "key" : 0.0,
     *           "doc_count" : 3
     *         },
     *         {
     *           "key" : 2000.0,
     *           "doc_count" : 5
     *         },
     *         {
     *           "key" : 4000.0,
     *           "doc_count" : 3
     *         },
     *         {
     *           "key" : 6000.0,
     *           "doc_count" : 1
     *         },
     *         {
     *           "key" : 8000.0,
     *           "doc_count" : 2
     *         }
     *       ]
     *     }
     *   }
     * }
     *
     * </pre>
     * <p>
     * 程序结果：
     * <pre>
     *
     * ----key：0.0
     * ----doc_count：3
     * ----------------------------------------
     * ----key：2000.0
     * ----doc_count：5
     * ----------------------------------------
     * ----key：4000.0
     * ----doc_count：3
```

```
     * ----------------------------------------
     * ----key：6000.0
     * ----doc_count：1
     * ----------------------------------------
     * ----key：8000.0
     * ----doc_count：2
     * ----------------------------------------
     *
     * </pre>
     *
     * @throws Exception Exception
     */
    @Test
    void aggregation5() throws Exception
    {
        //构建请求
        SearchRequest searchRequest = new SearchRequest("tvs");
        //构建请求体
        SearchSourceBuilder searchSourceBuilder = new SearchSourceBuilder();
        //查询
        searchSourceBuilder.query(QueryBuilders.matchAllQuery());
        //分页
        searchSourceBuilder.size(0);
        //聚合
        searchSourceBuilder.aggregation(

 AggregationBuilders.histogram("histogram_price").field("price").interval(2000)
        );

        //放入到请求中
        searchRequest.source(searchSourceBuilder);
        //发起请求
        SearchResponse searchResponse = client.search(searchRequest,
RequestOptions.DEFAULT);
        //获取数据
        //获取aggregations部分
        Aggregations aggregations = searchResponse.getAggregations();
        //获得histogram_price
        Histogram histogram_price = aggregations.get("histogram_price");
        //获取buckets部分
        List<? extends Histogram.Bucket> buckets = histogram_price.getBuckets();
        //遍历
        for (Histogram.Bucket bucket : buckets)
        {
            //获取数据
            Double key = (Double) bucket.getKey();
            long docCount = bucket.getDocCount();
            //打印
            System.out.println("----key：" + key);
            System.out.println("----doc_count：" + docCount);
            System.out.println("----------------------------------------");
        }
    }
```

搜索与聚合结合，查询某个品牌按颜色销量

```
/**
     * 搜索与聚合结合，查询某个品牌按颜色销量
     * <p>
     * 请求内容：
     * <pre>
     *
     * GET /tvs/_search
     * {
     *   "query":
     *   {
     *     "match":
     *     {
     *       "brand": "小米"
     *     }
     *   },
     *   "aggs": {
     *     "group_by_color":
     *     {
     *       "terms":
     *       {
     *         "field": "color"
     *       }
     *     }
     *   }
     * }
     *
     * </pre>
     * <p>
     * 结果：
     * <pre>
     *
     * {
     *   "took" : 1,
     *   "timed_out" : false,
     *   "_shards" : {
     *     "total" : 1,
     *     "successful" : 1,
     *     "skipped" : 0,
     *     "failed" : 0
     *   },
     *   "hits" : {
     *     "total" : {
     *       "value" : 5,
     *       "relation" : "eq"
     *     },
     *     "max_score" : 1.0033021,
     *     "hits" : [
     *       {
     *         "_index" : "tvs",
     *         "_id" : "7aouDoEBEpQthbP41cfj",
     *         "_score" : 1.0033021,
     *         "_source" : {
     *           "price" : 3000,
```

```
*           "color" : "绿色",
*           "brand" : "小米",
*           "sold_date" : "2019-05-18"
*         }
*       },
*       {
*         "_index" : "tvs",
*         "_id" : "8qouDoEBEpQthbP41cfj",
*         "_score" : 1.0033021,
*         "_source" : {
*           "price" : 2500,
*           "color" : "蓝色",
*           "brand" : "小米",
*           "sold_date" : "2020-02-12"
*         }
*       },
*       {
*         "_index" : "tvs",
*         "_id" : "86ouDoEBEpQthbP41cfj",
*         "_score" : 1.0033021,
*         "_source" : {
*           "price" : 4500,
*           "color" : "绿色",
*           "brand" : "小米",
*           "sold_date" : "2020-04-22"
*         }
*       },
*       {
*         "_index" : "tvs",
*         "_id" : "9qouDoEBEpQthbP41cfj",
*         "_score" : 1.0033021,
*         "_source" : {
*           "price" : 8500,
*           "color" : "红色",
*           "brand" : "小米",
*           "sold_date" : "2020-05-19"
*         }
*       },
*       {
*         "_index" : "tvs",
*         "_id" : "-KouDoEBEpQthbP41cfj",
*         "_score" : 1.0033021,
*         "_source" : {
*           "price" : 4800,
*           "color" : "黑色",
*           "brand" : "小米",
*           "sold_date" : "2020-06-10"
*         }
*       }
*     ]
*   },
*   "aggregations" : {
*     "group_by_color" : {
*       "doc_count_error_upper_bound" : 0,
*       "sum_other_doc_count" : 0,
*       "buckets" : [
*         {
*           "key" : "绿色",
```

```
     *            "doc_count" : 2
     *          },
     *          {
     *            "key" : "红色",
     *            "doc_count" : 1
     *          },
     *          {
     *            "key" : "蓝色",
     *            "doc_count" : 1
     *          },
     *          {
     *            "key" : "黑色",
     *            "doc_count" : 1
     *          }
     *        ]
     *      }
     *    }
     * }
     *
     * </pre>
     * <p>
     * 程序结果：
     * <pre>
     *
     * --数量：5
     * --数组数量：5
     * --最大分数：1.0033021
     * -->{color=绿色, price=3000, sold_date=2019-05-18, brand=小米}
     * -->{color=蓝色, price=2500, sold_date=2020-02-12, brand=小米}
     * -->{color=绿色, price=4500, sold_date=2020-04-22, brand=小米}
     * -->{color=红色, price=8500, sold_date=2020-05-19, brand=小米}
     * -->{color=黑色, price=4800, sold_date=2020-06-10, brand=小米}
     *
     * 聚合结果：
     *
     * ----key：绿色
     * ----doc_count：2
     * ----------------------------------------
     * ----key：红色
     * ----doc_count：1
     * ----------------------------------------
     * ----key：蓝色
     * ----doc_count：1
     * ----------------------------------------
     * ----key：黑色
     * ----doc_count：1
     * ----------------------------------------
     *
     * </pre>
     *
     * @throws Exception Exception
     */
    @Test
    void aggregation6() throws Exception
    {
        //构建请求
        SearchRequest searchRequest = new SearchRequest("tvs");
        //构建请求体
```

```
        SearchSourceBuilder searchSourceBuilder = new SearchSourceBuilder();
        //查询
        searchSourceBuilder.query(QueryBuilders.matchQuery("brand", "小米"));
        //分页
        //searchSourceBuilder.size(0);
        //聚合
        searchSourceBuilder.aggregation(
                AggregationBuilders.terms("group_by_color").field("color")
        );

        //放入到请求中
        searchRequest.source(searchSourceBuilder);
        //发起请求
        SearchResponse searchResponse = client.search(searchRequest,
RequestOptions.DEFAULT);
        //获取数据
        //获得hits
        SearchHits hits = searchResponse.getHits();
        long value = hits.getTotalHits().value;
        float maxScore = hits.getMaxScore();
        SearchHit[] hitsHits = hits.getHits();
        System.out.println("--数量: " + value);
        System.out.println("--数组数量: " + hitsHits.length);
        System.out.println("--最大分数: " + maxScore);
        for (SearchHit hitsHit : hitsHits)
        {
            Map<String, Object> sourceAsMap = hitsHit.getSourceAsMap();
            System.out.println("-->" + sourceAsMap);
        }
        System.out.println();
        System.out.println("聚合结果: ");
        System.out.println();

        //获取aggregations部分
        Aggregations aggregations = searchResponse.getAggregations();
        //获得group_by_color
        Terms group_by_color = aggregations.get("group_by_color");
        //获取buckets部分
        List<? extends Terms.Bucket> buckets = group_by_color.getBuckets();
        //遍历
        for (Terms.Bucket bucket : buckets)
        {
            //获取数据
            String key = (String) bucket.getKey();
            long docCount = bucket.getDocCount();
            //打印
            System.out.println("----key: " + key);
            System.out.println("----doc_count: " + docCount);
            System.out.println("----------------------------------------");
        }
    }
```

单个品牌与所有品牌销量对比

```
/**
     * 单个品牌与所有品牌销量对比
     * <p>
     * 请求内容:
     * <pre>
     *
     * GET /tvs/_search
     * {
     *   "query":
     *   {
     *     "term":
     *     {
     *       "brand":
     *       {
     *         "value": "小米"
     *       }
     *     }
     *   },
     *   "aggs":
     *   {
     *     "single_brand_avg_price":
     *     {
     *       "avg":
     *       {
     *         "field": "price"
     *       }
     *     },
     *     "all":
     *     {
     *       "global":
     *       {
     *
     *       },
     *       "aggs":
     *       {
     *         "all_brand_avg_price":
     *         {
     *           "avg":
     *           {
     *             "field": "price"
     *           }
     *         }
     *       }
     *     }
     *   }
     * }
     *
     * </pre>
     * <p>
     * 结果:
     * <pre>
     *
     * {
```

```
*   "took" : 2,
*   "timed_out" : false,
*   "_shards" : {
*     "total" : 1,
*     "successful" : 1,
*     "skipped" : 0,
*     "failed" : 0
*   },
*   "hits" : {
*     "total" : {
*       "value" : 5,
*       "relation" : "eq"
*     },
*     "max_score" : 1.0033021,
*     "hits" : [
*       {
*         "_index" : "tvs",
*         "_id" : "7aouDoEBEpQthbP41cfj",
*         "_score" : 1.0033021,
*         "_source" : {
*           "price" : 3000,
*           "color" : "绿色",
*           "brand" : "小米",
*           "sold_date" : "2019-05-18"
*         }
*       },
*       {
*         "_index" : "tvs",
*         "_id" : "8qouDoEBEpQthbP41cfj",
*         "_score" : 1.0033021,
*         "_source" : {
*           "price" : 2500,
*           "color" : "蓝色",
*           "brand" : "小米",
*           "sold_date" : "2020-02-12"
*         }
*       },
*       {
*         "_index" : "tvs",
*         "_id" : "86ouDoEBEpQthbP41cfj",
*         "_score" : 1.0033021,
*         "_source" : {
*           "price" : 4500,
*           "color" : "绿色",
*           "brand" : "小米",
*           "sold_date" : "2020-04-22"
*         }
*       },
*       {
*         "_index" : "tvs",
*         "_id" : "9qouDoEBEpQthbP41cfj",
*         "_score" : 1.0033021,
*         "_source" : {
*           "price" : 8500,
*           "color" : "红色",
*           "brand" : "小米",
*           "sold_date" : "2020-05-19"
*         }
```

```
     *       },
     *       {
     *         "_index" : "tvs",
     *         "_id" : "-KouDoEBEpQthbP41cfj",
     *         "_score" : 1.0033021,
     *         "_source" : {
     *           "price" : 4800,
     *           "color" : "黑色",
     *           "brand" : "小米",
     *           "sold_date" : "2020-06-10"
     *         }
     *       }
     *     ]
     *   },
     *   "aggregations" : {
     *     "all" : {
     *       "doc_count" : 14,
     *       "all_brand_avg_price" : {
     *         "value" : 3671.4285714285716
     *       }
     *     },
     *     "single_brand_avg_price" : {
     *       "value" : 4660.0
     *     }
     *   }
     * }
     *
     * </pre>
     * <p>
     * 程序结果：
     * <pre>
     *
     * --数量：5
     * --数组数量：5
     * --最大分数：1.0033021
     * -->{color=绿色, price=3000, sold_date=2019-05-18, brand=小米}
     * -->{color=蓝色, price=2500, sold_date=2020-02-12, brand=小米}
     * -->{color=绿色, price=4500, sold_date=2020-04-22, brand=小米}
     * -->{color=红色, price=8500, sold_date=2020-05-19, brand=小米}
     * -->{color=黑色, price=4800, sold_date=2020-06-10, brand=小米}
     *
     * 聚合结果：
     *
     * ----小米销售平均价格：4660.0
     * ---所有品牌销售总数：14
     * ----所有品牌销售平均价格：3671.4285714285716
     *
     * </pre>
     *
     * @throws Exception Exception
     */
    @Test
    void aggregation7() throws Exception
    {
        //构建请求
        SearchRequest searchRequest = new SearchRequest("tvs");
        //构建请求体
        SearchSourceBuilder searchSourceBuilder = new SearchSourceBuilder();
```

```
        //查询
        searchSourceBuilder.query(QueryBuilders.termQuery("brand", "小米"));
        //分页
        //searchSourceBuilder.size(0);
        //聚合

 searchSourceBuilder.aggregation((AggregationBuilders.avg("single_brand_avg_pric
e").field("price")))
                .aggregation(AggregationBuilders.global("all")

.subAggregation(AggregationBuilders.avg("all_brand_avg_price").field("price")));

        //放入到请求中
        searchRequest.source(searchSourceBuilder);
        //发起请求
        SearchResponse searchResponse = client.search(searchRequest,
RequestOptions.DEFAULT);
        //获取数据
        //获得hits
        SearchHits hits = searchResponse.getHits();
        long value = hits.getTotalHits().value;
        float maxScore = hits.getMaxScore();
        SearchHit[] hitsHits = hits.getHits();
        System.out.println("--数量: " + value);
        System.out.println("--数组数量: " + hitsHits.length);
        System.out.println("--最大分数: " + maxScore);
        for (SearchHit hitsHit : hitsHits)
        {
            Map<String, Object> sourceAsMap = hitsHit.getSourceAsMap();
            System.out.println("-->" + sourceAsMap);
        }
        System.out.println();
        System.out.println("聚合结果: ");
        System.out.println();

        //获取aggregations部分
        Aggregations aggregations = searchResponse.getAggregations();
        //获得single_brand_avg_price
        Avg single_brand_avg_price = aggregations.get("single_brand_avg_price");
        double single_brand_avg_priceValue = single_brand_avg_price.getValue();
        System.out.println("----小米销售平均价格: " + single_brand_avg_priceValue);
        Global all = aggregations.get("all");
        long docCount = all.getDocCount();
        System.out.println("---所有品牌销售总数: " + docCount);
        Avg all_brand_avg_price =
all.getAggregations().get("all_brand_avg_price");
        double all_brand_avg_priceValue = all_brand_avg_price.getValue();
        System.out.println("----所有品牌销售平均价格: " + all_brand_avg_priceValue);
    }
```

统计价格大于1200的电视平均价格

```
/**
     * 统计价格大于1200的电视平均价格
     * <p>
     * 请求内容:
     * <pre>
     *
     * GET /tvs/_search
     * {
     *   "query":
     *   {
     *     "constant_score":
     *     {
     *       "filter":
     *       {
     *         "range":
     *         {
     *           "price":
     *           {
     *             "gte": 1200
     *           }
     *         }
     *       }
     *     }
     *   },
     *   "aggs":
     *   {
     *     "avg_price":
     *     {
     *       "avg":
     *       {
     *         "field": "price"
     *       }
     *     }
     *   }
     *
     * }
     *
     * </pre>
     * <p>
     * 结果:
     * <pre>
     *
     * {
     *   "took" : 0,
     *   "timed_out" : false,
     *   "_shards" : {
     *     "total" : 1,
     *     "successful" : 1,
     *     "skipped" : 0,
     *     "failed" : 0
     *   },
     *   "hits" : {
     *     "total" : {
     *       "value" : 13,
     *       "relation" : "eq"
```

```
*     },
*     "max_score" : 1.0,
*     "hits" : [
*       {
*         "_index" : "tvs",
*         "_id" : "7KouDoEBEpQthbP41cfj",
*         "_score" : 1.0,
*         "_source" : {
*           "price" : 2000,
*           "color" : "红色",
*           "brand" : "长虹",
*           "sold_date" : "2019-11-05"
*         }
*       },
*       {
*         "_index" : "tvs",
*         "_id" : "7aouDoEBEpQthbP41cfj",
*         "_score" : 1.0,
*         "_source" : {
*           "price" : 3000,
*           "color" : "绿色",
*           "brand" : "小米",
*           "sold_date" : "2019-05-18"
*         }
*       },
*       {
*         "_index" : "tvs",
*         "_id" : "7qouDoEBEpQthbP41cfj",
*         "_score" : 1.0,
*         "_source" : {
*           "price" : 1500,
*           "color" : "蓝色",
*           "brand" : "TCL",
*           "sold_date" : "2019-07-02"
*         }
*       },
*       {
*         "_index" : "tvs",
*         "_id" : "76ouDoEBEpQthbP41cfj",
*         "_score" : 1.0,
*         "_source" : {
*           "price" : 1200,
*           "color" : "绿色",
*           "brand" : "TCL",
*           "sold_date" : "2019-08-19"
*         }
*       },
*       {
*         "_index" : "tvs",
*         "_id" : "8KouDoEBEpQthbP41cfj",
*         "_score" : 1.0,
*         "_source" : {
*           "price" : 2000,
*           "color" : "红色",
*           "brand" : "长虹",
*           "sold_date" : "2019-11-05"
*         }
*       },
```

```
*       {
*         "_index" : "tvs",
*         "_id" : "8aouDoEBEpQthbP41cfj",
*         "_score" : 1.0,
*         "_source" : {
*           "price" : 8000,
*           "color" : "红色",
*           "brand" : "三星",
*           "sold_date" : "2020-01-01"
*         }
*       },
*       {
*         "_index" : "tvs",
*         "_id" : "8qouDoEBEpQthbP41cfj",
*         "_score" : 1.0,
*         "_source" : {
*           "price" : 2500,
*           "color" : "蓝色",
*           "brand" : "小米",
*           "sold_date" : "2020-02-12"
*         }
*       },
*       {
*         "_index" : "tvs",
*         "_id" : "86ouDoEBEpQthbP41cfj",
*         "_score" : 1.0,
*         "_source" : {
*           "price" : 4500,
*           "color" : "绿色",
*           "brand" : "小米",
*           "sold_date" : "2020-04-22"
*         }
*       },
*       {
*         "_index" : "tvs",
*         "_id" : "9KouDoEBEpQthbP41cfj",
*         "_score" : 1.0,
*         "_source" : {
*           "price" : 6100,
*           "color" : "蓝色",
*           "brand" : "三星",
*           "sold_date" : "2020-05-16"
*         }
*       },
*       {
*         "_index" : "tvs",
*         "_id" : "9aouDoEBEpQthbP41cfj",
*         "_score" : 1.0,
*         "_source" : {
*           "price" : 2100,
*           "color" : "白色",
*           "brand" : "TCL",
*           "sold_date" : "2020-05-17"
*         }
*       }
*     ]
*   },
*   "aggregations" : {
```

```
     *     "avg_price" : {
     *       "value" : 3876.923076923077
     *     }
     *   }
     * }
     *
     * </pre>
     * <p>
     * 程序结果：
     * <pre>
     *
     * --数量：13
     * --数组数量：10
     * --最大分数：1.0
     * -->{color=红色, price=2000, sold_date=2019-11-05, brand=长虹}
     * -->{color=绿色, price=3000, sold_date=2019-05-18, brand=小米}
     * -->{color=蓝色, price=1500, sold_date=2019-07-02, brand=TCL}
     * -->{color=绿色, price=1200, sold_date=2019-08-19, brand=TCL}
     * -->{color=红色, price=2000, sold_date=2019-11-05, brand=长虹}
     * -->{color=红色, price=8000, sold_date=2020-01-01, brand=三星}
     * -->{color=蓝色, price=2500, sold_date=2020-02-12, brand=小米}
     * -->{color=绿色, price=4500, sold_date=2020-04-22, brand=小米}
     * -->{color=蓝色, price=6100, sold_date=2020-05-16, brand=三星}
     * -->{color=白色, price=2100, sold_date=2020-05-17, brand=TCL}
     *
     * 聚合结果：
     *
     * 大于1200元的平均价格：3876.923076923077
     *
     * </pre>
     *
     * @throws Exception Exception
     */
    @Test
    void aggregation8() throws Exception
    {
        //构建请求
        SearchRequest searchRequest = new SearchRequest("tvs");
        //构建请求体
        SearchSourceBuilder searchSourceBuilder = new SearchSourceBuilder();
        //查询

 searchSourceBuilder.query(QueryBuilders.constantScoreQuery(QueryBuilders.rangeQ
uery("price").gte(1200)));
        //分页
        //searchSourceBuilder.size(0);
        //聚合
        searchSourceBuilder.aggregation(
                AggregationBuilders.avg("avg_price").field("price")
        );

        //放入到请求中
        searchRequest.source(searchSourceBuilder);
        //发起请求
        SearchResponse searchResponse = client.search(searchRequest,
RequestOptions.DEFAULT);
        //获取数据
        //获得hits
```

```
        SearchHits hits = searchResponse.getHits();
        long value = hits.getTotalHits().value;
        float maxScore = hits.getMaxScore();
        SearchHit[] hitsHits = hits.getHits();
        System.out.println("--数量：" + value);
        System.out.println("--数组数量：" + hitsHits.length);
        System.out.println("--最大分数：" + maxScore);
        for (SearchHit hitsHit : hitsHits)
        {
            Map<String, Object> sourceAsMap = hitsHit.getSourceAsMap();
            System.out.println("-->" + sourceAsMap);
        }
        System.out.println();
        System.out.println("聚合结果：");
        System.out.println();

        //获取aggregations部分
        Aggregations aggregations = searchResponse.getAggregations();
        //获得avg_price
        Avg avg_price = aggregations.get("avg_price");
        double avg_priceValue = avg_price.getValue();
        System.out.println("大于1200元的平均价格：" + avg_priceValue);
    }
```

按每种颜色的平均销售额降序排序

```
/**
     * 按每种颜色的平均销售额降序排序
     * <p>
     * 请求内容:
     * <pre>
     *
     * GET /tvs/_search
     * {
     *   "query":
     *   {
     *     "match_all": {}
     *   },
     *   "size": 0,
     *   "aggs":
     *   {
     *     "group_by_color":
     *     {
     *       "terms":
     *       {
     *         "field": "color",
     *         "order":
     *         {
     *           "avg_price": "desc"
```

```
*         }
*       },
*       "aggs":
*       {
*         "avg_price":
*         {
*           "avg":
*           {
*             "field": "price"
*           }
*         }
*       }
*     }
*   }
* }
*
* </pre>
* <p>
* 结果:
* <pre>
*
* {
*   "took" : 1,
*   "timed_out" : false,
*   "_shards" : {
*     "total" : 1,
*     "successful" : 1,
*     "skipped" : 0,
*     "failed" : 0
*   },
*   "hits" : {
*     "total" : {
*       "value" : 14,
*       "relation" : "eq"
*     },
*     "max_score" : null,
*     "hits" : [ ]
*   },
*   "aggregations" : {
*     "group_by_color" : {
*       "doc_count_error_upper_bound" : 0,
*       "sum_other_doc_count" : 0,
*       "buckets" : [
*         {
*           "key" : "黑色",
*           "doc_count" : 1,
*           "avg_price" : {
*             "value" : 4800.0
*           }
*         },
*         {
*           "key" : "红色",
*           "doc_count" : 5,
*           "avg_price" : {
*             "value" : 4300.0
*           }
*         },
*         {
```

```
     *            "key" : "蓝色",
     *            "doc_count" : 4,
     *            "avg_price" : {
     *              "value" : 3575.0
     *            }
     *          },
     *          {
     *            "key" : "绿色",
     *            "doc_count" : 3,
     *            "avg_price" : {
     *              "value" : 2900.0
     *            }
     *          },
     *          {
     *            "key" : "白色",
     *            "doc_count" : 1,
     *            "avg_price" : {
     *              "value" : 2100.0
     *            }
     *          }
     *        ]
     *      }
     *    }
     * }
     *
     * </pre>
     * <p>
     * 程序结果:
     * <pre>
     *
     * ----key: 黑色
     * ----doc_count: 1
     * ----avg_price: 4800.0
     * ----------------------------------------
     * ----key: 红色
     * ----doc_count: 5
     * ----avg_price: 4300.0
     * ----------------------------------------
     * ----key: 蓝色
     * ----doc_count: 4
     * ----avg_price: 3575.0
     * ----------------------------------------
     * ----key: 绿色
     * ----doc_count: 3
     * ----avg_price: 2900.0
     * ----------------------------------------
     * ----key: 白色
     * ----doc_count: 1
     * ----avg_price: 2100.0
     * ----------------------------------------
     *
     * </pre>
     *
     * @throws Exception Exception
     */
    @Test
    void aggregation9() throws Exception
    {
```

```
        //构建请求
        SearchRequest searchRequest = new SearchRequest("tvs");
        //构建请求体
        SearchSourceBuilder searchSourceBuilder = new SearchSourceBuilder();
        //查询
        searchSourceBuilder.query(QueryBuilders.matchAllQuery());
        //分页
        searchSourceBuilder.size(0);
        //聚合
        searchSourceBuilder.aggregation(
                AggregationBuilders.terms("group_by_color").field("color")
                        .order(BucketOrder.aggregation("avg_price", false))

.subAggregation(AggregationBuilders.avg("avg_price").field("price"))
        );

        //放入到请求中
        searchRequest.source(searchSourceBuilder);
        //发起请求
        SearchResponse searchResponse = client.search(searchRequest,
RequestOptions.DEFAULT);
        //获取数据
        //获取aggregations部分
        Aggregations aggregations = searchResponse.getAggregations();
        //获得group_by_color
        Terms group_by_color = aggregations.get("group_by_color");
        //获取buckets部分
        List<? extends Terms.Bucket> buckets = group_by_color.getBuckets();
        //遍历
        for (Terms.Bucket bucket : buckets)
        {
            //获取数据
            String key = (String) bucket.getKey();
            long docCount = bucket.getDocCount();
            Avg avg_price = bucket.getAggregations().get("avg_price");
            double avg_priceValue = avg_price.getValue();
            //打印
            System.out.println("----key: " + key);
            System.out.println("----doc_count: " + docCount);
            System.out.println("----avg_price: " + avg_priceValue);
            System.out.println("----------------------------------------");
        }
    }
```

按每种颜色的每种品牌平均销售额降序排序

```
/**
     * 按每种颜色的每种品牌平均销售额降序排序
     * <p>
     * 请求内容:
     * <pre>
```

```
 *
 * GET /tvs/_search
 * {
 *   "query":
 *   {
 *     "match_all": {}
 *   },
 *   "size": 0,
 *   "aggs":
 *   {
 *     "group_by_color":
 *     {
 *       "terms":
 *       {
 *         "field": "color"
 *
 *       },
 *       "aggs":
 *       {
 *         "group_by_brand":
 *         {
 *           "terms":
 *           {
 *             "field": "brand",
 *             "order":
 *             {
 *               "avg_price": "desc"
 *             }
 *           },
 *           "aggs":
 *           {
 *             "avg_price":
 *             {
 *               "avg":
 *               {
 *                 "field": "price"
 *               }
 *             }
 *           }
 *         }
 *       }
 *     }
 *   }
 * }
 *
 * </pre>
 * <p>
 * 结果:
 * <pre>
 *
 * {
 *   "took" : 2,
 *   "timed_out" : false,
 *   "_shards" : {
 *     "total" : 1,
 *     "successful" : 1,
 *     "skipped" : 0,
 *     "failed" : 0
```

```
*   },
*   "hits" : {
*     "total" : {
*       "value" : 14,
*       "relation" : "eq"
*     },
*     "max_score" : null,
*     "hits" : [ ]
*   },
*   "aggregations" : {
*     "group_by_color" : {
*       "doc_count_error_upper_bound" : 0,
*       "sum_other_doc_count" : 0,
*       "buckets" : [
*         {
*           "key" : "红色",
*           "doc_count" : 5,
*           "group_by_brand" : {
*             "doc_count_error_upper_bound" : 0,
*             "sum_other_doc_count" : 0,
*             "buckets" : [
*               {
*                 "key" : "小米",
*                 "doc_count" : 1,
*                 "avg_price" : {
*                   "value" : 8500.0
*                 }
*               },
*               {
*                 "key" : "三星",
*                 "doc_count" : 1,
*                 "avg_price" : {
*                   "value" : 8000.0
*                 }
*               },
*               {
*                 "key" : "长虹",
*                 "doc_count" : 3,
*                 "avg_price" : {
*                   "value" : 1666.6666666666667
*                 }
*               }
*             ]
*           }
*         },
*         {
*           "key" : "蓝色",
*           "doc_count" : 4,
*           "group_by_brand" : {
*             "doc_count_error_upper_bound" : 0,
*             "sum_other_doc_count" : 0,
*             "buckets" : [
*               {
*                 "key" : "三星",
*                 "doc_count" : 1,
*                 "avg_price" : {
*                   "value" : 6100.0
*                 }
```

```
*                },
*                {
*                  "key" : "长虹",
*                  "doc_count" : 1,
*                  "avg_price" : {
*                    "value" : 4200.0
*                  }
*                },
*                {
*                  "key" : "小米",
*                  "doc_count" : 1,
*                  "avg_price" : {
*                    "value" : 2500.0
*                  }
*                },
*                {
*                  "key" : "TCL",
*                  "doc_count" : 1,
*                  "avg_price" : {
*                    "value" : 1500.0
*                  }
*                }
*              ]
*            }
*          },
*          {
*            "key" : "绿色",
*            "doc_count" : 3,
*            "group_by_brand" : {
*              "doc_count_error_upper_bound" : 0,
*              "sum_other_doc_count" : 0,
*              "buckets" : [
*                {
*                  "key" : "小米",
*                  "doc_count" : 2,
*                  "avg_price" : {
*                    "value" : 3750.0
*                  }
*                },
*                {
*                  "key" : "TCL",
*                  "doc_count" : 1,
*                  "avg_price" : {
*                    "value" : 1200.0
*                  }
*                }
*              ]
*            }
*          },
*          {
*            "key" : "白色",
*            "doc_count" : 1,
*            "group_by_brand" : {
*              "doc_count_error_upper_bound" : 0,
*              "sum_other_doc_count" : 0,
*              "buckets" : [
*                {
*                  "key" : "TCL",
```

```
 *                  "doc_count" : 1,
 *                  "avg_price" : {
 *                    "value" : 2100.0
 *                  }
 *                }
 *              ]
 *            }
 *          },
 *          {
 *            "key" : "黑色",
 *            "doc_count" : 1,
 *            "group_by_brand" : {
 *              "doc_count_error_upper_bound" : 0,
 *              "sum_other_doc_count" : 0,
 *              "buckets" : [
 *                {
 *                  "key" : "小米",
 *                  "doc_count" : 1,
 *                  "avg_price" : {
 *                    "value" : 4800.0
 *                  }
 *                }
 *              ]
 *            }
 *          }
 *        ]
 *      }
 *    }
 * }
 *
 * </pre>
 * <p>
 * 程序结果：
 * <pre>
 *
 * ----key：红色
 * ----doc_count：5
 * ----group_by_brand：
 * --------key：小米
 * --------doc_count：1
 * --------avg_price：8500.0
 * ------------------------------
 * --------key：三星
 * --------doc_count：1
 * --------avg_price：8000.0
 * ------------------------------
 * --------key：长虹
 * --------doc_count：3
 * --------avg_price：1666.6666666666667
 * ------------------------------
 * ----------------------------------------
 * ----key：蓝色
 * ----doc_count：4
 * ----group_by_brand：
 * --------key：三星
 * --------doc_count：1
 * --------avg_price：6100.0
 * ------------------------------
```

```
     * --------key：长虹
     * --------doc_count：1
     * --------avg_price：4200.0
     * ------------------------------
     * --------key：小米
     * --------doc_count：1
     * --------avg_price：2500.0
     * ------------------------------
     * --------key：TCL
     * --------doc_count：1
     * --------avg_price：1500.0
     * ------------------------------
     * ----------------------------------------
     * ----key：绿色
     * ----doc_count：3
     * ----group_by_brand：
     * --------key：小米
     * --------doc_count：2
     * --------avg_price：3750.0
     * ------------------------------
     * --------key：TCL
     * --------doc_count：1
     * --------avg_price：1200.0
     * ------------------------------
     * ----------------------------------------
     * ----key：白色
     * ----doc_count：1
     * ----group_by_brand：
     * --------key：TCL
     * --------doc_count：1
     * --------avg_price：2100.0
     * ------------------------------
     * ----------------------------------------
     * ----key：黑色
     * ----doc_count：1
     * ----group_by_brand：
     * --------key：小米
     * --------doc_count：1
     * --------avg_price：4800.0
     * ------------------------------
     * ----------------------------------------
     *
     * </pre>
     *
     * @throws Exception Exception
     */
    @Test
    void aggregation10() throws Exception
    {
        //构建请求
        SearchRequest searchRequest = new SearchRequest("tvs");
        //构建请求体
        SearchSourceBuilder searchSourceBuilder = new SearchSourceBuilder();
        //查询
        searchSourceBuilder.query(QueryBuilders.matchAllQuery());
        //分页
        searchSourceBuilder.size(0);
        //聚合
```

```
        searchSourceBuilder.aggregation(
                AggregationBuilders.terms("group_by_color").field("color")

.subAggregation(AggregationBuilders.terms("group_by_brand").field("brand")
                                .order(BucketOrder.aggregation("avg_price",
false))

.subAggregation(AggregationBuilders.avg("avg_price").field("price")))
        );

        //放入到请求中
        searchRequest.source(searchSourceBuilder);
        //发起请求
        SearchResponse searchResponse = client.search(searchRequest,
RequestOptions.DEFAULT);
        //获取数据
        //获取aggregations部分
        Aggregations aggregations = searchResponse.getAggregations();
        //获得group_by_color
        Terms group_by_color = aggregations.get("group_by_color");
        //获取buckets部分
        List<? extends Terms.Bucket> buckets = group_by_color.getBuckets();
        //遍历
        for (Terms.Bucket bucket : buckets)
        {
            //获取数据
            String key = (String) bucket.getKey();
            long docCount = bucket.getDocCount();
            Terms group_by_brand =
bucket.getAggregations().get("group_by_brand");
            List<? extends Terms.Bucket> buckets1 = group_by_brand.getBuckets();
            //打印
            System.out.println("----key: " + key);
            System.out.println("----doc_count: " + docCount);
            System.out.println("----group_by_brand: ");
            for (Terms.Bucket bucket1 : buckets1)
            {
                //获取数据
                //获取数据
                String key1 = (String) bucket1.getKey();
                long docCount1 = bucket1.getDocCount();
                Avg avg_price1 = bucket1.getAggregations().get("avg_price");
                double avgPrice1Value = avg_price1.getValue();
                //打印
                System.out.println("--------key: " + key1);
                System.out.println("--------doc_count: " + docCount1);
                System.out.println("--------avg_price: " + avgPrice1Value);
                System.out.println("------------------------------");
            }
            System.out.println("----------------------------------------");
        }
    }
```

完整代码

项目地址：https://github.com/maomao124/elasticsearch_Implement_aggregation.git

```
package mao.elasticsearch_implement_aggregation;

import org.apache.http.HttpHost;
import org.elasticsearch.action.search.SearchRequest;
import org.elasticsearch.action.search.SearchResponse;
import org.elasticsearch.client.RequestOptions;
import org.elasticsearch.client.RestClient;
import org.elasticsearch.client.RestHighLevelClient;
import org.elasticsearch.index.query.QueryBuilders;
import org.elasticsearch.search.SearchHit;
import org.elasticsearch.search.SearchHits;
import org.elasticsearch.search.aggregations.*;
import org.elasticsearch.search.aggregations.bucket.global.Global;
import org.elasticsearch.search.aggregations.bucket.histogram.Histogram;
import org.elasticsearch.search.aggregations.bucket.terms.Terms;
import org.elasticsearch.search.aggregations.metrics.Avg;
import org.elasticsearch.search.aggregations.metrics.Max;
import org.elasticsearch.search.aggregations.metrics.Min;
import org.elasticsearch.search.aggregations.metrics.Sum;
import org.elasticsearch.search.builder.SearchSourceBuilder;
import org.junit.jupiter.api.AfterAll;
import org.junit.jupiter.api.BeforeAll;
import org.junit.jupiter.api.Test;
import org.springframework.boot.test.context.SpringBootTest;

import java.io.IOException;
import java.util.List;
import java.util.Map;


/**
 * Project name(项目名称)：elasticsearch_Implement_aggregation
 * Package(包名): mao.elasticsearch_implement_aggregation
 * Class(类名): ElasticSearchTest
 * Author(作者）: mao
 * Author QQ：1296193245
 * GitHub：https://github.com/maomao124/
 * Date(创建日期)： 2022/5/29
 * Time(创建时间)： 21:03
 * Version(版本): 1.0
 * Description(描述)： SpringBootTest
 * <p>
 * <p>
 * 电视案例
 * 索引：
 * <pre>
 *
 * {
 *   "tvs" : {
 *     "aliases" : { },
 *     "mappings" : {
```

```
*       "properties" : {
*         "brand" : {
*           "type" : "keyword"
*         },
*         "color" : {
*           "type" : "keyword"
*         },
*         "price" : {
*           "type" : "long"
*         },
*         "sold_date" : {
*           "type" : "date"
*         }
*       }
*     },
*     "settings" : {
*       "index" : {
*         "routing" : {
*           "allocation" : {
*             "include" : {
*               "_tier_preference" : "data_content"
*             }
*           }
*         },
*         "number_of_shards" : "1",
*         "provided_name" : "tvs",
*         "creation_date" : "1653799931947",
*         "number_of_replicas" : "1",
*         "uuid" : "UPq3NuVHTlWq1r9GrCj4fw",
*         "version" : {
*           "created" : "8010399"
*         }
*       }
*     }
*   }
* }
*
* </pre>
* <p>
* 数据:
* <pre>
*
* {
*   "took" : 0,
*   "timed_out" : false,
*   "_shards" : {
*     "total" : 1,
*     "successful" : 1,
*     "skipped" : 0,
*     "failed" : 0
*   },
*   "hits" : {
*     "total" : {
*       "value" : 14,
*       "relation" : "eq"
*     },
*     "max_score" : 1.0,
*     "hits" : [
```

```
*        {
*          "_index" : "tvs",
*          "_id" : "66ouDoEBEpQthbP41cfj",
*          "_score" : 1.0,
*          "_source" : {
*            "price" : 1000,
*            "color" : "红色",
*            "brand" : "长虹",
*            "sold_date" : "2019-10-28"
*          }
*        },
*        {
*          "_index" : "tvs",
*          "_id" : "7KouDoEBEpQthbP41cfj",
*          "_score" : 1.0,
*          "_source" : {
*            "price" : 2000,
*            "color" : "红色",
*            "brand" : "长虹",
*            "sold_date" : "2019-11-05"
*          }
*        },
*        {
*          "_index" : "tvs",
*          "_id" : "7aouDoEBEpQthbP41cfj",
*          "_score" : 1.0,
*          "_source" : {
*            "price" : 3000,
*            "color" : "绿色",
*            "brand" : "小米",
*            "sold_date" : "2019-05-18"
*          }
*        },
*        {
*          "_index" : "tvs",
*          "_id" : "7qouDoEBEpQthbP41cfj",
*          "_score" : 1.0,
*          "_source" : {
*            "price" : 1500,
*            "color" : "蓝色",
*            "brand" : "TCL",
*            "sold_date" : "2019-07-02"
*          }
*        },
*        {
*          "_index" : "tvs",
*          "_id" : "76ouDoEBEpQthbP41cfj",
*          "_score" : 1.0,
*          "_source" : {
*            "price" : 1200,
*            "color" : "绿色",
*            "brand" : "TCL",
*            "sold_date" : "2019-08-19"
*          }
*        },
*        {
*          "_index" : "tvs",
*          "_id" : "8KouDoEBEpQthbP41cfj",
```

```
*         "_score" : 1.0,
*         "_source" : {
*           "price" : 2000,
*           "color" : "红色",
*           "brand" : "长虹",
*           "sold_date" : "2019-11-05"
*         }
*       },
*       {
*         "_index" : "tvs",
*         "_id" : "8aouDoEBEpQthbP41cfj",
*         "_score" : 1.0,
*         "_source" : {
*           "price" : 8000,
*           "color" : "红色",
*           "brand" : "三星",
*           "sold_date" : "2020-01-01"
*         }
*       },
*       {
*         "_index" : "tvs",
*         "_id" : "8qouDoEBEpQthbP41cfj",
*         "_score" : 1.0,
*         "_source" : {
*           "price" : 2500,
*           "color" : "蓝色",
*           "brand" : "小米",
*           "sold_date" : "2020-02-12"
*         }
*       },
*       {
*         "_index" : "tvs",
*         "_id" : "86ouDoEBEpQthbP41cfj",
*         "_score" : 1.0,
*         "_source" : {
*           "price" : 4500,
*           "color" : "绿色",
*           "brand" : "小米",
*           "sold_date" : "2020-04-22"
*         }
*       },
*       {
*         "_index" : "tvs",
*         "_id" : "9KouDoEBEpQthbP41cfj",
*         "_score" : 1.0,
*         "_source" : {
*           "price" : 6100,
*           "color" : "蓝色",
*           "brand" : "三星",
*           "sold_date" : "2020-05-16"
*         }
*       }
*     ]
*   }
* }
*
* </pre>
*/
```

```
@SpringBootTest
public class ElasticSearchTest
{

    private static RestHighLevelClient client;

    /**
     * Before all.
     */
    @BeforeAll
    static void beforeAll()
    {
        client = new RestHighLevelClient(RestClient.builder(
                new HttpHost("localhost", 9200, "http")
        ));
    }

    /**
     * After all.
     *
     * @throws IOException the io exception
     */
    @AfterAll
    static void afterAll() throws IOException
    {
        client.close();
    }

    /**
     * Ping.
     *
     * @throws IOException the io exception
     */
    @Test
    void ping() throws IOException
    {
        boolean ping = client.ping(RequestOptions.DEFAULT);
        System.out.println(ping);
    }


    /**
     * 统计哪种颜色的电视销量最高
     * <p>
     * 请求内容：
     * <pre>
     *
     * GET /tvs/_search
     * {
     *   "query":
     *   {
     *     "match_all": {}
     *   },
     *   "size": 0,
     *   "aggs":
     *   {
     *     "popular_colors":
```

```
*     {
*       "terms":
*       {
*         "field": "color"
*       }
*     }
*   }
* }
*
* </pre>
* <p>
* 结果:
* <pre>
*
* {
*   "took" : 12,
*   "timed_out" : false,
*   "_shards" : {
*     "total" : 1,
*     "successful" : 1,
*     "skipped" : 0,
*     "failed" : 0
*   },
*   "hits" : {
*     "total" : {
*       "value" : 14,
*       "relation" : "eq"
*     },
*     "max_score" : null,
*     "hits" : [ ]
*   },
*   "aggregations" : {
*     "popular_colors" : {
*       "doc_count_error_upper_bound" : 0,
*       "sum_other_doc_count" : 0,
*       "buckets" : [
*         {
*           "key" : "红色",
*           "doc_count" : 5
*         },
*         {
*           "key" : "蓝色",
*           "doc_count" : 4
*         },
*         {
*           "key" : "绿色",
*           "doc_count" : 3
*         },
*         {
*           "key" : "白色",
*           "doc_count" : 1
*         },
*         {
*           "key" : "黑色",
*           "doc_count" : 1
*         }
*       ]
*     }
```

```
     *   }
     * }
     *
     * </pre>
     * <p>
     * 程序结果：
     * <pre>
     *
     * ----key：红色
     * ----doc_count：5
     * ------------------------------------
     * ----key：蓝色
     * ----doc_count：4
     * ------------------------------------
     * ----key：绿色
     * ----doc_count：3
     * ------------------------------------
     * ----key：白色
     * ----doc_count：1
     * ------------------------------------
     * ----key：黑色
     * ----doc_count：1
     * ------------------------------------
     *
     * </pre>
     *
     * @throws Exception Exception
     */
    @Test
    void aggregation1() throws Exception
    {
        //构建请求
        SearchRequest searchRequest = new SearchRequest("tvs");
        //构建请求体
        SearchSourceBuilder searchSourceBuilder = new SearchSourceBuilder();
        //查询
        searchSourceBuilder.query(QueryBuilders.matchAllQuery());
        //分页
        searchSourceBuilder.size(0);
        //聚合

 searchSourceBuilder.aggregation(AggregationBuilders.terms("popular_colors").fie
ld("color"));
        //放入到请求中
        searchRequest.source(searchSourceBuilder);
        //发起请求
        SearchResponse searchResponse = client.search(searchRequest,
RequestOptions.DEFAULT);
        //获取数据
        //获取aggregations部分
        Aggregations aggregations = searchResponse.getAggregations();
        //获得popular_colors
        Terms popular_colors = aggregations.get("popular_colors");
        //获取buckets部分
        List<? extends Terms.Bucket> buckets = popular_colors.getBuckets();
        //遍历
        for (Terms.Bucket bucket : buckets)
        {
```

```
            //获取数据
            String key = (String) bucket.getKey();
            long docCount = bucket.getDocCount();
            //打印
            System.out.println("----key: " + key);
            System.out.println("----doc_count: " + docCount);
            System.out.println("----------------------------------------");
        }
    }


    /**
     * 统计每种颜色电视平均价格
     * <p>
     * 请求内容:
     * <pre>
     *
     * GET /tvs/_search
     * {
     *   "query":
     *   {
     *     "match_all": {}
     *   },
     *   "size": 0,
     *   "aggs":
     *   {
     *     "group_by_colors":
     *     {
     *       "terms":
     *       {
     *         "field": "color"
     *       },
     *       "aggs": {
     *         "avg_price":
     *         {
     *           "avg":
     *           {
     *             "field": "price"
     *           }
     *         }
     *       }
     *     }
     *   }
     * }
     *
     *
     *
     * </pre>
     * <p>
     * 结果:
     * <pre>
     *
     * {
     *   "took" : 1,
     *   "timed_out" : false,
     *   "_shards" : {
     *     "total" : 1,
     *     "successful" : 1,
```

```
 *     "skipped" : 0,
 *     "failed" : 0
 *   },
 *   "hits" : {
 *     "total" : {
 *       "value" : 14,
 *       "relation" : "eq"
 *     },
 *     "max_score" : null,
 *     "hits" : [ ]
 *   },
 *   "aggregations" : {
 *     "group_by_colors" : {
 *       "doc_count_error_upper_bound" : 0,
 *       "sum_other_doc_count" : 0,
 *       "buckets" : [
 *         {
 *           "key" : "红色",
 *           "doc_count" : 5,
 *           "avg_price" : {
 *             "value" : 4300.0
 *           }
 *         },
 *         {
 *           "key" : "蓝色",
 *           "doc_count" : 4,
 *           "avg_price" : {
 *             "value" : 3575.0
 *           }
 *         },
 *         {
 *           "key" : "绿色",
 *           "doc_count" : 3,
 *           "avg_price" : {
 *             "value" : 2900.0
 *           }
 *         },
 *         {
 *           "key" : "白色",
 *           "doc_count" : 1,
 *           "avg_price" : {
 *             "value" : 2100.0
 *           }
 *         },
 *         {
 *           "key" : "黑色",
 *           "doc_count" : 1,
 *           "avg_price" : {
 *             "value" : 4800.0
 *           }
 *         }
 *       ]
 *     }
 *   }
 * }
 *
 * </pre>
 * <p>
```

```
     * 程序结果：
     * <pre>
     *
     * ----key：红色
     * ----doc_count：5
     * ----平均价格：4300.0
     * ----------------------------------------
     * ----key：蓝色
     * ----doc_count：4
     * ----平均价格：3575.0
     * ----------------------------------------
     * ----key：绿色
     * ----doc_count：3
     * ----平均价格：2900.0
     * ----------------------------------------
     * ----key：白色
     * ----doc_count：1
     * ----平均价格：2100.0
     * ----------------------------------------
     * ----key：黑色
     * ----doc_count：1
     * ----平均价格：4800.0
     * ----------------------------------------
     *
     * </pre>
     *
     * @throws Exception Exception
     */
    @Test
    void aggregation2() throws Exception
    {
        //构建请求
        SearchRequest searchRequest = new SearchRequest("tvs");
        //构建请求体
        SearchSourceBuilder searchSourceBuilder = new SearchSourceBuilder();
        //查询
        searchSourceBuilder.query(QueryBuilders.matchAllQuery());
        //分页
        searchSourceBuilder.size(0);
        //聚合

 searchSourceBuilder.aggregation(AggregationBuilders.terms("group_by_colors").fi
eld("color")

.subAggregation(AggregationBuilders.avg("avg_price").field("price")));
        //放入到请求中
        searchRequest.source(searchSourceBuilder);
        //发起请求
        SearchResponse searchResponse = client.search(searchRequest,
RequestOptions.DEFAULT);
        //获取数据
        //获取aggregations部分
        Aggregations aggregations = searchResponse.getAggregations();
        //获得group_by_colors
        Terms group_by_colors = aggregations.get("group_by_colors");
        //获取buckets部分
        List<? extends Terms.Bucket> buckets = group_by_colors.getBuckets();
        //遍历
```

```
        for (Terms.Bucket bucket : buckets)
        {
            //获取数据
            String key = (String) bucket.getKey();
            long docCount = bucket.getDocCount();
            Avg avg_price = bucket.getAggregations().get("avg_price");
            double avgPriceValue = avg_price.getValue();
            //打印
            System.out.println("----key: " + key);
            System.out.println("----doc_count: " + docCount);
            System.out.println("----平均价格: " + avgPriceValue);
            System.out.println("----------------------------------------");
        }
    }


    /**
     * 统计每个颜色下，平均价格及每个颜色下，每个品牌的平均价格
     * <p>
     * 请求内容:
     * <pre>
     *
     * GET /tvs/_search
     * {
     *   "query":
     *   {
     *     "match_all": {}
     *   },
     *   "size": 0,
     *   "aggs":
     *   {
     *     "group_by_colors":
     *     {
     *       "terms":
     *       {
     *         "field": "color"
     *       },
     *       "aggs":
     *       {
     *         "avg_price":
     *         {
     *           "avg":
     *           {
     *             "field": "price"
     *           }
     *         },
     *         "group_by_brand":
     *         {
     *           "terms":
     *           {
     *             "field": "brand"
     *           },
     *           "aggs":
     *           {
     *             "avg_price":
     *             {
     *               "avg":
     *               {
```

```
*                 "field": "price"
*               }
*             }
*           }
*         }
*       }
*     }
*   }
* }
*
* </pre>
* <p>
* 结果:
* <pre>
*
* {
*   "took" : 1,
*   "timed_out" : false,
*   "_shards" : {
*     "total" : 1,
*     "successful" : 1,
*     "skipped" : 0,
*     "failed" : 0
*   },
*   "hits" : {
*     "total" : {
*       "value" : 14,
*       "relation" : "eq"
*     },
*     "max_score" : null,
*     "hits" : [ ]
*   },
*   "aggregations" : {
*     "group_by_colors" : {
*       "doc_count_error_upper_bound" : 0,
*       "sum_other_doc_count" : 0,
*       "buckets" : [
*         {
*           "key" : "红色",
*           "doc_count" : 5,
*           "avg_price" : {
*             "value" : 4300.0
*           },
*           "group_by_brand" : {
*             "doc_count_error_upper_bound" : 0,
*             "sum_other_doc_count" : 0,
*             "buckets" : [
*               {
*                 "key" : "长虹",
*                 "doc_count" : 3,
*                 "avg_price" : {
*                   "value" : 1666.6666666666667
*                 }
*               },
*               {
*                 "key" : "三星",
*                 "doc_count" : 1,
*                 "avg_price" : {
```

```
*                   "value" : 8000.0
*                 }
*               },
*               {
*                 "key" : "小米",
*                 "doc_count" : 1,
*                 "avg_price" : {
*                   "value" : 8500.0
*                 }
*               }
*             ]
*           }
*         },
*         {
*           "key" : "蓝色",
*           "doc_count" : 4,
*           "avg_price" : {
*             "value" : 3575.0
*           },
*           "group_by_brand" : {
*             "doc_count_error_upper_bound" : 0,
*             "sum_other_doc_count" : 0,
*             "buckets" : [
*               {
*                 "key" : "TCL",
*                 "doc_count" : 1,
*                 "avg_price" : {
*                   "value" : 1500.0
*                 }
*               },
*               {
*                 "key" : "三星",
*                 "doc_count" : 1,
*                 "avg_price" : {
*                   "value" : 6100.0
*                 }
*               },
*               {
*                 "key" : "小米",
*                 "doc_count" : 1,
*                 "avg_price" : {
*                   "value" : 2500.0
*                 }
*               },
*               {
*                 "key" : "长虹",
*                 "doc_count" : 1,
*                 "avg_price" : {
*                   "value" : 4200.0
*                 }
*               }
*             ]
*           }
*         },
*         {
*           "key" : "绿色",
*           "doc_count" : 3,
*           "avg_price" : {
```

```
*              "value" : 2900.0
*            },
*            "group_by_brand" : {
*              "doc_count_error_upper_bound" : 0,
*              "sum_other_doc_count" : 0,
*              "buckets" : [
*                {
*                  "key" : "小米",
*                  "doc_count" : 2,
*                  "avg_price" : {
*                    "value" : 3750.0
*                  }
*                },
*                {
*                  "key" : "TCL",
*                  "doc_count" : 1,
*                  "avg_price" : {
*                    "value" : 1200.0
*                  }
*                }
*              ]
*            }
*          },
*          {
*            "key" : "白色",
*            "doc_count" : 1,
*            "avg_price" : {
*              "value" : 2100.0
*            },
*            "group_by_brand" : {
*              "doc_count_error_upper_bound" : 0,
*              "sum_other_doc_count" : 0,
*              "buckets" : [
*                {
*                  "key" : "TCL",
*                  "doc_count" : 1,
*                  "avg_price" : {
*                    "value" : 2100.0
*                  }
*                }
*              ]
*            }
*          },
*          {
*            "key" : "黑色",
*            "doc_count" : 1,
*            "avg_price" : {
*              "value" : 4800.0
*            },
*            "group_by_brand" : {
*              "doc_count_error_upper_bound" : 0,
*              "sum_other_doc_count" : 0,
*              "buckets" : [
*                {
*                  "key" : "小米",
*                  "doc_count" : 1,
*                  "avg_price" : {
*                    "value" : 4800.0
```

```
 *                 }
 *               }
 *             ]
 *           }
 *         }
 *       ]
 *     }
 *   }
 * }
 *
 * </pre>
 * <p>
 * 程序结果：
 * <pre>
 *
 * ----key：红色
 * ----doc_count：5
 * ----平均价格：4300.0
 * ----group_by_brand：
 * --------key：长虹
 * --------doc_count：3
 * --------平均价格：1666.6666666666667
 * ----------------
 * --------key：三星
 * --------doc_count：1
 * --------平均价格：8000.0
 * ----------------
 * --------key：小米
 * --------doc_count：1
 * --------平均价格：8500.0
 * ----------------
 * ----------------------------------------
 * ----key：蓝色
 * ----doc_count：4
 * ----平均价格：3575.0
 * ----group_by_brand：
 * --------key：TCL
 * --------doc_count：1
 * --------平均价格：1500.0
 * ----------------
 * --------key：三星
 * --------doc_count：1
 * --------平均价格：6100.0
 * ----------------
 * --------key：小米
 * --------doc_count：1
 * --------平均价格：2500.0
 * ----------------
 * --------key：长虹
 * --------doc_count：1
 * --------平均价格：4200.0
 * ----------------
 * ----------------------------------------
 * ----key：绿色
 * ----doc_count：3
 * ----平均价格：2900.0
 * ----group_by_brand：
 * --------key：小米
```

```
     * --------doc_count：2
     * --------平均价格：3750.0
     * ----------------
     * --------key：TCL
     * --------doc_count：1
     * --------平均价格：1200.0
     * ----------------
     * ----------------------------------------
     * ----key：白色
     * ----doc_count：1
     * ----平均价格：2100.0
     * ----group_by_brand：
     * --------key：TCL
     * --------doc_count：1
     * --------平均价格：2100.0
     * ----------------
     * ----------------------------------------
     * ----key：黑色
     * ----doc_count：1
     * ----平均价格：4800.0
     * ----group_by_brand：
     * --------key：小米
     * --------doc_count：1
     * --------平均价格：4800.0
     * ----------------
     * ----------------------------------------
     *
     * </pre>
     *
     * @throws Exception Exception
     */
    @Test
    void aggregation3() throws Exception
    {
        //构建请求
        SearchRequest searchRequest = new SearchRequest("tvs");
        //构建请求体
        SearchSourceBuilder searchSourceBuilder = new SearchSourceBuilder();
        //查询
        searchSourceBuilder.query(QueryBuilders.matchAllQuery());
        //分页
        searchSourceBuilder.size(0);
        //聚合
        searchSourceBuilder.aggregation(
                AggregationBuilders.terms("group_by_colors").field("color")
                        .subAggregations(AggregatorFactories.builder()

.addAggregator(AggregationBuilders.avg("avg_price").field("price"))

.addAggregator(AggregationBuilders.terms("group_by_brand").field("brand")

.subAggregation(AggregationBuilders.avg("avg_price").field("price"))))));
        //放入到请求中
        searchRequest.source(searchSourceBuilder);
        //发起请求
        SearchResponse searchResponse = client.search(searchRequest,
RequestOptions.DEFAULT);
        //获取数据
```

```
        //获取aggregations部分
        Aggregations aggregations = searchResponse.getAggregations();
        //获得group_by_colors
        Terms group_by_colors = aggregations.get("group_by_colors");
        //获取buckets部分
        List<? extends Terms.Bucket> buckets = group_by_colors.getBuckets();
        //遍历
        for (Terms.Bucket bucket : buckets)
        {
            //获取数据
            String key = (String) bucket.getKey();
            long docCount = bucket.getDocCount();
            Avg avg_price = bucket.getAggregations().get("avg_price");
            Terms group_by_brand =
bucket.getAggregations().get("group_by_brand");
            List<? extends Terms.Bucket> buckets1 = group_by_brand.getBuckets();
            double avgPriceValue = avg_price.getValue();
            //打印
            System.out.println("----key: " + key);
            System.out.println("----doc_count: " + docCount);
            System.out.println("----平均价格: " + avgPriceValue);
            System.out.println("----group_by_brand: ");
            for (Terms.Bucket bucket1 : buckets1)
            {
                //获取数据
                String key1 = (String) bucket1.getKey();
                long docCount1 = bucket1.getDocCount();
                Avg avg_price1 = bucket1.getAggregations().get("avg_price");
                double avgPrice1Value = avg_price1.getValue();
                //打印
                System.out.println("--------key: " + key1);
                System.out.println("--------doc_count: " + docCount1);
                System.out.println("--------平均价格: " + avgPrice1Value);
                System.out.println("-----------------");
            }
            System.out.println("----------------------------------------");
        }
    }


    /**
     * 求每个颜色的最大价格、最小价格、平均价格和总价格
     * 请求内容:
     * <pre>
     *
     * GET /tvs/_search
     * {
     *   "query":
     *   {
     *     "match_all": {}
     *   },
     *   "size": 0,
     *   "aggs":
     *   {
     *     "group_by_color":
     *     {
     *       "terms":
     *       {
```

```
*         "field": "color"
*       },
*       "aggs":
*       {
*         "max_price":
*         {
*           "max":
*           {
*             "field": "price"
*           }
*         },
*         "min_price":
*         {
*           "min":
*           {
*             "field": "price"
*           }
*         },
*         "avg_price":
*         {
*           "avg":
*           {
*             "field": "price"
*           }
*         },
*         "sum_price":
*         {
*           "sum":
*           {
*             "field": "price"
*           }
*         }
*       }
*     }
*   }
* }
*
* </pre>
* <p>
* 结果:
* <pre>
*
* {
*   "took" : 1,
*   "timed_out" : false,
*   "_shards" : {
*     "total" : 1,
*     "successful" : 1,
*     "skipped" : 0,
*     "failed" : 0
*   },
*   "hits" : {
*     "total" : {
*       "value" : 14,
*       "relation" : "eq"
*     },
*     "max_score" : null,
*     "hits" : [ ]
```

```
*   },
*   "aggregations" : {
*     "group_by_color" : {
*       "doc_count_error_upper_bound" : 0,
*       "sum_other_doc_count" : 0,
*       "buckets" : [
*         {
*           "key" : "红色",
*           "doc_count" : 5,
*           "max_price" : {
*             "value" : 8500.0
*           },
*           "min_price" : {
*             "value" : 1000.0
*           },
*           "avg_price" : {
*             "value" : 4300.0
*           },
*           "sum_price" : {
*             "value" : 21500.0
*           }
*         },
*         {
*           "key" : "蓝色",
*           "doc_count" : 4,
*           "max_price" : {
*             "value" : 6100.0
*           },
*           "min_price" : {
*             "value" : 1500.0
*           },
*           "avg_price" : {
*             "value" : 3575.0
*           },
*           "sum_price" : {
*             "value" : 14300.0
*           }
*         },
*         {
*           "key" : "绿色",
*           "doc_count" : 3,
*           "max_price" : {
*             "value" : 4500.0
*           },
*           "min_price" : {
*             "value" : 1200.0
*           },
*           "avg_price" : {
*             "value" : 2900.0
*           },
*           "sum_price" : {
*             "value" : 8700.0
*           }
*         },
*         {
*           "key" : "白色",
*           "doc_count" : 1,
*           "max_price" : {
```

```
*              "value" : 2100.0
*            },
*            "min_price" : {
*              "value" : 2100.0
*            },
*            "avg_price" : {
*              "value" : 2100.0
*            },
*            "sum_price" : {
*              "value" : 2100.0
*            }
*          },
*          {
*            "key" : "黑色",
*            "doc_count" : 1,
*            "max_price" : {
*              "value" : 4800.0
*            },
*            "min_price" : {
*              "value" : 4800.0
*            },
*            "avg_price" : {
*              "value" : 4800.0
*            },
*            "sum_price" : {
*              "value" : 4800.0
*            }
*          }
*        ]
*      }
*    }
* }
*
* </pre>
* <p>
* 程序结果:
* <pre>
*
* ----key：红色
* ----doc_count：5
* ----group_by_brand:
* ----max_price：8500.0
* ----min_price：1000.0
* ----avg_price：4300.0
* ----sum_price：21500.0
* ----------------------------------------
* ----key：蓝色
* ----doc_count：4
* ----group_by_brand:
* ----max_price：6100.0
* ----min_price：1500.0
* ----avg_price：3575.0
* ----sum_price：14300.0
* ----------------------------------------
* ----key：绿色
* ----doc_count：3
* ----group_by_brand:
* ----max_price：4500.0
```

```
     * ----min_price: 1200.0
     * ----avg_price: 2900.0
     * ----sum_price: 8700.0
     * ----------------------------------------
     * ----key: 白色
     * ----doc_count: 1
     * ----group_by_brand:
     * ----max_price: 2100.0
     * ----min_price: 2100.0
     * ----avg_price: 2100.0
     * ----sum_price: 2100.0
     * ----------------------------------------
     * ----key: 黑色
     * ----doc_count: 1
     * ----group_by_brand:
     * ----max_price: 4800.0
     * ----min_price: 4800.0
     * ----avg_price: 4800.0
     * ----sum_price: 4800.0
     * ----------------------------------------
     *
     * </pre>
     *
     * @throws Exception Exception
     */
    @Test
    void aggregation4() throws Exception
    {
        //构建请求
        SearchRequest searchRequest = new SearchRequest("tvs");
        //构建请求体
        SearchSourceBuilder searchSourceBuilder = new SearchSourceBuilder();
        //查询
        searchSourceBuilder.query(QueryBuilders.matchAllQuery());
        //分页
        searchSourceBuilder.size(0);
        //聚合
        searchSourceBuilder.aggregation(
                AggregationBuilders.terms("group_by_colors").field("color")
                        .subAggregations(AggregatorFactories.builder()

.addAggregator(AggregationBuilders.max("max_price").field("price"))

.addAggregator(AggregationBuilders.min("min_price").field("price"))

.addAggregator(AggregationBuilders.avg("avg_price").field("price"))

.addAggregator(AggregationBuilders.sum("sum_price").field("price")))
        );

        //放入到请求中
        searchRequest.source(searchSourceBuilder);
        //发起请求
        SearchResponse searchResponse = client.search(searchRequest,
RequestOptions.DEFAULT);
        //获取数据
        //获取aggregations部分
        Aggregations aggregations = searchResponse.getAggregations();
```

```
        //获得group_by_colors
        Terms group_by_colors = aggregations.get("group_by_colors");
        //获取buckets部分
        List<? extends Terms.Bucket> buckets = group_by_colors.getBuckets();
        //遍历
        for (Terms.Bucket bucket : buckets)
        {
            //获取数据
            String key = (String) bucket.getKey();
            long docCount = bucket.getDocCount();
            Max max_price = bucket.getAggregations().get("max_price");
            Min min_price = bucket.getAggregations().get("min_price");
            Avg avg_price = bucket.getAggregations().get("avg_price");
            Sum sum_price = bucket.getAggregations().get("sum_price");
            double maxPriceValue = max_price.getValue();
            double minPriceValue = min_price.getValue();
            double avgPriceValue = avg_price.getValue();
            double sumPriceValue = sum_price.getValue();

            //打印
            System.out.println("----key: " + key);
            System.out.println("----doc_count: " + docCount);
            System.out.println("----group_by_brand: ");
            System.out.println("----max_price: " + maxPriceValue);
            System.out.println("----min_price: " + minPriceValue);
            System.out.println("----avg_price: " + avgPriceValue);
            System.out.println("----sum_price: " + sumPriceValue);


            System.out.println("----------------------------------------");
        }
    }


    /**
     * 划分范围 histogram
     * <p>
     * 请求内容:
     * <pre>
     *
     * GET /tvs/_search
     * {
     *   "query":
     *   {
     *     "match_all": {}
     *   },
     *   "size": 0,
     *   "aggs":
     *   {
     *     "histogram_price":
     *     {
     *       "histogram":
     *       {
     *         "field": "price",
     *         "interval": 2000
     *       }
     *     }
     *   }
```

```
 * }
 *
 * </pre>
 * <p>
 * 结果：
 * <pre>
 *
 * {
 *   "took" : 1,
 *   "timed_out" : false,
 *   "_shards" : {
 *     "total" : 1,
 *     "successful" : 1,
 *     "skipped" : 0,
 *     "failed" : 0
 *   },
 *   "hits" : {
 *     "total" : {
 *       "value" : 14,
 *       "relation" : "eq"
 *     },
 *     "max_score" : null,
 *     "hits" : [ ]
 *   },
 *   "aggregations" : {
 *     "histogram_price" : {
 *       "buckets" : [
 *         {
 *           "key" : 0.0,
 *           "doc_count" : 3
 *         },
 *         {
 *           "key" : 2000.0,
 *           "doc_count" : 5
 *         },
 *         {
 *           "key" : 4000.0,
 *           "doc_count" : 3
 *         },
 *         {
 *           "key" : 6000.0,
 *           "doc_count" : 1
 *         },
 *         {
 *           "key" : 8000.0,
 *           "doc_count" : 2
 *         }
 *       ]
 *     }
 *   }
 * }
 *
 * </pre>
 * <p>
 * 程序结果：
 * <pre>
 *
 * ----key：0.0
```

```
     * ----doc_count：3
     * ----------------------------------------
     * ----key：2000.0
     * ----doc_count：5
     * ----------------------------------------
     * ----key：4000.0
     * ----doc_count：3
     * ----------------------------------------
     * ----key：6000.0
     * ----doc_count：1
     * ----------------------------------------
     * ----key：8000.0
     * ----doc_count：2
     * ----------------------------------------
     *
     * </pre>
     *
     * @throws Exception Exception
     */
    @Test
    void aggregation5() throws Exception
    {
        //构建请求
        SearchRequest searchRequest = new SearchRequest("tvs");
        //构建请求体
        SearchSourceBuilder searchSourceBuilder = new SearchSourceBuilder();
        //查询
        searchSourceBuilder.query(QueryBuilders.matchAllQuery());
        //分页
        searchSourceBuilder.size(0);
        //聚合
        searchSourceBuilder.aggregation(

 AggregationBuilders.histogram("histogram_price").field("price").interval(2000)
        );

        //放入到请求中
        searchRequest.source(searchSourceBuilder);
        //发起请求
        SearchResponse searchResponse = client.search(searchRequest,
RequestOptions.DEFAULT);
        //获取数据
        //获取aggregations部分
        Aggregations aggregations = searchResponse.getAggregations();
        //获得histogram_price
        Histogram histogram_price = aggregations.get("histogram_price");
        //获取buckets部分
        List<? extends Histogram.Bucket> buckets = histogram_price.getBuckets();
        //遍历
        for (Histogram.Bucket bucket : buckets)
        {
            //获取数据
            Double key = (Double) bucket.getKey();
            long docCount = bucket.getDocCount();
            //打印
            System.out.println("----key：" + key);
            System.out.println("----doc_count：" + docCount);
            System.out.println("----------------------------------------");
```

```
    }
}


/**
 * 搜索与聚合结合，查询某个品牌按颜色销量
 * <p>
 * 请求内容：
 * <pre>
 *
 * GET /tvs/_search
 * {
 *   "query":
 *   {
 *     "match":
 *     {
 *       "brand": "小米"
 *     }
 *   },
 *   "aggs": {
 *     "group_by_color":
 *     {
 *       "terms":
 *       {
 *         "field": "color"
 *       }
 *     }
 *   }
 * }
 *
 * </pre>
 * <p>
 * 结果：
 * <pre>
 *
 * {
 *   "took" : 1,
 *   "timed_out" : false,
 *   "_shards" : {
 *     "total" : 1,
 *     "successful" : 1,
 *     "skipped" : 0,
 *     "failed" : 0
 *   },
 *   "hits" : {
 *     "total" : {
 *       "value" : 5,
 *       "relation" : "eq"
 *     },
 *     "max_score" : 1.0033021,
 *     "hits" : [
 *       {
 *         "_index" : "tvs",
 *         "_id" : "7aouDoEBEpQthbP41cfj",
 *         "_score" : 1.0033021,
 *         "_source" : {
 *           "price" : 3000,
 *           "color" : "绿色",
```

```
*           "brand" : "小米",
*           "sold_date" : "2019-05-18"
*         }
*       },
*       {
*         "_index" : "tvs",
*         "_id" : "8qouDoEBEpQthbP41cfj",
*         "_score" : 1.0033021,
*         "_source" : {
*           "price" : 2500,
*           "color" : "蓝色",
*           "brand" : "小米",
*           "sold_date" : "2020-02-12"
*         }
*       },
*       {
*         "_index" : "tvs",
*         "_id" : "86ouDoEBEpQthbP41cfj",
*         "_score" : 1.0033021,
*         "_source" : {
*           "price" : 4500,
*           "color" : "绿色",
*           "brand" : "小米",
*           "sold_date" : "2020-04-22"
*         }
*       },
*       {
*         "_index" : "tvs",
*         "_id" : "9qouDoEBEpQthbP41cfj",
*         "_score" : 1.0033021,
*         "_source" : {
*           "price" : 8500,
*           "color" : "红色",
*           "brand" : "小米",
*           "sold_date" : "2020-05-19"
*         }
*       },
*       {
*         "_index" : "tvs",
*         "_id" : "-KouDoEBEpQthbP41cfj",
*         "_score" : 1.0033021,
*         "_source" : {
*           "price" : 4800,
*           "color" : "黑色",
*           "brand" : "小米",
*           "sold_date" : "2020-06-10"
*         }
*       }
*     ]
*   },
*   "aggregations" : {
*     "group_by_color" : {
*       "doc_count_error_upper_bound" : 0,
*       "sum_other_doc_count" : 0,
*       "buckets" : [
*         {
*           "key" : "绿色",
*           "doc_count" : 2
```

```
     *          },
     *          {
     *            "key" : "红色",
     *            "doc_count" : 1
     *          },
     *          {
     *            "key" : "蓝色",
     *            "doc_count" : 1
     *          },
     *          {
     *            "key" : "黑色",
     *            "doc_count" : 1
     *          }
     *        ]
     *      }
     *    }
     * }
     *
     * </pre>
     * <p>
     * 程序结果：
     * <pre>
     *
     * --数量：5
     * --数组数量：5
     * --最大分数：1.0033021
     * -->{color=绿色, price=3000, sold_date=2019-05-18, brand=小米}
     * -->{color=蓝色, price=2500, sold_date=2020-02-12, brand=小米}
     * -->{color=绿色, price=4500, sold_date=2020-04-22, brand=小米}
     * -->{color=红色, price=8500, sold_date=2020-05-19, brand=小米}
     * -->{color=黑色, price=4800, sold_date=2020-06-10, brand=小米}
     *
     * 聚合结果：
     *
     * ----key：绿色
     * ----doc_count：2
     * ----------------------------------------
     * ----key：红色
     * ----doc_count：1
     * ----------------------------------------
     * ----key：蓝色
     * ----doc_count：1
     * ----------------------------------------
     * ----key：黑色
     * ----doc_count：1
     * ----------------------------------------
     *
     * </pre>
     *
     * @throws Exception Exception
     */
    @Test
    void aggregation6() throws Exception
    {
        //构建请求
        SearchRequest searchRequest = new SearchRequest("tvs");
        //构建请求体
        SearchSourceBuilder searchSourceBuilder = new SearchSourceBuilder();
```

```
        //查询
        searchSourceBuilder.query(QueryBuilders.matchQuery("brand", "小米"));
        //分页
        //searchSourceBuilder.size(0);
        //聚合
        searchSourceBuilder.aggregation(
                AggregationBuilders.terms("group_by_color").field("color")
        );

        //放入到请求中
        searchRequest.source(searchSourceBuilder);
        //发起请求
        SearchResponse searchResponse = client.search(searchRequest,
RequestOptions.DEFAULT);
        //获取数据
        //获得hits
        SearchHits hits = searchResponse.getHits();
        long value = hits.getTotalHits().value;
        float maxScore = hits.getMaxScore();
        SearchHit[] hitsHits = hits.getHits();
        System.out.println("--数量: " + value);
        System.out.println("--数组数量: " + hitsHits.length);
        System.out.println("--最大分数: " + maxScore);
        for (SearchHit hitsHit : hitsHits)
        {
            Map<String, Object> sourceAsMap = hitsHit.getSourceAsMap();
            System.out.println("-->" + sourceAsMap);
        }
        System.out.println();
        System.out.println("聚合结果: ");
        System.out.println();

        //获取aggregations部分
        Aggregations aggregations = searchResponse.getAggregations();
        //获得group_by_color
        Terms group_by_color = aggregations.get("group_by_color");
        //获取buckets部分
        List<? extends Terms.Bucket> buckets = group_by_color.getBuckets();
        //遍历
        for (Terms.Bucket bucket : buckets)
        {
            //获取数据
            String key = (String) bucket.getKey();
            long docCount = bucket.getDocCount();
            //打印
            System.out.println("----key: " + key);
            System.out.println("----doc_count: " + docCount);
            System.out.println("----------------------------------------");
        }
    }


    /**
     * 单个品牌与所有品牌销量对比
     * <p>
     * 请求内容:
     * <pre>
     *
```

```
     * GET /tvs/_search
     * {
     *   "query":
     *   {
     *     "term":
     *     {
     *       "brand":
     *       {
     *         "value": "小米"
     *       }
     *     }
     *   },
     *   "aggs":
     *   {
     *     "single_brand_avg_price":
     *     {
     *       "avg":
     *       {
     *         "field": "price"
     *       }
     *     },
     *     "all":
     *     {
     *       "global":
     *       {
     *
     *       },
     *       "aggs":
     *       {
     *         "all_brand_avg_price":
     *         {
     *           "avg":
     *           {
     *             "field": "price"
     *           }
     *         }
     *       }
     *     }
     *   }
     * }
     *
     * </pre>
     * <p>
     * 结果:
     * <pre>
     *
     * {
     *   "took" : 2,
     *   "timed_out" : false,
     *   "_shards" : {
     *     "total" : 1,
     *     "successful" : 1,
     *     "skipped" : 0,
     *     "failed" : 0
     *   },
     *   "hits" : {
     *     "total" : {
     *       "value" : 5,
```

```
*       "relation" : "eq"
*     },
*     "max_score" : 1.0033021,
*     "hits" : [
*       {
*         "_index" : "tvs",
*         "_id" : "7aouDoEBEpQthbP41cfj",
*         "_score" : 1.0033021,
*         "_source" : {
*           "price" : 3000,
*           "color" : "绿色",
*           "brand" : "小米",
*           "sold_date" : "2019-05-18"
*         }
*       },
*       {
*         "_index" : "tvs",
*         "_id" : "8qouDoEBEpQthbP41cfj",
*         "_score" : 1.0033021,
*         "_source" : {
*           "price" : 2500,
*           "color" : "蓝色",
*           "brand" : "小米",
*           "sold_date" : "2020-02-12"
*         }
*       },
*       {
*         "_index" : "tvs",
*         "_id" : "86ouDoEBEpQthbP41cfj",
*         "_score" : 1.0033021,
*         "_source" : {
*           "price" : 4500,
*           "color" : "绿色",
*           "brand" : "小米",
*           "sold_date" : "2020-04-22"
*         }
*       },
*       {
*         "_index" : "tvs",
*         "_id" : "9qouDoEBEpQthbP41cfj",
*         "_score" : 1.0033021,
*         "_source" : {
*           "price" : 8500,
*           "color" : "红色",
*           "brand" : "小米",
*           "sold_date" : "2020-05-19"
*         }
*       },
*       {
*         "_index" : "tvs",
*         "_id" : "-KouDoEBEpQthbP41cfj",
*         "_score" : 1.0033021,
*         "_source" : {
*           "price" : 4800,
*           "color" : "黑色",
*           "brand" : "小米",
*           "sold_date" : "2020-06-10"
*         }
```

```
     *       }
     *     ]
     *   },
     *   "aggregations" : {
     *     "all" : {
     *       "doc_count" : 14,
     *       "all_brand_avg_price" : {
     *         "value" : 3671.4285714285716
     *       }
     *     },
     *     "single_brand_avg_price" : {
     *       "value" : 4660.0
     *     }
     *   }
     * }
     *
     * </pre>
     * <p>
     * 程序结果：
     * <pre>
     *
     * --数量：5
     * --数组数量：5
     * --最大分数：1.0033021
     * -->{color=绿色, price=3000, sold_date=2019-05-18, brand=小米}
     * -->{color=蓝色, price=2500, sold_date=2020-02-12, brand=小米}
     * -->{color=绿色, price=4500, sold_date=2020-04-22, brand=小米}
     * -->{color=红色, price=8500, sold_date=2020-05-19, brand=小米}
     * -->{color=黑色, price=4800, sold_date=2020-06-10, brand=小米}
     *
     * 聚合结果：
     *
     * ----小米销售平均价格：4660.0
     * ---所有品牌销售总数：14
     * ----所有品牌销售平均价格：3671.4285714285716
     *
     * </pre>
     *
     * @throws Exception Exception
     */
    @Test
    void aggregation7() throws Exception
    {
        //构建请求
        SearchRequest searchRequest = new SearchRequest("tvs");
        //构建请求体
        SearchSourceBuilder searchSourceBuilder = new SearchSourceBuilder();
        //查询
        searchSourceBuilder.query(QueryBuilders.termQuery("brand", "小米"));
        //分页
        //searchSourceBuilder.size(0);
        //聚合

 searchSourceBuilder.aggregation((AggregationBuilders.avg("single_brand_avg_pric
e").field("price")))
                .aggregation(AggregationBuilders.global("all")

.subAggregation(AggregationBuilders.avg("all_brand_avg_price").field("price")));
```

```
        //放入到请求中
        searchRequest.source(searchSourceBuilder);
        //发起请求
        SearchResponse searchResponse = client.search(searchRequest,
RequestOptions.DEFAULT);
        //获取数据
        //获得hits
        SearchHits hits = searchResponse.getHits();
        long value = hits.getTotalHits().value;
        float maxScore = hits.getMaxScore();
        SearchHit[] hitsHits = hits.getHits();
        System.out.println("--数量：" + value);
        System.out.println("--数组数量：" + hitsHits.length);
        System.out.println("--最大分数：" + maxScore);
        for (SearchHit hitsHit : hitsHits)
        {
            Map<String, Object> sourceAsMap = hitsHit.getSourceAsMap();
            System.out.println("-->" + sourceAsMap);
        }
        System.out.println();
        System.out.println("聚合结果：");
        System.out.println();

        //获取aggregations部分
        Aggregations aggregations = searchResponse.getAggregations();
        //获得single_brand_avg_price
        Avg single_brand_avg_price = aggregations.get("single_brand_avg_price");
        double single_brand_avg_priceValue = single_brand_avg_price.getValue();
        System.out.println("----小米销售平均价格：" + single_brand_avg_priceValue);
        Global all = aggregations.get("all");
        long docCount = all.getDocCount();
        System.out.println("---所有品牌销售总数：" + docCount);
        Avg all_brand_avg_price =
all.getAggregations().get("all_brand_avg_price");
        double all_brand_avg_priceValue = all_brand_avg_price.getValue();
        System.out.println("----所有品牌销售平均价格：" + all_brand_avg_priceValue);
    }


    /**
     * 统计价格大于1200的电视平均价格
     * <p>
     * 请求内容：
     * <pre>
     *
     * GET /tvs/_search
     * {
     *   "query":
     *   {
     *     "constant_score":
     *     {
     *       "filter":
     *       {
     *         "range":
     *         {
     *           "price":
     *           {
```

```
 *             "gte": 1200
 *           }
 *         }
 *       }
 *     }
 *   },
 *   "aggs":
 *   {
 *     "avg_price":
 *     {
 *       "avg":
 *       {
 *         "field": "price"
 *       }
 *     }
 *   }
 *
 * }
 *
 * </pre>
 * <p>
 * 结果:
 * <pre>
 *
 * {
 *   "took" : 0,
 *   "timed_out" : false,
 *   "_shards" : {
 *     "total" : 1,
 *     "successful" : 1,
 *     "skipped" : 0,
 *     "failed" : 0
 *   },
 *   "hits" : {
 *     "total" : {
 *       "value" : 13,
 *       "relation" : "eq"
 *     },
 *     "max_score" : 1.0,
 *     "hits" : [
 *       {
 *         "_index" : "tvs",
 *         "_id" : "7KouDoEBEpQthbP41cfj",
 *         "_score" : 1.0,
 *         "_source" : {
 *           "price" : 2000,
 *           "color" : "红色",
 *           "brand" : "长虹",
 *           "sold_date" : "2019-11-05"
 *         }
 *       },
 *       {
 *         "_index" : "tvs",
 *         "_id" : "7aouDoEBEpQthbP41cfj",
 *         "_score" : 1.0,
 *         "_source" : {
 *           "price" : 3000,
 *           "color" : "绿色",
```

```
*           "brand" : "小米",
*           "sold_date" : "2019-05-18"
*         }
*       },
*       {
*         "_index" : "tvs",
*         "_id" : "7qouDoEBEpQthbP41cfj",
*         "_score" : 1.0,
*         "_source" : {
*           "price" : 1500,
*           "color" : "蓝色",
*           "brand" : "TCL",
*           "sold_date" : "2019-07-02"
*         }
*       },
*       {
*         "_index" : "tvs",
*         "_id" : "76ouDoEBEpQthbP41cfj",
*         "_score" : 1.0,
*         "_source" : {
*           "price" : 1200,
*           "color" : "绿色",
*           "brand" : "TCL",
*           "sold_date" : "2019-08-19"
*         }
*       },
*       {
*         "_index" : "tvs",
*         "_id" : "8KouDoEBEpQthbP41cfj",
*         "_score" : 1.0,
*         "_source" : {
*           "price" : 2000,
*           "color" : "红色",
*           "brand" : "长虹",
*           "sold_date" : "2019-11-05"
*         }
*       },
*       {
*         "_index" : "tvs",
*         "_id" : "8aouDoEBEpQthbP41cfj",
*         "_score" : 1.0,
*         "_source" : {
*           "price" : 8000,
*           "color" : "红色",
*           "brand" : "三星",
*           "sold_date" : "2020-01-01"
*         }
*       },
*       {
*         "_index" : "tvs",
*         "_id" : "8qouDoEBEpQthbP41cfj",
*         "_score" : 1.0,
*         "_source" : {
*           "price" : 2500,
*           "color" : "蓝色",
*           "brand" : "小米",
*           "sold_date" : "2020-02-12"
*         }
```

```
*       },
*       {
*         "_index" : "tvs",
*         "_id" : "86ouDoEBEpQthbP41cfj",
*         "_score" : 1.0,
*         "_source" : {
*           "price" : 4500,
*           "color" : "绿色",
*           "brand" : "小米",
*           "sold_date" : "2020-04-22"
*         }
*       },
*       {
*         "_index" : "tvs",
*         "_id" : "9KouDoEBEpQthbP41cfj",
*         "_score" : 1.0,
*         "_source" : {
*           "price" : 6100,
*           "color" : "蓝色",
*           "brand" : "三星",
*           "sold_date" : "2020-05-16"
*         }
*       },
*       {
*         "_index" : "tvs",
*         "_id" : "9aouDoEBEpQthbP41cfj",
*         "_score" : 1.0,
*         "_source" : {
*           "price" : 2100,
*           "color" : "白色",
*           "brand" : "TCL",
*           "sold_date" : "2020-05-17"
*         }
*       }
*     ]
*   },
*   "aggregations" : {
*     "avg_price" : {
*       "value" : 3876.923076923077
*     }
*   }
* }
*
* </pre>
* <p>
* 程序结果：
* <pre>
*
* --数量：13
* --数组数量：10
* --最大分数：1.0
* -->{color=红色, price=2000, sold_date=2019-11-05, brand=长虹}
* -->{color=绿色, price=3000, sold_date=2019-05-18, brand=小米}
* -->{color=蓝色, price=1500, sold_date=2019-07-02, brand=TCL}
* -->{color=绿色, price=1200, sold_date=2019-08-19, brand=TCL}
* -->{color=红色, price=2000, sold_date=2019-11-05, brand=长虹}
* -->{color=红色, price=8000, sold_date=2020-01-01, brand=三星}
* -->{color=蓝色, price=2500, sold_date=2020-02-12, brand=小米}
```

```
     * -->{color=绿色, price=4500, sold_date=2020-04-22, brand=小米}
     * -->{color=蓝色, price=6100, sold_date=2020-05-16, brand=三星}
     * -->{color=白色, price=2100, sold_date=2020-05-17, brand=TCL}
     *
     * 聚合结果:
     *
     * 大于1200元的平均价格：3876.923076923077
     *
     * </pre>
     *
     * @throws Exception Exception
     */
    @Test
    void aggregation8() throws Exception
    {
        //构建请求
        SearchRequest searchRequest = new SearchRequest("tvs");
        //构建请求体
        SearchSourceBuilder searchSourceBuilder = new SearchSourceBuilder();
        //查询

 searchSourceBuilder.query(QueryBuilders.constantScoreQuery(QueryBuilders.rangeQ
uery("price").gte(1200)));
        //分页
        //searchSourceBuilder.size(0);
        //聚合
        searchSourceBuilder.aggregation(
                AggregationBuilders.avg("avg_price").field("price")
        );

        //放入到请求中
        searchRequest.source(searchSourceBuilder);
        //发起请求
        SearchResponse searchResponse = client.search(searchRequest,
RequestOptions.DEFAULT);
        //获取数据
        //获得hits
        SearchHits hits = searchResponse.getHits();
        long value = hits.getTotalHits().value;
        float maxScore = hits.getMaxScore();
        SearchHit[] hitsHits = hits.getHits();
        System.out.println("--数量：" + value);
        System.out.println("--数组数量：" + hitsHits.length);
        System.out.println("--最大分数：" + maxScore);
        for (SearchHit hitsHit : hitsHits)
        {
            Map<String, Object> sourceAsMap = hitsHit.getSourceAsMap();
            System.out.println("-->" + sourceAsMap);
        }
        System.out.println();
        System.out.println("聚合结果：");
        System.out.println();

        //获取aggregations部分
        Aggregations aggregations = searchResponse.getAggregations();
        //获得avg_price
        Avg avg_price = aggregations.get("avg_price");
        double avg_priceValue = avg_price.getValue();
```

```
        System.out.println("大于1200元的平均价格：" + avg_priceValue);
    }


    /**
     * 按每种颜色的平均销售额降序排序
     * <p>
     * 请求内容：
     * <pre>
     *
     * GET /tvs/_search
     * {
     *   "query":
     *   {
     *     "match_all": {}
     *   },
     *   "size": 0,
     *   "aggs":
     *   {
     *     "group_by_color":
     *     {
     *       "terms":
     *       {
     *         "field": "color",
     *         "order":
     *         {
     *           "avg_price": "desc"
     *         }
     *       },
     *       "aggs":
     *       {
     *         "avg_price":
     *         {
     *           "avg":
     *           {
     *             "field": "price"
     *           }
     *         }
     *       }
     *     }
     *   }
     * }
     *
     * </pre>
     * <p>
     * 结果：
     * <pre>
     *
     * {
     *   "took" : 1,
     *   "timed_out" : false,
     *   "_shards" : {
     *     "total" : 1,
     *     "successful" : 1,
     *     "skipped" : 0,
     *     "failed" : 0
     *   },
     *   "hits" : {
```

```
*     "total" : {
*       "value" : 14,
*       "relation" : "eq"
*     },
*     "max_score" : null,
*     "hits" : [ ]
*   },
*   "aggregations" : {
*     "group_by_color" : {
*       "doc_count_error_upper_bound" : 0,
*       "sum_other_doc_count" : 0,
*       "buckets" : [
*         {
*           "key" : "黑色",
*           "doc_count" : 1,
*           "avg_price" : {
*             "value" : 4800.0
*           }
*         },
*         {
*           "key" : "红色",
*           "doc_count" : 5,
*           "avg_price" : {
*             "value" : 4300.0
*           }
*         },
*         {
*           "key" : "蓝色",
*           "doc_count" : 4,
*           "avg_price" : {
*             "value" : 3575.0
*           }
*         },
*         {
*           "key" : "绿色",
*           "doc_count" : 3,
*           "avg_price" : {
*             "value" : 2900.0
*           }
*         },
*         {
*           "key" : "白色",
*           "doc_count" : 1,
*           "avg_price" : {
*             "value" : 2100.0
*           }
*         }
*       ]
*     }
*   }
* }
*
* </pre>
* <p>
* 程序结果：
* <pre>
*
* ----key：黑色
```

```
     * ----doc_count：1
     * ----avg_price：4800.0
     * ----------------------------------------
     * ----key：红色
     * ----doc_count：5
     * ----avg_price：4300.0
     * ----------------------------------------
     * ----key：蓝色
     * ----doc_count：4
     * ----avg_price：3575.0
     * ----------------------------------------
     * ----key：绿色
     * ----doc_count：3
     * ----avg_price：2900.0
     * ----------------------------------------
     * ----key：白色
     * ----doc_count：1
     * ----avg_price：2100.0
     * ----------------------------------------
     *
     * </pre>
     *
     * @throws Exception Exception
     */
    @Test
    void aggregation9() throws Exception
    {
        //构建请求
        SearchRequest searchRequest = new SearchRequest("tvs");
        //构建请求体
        SearchSourceBuilder searchSourceBuilder = new SearchSourceBuilder();
        //查询
        searchSourceBuilder.query(QueryBuilders.matchAllQuery());
        //分页
        searchSourceBuilder.size(0);
        //聚合
        searchSourceBuilder.aggregation(
                AggregationBuilders.terms("group_by_color").field("color")
                        .order(BucketOrder.aggregation("avg_price", false))

.subAggregation(AggregationBuilders.avg("avg_price").field("price"))
        );

        //放入到请求中
        searchRequest.source(searchSourceBuilder);
        //发起请求
        SearchResponse searchResponse = client.search(searchRequest,
RequestOptions.DEFAULT);
        //获取数据
        //获取aggregations部分
        Aggregations aggregations = searchResponse.getAggregations();
        //获得group_by_color
        Terms group_by_color = aggregations.get("group_by_color");
        //获取buckets部分
        List<? extends Terms.Bucket> buckets = group_by_color.getBuckets();
        //遍历
        for (Terms.Bucket bucket : buckets)
        {
```

```
            //获取数据
            String key = (String) bucket.getKey();
            long docCount = bucket.getDocCount();
            Avg avg_price = bucket.getAggregations().get("avg_price");
            double avg_priceValue = avg_price.getValue();
            //打印
            System.out.println("----key: " + key);
            System.out.println("----doc_count: " + docCount);
            System.out.println("----avg_price: " + avg_priceValue);
            System.out.println("----------------------------------------");
        }
    }


    /**
     * 按每种颜色的每种品牌平均销售额降序排序
     * <p>
     * 请求内容:
     * <pre>
     *
     * GET /tvs/_search
     * {
     *   "query":
     *   {
     *     "match_all": {}
     *   },
     *   "size": 0,
     *   "aggs":
     *   {
     *     "group_by_color":
     *     {
     *       "terms":
     *       {
     *         "field": "color"
     *
     *       },
     *       "aggs":
     *       {
     *         "group_by_brand":
     *         {
     *           "terms":
     *           {
     *             "field": "brand",
     *             "order":
     *             {
     *               "avg_price": "desc"
     *             }
     *           },
     *           "aggs":
     *           {
     *             "avg_price":
     *             {
     *               "avg":
     *               {
     *                 "field": "price"
     *               }
     *             }
     *           }
```

```
*         }
*       }
*     }
*   }
* }
*
* </pre>
* <p>
* 结果:
* <pre>
*
* {
*   "took" : 2,
*   "timed_out" : false,
*   "_shards" : {
*     "total" : 1,
*     "successful" : 1,
*     "skipped" : 0,
*     "failed" : 0
*   },
*   "hits" : {
*     "total" : {
*       "value" : 14,
*       "relation" : "eq"
*     },
*     "max_score" : null,
*     "hits" : [ ]
*   },
*   "aggregations" : {
*     "group_by_color" : {
*       "doc_count_error_upper_bound" : 0,
*       "sum_other_doc_count" : 0,
*       "buckets" : [
*         {
*           "key" : "红色",
*           "doc_count" : 5,
*           "group_by_brand" : {
*             "doc_count_error_upper_bound" : 0,
*             "sum_other_doc_count" : 0,
*             "buckets" : [
*               {
*                 "key" : "小米",
*                 "doc_count" : 1,
*                 "avg_price" : {
*                   "value" : 8500.0
*                 }
*               },
*               {
*                 "key" : "三星",
*                 "doc_count" : 1,
*                 "avg_price" : {
*                   "value" : 8000.0
*                 }
*               },
*               {
*                 "key" : "长虹",
*                 "doc_count" : 3,
*                 "avg_price" : {
```

```
*                  "value" : 1666.6666666666667
*                }
*              }
*            ]
*          }
*        },
*        {
*          "key" : "蓝色",
*          "doc_count" : 4,
*          "group_by_brand" : {
*            "doc_count_error_upper_bound" : 0,
*            "sum_other_doc_count" : 0,
*            "buckets" : [
*              {
*                "key" : "三星",
*                "doc_count" : 1,
*                "avg_price" : {
*                  "value" : 6100.0
*                }
*              },
*              {
*                "key" : "长虹",
*                "doc_count" : 1,
*                "avg_price" : {
*                  "value" : 4200.0
*                }
*              },
*              {
*                "key" : "小米",
*                "doc_count" : 1,
*                "avg_price" : {
*                  "value" : 2500.0
*                }
*              },
*              {
*                "key" : "TCL",
*                "doc_count" : 1,
*                "avg_price" : {
*                  "value" : 1500.0
*                }
*              }
*            ]
*          }
*        },
*        {
*          "key" : "绿色",
*          "doc_count" : 3,
*          "group_by_brand" : {
*            "doc_count_error_upper_bound" : 0,
*            "sum_other_doc_count" : 0,
*            "buckets" : [
*              {
*                "key" : "小米",
*                "doc_count" : 2,
*                "avg_price" : {
*                  "value" : 3750.0
*                }
*              },
```

```
 *                {
 *                  "key" : "TCL",
 *                  "doc_count" : 1,
 *                  "avg_price" : {
 *                    "value" : 1200.0
 *                  }
 *                }
 *              ]
 *            }
 *          },
 *          {
 *            "key" : "白色",
 *            "doc_count" : 1,
 *            "group_by_brand" : {
 *              "doc_count_error_upper_bound" : 0,
 *              "sum_other_doc_count" : 0,
 *              "buckets" : [
 *                {
 *                  "key" : "TCL",
 *                  "doc_count" : 1,
 *                  "avg_price" : {
 *                    "value" : 2100.0
 *                  }
 *                }
 *              ]
 *            }
 *          },
 *          {
 *            "key" : "黑色",
 *            "doc_count" : 1,
 *            "group_by_brand" : {
 *              "doc_count_error_upper_bound" : 0,
 *              "sum_other_doc_count" : 0,
 *              "buckets" : [
 *                {
 *                  "key" : "小米",
 *                  "doc_count" : 1,
 *                  "avg_price" : {
 *                    "value" : 4800.0
 *                  }
 *                }
 *              ]
 *            }
 *          }
 *        ]
 *      }
 *    }
 * }
 *
 * </pre>
 * <p>
 * 程序结果：
 * <pre>
 *
 * ----key：红色
 * ----doc_count：5
 * ----group_by_brand：
 * --------key：小米
```

```
* --------doc_count：1
* --------avg_price：8500.0
* ------------------------------
* --------key：三星
* --------doc_count：1
* --------avg_price：8000.0
* ------------------------------
* --------key：长虹
* --------doc_count：3
* --------avg_price：1666.6666666666667
* ------------------------------
* ----------------------------------------
* ----key：蓝色
* ----doc_count：4
* ----group_by_brand：
* --------key：三星
* --------doc_count：1
* --------avg_price：6100.0
* ------------------------------
* --------key：长虹
* --------doc_count：1
* --------avg_price：4200.0
* ------------------------------
* --------key：小米
* --------doc_count：1
* --------avg_price：2500.0
* ------------------------------
* --------key：TCL
* --------doc_count：1
* --------avg_price：1500.0
* ------------------------------
* ----------------------------------------
* ----key：绿色
* ----doc_count：3
* ----group_by_brand：
* --------key：小米
* --------doc_count：2
* --------avg_price：3750.0
* ------------------------------
* --------key：TCL
* --------doc_count：1
* --------avg_price：1200.0
* ------------------------------
* ----------------------------------------
* ----key：白色
* ----doc_count：1
* ----group_by_brand：
* --------key：TCL
* --------doc_count：1
* --------avg_price：2100.0
* ------------------------------
* ----------------------------------------
* ----key：黑色
* ----doc_count：1
* ----group_by_brand：
* --------key：小米
* --------doc_count：1
* --------avg_price：4800.0
```

```
     * ----------------------------
     * ----------------------------------------
     *
     * </pre>
     *
     * @throws Exception Exception
     */
    @Test
    void aggregation10() throws Exception
    {
        //构建请求
        SearchRequest searchRequest = new SearchRequest("tvs");
        //构建请求体
        SearchSourceBuilder searchSourceBuilder = new SearchSourceBuilder();
        //查询
        searchSourceBuilder.query(QueryBuilders.matchAllQuery());
        //分页
        searchSourceBuilder.size(0);
        //聚合
        searchSourceBuilder.aggregation(
                AggregationBuilders.terms("group_by_color").field("color")

.subAggregation(AggregationBuilders.terms("group_by_brand").field("brand")
                                .order(BucketOrder.aggregation("avg_price",
false))

.subAggregation(AggregationBuilders.avg("avg_price").field("price")))
        );

        //放入到请求中
        searchRequest.source(searchSourceBuilder);
        //发起请求
        SearchResponse searchResponse = client.search(searchRequest,
RequestOptions.DEFAULT);
        //获取数据
        //获取aggregations部分
        Aggregations aggregations = searchResponse.getAggregations();
        //获得group_by_color
        Terms group_by_color = aggregations.get("group_by_color");
        //获取buckets部分
        List<? extends Terms.Bucket> buckets = group_by_color.getBuckets();
        //遍历
        for (Terms.Bucket bucket : buckets)
        {
            //获取数据
            String key = (String) bucket.getKey();
            long docCount = bucket.getDocCount();
            Terms group_by_brand =
bucket.getAggregations().get("group_by_brand");
            List<? extends Terms.Bucket> buckets1 = group_by_brand.getBuckets();
            //打印
            System.out.println("----key: " + key);
            System.out.println("----doc_count: " + docCount);
            System.out.println("----group_by_brand: ");
            for (Terms.Bucket bucket1 : buckets1)
            {
                //获取数据
                //获取数据
```

```
                String key1 = (String) bucket1.getKey();
                long docCount1 = bucket1.getDocCount();
                Avg avg_price1 = bucket1.getAggregations().get("avg_price");
                double avgPrice1Value = avg_price1.getValue();
                //打印
                System.out.println("--------key: " + key1);
                System.out.println("--------doc_count: " + docCount1);
                System.out.println("--------avg_price: " + avgPrice1Value);
                System.out.println("------------------------------");
            }
            System.out.println("----------------------------------------");
        }
    }
}
```

elasticsearch sql

使用方式

http请求

```
POST /_sql?format=txt
{
  "query": """
  SELECT * FROM "tvs"
  """
}
```

返回:

```
     brand      |     color      |     price      |       sold_date
---------------+---------------+---------------+------------------------
长虹             |红色             |1000           |2019-10-28T00:00:00.000Z
长虹             |红色             |2000           |2019-11-05T00:00:00.000Z
小米             |绿色             |3000           |2019-05-18T00:00:00.000Z
TCL             |蓝色             |1500           |2019-07-02T00:00:00.000Z
TCL             |绿色             |1200           |2019-08-19T00:00:00.000Z
长虹             |红色             |2000           |2019-11-05T00:00:00.000Z
三星             |红色             |8000           |2020-01-01T00:00:00.000Z
小米             |蓝色             |2500           |2020-02-12T00:00:00.000Z
小米             |绿色             |4500           |2020-04-22T00:00:00.000Z
三星             |蓝色             |6100           |2020-05-16T00:00:00.000Z
TCL             |白色             |2100           |2020-05-17T00:00:00.000Z
小米             |红色             |8500           |2020-05-19T00:00:00.000Z
长虹             |蓝色             |4200           |2020-05-23T00:00:00.000Z
小米             |黑色             |4800           |2020-06-10T00:00:00.000Z
```

使用客户端

1. 进入elasticsearch安装目录
2. 进入bin目录
3. 找到elasticsearch-sql-cli.bat文件
4. 运行elasticsearch-sql-cli.bat文件，或者在此目录下打开控制台，输入elasticsearch-sql-cli运行

运行结果：

```
                      asticElasticE
                    ElasticE  sticEla
          sticEl  ticEl            Elast
        lasti Elasti                   tic
      cEl       ast                    icE
     icE        as                      cEl
     icE        as                      cEl
     icEla     las                       El
   sticElasticElast                    icElas
 las           last                    ticElast
El              asti                 asti    stic
El              asticEla           Elas        icE
El            Elas  cElasticE   ticEl           cE
Ela        ticEl         ticElasti             cE
 las     astic               last              icE
   sticElas                  asti           stic
     icEl                     sticElasticElast
     icE                      sticE   ticEla
     icE                      sti       cEla
     icEl                     sti        Ela
      cEl                     sti       cEl
       Ela                   astic    ticE
         asti               ElasticElasti
           ticElasti  lasticElas
              ElasticElast

                       SQL
                      8.1.3

sql>
```

5. 测试输入

```
select * from tvs;
```

结果：

```
                      asticElasticE
                    ElasticE  sticEla
          sticEl  ticEl            Elast
        lasti Elasti                   tic
```

```
      cEl       ast                    icE
     icE        as                      cEl
     icE        as                      cEl
     icEla     las                       El
   sticElasticElast                    icElas
 las           last                   ticElast
El              asti                asti    stic
El              asticEla          Elas        icE
El            Elas  cElasticE   ticEl          cE
Ela        ticEl         ticElasti             cE
 las     astic               last              icE
   sticElas                  asti           stic
     icEl                     sticElasticElast
     icE                      sticE   ticEla
     icE                      sti       cEla
     icEl                     sti        Ela
      cEl                     sti       cEl
       Ela                   astic    ticE
         asti              ElasticElasti
           ticElasti  lasticElas
              ElasticElast

                      SQL
                     8.1.3

sql> select * from tvs;
     brand     |     color     |     price     |       sold_date
---------------+---------------+---------------+------------------------
长虹             |红色             |1000           |2019-10-28T00:00:00.000Z
长虹             |红色             |2000           |2019-11-05T00:00:00.000Z
小米             |绿色             |3000           |2019-05-18T00:00:00.000Z
TCL             |蓝色             |1500           |2019-07-02T00:00:00.000Z
TCL             |绿色             |1200           |2019-08-19T00:00:00.000Z
长虹             |红色             |2000           |2019-11-05T00:00:00.000Z
三星             |红色             |8000           |2020-01-01T00:00:00.000Z
小米             |蓝色             |2500           |2020-02-12T00:00:00.000Z
小米             |绿色             |4500           |2020-04-22T00:00:00.000Z
三星             |蓝色             |6100           |2020-05-16T00:00:00.000Z
TCL             |白色             |2100           |2020-05-17T00:00:00.000Z
小米             |红色             |8500           |2020-05-19T00:00:00.000Z
长虹             |蓝色             |4200           |2020-05-23T00:00:00.000Z
小米             |黑色             |4800           |2020-06-10T00:00:00.000Z

sql>
```

显示方式

txt

```
POST /_sql?format=txt
{
  "query": """
  SELECT * FROM "tvs"
  """
}
```

结果:

```
     brand      |     color      |     price      |       sold_date
----------------+----------------+----------------+------------------------
长虹             |红色             |1000            |2019-10-28T00:00:00.000Z
长虹             |红色             |2000            |2019-11-05T00:00:00.000Z
小米             |绿色             |3000            |2019-05-18T00:00:00.000Z
TCL             |蓝色             |1500            |2019-07-02T00:00:00.000Z
TCL             |绿色             |1200            |2019-08-19T00:00:00.000Z
长虹             |红色             |2000            |2019-11-05T00:00:00.000Z
三星             |红色             |8000            |2020-01-01T00:00:00.000Z
小米             |蓝色             |2500            |2020-02-12T00:00:00.000Z
小米             |绿色             |4500            |2020-04-22T00:00:00.000Z
三星             |蓝色             |6100            |2020-05-16T00:00:00.000Z
TCL             |白色             |2100            |2020-05-17T00:00:00.000Z
小米             |红色             |8500            |2020-05-19T00:00:00.000Z
长虹             |蓝色             |4200            |2020-05-23T00:00:00.000Z
小米             |黑色             |4800            |2020-06-10T00:00:00.000Z
```

CSV

```
POST /_sql?format=csv
{
  "query": """
  SELECT * FROM "tvs"
  """
}
```

结果:

```
brand,color,price,sold_date
长虹,红色,1000,2019-10-28T00:00:00.000Z
长虹,红色,2000,2019-11-05T00:00:00.000Z
小米,绿色,3000,2019-05-18T00:00:00.000Z
TCL,蓝色,1500,2019-07-02T00:00:00.000Z
TCL,绿色,1200,2019-08-19T00:00:00.000Z
长虹,红色,2000,2019-11-05T00:00:00.000Z
三星,红色,8000,2020-01-01T00:00:00.000Z
小米,蓝色,2500,2020-02-12T00:00:00.000Z
小米,绿色,4500,2020-04-22T00:00:00.000Z
三星,蓝色,6100,2020-05-16T00:00:00.000Z
TCL,白色,2100,2020-05-17T00:00:00.000Z
小米,红色,8500,2020-05-19T00:00:00.000Z
长虹,蓝色,4200,2020-05-23T00:00:00.000Z
```

```
小米,黑色,4800,2020-06-10T00:00:00.000Z
```

json

```
POST /_sql?format=json
{
  "query": """
  SELECT * FROM "tvs"
  """
}
```

结果:


```
{
  "columns" : [
    {
      "name" : "brand",
      "type" : "keyword"
    },
    {
      "name" : "color",
      "type" : "keyword"
    },
    {
      "name" : "price",
      "type" : "long"
    },
    {
      "name" : "sold_date",
      "type" : "datetime"
    }
  ],
  "rows" : [
    [
      "长虹",
      "红色",
      1000,
      "2019-10-28T00:00:00.000Z"
    ],
    [
      "长虹",
      "红色",
      2000,
      "2019-11-05T00:00:00.000Z"
    ],
    [
      "小米",
      "绿色",
      3000,
      "2019-05-18T00:00:00.000Z"
    ],
    [
      "TCL",
```

```
      "蓝色",
      1500,
      "2019-07-02T00:00:00.000Z"
    ],
    [
      "TCL",
      "绿色",
      1200,
      "2019-08-19T00:00:00.000Z"
    ],
    [
      "长虹",
      "红色",
      2000,
      "2019-11-05T00:00:00.000Z"
    ],
    [
      "三星",
      "红色",
      8000,
      "2020-01-01T00:00:00.000Z"
    ],
    [
      "小米",
      "蓝色",
      2500,
      "2020-02-12T00:00:00.000Z"
    ],
    [
      "小米",
      "绿色",
      4500,
      "2020-04-22T00:00:00.000Z"
    ],
    [
      "三星",
      "蓝色",
      6100,
      "2020-05-16T00:00:00.000Z"
    ],
    [
      "TCL",
      "白色",
      2100,
      "2020-05-17T00:00:00.000Z"
    ],
    [
      "小米",
      "红色",
      8500,
      "2020-05-19T00:00:00.000Z"
    ],
    [
      "长虹",
      "蓝色",
      4200,
      "2020-05-23T00:00:00.000Z"
    ],
```

```
    [
      "小米",
      "黑色",
      4800,
      "2020-06-10T00:00:00.000Z"
    ]
  ]
}
```

tsv

```
POST /_sql?format=tsv
{
  "query": """
  SELECT * FROM "tvs"
  """
}
```

结果:

```
brand   color   price   sold_date
长虹  红色  1000    2019-10-28T00:00:00.000Z
长虹  红色  2000    2019-11-05T00:00:00.000Z
小米  绿色  3000    2019-05-18T00:00:00.000Z
TCL 蓝色  1500    2019-07-02T00:00:00.000Z
TCL 绿色  1200    2019-08-19T00:00:00.000Z
长虹  红色  2000    2019-11-05T00:00:00.000Z
三星  红色  8000    2020-01-01T00:00:00.000Z
小米  蓝色  2500    2020-02-12T00:00:00.000Z
小米  绿色  4500    2020-04-22T00:00:00.000Z
三星  蓝色  6100    2020-05-16T00:00:00.000Z
TCL 白色  2100    2020-05-17T00:00:00.000Z
小米  红色  8500    2020-05-19T00:00:00.000Z
长虹  蓝色  4200    2020-05-23T00:00:00.000Z
小米  黑色  4800    2020-06-10T00:00:00.000Z
```

yaml

```
POST /_sql?format=yaml
{
  "query": """
  SELECT * FROM "tvs"
  """
}
```

结果:

```yaml
---
columns:
- name: "brand"
  type: "keyword"
- name: "color"
  type: "keyword"
- name: "price"
  type: "long"
- name: "sold_date"
  type: "datetime"
rows:
- - "长虹"
  - "红色"
  - 1000
  - "2019-10-28T00:00:00.000Z"
- - "长虹"
  - "红色"
  - 2000
  - "2019-11-05T00:00:00.000Z"
- - "小米"
  - "绿色"
  - 3000
  - "2019-05-18T00:00:00.000Z"
- - "TCL"
  - "蓝色"
  - 1500
  - "2019-07-02T00:00:00.000Z"
- - "TCL"
  - "绿色"
  - 1200
  - "2019-08-19T00:00:00.000Z"
- - "长虹"
  - "红色"
  - 2000
  - "2019-11-05T00:00:00.000Z"
- - "三星"
  - "红色"
  - 8000
  - "2020-01-01T00:00:00.000Z"
- - "小米"
  - "蓝色"
  - 2500
  - "2020-02-12T00:00:00.000Z"
- - "小米"
  - "绿色"
  - 4500
  - "2020-04-22T00:00:00.000Z"
- - "三星"
  - "蓝色"
  - 6100
  - "2020-05-16T00:00:00.000Z"
- - "TCL"
  - "白色"
  - 2100
  - "2020-05-17T00:00:00.000Z"
- - "小米"
  - "红色"
```

```
  - 8500
  - "2020-05-19T00:00:00.000Z"
- - "长虹"
  - "蓝色"
  - 4200
  - "2020-05-23T00:00:00.000Z"
- - "小米"
  - "黑色"
  - 4800
  - "2020-06-10T00:00:00.000Z"
```

cbor

```
POST /_sql?format=cbor
{
  "query": """
  SELECT * FROM "tvs"
  """
}
```

结果:

```
gcolumns  dnameebranddtypegkeyword  dnameecolordtypegkeyword  dnameepricedtypedlong  dnameisold_datedtypehdatetime  drows  f长虹f红色  x2019-10-28T00:00:00.000Z  f长虹f红色  x2019-11-05T00:00:00.000Z  f小米f绿色  x2019-05-18T00:00:00.000Z  cTCLf蓝色  x2019-07-02T00:00:00.000Z  cTCLf绿色  x2019-08-19T00:00:00.000Z  f长虹f红色  x2019-11-05T00:00:00.000Z  f三星f红色  @x2020-01-01T00:00:00.000Z  f小米f蓝色  x2020-02-12T00:00:00.000Z  f小米f绿色  x2020-04-22T00:00:00.000Z  f三星f蓝色  x2020-05-16T00:00:00.000Z  cTCLf白色  4x2020-05-17T00:00:00.000Z  f小米f红色  !4x2020-05-19T00:00:00.000Z  f长虹f蓝色  hx2020-05-23T00:00:00.000Z  f小米f黑色  x2020-06-10T00:00:00.000Z
```

smile

```
POST /_sql?format=smile
{
  "query": """
  SELECT * FROM "tvs"
  """
}
```

结果:

```
:)
•��columns���nameDbrand�typeFkeyword��ADcolorBFkeyword��ADpriceBClong��AH
sold_dateBGdatetime���rows���长虹�红色$•�W2019-10-28T00:00:00.000Z���长虹�红
色$>�W2019-11-05T00:00:00.000Z���小米�绿色$]�W2019-05-18T00:00:00.000Z��BTCL�
蓝色$.�W2019-07-02T00:00:00.000Z��BTCL�绿色$%�W2019-08-19T00:00:00.000Z���长虹
�红色$>�W2019-11-05T00:00:00.000Z���三星�红色$•z�W2020-01-
01T00:00:00.000Z���小米�蓝色$N�W2020-02-12T00:00:00.000Z���小米�绿色
$••�W2020-04-22T00:00:00.000Z���三星�蓝色$•>�W2020-05-
16T00:00:00.000Z��BTCL�白色$A�W2020-05-17T00:00:00.000Z���小米�红色$• •W2020-
05-19T00:00:00.000Z���长虹�蓝色$••�W2020-05-23T00:00:00.000Z���小米�黑色
$••W2020-06-10T00:00:00.000Z���
```

sql翻译

```
POST /_sql/translate
{
  "query": """
  SELECT * FROM "tvs"
  """
}
```

结果:

```
{
  "size" : 1000,
  "_source" : false,
  "fields" : [
    {
      "field" : "brand"
    },
    {
      "field" : "color"
    },
    {
      "field" : "price"
    },
    {
      "field" : "sold_date",
      "format" : "strict_date_optional_time_nanos"
    }
  ],
  "sort" : [
    {
      "_doc" : {
        "order" : "asc"
      }
    }
  ]
}
```

sql与DSL结合

```
POST /_sql?format=txt
{
    "query": "SELECT * FROM tvs",
    "filter": {
        "range":
        {
            "price":
            {
                "gte" : 1200,
                "lte" : 3000
            }
        }
    }
}
```

结果:

```
     brand      |     color      |     price      |       sold_date
---------------+---------------+---------------+------------------------
长虹             |红色             |2000           |2019-11-05T00:00:00.000Z
小米             |绿色             |3000           |2019-05-18T00:00:00.000Z
TCL             |蓝色             |1500           |2019-07-02T00:00:00.000Z
TCL             |绿色             |1200           |2019-08-19T00:00:00.000Z
长虹             |红色             |2000           |2019-11-05T00:00:00.000Z
小米             |蓝色             |2500           |2020-02-12T00:00:00.000Z
TCL             |白色             |2100           |2020-05-17T00:00:00.000Z
```

java API实现

```
package mao.elasticsearch_jdbc_sql;

import org.junit.jupiter.api.AfterEach;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import org.springframework.boot.test.context.SpringBootTest;

import java.sql.*;

/**
 * Project name(项目名称): elasticsearch_JDBC_sql
 * Package(包名): mao.elasticsearch_jdbc_sql
```

```
 * Class(类名): ElasticSearchJDBCTest
 * Author(作者）: mao
 * Author QQ：1296193245
 * GitHub：https://github.com/maomao124/
 * Date(创建日期)： 2022/5/30
 * Time(创建时间)： 22:34
 * Version(版本): 1.0
 * Description(描述)： ElasticSearchJDBCTest
 */

@SpringBootTest
public class ElasticSearchJDBCTest
{
    private Connection connection;

    @BeforeEach
    void setUp() throws SQLException
    {
        //创建链接
        connection =
DriverManager.getConnection("jdbc:es://http://localhost:9200");
    }

    @AfterEach
    void tearDown() throws SQLException
    {
        //关闭链接
        connection.close();
    }

    @Test
    void select() throws SQLException
    {
        Statement statement = connection.createStatement();
        ResultSet resultSet = statement.executeQuery("select * from tvs");
        while (resultSet.next())
        {
            System.out.println(resultSet.getString(1));
            System.out.println(resultSet.getString(2));
            System.out.println(resultSet.getString(3));
            System.out.println(resultSet.getString(4));
            System.out.println("-----");
        }
    }
}
```

Logstash

什么是Logstash

logstash是一个数据抽取工具，将数据从一个地方转移到另一个地方。如hadoop生态圈的sqoop等。下载地址：https://www.elastic.co/cn/downloads/logstash

logstash之所以功能强大和流行，还与其丰富的过滤器插件是分不开的，过滤器提供的并不单单是过滤的功能，还可以对进入过滤器的原始数据进行复杂的逻辑处理，甚至添加独特的事件到后续流程中。

Logstash配置文件有如下三部分组成，其中input、output部分是必须配置，filter部分是可选配置，而filter就是过滤器插件，可以在这部分实现各种日志过滤功能。

配置文件

```
input {
    #输入插件
}
filter {
    #过滤匹配插件
}
output {
    #输出插件
}
```

启动操作

```
logstash -e 'input{stdin{}} output{stdout{}}'
```

```
PS H:\opensoft\logstash-8.1.3\bin> .\logstash -e 'input{stdin{}}
output{stdout{}}'
"Using bundled JDK: H:\opensoft\logstash-8.1.3\jdk\bin\java.exe"
OpenJDK 64-Bit Server VM warning: Option UseConcMarkSweepGC was deprecated in
version 9.0 and will likely be removed in a future release.
Sending Logstash logs to H:/opensoft/logstash-8.1.3/logs which is now configured
via log4j2.properties
[2022-05-31T13:01:22,223][INFO ][logstash.runner          ] Log4j configuration
path used is: H:\opensoft\logstash-8.1.3\config\log4j2.properties
[2022-05-31T13:01:22,267][WARN ][logstash.runner          ] The use of JAVA_HOME
has been deprecated. Logstash 8.0 and later ignores JAVA_HOME and uses the
bundled JDK. Running Logstash with the bundled JDK is recommended. The bundled
JDK has been verified to work with each specific version of Logstash, and
generally provides best performance and reliability. If you have compelling
reasons for using your own JDK (organizational-specific compliance requirements,
for example), you can configure LS_JAVA_HOME to use that version instead.
[2022-05-31T13:01:22,268][INFO ][logstash.runner          ] Starting Logstash
{"logstash.version"=>"8.1.3", "jruby.version"=>"jruby 9.2.20.1 (2.5.8) 2021-11-
30 2a2962fbd1 OpenJDK 64-Bit Server VM 11.0.14.1+1 on 11.0.14.1+1 +indy +jit
[mswin32-x86_64]"}
```

```
[2022-05-31T13:01:22,270][INFO ][logstash.runner          ] JVM bootstrap flags:
[-Xms1g, -Xmx4g, -XX:+UseConcMarkSweepGC, -XX:CMSInitiatingOccupancyFraction=75,
-XX:+UseCMSInitiatingOccupancyOnly, -Djava.awt.headless=true, -
Dfile.encoding=UTF-8, -Djruby.compile.invokedynamic=true, -
Djruby.jit.threshold=0, -Djruby.regexp.interruptible=true, -
XX:+HeapDumpOnOutOfMemoryError, -Djava.security.egd=file:/dev/urandom, -
Dlog4j2.isThreadContextMapInheritable=true, --add-
opens=java.base/java.security=ALL-UNNAMED, --add-opens=java.base/java.io=ALL-
UNNAMED, --add-opens=java.base/java.nio.channels=ALL-UNNAMED, --add-
opens=java.base/sun.nio.ch=ALL-UNNAMED, --add-
opens=java.management/sun.management=ALL-UNNAMED]
[2022-05-31T13:01:22,345][WARN ][logstash.config.source.multilocal] Ignoring the
'pipelines.yml' file because modules or command line options are specified
[2022-05-31T13:01:22,370][INFO ][logstash.agent           ] No persistent UUID
file found. Generating new UUID {:uuid=>"494789aa-3671-4c49-9cf9-048e1e097ec6",
:path=>"H:/opensoft/logstash-8.1.3/data/uuid"}
[2022-05-31T13:01:25,188][INFO ][logstash.agent           ] Successfully started
Logstash API endpoint {:port=>9600, :ssl_enabled=>false}
[2022-05-31T13:01:25,538][INFO ][org.reflections.Reflections] Reflections took
61 ms to scan 1 urls, producing 120 keys and 419 values
[2022-05-31T13:01:26,171][INFO ][logstash.javapipeline    ] Pipeline `main` is
configured with `pipeline.ecs_compatibility: v8` setting. All plugins in this
pipeline will default to `ecs_compatibility => v8` unless explicitly configured
otherwise.
[2022-05-31T13:01:26,259][INFO ][logstash.javapipeline    ][main] Starting
pipeline {:pipeline_id=>"main", "pipeline.workers"=>16,
"pipeline.batch.size"=>125, "pipeline.batch.delay"=>50,
"pipeline.max_inflight"=>2000, "pipeline.sources"=>["config string"],
:thread=>"#<Thread:0x6876925e run>"}
[2022-05-31T13:01:27,088][INFO ][logstash.javapipeline    ][main] Pipeline Java
execution initialization time {"seconds"=>0.83}
[2022-05-31T13:01:27,247][INFO ][logstash.javapipeline    ][main] Pipeline
started {"pipeline.id"=>"main"}
The stdin plugin is now waiting for input:
[2022-05-31T13:01:27,289][INFO ][logstash.agent           ] Pipelines running
{:count=>1, :running_pipelines=>[:main], :non_running_pipelines=>[]}
test
{
      "@version" => "1",
         "event" => {
        "original" => "test\r"
    },
          "host" => {
        "hostname" => "mao"
    },
       "message" => "test\r",
    "@timestamp" => 2022-05-31T05:01:40.200831200Z
}
```

启动

```
logstash.bat -f ../config/test.conf
```

Logstash输入插件

Plugin	Description	Github repository
azure_event_hubs	Receives events from Azure Event Hubs	azure_event_hubs
beats	Receives events from the Elastic Beats framework	logstash-input-beats
cloudwatch	Pulls events from the Amazon Web Services CloudWatch API	logstash-input-cloudwatch
couchdb_changes	Streams events from CouchDB's `_changes` URI	logstash-input-couchdb_changes
dead_letter_queue	read events from Logstash's dead letter queue	logstash-input-dead_letter_queue
elastic_agent	Receives events from the Elastic Agent framework	logstash-input-beats (shared)
elasticsearch	Reads query results from an Elasticsearch cluster	logstash-input-elasticsearch
exec	Captures the output of a shell command as an event	logstash-input-exec
file	Streams events from files	logstash-input-file
ganglia	Reads Ganglia packets over UDP	logstash-input-ganglia
gelf	Reads GELF-format messages from Graylog2 as events	logstash-input-gelf
generator	Generates random log events for test purposes	logstash-input-generator
github	Reads events from a GitHub webhook	logstash-input-github
google_cloud_storage	Extract events from files in a Google Cloud Storage bucket	logstash-input-google_cloud_storage
google_pubsub	Consume events from a Google Cloud PubSub service	logstash-input-google_pubsub
graphite	Reads metrics from the `graphite` tool	logstash-input-graphite
heartbeat	Generates heartbeat events for testing	logstash-input-heartbeat
http	Receives events over HTTP or HTTPS	logstash-input-http
http_poller	Decodes the output of an HTTP API into events	logstash-input-http_poller
imap	Reads mail from an IMAP server	logstash-input-imap
irc	Reads events from an IRC server	logstash-input-irc

Plugin	Description	Github repository
java_generator	Generates synthetic log events	core plugin
java_stdin	Reads events from standard input	core plugin
jdbc	Creates events from JDBC data	logstash-integration-jdbc
jms	Reads events from a Jms Broker	logstash-input-jms
jmx	Retrieves metrics from remote Java applications over JMX	logstash-input-jmx
kafka	Reads events from a Kafka topic	logstash-integration-kafka
kinesis	Receives events through an AWS Kinesis stream	logstash-input-kinesis
log4j	Reads events over a TCP socket from a Log4j `SocketAppender` object	logstash-input-log4j
lumberjack	Receives events using the Lumberjack protocl	logstash-input-lumberjack
meetup	Captures the output of command line tools as an event	logstash-input-meetup
pipe	Streams events from a long-running command pipe	logstash-input-pipe
puppet_facter	Receives facts from a Puppet server	logstash-input-puppet_facter
rabbitmq	Pulls events from a RabbitMQ exchange	logstash-integration-rabbitmq
redis	Reads events from a Redis instance	logstash-input-redis
relp	Receives RELP events over a TCP socket	logstash-input-relp
rss	Captures the output of command line tools as an event	logstash-input-rss
s3	Streams events from files in a S3 bucket	logstash-input-s3
s3-sns-sqs	Reads logs from AWS S3 buckets using sqs	logstash-input-s3-sns-sqs
salesforce	Creates events based on a Salesforce SOQL query	logstash-input-salesforce
snmp	Polls network devices using Simple Network Management Protocol (SNMP)	logstash-input-snmp
snmptrap	Creates events based on SNMP trap messages	logstash-input-snmptrap

Plugin	Description	Github repository
sqlite	Creates events based on rows in an SQLite database	logstash-input-sqlite
sqs	Pulls events from an Amazon Web Services Simple Queue Service queue	logstash-input-sqs
stdin	Reads events from standard input	logstash-input-stdin
stomp	Creates events received with the STOMP protocol	logstash-input-stomp
syslog	Reads syslog messages as events	logstash-input-syslog
tcp	Reads events from a TCP socket	logstash-input-tcp
twitter	Reads events from the Twitter Streaming API	logstash-input-twitter
udp	Reads events over UDP	logstash-input-udp
unix	Reads events over a UNIX socket	logstash-input-unix
varnishlog	Reads from the `varnish` cache shared memory log	logstash-input-varnishlog
websocket	Reads events from a websocket	logstash-input-websocket
wmi	Creates events based on the results of a WMI query	logstash-input-wmi
xmpp	Receives events over the XMPP/Jabber protocol	logstash-input-xmpp

标准输入(Stdin)

```
input{
    stdin{

    }
}
output {
    stdout{
        codec=>rubydebug
    }
}
```

文件(File)

```
input {
    file {
        path => ["/var/*/*"]
        start_position => "beginning"
    }
}
output {
    stdout{
        codec=>rubydebug
    }
}
```

TCP网络数据

```
input {
  tcp {
    port => "端口号"
  }
}

filter {
  grok {
    match => { "message" => "%{SYSLOGLINE}" }
  }
}

output {
    stdout{
        codec=>rubydebug
    }
}
```

Logstash过滤器插件

Plugin	Description	Github repository
age	Calculates the age of an event by subtracting the event timestamp from the current timestamp	logstash-filter-age
aggregate	Aggregates information from several events originating with a single task	logstash-filter-aggregate
alter	Performs general alterations to fields that the `mutate` filter does not handle	logstash-filter-alter
bytes	Parses string representations of computer storage sizes, such as "123 MB" or "5.6gb", into their numeric value in bytes	logstash-filter-bytes
cidr	Checks IP addresses against a list of network blocks	logstash-filter-cidr
cipher	Applies or removes a cipher to an event	logstash-filter-cipher
clone	Duplicates events	logstash-filter-clone
csv	Parses comma-separated value data into individual fields	logstash-filter-csv
date	Parses dates from fields to use as the Logstash timestamp for an event	logstash-filter-date
de_dot	Computationally expensive filter that removes dots from a field name	logstash-filter-de_dot
dissect	Extracts unstructured event data into fields using delimiters	logstash-filter-dissect
dns	Performs a standard or reverse DNS lookup	logstash-filter-dns
drop	Drops all events	logstash-filter-drop
elapsed	Calculates the elapsed time between a pair of events	logstash-filter-elapsed
elasticsearch	Copies fields from previous log events in Elasticsearch to current events	logstash-filter-elasticsearch
environment	Stores environment variables as metadata sub-fields	logstash-filter-environment
extractnumbers	Extracts numbers from a string	logstash-filter-extractnumbers
fingerprint	Fingerprints fields by replacing values with a consistent hash	logstash-filter-fingerprint

Plugin	Description	Github repository
geoip	Adds geographical information about an IP address	logstash-filter-geoip
grok	Parses unstructured event data into fields	logstash-filter-grok
http	Provides integration with external web services/REST APIs	logstash-filter-http
i18n	Removes special characters from a field	logstash-filter-i18n
java_uuid	Generates a UUID and adds it to each processed event	core plugin
jdbc_static	Enriches events with data pre-loaded from a remote database	logstash-integration-jdbc
jdbc_streaming	Enrich events with your database data	logstash-integration-jdbc
json	Parses JSON events	logstash-filter-json
json_encode	Serializes a field to JSON	logstash-filter-json_encode
kv	Parses key-value pairs	logstash-filter-kv
memcached	Provides integration with external data in Memcached	logstash-filter-memcached
metricize	Takes complex events containing a number of metrics and splits these up into multiple events, each holding a single metric	logstash-filter-metricize
metrics	Aggregates metrics	logstash-filter-metrics
mutate	Performs mutations on fields	logstash-filter-mutate
prune	Prunes event data based on a list of fields to blacklist or whitelist	logstash-filter-prune
range	Checks that specified fields stay within given size or length limits	logstash-filter-range
ruby	Executes arbitrary Ruby code	logstash-filter-ruby
sleep	Sleeps for a specified time span	logstash-filter-sleep
split	Splits multi-line messages, strings, or arrays into distinct events	logstash-filter-split
syslog_pri	Parses the `PRI` (priority) field of a `syslog` message	logstash-filter-syslog_pri

Plugin	Description	Github repository
threats_classifier	Enriches security logs with information about the attacker's intent	logstash-filter-threats_classifier
throttle	Throttles the number of events	logstash-filter-throttle
tld	Replaces the contents of the default message field with whatever you specify in the configuration	logstash-filter-tld
translate	Replaces field contents based on a hash or YAML file	logstash-filter-translate
truncate	Truncates fields longer than a given length	logstash-filter-truncate
urldecode	Decodes URL-encoded fields	logstash-filter-urldecode
useragent	Parses user agent strings into fields	logstash-filter-useragent
uuid	Adds a UUID to events	logstash-filter-uuid
wurfl_device_detection	Enriches logs with device information such as brand, model, OS	logstash-filter-wurfl_device_detection
xml	Parses XML into fields	logstash-filter-xml

Grok 正则捕获

语法：

```
%{语法：语义}
```

输入的内容：

```
172.16.213.132 [07/Feb/2019:16:24:19 +0800] "GET / HTTP/1.1" 403 5039
```

%{IP:clientip}匹配模式将获得的结果为：clientip: 172.16.213.132
%{HTTPDATE:timestamp}匹配模式将获得的结果为：timestamp: 07/Feb/2018:16:24:19 +0800
而%{QS:referrer}匹配模式将获得的结果为：referrer: "GET / HTTP/1.1"

结果：

```
%{IP:clientip}\ \[%{HTTPDATE:timestamp}\]\ %{QS:referrer}\ %{NUMBER:response}\ %{NUMBER:bytes}
```

```
input{
    stdin{}
}
filter{
    grok{
        match => ["message","%{IP:clientip}\ \[%{HTTPDATE:timestamp}\]\ %{QS:referrer}\ %{NUMBER:response}\ %{NUMBER:bytes}"]
    }
}
output{
    stdout{
        codec => "rubydebug"
    }
}
```

时间处理

```
filter {
    grok {
        match => ["message", "%{HTTPDATE:timestamp}"]
    }
    date {
        match => ["timestamp", "dd/MMM/yyyy:HH:mm:ss Z"]
    }
}
```

数据修改

将filed_name_1字段中所有"/"字符替换为"_"

```
filter {
    mutate {
        gsub => ["filed_name_1", "/" , "_"]
    }
}
```

分隔符分割字符串为数组

```
filter {
    mutate {
        split => ["filed_name_2", "|"]
    }
}
```

重命名字段

```
filter {
    mutate {
        rename => { "old_field" => "new_field" }
    }
}
```

删除字段

```
filter {
    mutate {
        remove_field  =>  ["timestamp"]
    }
}
```

GeoIP 地址查询归类

```
filter {
    geoip {
        source => "ip_field"
    }
}
```

综合示例

```
input
{
    stdin {}
}
```

```
filter
{
    grok
    {
        match => { "message" => "%{IP:clientip}\ \[%{HTTPDATE:timestamp}\]\ %
{QS:referrer}\ %{NUMBER:response}\ %{NUMBER:bytes}" }
        remove_field => [ "message" ]
   }
date
{
        match => ["timestamp", "dd/MMM/yyyy:HH:mm:ss Z"]
    }
mutate {
          convert => [ "response","float" ]
           rename => { "response" => "response_new" }
           gsub => ["referrer","\"",""]
           split => ["clientip", "."]
        }
}
output
{
    stdout
    {
        codec => "rubydebug"
    }
```

Logstash输出插件

Plugin	Description	Github repository
app_search	Sends events to the Elastic App Search solution	logstash-integration-elastic_enterprise_search
boundary	Sends annotations to Boundary based on Logstash events	logstash-output-boundary
circonus	Sends annotations to Circonus based on Logstash events	logstash-output-circonus
cloudwatch	Aggregates and sends metric data to AWS CloudWatch	logstash-output-cloudwatch
csv	Writes events to disk in a delimited format	logstash-output-csv
datadog	Sends events to DataDogHQ based on Logstash events	logstash-output-datadog
datadog_metrics	Sends metrics to DataDogHQ based on Logstash events	logstash-output-datadog_metrics
dynatrace	Sends events to Dynatrace based on Logstash events	logstash-output-dynatrace
elastic_app_search	Sends events to the Elastic App Search solution	logstash-integration-elastic_enterprise_search
elastic_workplace_search	Sends events to the Elastic Workplace Search solution	logstash-integration-elastic_enterprise_search
elasticsearch	Stores logs in Elasticsearch	logstash-output-elasticsearch
email	Sends email to a specified address when output is received	logstash-output-email
exec	Runs a command for a matching event	logstash-output-exec
file	Writes events to files on disk	logstash-output-file
ganglia	Writes metrics to Ganglia's `gmond`	logstash-output-ganglia
gelf	Generates GELF formatted output for Graylog2	logstash-output-gelf
google_bigquery	Writes events to Google BigQuery	logstash-output-google_bigquery
google_cloud_storage	Uploads log events to Google Cloud Storage	logstash-output-google_cloud_storage
google_pubsub	Uploads log events to Google Cloud Pubsub	logstash-output-google_pubsub
graphite	Writes metrics to Graphite	logstash-output-graphite

Plugin	Description	Github repository
graphtastic	Sends metric data on Windows	logstash-output-graphtastic
http	Sends events to a generic HTTP or HTTPS endpoint	logstash-output-http
influxdb	Writes metrics to InfluxDB	logstash-output-influxdb
irc	Writes events to IRC	logstash-output-irc
java_stdout	Prints events to the STDOUT of the shell	core plugin
juggernaut	Pushes messages to the Juggernaut websockets server	logstash-output-juggernaut
kafka	Writes events to a Kafka topic	logstash-integration-kafka
librato	Sends metrics, annotations, and alerts to Librato based on Logstash events	logstash-output-librato
loggly	Ships logs to Loggly	logstash-output-loggly
lumberjack	Sends events using the `lumberjack` protocol	logstash-output-lumberjack
metriccatcher	Writes metrics to MetricCatcher	logstash-output-metriccatcher
mongodb	Writes events to MongoDB	logstash-output-mongodb
nagios	Sends passive check results to Nagios	logstash-output-nagios
nagios_nsca	Sends passive check results to Nagios using the NSCA protocol	logstash-output-nagios_nsca
opentsdb	Writes metrics to OpenTSDB	logstash-output-opentsdb
pagerduty	Sends notifications based on preconfigured services and escalation policies	logstash-output-pagerduty
pipe	Pipes events to another program's standard input	logstash-output-pipe
rabbitmq	Pushes events to a RabbitMQ exchange	logstash-integration-rabbitmq
redis	Sends events to a Redis queue using the `RPUSH` command	logstash-output-redis

Plugin	Description	Github repository
redmine	Creates tickets using the Redmine API	logstash-output-redmine
riak	Writes events to the Riak distributed key/value store	logstash-output-riak
riemann	Sends metrics to Riemann	logstash-output-riemann
s3	Sends Logstash events to the Amazon Simple Storage Service	logstash-output-s3
sink	Discards any events received	core plugin
sns	Sends events to Amazon's Simple Notification Service	logstash-output-sns
solr_http	Stores and indexes logs in Solr	logstash-output-solr_http
sqs	Pushes events to an Amazon Web Services Simple Queue Service queue	logstash-output-sqs
statsd	Sends metrics using the `statsd` network daemon	logstash-output-statsd
stdout	Prints events to the standard output	logstash-output-stdout
stomp	Writes events using the STOMP protocol	logstash-output-stomp
syslog	Sends events to a `syslog` server	logstash-output-syslog
tcp	Writes events over a TCP socket	logstash-output-tcp
timber	Sends events to the Timber.io logging service	logstash-output-timber
udp	Sends events over UDP	logstash-output-udp
webhdfs	Sends Logstash events to HDFS using the `webhdfs` REST API	logstash-output-webhdfs
websocket	Publishes messages to a websocket	logstash-output-websocket
workplace_search	Sends events to the Elastic Workplace Search solution	logstash-integration-elastic_enterprise_search
xmpp	Posts events over XMPP	logstash-output-xmpp
zabbix	Sends events to a Zabbix server	logstash-output-zabbix

输出到标准输出

```
output {
    stdout {
        codec => rubydebug
    }
}
```

保存为文件

```
output {
    file {
        path => "/data/log/%{+yyyy-MM-dd}/%{host}_%{+HH}.log"
    }
}
```

输出到elasticsearch

```
output {
    elasticsearch {
        host => ["127.0.0.1:9200","127.0.0.1:9201"]
        index => "logstash-%{+YYYY.MM.dd}"
    }
}
```

- host：是一个数组类型的值，后面跟的值是elasticsearch节点的地址与端口，默认端口是9200。可添加多个地址。
- index：写入elasticsearch的索引的名称，这里可以使用变量。Logstash提供了%{+YYYY.MM.dd}这种写法。在语法解析的时候，看到以+ 号开头的，就会自动认为后面是时间格式，尝试用时间格式来解析后续字符串。这种以天为单位分割的写法，可以很容易的删除老的数据或者搜索指定时间范围内的数据。此外，注意索引名中不能有大写字母。
- manage_template:用来设置是否开启logstash自动管理模板功能，如果设置为false将关闭自动管理模板功能。如果我们自定义了模板，那么应该设置为false。
- template_name:这个配置项用来设置在Elasticsearch中模板的名称。

示例

```
input {
    file {
        path => ["D:/ES/logstash-7.3.0/nginx.log"]
        start_position => "beginning"
    }
```

```
}

filter {
    grok {
        match => { "message" => "%{IP:clientip}\ \[%{HTTPDATE:timestamp}\]\ %
{QS:referrer}\ %{NUMBER:response}\ %{NUMBER:bytes}" }
        remove_field => [ "message" ]
   }
    date {
        match => ["timestamp", "dd/MMM/yyyy:HH:mm:ss Z"]
    }
    mutate {
           rename => { "response" => "response_new" }
           convert => [ "response","float" ]
           gsub => ["referrer","\"",""]
           remove_field => ["timestamp"]
           split => ["clientip", "."]
        }
}

output {
    stdout {
        codec => "rubydebug"
    }

elasticsearch {
    host => ["localhost:9200"]
    index => "logstash-%{+YYYY.MM.dd}"
}

}
```

高亮显示

语法:

```
GET /{index}/_search
{
  "query": {
    "match": {
      "FIELD": "TEXT" // 查询条件，高亮一定要使用全文检索查询
    }
  },
  "highlight": {
    "fields": { // 指定要高亮的字段
      "FIELD": {
        "pre_tags": "<em>",  // 用来标记高亮字段的前置标签
        "post_tags": "</em>" // 用来标记高亮字段的后置标签
      }
```

```
    }
  }
}
```

- 高亮是对关键字高亮，因此**搜索条件必须带有关键字**，而不能是范围这样的查询。
- 默认情况下，**高亮的字段，必须与搜索指定的字段一致**，否则无法高亮
- 如果要对非搜索字段高亮，则需要添加一个属性：required_field_match=false

```
GET /tvs/_search
{
  "query": {"match": {
    "brand": "小米"
  }},
  "highlight": {
    "fields": {
      "brand": {

      }
    }
  }
}
```

结果：

```
{
  "took" : 1,
  "timed_out" : false,
  "_shards" : {
    "total" : 1,
    "successful" : 1,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : {
      "value" : 5,
      "relation" : "eq"
    },
    "max_score" : 1.0033021,
    "hits" : [
      {
        "_index" : "tvs",
        "_id" : "7aouDoEBEpQthbP41cfj",
        "_score" : 1.0033021,
        "_source" : {
          "price" : 3000,
          "color" : "绿色",
          "brand" : "小米",
          "sold_date" : "2019-05-18"
        },
        "highlight" : {
          "brand" : [
            "<em>小米</em>"
```

```
        ]
      }
    },
    {
      "_index" : "tvs",
      "_id" : "8qouDoEBEpQthbP41cfj",
      "_score" : 1.0033021,
      "_source" : {
        "price" : 2500,
        "color" : "蓝色",
        "brand" : "小米",
        "sold_date" : "2020-02-12"
      },
      "highlight" : {
        "brand" : [
          "<em>小米</em>"
        ]
      }
    },
    {
      "_index" : "tvs",
      "_id" : "86ouDoEBEpQthbP41cfj",
      "_score" : 1.0033021,
      "_source" : {
        "price" : 4500,
        "color" : "绿色",
        "brand" : "小米",
        "sold_date" : "2020-04-22"
      },
      "highlight" : {
        "brand" : [
          "<em>小米</em>"
        ]
      }
    },
    {
      "_index" : "tvs",
      "_id" : "9qouDoEBEpQthbP41cfj",
      "_score" : 1.0033021,
      "_source" : {
        "price" : 8500,
        "color" : "红色",
        "brand" : "小米",
        "sold_date" : "2020-05-19"
      },
      "highlight" : {
        "brand" : [
          "<em>小米</em>"
        ]
      }
    },
    {
      "_index" : "tvs",
      "_id" : "-KouDoEBEpQthbP41cfj",
      "_score" : 1.0033021,
      "_source" : {
        "price" : 4800,
        "color" : "黑色",
```

```
          "brand" : "小米",
          "sold_date" : "2020-06-10"
        },
        "highlight" : {
          "brand" : [
            "<em>小米</em>"
          ]
        }
      }
    ]
  }
}
```

```
GET /book/_search
{
  "query": {
    "match": {
      "description": "java"
    }
  },
  "highlight": {
    "fields": {
      "description": {

      }
    }
  }
}
```

结果:

```
{
  "took" : 2,
  "timed_out" : false,
  "_shards" : {
    "total" : 1,
    "successful" : 1,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : {
      "value" : 4,
      "relation" : "eq"
    },
    "max_score" : 0.4745544,
    "hits" : [
      {
        "_index" : "book",
        "_id" : "3",
        "_score" : 0.4745544,
        "_source" : {
          "name" : "spring开发基础",
```

```
      "description" : "spring 在java领域非常流行，java程序员都在用。",
      "studymodel" : "201001",
      "price" : 78.6,
      "timestamp" : "2019-08-24 19:21:35",
      "pic" : "group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg",
      "tags" : [
        "spring",
        "java"
      ]
    },
    "highlight" : {
      "description" : [
        "spring 在<em>java</em>领域非常流行，<em>java</em>程序员都在用。"
      ]
    }
  },
  {
    "_index" : "book",
    "_id" : "2",
    "_score" : 0.33773077,
    "_source" : {
      "name" : "java编程思想",
      "description" : "java语言是世界第一编程语言，在软件开发领域使用人数最多。",
      "studymodel" : "201001",
      "price" : 68.6,
      "timestamp" : "2019-08-25 19:11:35",
      "pic" : "group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg",
      "tags" : [
        "java",
        "dev"
      ]
    },
    "highlight" : {
      "description" : [
        "<em>java</em>语言是世界第一编程语言，在软件开发领域使用人数最多。"
      ]
    }
  },
  {
    "_index" : "book",
    "_id" : "5",
    "_score" : 0.33773077,
    "_source" : {
      "name" : "java编程思想",
      "description" : "java语言是世界第一编程语言，在软件开发领域使用人数最多。",
      "studymodel" : "201001",
      "price" : 68.6,
      "timestamp" : "2022-5-25 19:11:35",
      "pic" : "group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg",
      "tags" : [
        "bootstrap",
        "dev"
      ]
    },
    "highlight" : {
      "description" : [
        "<em>java</em>语言是世界第一编程语言，在软件开发领域使用人数最多。"
      ]
```

```
        }
      },
      {
        "_index" : "book",
        "_id" : "6",
        "_score" : 0.33773077,
        "_source" : {
          "name" : "java编程思想",
          "description" : "java语言是世界第一编程语言，在软件开发领域使用人数最多。",
          "studymodel" : "201001",
          "price" : 68.6,
          "timestamp" : "2022-5-25 19:11:35",
          "pic" : "group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg",
          "tags" : [
            "bootstrap",
            "dev"
          ]
        },
        "highlight" : {
          "description" : [
            "<em>java</em>语言是世界第一编程语言，在软件开发领域使用人数最多。"
          ]
        }
      }
    ]
  }
}
```

自定义标签：

```
GET /book/_search
{
  "query": {
    "multi_match":
    {
      "query": "java",
      "fields": ["name","description"]
    }
  },
  "highlight":
  {
    "fields":
    {
      "description":
      {
        "pre_tags": "<div class=\"s\">",
        "post_tags": "</div>",
        "require_field_match": "false"
      },
      "name":
      {
        "pre_tags": "<div class=\"s\">",
        "post_tags": "</div>",
```

```
        "require_field_match": "false"
      }
    }
  }
}
```

结果:

```
{
  "took" : 1,
  "timed_out" : false,
  "_shards" : {
    "total" : 1,
    "successful" : 1,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : {
      "value" : 4,
      "relation" : "eq"
    },
    "max_score" : 0.52048135,
    "hits" : [
      {
        "_index" : "book",
        "_id" : "2",
        "_score" : 0.52048135,
        "_source" : {
          "name" : "java编程思想",
          "description" : "java语言是世界第一编程语言，在软件开发领域使用人数最多。",
          "studymodel" : "201001",
          "price" : 68.6,
          "timestamp" : "2019-08-25 19:11:35",
          "pic" : "group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg",
          "tags" : [
            "java",
            "dev"
          ]
        },
        "highlight" : {
          "name" : [
            """<div class="s">java</div>编程思想"""
          ],
          "description" : [
            """<div class="s">java</div>语言是世界第一编程语言，在软件开发领域使用人数最多。"""
          ]
        }
      },
      {
        "_index" : "book",
        "_id" : "5",
        "_score" : 0.52048135,
        "_source" : {
          "name" : "java编程思想",
          "description" : "java语言是世界第一编程语言，在软件开发领域使用人数最多。",
```

```
        "studymodel" : "201001",
        "price" : 68.6,
        "timestamp" : "2022-5-25 19:11:35",
        "pic" : "group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg",
        "tags" : [
          "bootstrap",
          "dev"
        ]
      },
      "highlight" : {
        "name" : [
          """<div class="s">java</div>编程思想"""
        ],
        "description" : [
          """<div class="s">java</div>语言是世界第一编程语言，在软件开发领域使用人数最多。"""
        ]
      }
    },
    {
      "_index" : "book",
      "_id" : "6",
      "_score" : 0.52048135,
      "_source" : {
        "name" : "java编程思想",
        "description" : "java语言是世界第一编程语言，在软件开发领域使用人数最多。",
        "studymodel" : "201001",
        "price" : 68.6,
        "timestamp" : "2022-5-25 19:11:35",
        "pic" : "group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg",
        "tags" : [
          "bootstrap",
          "dev"
        ]
      },
      "highlight" : {
        "name" : [
          """<div class="s">java</div>编程思想"""
        ],
        "description" : [
          """<div class="s">java</div>语言是世界第一编程语言，在软件开发领域使用人数最多。"""
        ]
      }
    },
    {
      "_index" : "book",
      "_id" : "3",
      "_score" : 0.4745544,
      "_source" : {
        "name" : "spring开发基础",
        "description" : "spring 在java领域非常流行，java程序员都在用。",
        "studymodel" : "201001",
        "price" : 78.6,
        "timestamp" : "2019-08-24 19:21:35",
        "pic" : "group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg",
        "tags" : [
          "spring",
```

```
            "java"
          ]
        },
        "highlight" : {
          "description" : [
            """spring 在<div class="s">java</div>领域非常流行，<div
class="s">java</div>程序员都在用。"""
          ]
        }
      }
    ]
  }
}
```

java API实现高亮显示

```
package mao.elasticsearch_implement_highlight;

import org.apache.http.HttpHost;
import org.elasticsearch.action.search.SearchRequest;
import org.elasticsearch.action.search.SearchResponse;
import org.elasticsearch.client.RequestOptions;
import org.elasticsearch.client.RestClient;
import org.elasticsearch.client.RestHighLevelClient;
import org.elasticsearch.index.query.QueryBuilders;
import org.elasticsearch.search.SearchHit;
import org.elasticsearch.search.SearchHits;
import org.elasticsearch.search.builder.SearchSourceBuilder;
import org.elasticsearch.search.fetch.subphase.highlight.HighlightBuilder;
import org.elasticsearch.search.fetch.subphase.highlight.HighlightField;
import org.junit.jupiter.api.AfterAll;
import org.junit.jupiter.api.BeforeAll;
import org.junit.jupiter.api.Test;
import org.springframework.boot.test.context.SpringBootTest;

import java.io.IOException;
import java.util.List;
import java.util.Map;

/**
 * Project name(项目名称)：elasticsearch_Implement_highlight
 * Package(包名): mao.elasticsearch_implement_highlight
 * Class(类名): ElasticSearchTest
 * Author(作者）: mao
 * Author QQ：1296193245
 * GitHub：https://github.com/maomao124/
 * Date(创建日期)： 2022/6/1
 * Time(创建时间)： 13:55
 * Version(版本): 1.0
 * Description(描述)： 测试高亮
```

```
 */


@SpringBootTest
public class ElasticSearchTest
{
    private static RestHighLevelClient client;

    /**
     * Before all.
     */
    @BeforeAll
    static void beforeAll()
    {
        client = new RestHighLevelClient(RestClient.builder(
                new HttpHost("localhost", 9200, "http")));
    }

    /**
     * After all.
     *
     * @throws IOException the io exception
     */
    @AfterAll
    static void afterAll() throws IOException
    {
        client.close();
    }

    /**
     * 测试高亮
     * <p>
     * 请求内容:
     * <pre>
     *
     * GET /book/_search
     * {
     *   "query": {
     *     "multi_match":
     *     {
     *       "query": "java",
     *       "fields": ["name","description"]
     *     }
     *   },
     *   "highlight":
     *   {
     *     "fields":
     *     {
     *       "description":
     *       {
     *         "pre_tags": "<div class=\"s\">",
     *         "post_tags": "</div>",
     *         "require_field_match": "false"
     *       },
     *       "name":
     *       {
     *         "pre_tags": "<div class=\"s\">",
     *         "post_tags": "</div>",
```

```
     *         "require_field_match": "false"
     *       }
     *     }
     *   }
     * }
     *
     * </pre>
     * <p>
     * 结果:
     * <pre>
     *
     * {
     *   "took" : 1,
     *   "timed_out" : false,
     *   "_shards" : {
     *     "total" : 1,
     *     "successful" : 1,
     *     "skipped" : 0,
     *     "failed" : 0
     *   },
     *   "hits" : {
     *     "total" : {
     *       "value" : 4,
     *       "relation" : "eq"
     *     },
     *     "max_score" : 0.52048135,
     *     "hits" : [
     *       {
     *         "_index" : "book",
     *         "_id" : "2",
     *         "_score" : 0.52048135,
     *         "_source" : {
     *           "name" : "java编程思想",
     *           "description" : "java语言是世界第一编程语言，在软件开发领域使用人数最
多。",
     *           "studymodel" : "201001",
     *           "price" : 68.6,
     *           "timestamp" : "2019-08-25 19:11:35",
     *           "pic" : "group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg",
     *           "tags" : [
     *             "java",
     *             "dev"
     *           ]
     *         },
     *         "highlight" : {
     *           "name" : [
     *             """<div class="s">java</div>编程思想"""
     *           ],
     *           "description" : [
     *             """<div class="s">java</div>语言是世界第一编程语言，在软件开发领域使
用人数最多。"""
     *           ]
     *         }
     *       },
     *       {
     *         "_index" : "book",
     *         "_id" : "5",
     *         "_score" : 0.52048135,
```

```
     *          "_source" : {
     *            "name" : "java编程思想",
     *            "description" : "java语言是世界第一编程语言，在软件开发领域使用人数最
多。",
     *            "studymodel" : "201001",
     *            "price" : 68.6,
     *            "timestamp" : "2022-5-25 19:11:35",
     *            "pic" : "group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg",
     *            "tags" : [
     *              "bootstrap",
     *              "dev"
     *            ]
     *          },
     *          "highlight" : {
     *            "name" : [
     *              """<div class="s">java</div>编程思想"""
     *            ],
     *            "description" : [
     *              """<div class="s">java</div>语言是世界第一编程语言，在软件开发领域使
用人数最多。"""
     *            ]
     *          }
     *        },
     *        {
     *          "_index" : "book",
     *          "_id" : "6",
     *          "_score" : 0.52048135,
     *          "_source" : {
     *            "name" : "java编程思想",
     *            "description" : "java语言是世界第一编程语言，在软件开发领域使用人数最
多。",
     *            "studymodel" : "201001",
     *            "price" : 68.6,
     *            "timestamp" : "2022-5-25 19:11:35",
     *            "pic" : "group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg",
     *            "tags" : [
     *              "bootstrap",
     *              "dev"
     *            ]
     *          },
     *          "highlight" : {
     *            "name" : [
     *              """<div class="s">java</div>编程思想"""
     *            ],
     *            "description" : [
     *              """<div class="s">java</div>语言是世界第一编程语言，在软件开发领域使
用人数最多。"""
     *            ]
     *          }
     *        },
     *        {
     *          "_index" : "book",
     *          "_id" : "3",
     *          "_score" : 0.4745544,
     *          "_source" : {
     *            "name" : "spring开发基础",
     *            "description" : "spring 在java领域非常流行，java程序员都在用。",
     *            "studymodel" : "201001",
```

```
     *           "price" : 78.6,
     *           "timestamp" : "2019-08-24 19:21:35",
     *           "pic" : "group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg",
     *           "tags" : [
     *             "spring",
     *             "java"
     *           ]
     *         },
     *         "highlight" : {
     *           "description" : [
     *             """spring 在<div class="s">java</div>领域非常流行，<div
class="s">java</div>程序员都在用。"""
     *           ]
     *         }
     *       }
     *     ]
     *   }
     * }
     *
     * </pre>
     * <p>
     * 程序结果：
     * <pre>
     *
     * 总数：4
     * 最大分数：0.52048135
     * 数据：
     * ----price：68.6
     * ----studymodel：201001
     * ----name：java编程思想
     * ----description：java语言是世界第一编程语言，在软件开发领域使用人数最多。
     * ----pic：group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
     * ----timestamp：2019-08-25 19:11:35
     * ----tags：[java, dev]
     * ------高亮：
     * ------name：<div class=\"s\">java</div>编程思想
     * ------description：<div class=\"s\">java</div>语言是世界第一编程语言，在软件开发
领域使用人数最多。
     * ----------------------------------------
     * ----price：68.6
     * ----studymodel：201001
     * ----name：java编程思想
     * ----description：java语言是世界第一编程语言，在软件开发领域使用人数最多。
     * ----pic：group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
     * ----timestamp：2022-5-25 19:11:35
     * ----tags：[bootstrap, dev]
     * ------高亮：
     * ------name：<div class=\"s\">java</div>编程思想
     * ------description：<div class=\"s\">java</div>语言是世界第一编程语言，在软件开发
领域使用人数最多。
     * ----------------------------------------
     * ----price：68.6
     * ----studymodel：201001
     * ----name：java编程思想
     * ----description：java语言是世界第一编程语言，在软件开发领域使用人数最多。
     * ----pic：group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
     * ----timestamp：2022-5-25 19:11:35
     * ----tags：[bootstrap, dev]
```

```
     * ------高亮：
     * ------name：<div class=\"s\">java</div>编程思想
     * ------description：<div class=\"s\">java</div>语言是世界第一编程语言，在软件开发领域使用人数最多。
     * ----------------------------------
     * ----price：78.6
     * ----studymodel：201001
     * ----name：spring开发基础
     * ----description：spring 在java领域非常流行，java程序员都在用。
     * ----pic：group1/M00/00/00/wKhlQFs6RCeAY0pHAAJx5ZjNDEM428.jpg
     * ----timestamp：2019-08-24 19:21:35
     * ----tags：[spring, java]
     * ------高亮：
     * ------description：spring 在<div class=\"s\">java</div>领域非常流行，<div class=\"s\">java</div>程序员都在用。
     * ----------------------------------
     *
     * </pre>
     *
     * @throws Exception Exception
     */
    @Test
    void highlight() throws Exception
    {
        //构建请求
        SearchRequest searchRequest = new SearchRequest("book");
        //构建请求体
        SearchSourceBuilder searchSourceBuilder = new SearchSourceBuilder();
        //查询
        searchSourceBuilder.query(QueryBuilders.multiMatchQuery("java", "name", "description"));
        //高亮
        /*HighlightBuilder highlightBuilder = new HighlightBuilder();
        List<HighlightBuilder.Field> fields = highlightBuilder.fields();
        fields.add(new HighlightBuilder.Field("name")
                .requireFieldMatch(false)
                .preTags("<div class=\\\"s\\\">")
                .postTags("</div>"));
        fields.add(new HighlightBuilder.Field("description")
                .requireFieldMatch(false)
                .preTags("<div class=\\\"s\\\">")
                .postTags("</div>"));
        searchSourceBuilder.highlighter(highlightBuilder);*/

        //方法二
        searchSourceBuilder.highlighter(new HighlightBuilder()
                .field("name")
                .requireFieldMatch(false)
                .preTags("<div class=\\\"s\\\">")
                .postTags("</div>")
                .field("description")
                .requireFieldMatch(false)
                .preTags("<div class=\\\"s\\\">")
                .postTags("</div>"));

        //放入到请求中
        searchRequest.source(searchSourceBuilder);
        //发起请求
```

```
        SearchResponse searchResponse = client.search(searchRequest,
RequestOptions.DEFAULT);
        //获取数据
        SearchHits hits = searchResponse.getHits();
        //总数
        long total = hits.getTotalHits().value;
        //最大分数
        float maxScore = hits.getMaxScore();
        //数据
        SearchHit[] hitsHits = hits.getHits();
        //打印
        System.out.println("总数：" + total);
        System.out.println("最大分数：" + maxScore);
        System.out.println("数据：");
        for (SearchHit hitsHit : hitsHits)
        {
            //数据
            Map<String, Object> sourceAsMap = hitsHit.getSourceAsMap();
            for (String key : sourceAsMap.keySet())
            {
                Object value = sourceAsMap.get(key);
                System.out.println("----" + key + "：" + value);
            }
            System.out.println("------高亮：");
            Map<String, HighlightField> highlightFields =
hitsHit.getHighlightFields();
            for (String key : highlightFields.keySet())
            {
                HighlightField highlightField = highlightFields.get(key);
                String name = highlightField.getName();
                String field = highlightField.getFragments()[0].string();
                System.out.println("------" + name + "：" + field);
            }
            System.out.println("---------------------------------------");

        }

    }
}
```

地理坐标

使用场景：

- 搜索我附近的酒店
- 搜索我附近的出租车
- 搜索我附近的人
- ...

```
PUT /my_locations
```

```
{
  "mappings": {
    "properties": {
      "pin": {
        "properties": {
          "location": {
            "type": "geo_point"
          }
        }
      }
    }
  }
}

PUT /my_locations/_doc/1
{
  "pin": {
    "location": {
      "lat": 40.12,
      "lon": -71.34
    }
  }
}

PUT /my_geoshapes
{
  "mappings": {
    "properties": {
      "pin": {
        "properties": {
          "location": {
            "type": "geo_shape"
          }
        }
      }
    }
  }
}

PUT /my_geoshapes/_doc/1
{
  "pin": {
    "location": {
      "type" : "polygon",
      "coordinates" : [[[13.0 ,51.5], [15.0, 51.5], [15.0, 54.0], [13.0, 54.0],
[13.0 ,51.5]]]
    }
  }
}
```

矩形范围查询

查询时，需要指定矩形的**左上**、**右下**两个点的坐标，然后画出一个矩形，落在该矩形内的都是符合条件的点。

```
GET /indexName/_search
{
  "query": {
    "geo_bounding_box": {
      "FIELD": {
        "top_left": { // 左上点
          "lat": 31.1,
          "lon": 121.5
        },
        "bottom_right": { // 右下点
          "lat": 30.9,
          "lon": 121.7
        }
      }
    }
  }
}
```

附近查询

附近查询，也叫做距离查询（geo_distance）：查询到指定中心点小于某个距离值的所有文档。

```
GET /indexName/_search
{
  "query": {
    "geo_distance": {
      "distance": "15km", // 半径
      "FIELD": "31.21,121.5" // 圆心
    }
  }
}
```

排序

```
GET /indexName/_search
{
  "query": {
    "match_all": {}
  },
  "sort": [
    {
      "_geo_distance" : {
          "FIELD" : "纬度，经度", // 文档中geo_point类型的字段名、目标坐标点
          "order" : "asc", // 排序方式
          "unit" : "km" // 排序的距离单位
      }
    }
  ]
```

```
}
```

- 指定一个坐标，作为目标点
- 计算每一个文档中，指定字段（必须是geo_point类型）的坐标 到目标点的距离是多少
- 根据距离排序

自动补全

拼音分词器

要实现根据字母做补全，就必须对文档按照拼音分词。

拼音分词插件地址：https://github.com/medcl/elasticsearch-analysis-pinyin

测试：

```
POST /_analyze
{
  "text": "测试字段",
  "analyzer": "pinyin"
}
```

自定义分词器

elasticsearch中分词器（analyzer）的组成包含三部分：

- character filters：在tokenizer之前对文本进行处理。例如删除字符、替换字符
- tokenizer：将文本按照一定的规则切割成词条（term）。例如keyword，就是不分词；还有ik_smart
- tokenizer filter：将tokenizer输出的词条做进一步处理。例如大小写转换、同义词处理、拼音处理等

自定义分词器：

```
PUT /test
{
  "settings": {
    "analysis": {
      "analyzer": { // 自定义分词器
        "my_analyzer": {  // 分词器名称
          "tokenizer": "ik_max_word",
          "filter": "py"
        }
      },
      "filter": { // 自定义tokenizer filter
        "py": { // 过滤器名称
          "type": "pinyin", // 过滤器类型，这里是pinyin
```

```
          "keep_full_pinyin": false,
          "keep_joined_full_pinyin": true,
          "keep_original": true,
          "limit_first_letter_length": 16,
          "remove_duplicated_term": true,
          "none_chinese_pinyin_tokenize": false
        }
      }
    }
  },
  "mappings": {
    "properties": {
      "name": {
        "type": "text",
        "analyzer": "my_analyzer",
        "search_analyzer": "ik_smart"
      }
    }
  }
}
```

自动补全查询

elasticsearch提供了Completion Suggester查询来实现自动补全功能。这个查询会匹配以用户输入内容开头的词条并返回。为了提高补全查询的效率，对于文档中字段的类型有一些约束：

- 参与补全查询的字段必须是completion类型。
- 字段的内容一般是用来补全的多个词条形成的数组。

示例：

```
put test2
{
  "mappings":
  {
    "properties":
    {
      "title":
      {
        "type":"completion"
      }
    }
  }
}
```

插入数据：

```
POST test/_doc
{
```

```
 "title": ["Sony", "WH-1000XM3"]
}

POST test/_doc
{
 "title": ["SK-II", "PITERA"]
}

POST test/_doc
{
 "title": ["Nintendo", "switch"]
}
```

查询：

```
GET /test2/_search
{
  "suggest":
  {
    "title_suggest":
    {
      "text": "s",
      "completion":
      {
        "field": "title",
        "skip_duplicates": true,
        "size": 10
      }
    }
  }
}
```

- text：关键字
- field：补全查询的字段
- skip_duplicates：是否跳过重复的
- size：只获取前n条结果

结果：

```
{
  "took" : 1,
  "timed_out" : false,
  "_shards" : {
    "total" : 1,
    "successful" : 1,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : {
      "value" : 0,
```

```
      "relation" : "eq"
    },
    "max_score" : null,
    "hits" : [ ]
  },
  "suggest" : {
    "title_suggest" : [
      {
        "text" : "s",
        "offset" : 0,
        "length" : 1,
        "options" : [
          {
            "text" : "SK-II",
            "_index" : "test2",
            "_id" : "nVUsH4EBfwatmgrgIAsV",
            "_score" : 1.0,
            "_source" : {
              "title" : [
                "SK-II",
                "PITERA"
              ]
            }
          },
          {
            "text" : "Sony",
            "_index" : "test2",
            "_id" : "nFUsH4EBfwatmgrgFgu5",
            "_score" : 1.0,
            "_source" : {
              "title" : [
                "Sony",
                "WH-1000XM3"
              ]
            }
          },
          {
            "text" : "switch",
            "_index" : "test2",
            "_id" : "nlUsH4EBfwatmgrgKAsR",
            "_score" : 1.0,
            "_source" : {
              "title" : [
                "Nintendo",
                "switch"
              ]
            }
          }
        ]
      }
    ]
  }
}
```

java API 实现自动补全

```java
package mao.elasticsearch_implement_automatic_completion;

import org.apache.http.HttpHost;
import org.elasticsearch.action.search.SearchRequest;
import org.elasticsearch.action.search.SearchResponse;
import org.elasticsearch.client.RequestOptions;
import org.elasticsearch.client.RestClient;
import org.elasticsearch.client.RestHighLevelClient;
import org.elasticsearch.search.builder.SearchSourceBuilder;
import org.elasticsearch.search.suggest.Suggest;
import org.elasticsearch.search.suggest.SuggestBuilder;
import org.elasticsearch.search.suggest.completion.CompletionSuggestionBuilder;
import org.junit.jupiter.api.AfterAll;
import org.junit.jupiter.api.BeforeAll;
import org.junit.jupiter.api.Test;
import org.springframework.boot.test.context.SpringBootTest;

import java.io.IOException;
import java.util.List;
import java.util.Scanner;

/**
 * Project name(项目名称)：elasticsearch_Implement_automatic_completion
 * Package(包名): mao.elasticsearch_implement_automatic_completion
 * Class(类名): ElasticSearchTest
 * Author(作者）: mao
 * Author QQ：1296193245
 * GitHub：https://github.com/maomao124/
 * Date(创建日期)： 2022/6/1
 * Time(创建时间)： 20:19
 * Version(版本): 1.0
 * Description(描述)： 测试自动补全
 */


@SpringBootTest
public class ElasticSearchTest
{

    private static RestHighLevelClient client;

    /**
     * Before all.
     */
    @BeforeAll
    static void beforeAll()
    {
        client = new RestHighLevelClient(RestClient.builder(
                new HttpHost("localhost", 9200, "http")
        ));
    }
```

```
/**
 * After all.
 *
 * @throws IOException the io exception
 */
@AfterAll
static void afterAll() throws IOException
{
    client.close();
}

/**
 * 测试自动补全功能
 * <p>
 * 请求内容:
 * <pre>
 *
 * GET /test2/_search
 * {
 *   "suggest":
 *   {
 *     "title_suggest":
 *     {
 *       "text": "s",
 *       "completion":
 *       {
 *         "field": "title",
 *         "skip_duplicates": true,
 *         "size": 10
 *       }
 *     }
 *   }
 * }
 *
 * </pre>
 * <p>
 * 结果:
 * <pre>
 *
 * {
 *   "took" : 1,
 *   "timed_out" : false,
 *   "_shards" : {
 *     "total" : 1,
 *     "successful" : 1,
 *     "skipped" : 0,
 *     "failed" : 0
 *   },
 *   "hits" : {
 *     "total" : {
 *       "value" : 0,
 *       "relation" : "eq"
 *     },
 *     "max_score" : null,
 *     "hits" : [ ]
 *   },
 *   "suggest" : {
```

```
*     "title_suggest" : [
*       {
*         "text" : "s",
*         "offset" : 0,
*         "length" : 1,
*         "options" : [
*           {
*             "text" : "SK-II",
*             "_index" : "test2",
*             "_id" : "nVUsH4EBfwatmgrgIAsV",
*             "_score" : 1.0,
*             "_source" : {
*               "title" : [
*                 "SK-II",
*                 "PITERA"
*               ]
*             }
*           },
*           {
*             "text" : "Sony",
*             "_index" : "test2",
*             "_id" : "nFUsH4EBfwatmgrgFgu5",
*             "_score" : 1.0,
*             "_source" : {
*               "title" : [
*                 "Sony",
*                 "WH-1000XM3"
*               ]
*             }
*           },
*           {
*             "text" : "switch",
*             "_index" : "test2",
*             "_id" : "nlUsH4EBfwatmgrgKAsR",
*             "_score" : 1.0,
*             "_source" : {
*               "title" : [
*                 "Nintendo",
*                 "switch"
*               ]
*             }
*           }
*         ]
*       }
*     ]
*   }
* }
*
* </pre>
* <p>
* 程序结果：
* <pre>
*
* 补全字段：s
* 结果：
* -->SK-II
* -->Sony
* -->switch
```

```
     *
     * </pre>
     *
     * @throws Exception Exception
     */
    @Test
    void automatic_completion() throws Exception
    {
        //构建请求
        SearchRequest searchRequest = new SearchRequest("test2");
        //构建请求体
        SearchSourceBuilder searchSourceBuilder = new SearchSourceBuilder();
        //补全
        searchSourceBuilder.suggest(new SuggestBuilder().addSuggestion(
                "title_suggest",
                new
CompletionSuggestionBuilder("title").text("s").skipDuplicates(true).size(10)));
        //放入到请求中
        searchRequest.source(searchSourceBuilder);
        //发起请求
        SearchResponse searchResponse = client.search(searchRequest,
RequestOptions.DEFAULT);
        //获取数据
        //获取suggest部分
        Suggest suggest = searchResponse.getSuggest();
        Suggest.Suggestion<? extends Suggest.Suggestion.Entry<? extends
Suggest.Suggestion.Entry.Option>>
                title_suggest = suggest.getSuggestion("title_suggest");
        List<? extends Suggest.Suggestion.Entry<? extends
Suggest.Suggestion.Entry.Option>> entries = title_suggest.getEntries();
        for (Suggest.Suggestion.Entry<? extends Suggest.Suggestion.Entry.Option>
entry : entries)
        {
            String text = entry.getText().string();
            System.out.println("补全字段：" + text);
            System.out.println("结果：");
            for (Suggest.Suggestion.Entry.Option option : entry)
            {
                String result = option.getText().string();
                System.out.println("-->" + result);
            }
        }
    }


    /**
     * 自动补全
     *
     * @param automaticCompletionText 自动补全的文本
     * @throws Exception Exception
     */
    void automaticCompletionService(String automaticCompletionText) throws
Exception
    {
        //构建请求
        SearchRequest searchRequest = new SearchRequest("test2");
        //构建请求体
        SearchSourceBuilder searchSourceBuilder = new SearchSourceBuilder();
```

```
        //补全
        searchSourceBuilder.suggest(new SuggestBuilder().addSuggestion(
                "title_suggest",
                new
CompletionSuggestionBuilder("title").text(automaticCompletionText).skipDuplicate
s(true).size(10)));
        //放入到请求中
        searchRequest.source(searchSourceBuilder);
        //发起请求
        SearchResponse searchResponse = client.search(searchRequest,
RequestOptions.DEFAULT);
        //获取数据
        //获取suggest部分
        Suggest suggest = searchResponse.getSuggest();
        Suggest.Suggestion<? extends Suggest.Suggestion.Entry<? extends
Suggest.Suggestion.Entry.Option>>
                title_suggest = suggest.getSuggestion("title_suggest");
        List<? extends Suggest.Suggestion.Entry<? extends
Suggest.Suggestion.Entry.Option>> entries = title_suggest.getEntries();
        for (Suggest.Suggestion.Entry<? extends Suggest.Suggestion.Entry.Option>
entry : entries)
        {
            String text = entry.getText().string();
            System.out.println("补全字段：" + text);
            System.out.println("结果：");
            for (Suggest.Suggestion.Entry.Option option : entry)
            {
                String result = option.getText().string();
                System.out.println("-->" + result);
            }
        }
    }

    public static void main(String[] args) throws Exception
    {
        beforeAll();
        while (true)
        {
            Scanner input = new Scanner(System.in);
            System.out.print("请输入要补全的关键字：");
            String inputString = input.next();
            if (inputString.equals("exit"))
            {
                afterAll();
                return;
            }
            new ElasticSearchTest().automaticCompletionService(inputString);
            System.out.println();
            System.out.println("--------");
            System.out.println();
        }
    }
}
```

数据同步

一般情况下，elasticsearch中的数据来自于mysql数据库，因此mysql数据发生改变时，elasticsearch也必须跟着改变，这个就是elasticsearch与mysql之间的**数据同步。**

数据同步方案有：

- 同步调用
- 异步通知
- 监听binlog
- 使用Logstash

同步调用

对数据库进行修改操作时，同时修改elasticsearch

异步通知

数据库数据完成增、删、改后，发送MQ消息

然后监听MQ，接收到消息后完成elasticsearch数据修改

监听binlog

给mysql开启binlog功能

mysql完成增、删、改操作都会记录在binlog中

使用canal监听binlog变化，实时更新elasticsearch中的内容

选择

方式一：同步调用

- 优点：实现简单，粗暴
- 缺点：业务耦合度高

方式二：异步通知

- 优点：低耦合，实现难度一般
- 缺点：依赖mq的可靠性

方式三：监听binlog

- 优点：完全解除服务间耦合
- 缺点：开启binlog增加数据库负担、实现复杂度高

java API 实现数据同步

代码在实战学习笔记里

cerebro

https://github.com/cerebroapp/cerebro

下载地址：

https://github.com/lmenezes/cerebro/releases

双击其中的cerebro.bat文件即可启动服务。

访问http://localhost:9000 即可进入管理界面

输入你的elasticsearch的任意节点的地址和端口，点击connect即可

绿色的条，代表集群处于绿色（健康状态）

elasticsearch集群

windows实现

1. 创建文件夹

创建 一个elasticsearch-cluster 文件夹，在内部复制三个 elasticsearch 服务

文件夹名称分别为elasticsearch1、elasticsearch2和elasticsearch3

2. 删除data和logs文件夹

3. 修改配置文件

进入elasticsearch1/config目录，找到elasticsearch.yml文件，修改配置文件为：

```
#节点 1 的配置信息：
#集群名称，节点之间要保持一致
cluster.name: my-elasticsearch
#节点名称，集群内要唯一
node.name: node-9201

# 配置该结点有资格被选举为主结点（候选主结点），用于处理请求和管理集群。如果结点没有资格成为主结点，那么该结点永远不可能成为主结点；如果结点有资格成为主结点，只有在被其他候选主结点认可和被选举为主结点之后，才真正成为主结点。
node.master: true
# 配置该结点是数据结点，用于保存数据，执行数据相关的操作（CRUD，Aggregation）
node.data: true

#ip 地址
network.host: localhost
#http 端口
http.port: 9201
#tcp 监听端口
transport.tcp.port: 9301

discovery.seed_hosts: ["localhost:9302","localhost:9303"]
discovery.zen.fd.ping_timeout: 1m
```

```
discovery.zen.fd.ping_retries: 5

#集群内的可以被选为主节点的节点列表
cluster.initial_master_nodes: ["node-9201", "node-9202","node-9203"]

#跨域配置
#action.destructive_requires_name: true
http.cors.enabled: true
http.cors.allow-origin: "*"
```

进入elasticsearch2/config目录，找到elasticsearch.yml文件，修改配置文件为：

```
#节点 2 的配置信息：
#集群名称，节点之间要保持一致
cluster.name: my-elasticsearch
#节点名称，集群内要唯一
node.name: node-9202
# 配置该结点有资格被选举为主结点（候选主结点），用于处理请求和管理集群。如果结点没有资格成为主结点，那么该结点永远不可能成为主结点；如果结点有资格成为主结点，只有在被其他候选主结点认可和被选举为主结点之后，才真正成为主结点。
node.master: true
# 配置该结点是数据结点，用于保存数据，执行数据相关的操作（CRUD，Aggregation）
node.data: true

#ip 地址
network.host: localhost
#http 端口
http.port: 9202
#tcp 监听端口
transport.tcp.port: 9302

discovery.seed_hosts: ["localhost:9301","localhost:9303"]
discovery.zen.fd.ping_timeout: 1m
discovery.zen.fd.ping_retries: 5

#集群内的可以被选为主节点的节点列表
cluster.initial_master_nodes: ["node-9201", "node-9202","node-9203"]

#跨域配置
#action.destructive_requires_name: true
http.cors.enabled: true
http.cors.allow-origin: "*"
```

进入elasticsearch3/config目录，找到elasticsearch.yml文件，修改配置文件为：

```
#节点 3 的配置信息：
#集群名称，节点之间要保持一致
cluster.name: my-elasticsearch

#节点名称，集群内要唯一
node.name: node-9203
```

```
# 配置该结点有资格被选举为主结点（候选主结点），用于处理请求和管理集群。如果结点没有资格成为主结点，那么该结点永远不可能成为主结点；如果结点有资格成为主结点，只有在被其他候选主结点认可和被选举为主结点之后，才真正成为主结点。
node.master: true
# 配置该结点是数据结点，用于保存数据，执行数据相关的操作（CRUD，Aggregation）
node.data: true

#ip 地址
network.host: localhost
#http 端口
http.port: 9203
#tcp 监听端口
transport.tcp.port: 9303

#候选主节点的地址，在开启服务后可以被选为主节点
discovery.seed_hosts: ["localhost:9301", "localhost:9302"]
discovery.zen.fd.ping_timeout: 1m
discovery.zen.fd.ping_retries: 5

#集群内的可以被选为主节点的节点列表
cluster.initial_master_nodes: ["node-9201", "node-9202","node-9203"]

#跨域配置
#action.destructive_requires_name: true
http.cors.enabled: true
http.cors.allow-origin: "*"
```

注意：

如果你的版本比较高，配置上面的配置无效，因为部分配置项不存在，部分配置项已经更改名称，我使用的8.1.3版本，这一步浪费了我很多时间，下面的是我的配置：

9201：

```
#节点 1 的配置信息：
#集群名称，节点之间要保持一致
cluster.name:  my-elasticsearch
#节点名称，集群内要唯一
node.name:  node-9201

# 配置该结点有资格被选举为主结点（候选主结点），用于处理请求和管理集群。如果结点没有资格成为主结点，那么该结点永远不可能成为主结点；如果结点有资格成为主结点，只有在被其他候选主结点认可和被选举为主结点之后，才真正成为主结点。
#node.master: true
# 配置该结点是数据结点，用于保存数据，执行数据相关的操作（CRUD，Aggregation）
#node.data:  true

#ip 地址
network.host: localhost
#http 端口
http.port:  9201
#tcp 监听端口
```

```
transport.host: localhost
transport.port:  9301

discovery.seed_hosts: ["localhost:9302","localhost:9303"]
# discovery.zen.fd.ping_timeout: 1m
# discovery.zen.fd.ping_retries:  5

#集群内的可以被选为主节点的节点列表
cluster.initial_master_nodes: ["node-9201", "node-9202","node-9203"]

#跨域配置
#action.destructive_requires_name: true
http.cors.enabled: true
http.cors.allow-origin: "*"

xpack.security.enabled: false
xpack.security.enrollment.enabled: true
xpack.security.http.ssl:
  enabled: false
  keystore.path: certs/http.p12

xpack.security.transport.ssl:
  enabled: true
  verification_mode: certificate
  keystore.path: certs/transport.p12
  truststore.path: certs/transport.p12
```

9202:

```
#节点 2 的配置信息:
#集群名称，节点之间要保持一致
cluster.name: my-elasticsearch
#节点名称，集群内要唯一
node.name: node-9202
# 配置该结点有资格被选举为主结点（候选主结点），用于处理请求和管理集群。如果结点没有资格成为主结点，那么该结点永远不可能成为主结点；如果结点有资格成为主结点，只有在被其他候选主结点认可和被选举为主结点之后，才真正成为主结点。
#node.master: true
# 配置该结点是数据结点，用于保存数据，执行数据相关的操作（CRUD，Aggregation）
#node.data: true

#ip 地址
network.host: localhost
#http 端口
http.port: 9202
#tcp 监听端口
transport.host: localhost
transport.port: 9302

discovery.seed_hosts: ["localhost:9301","localhost:9303"]
#discovery.zen.fd.ping_timeout: 1m
#discovery.zen.fd.ping_retries: 5

#集群内的可以被选为主节点的节点列表
cluster.initial_master_nodes: ["node-9201", "node-9202","node-9203"]
```

```
#跨域配置
#action.destructive_requires_name: true
http.cors.enabled: true
http.cors.allow-origin: "*"


xpack.security.enabled: false
xpack.security.enrollment.enabled: true
xpack.security.http.ssl:
  enabled: false
  keystore.path: certs/http.p12

xpack.security.transport.ssl:
  enabled: true
  verification_mode: certificate
  keystore.path: certs/transport.p12
  truststore.path: certs/transport.p12
```

9303:

```
#节点 3 的配置信息：
#集群名称，节点之间要保持一致
cluster.name: my-elasticsearch

#节点名称，集群内要唯一
node.name: node-9203
# 配置该结点有资格被选举为主结点（候选主结点），用于处理请求和管理集群。如果结点没有资格成为主结点，那么该结点永远不可能成为主结点；如果结点有资格成为主结点，只有在被其他候选主结点认可和被选举为主结点之后，才真正成为主结点。
#node.master: true
# 配置该结点是数据结点，用于保存数据，执行数据相关的操作（CRUD，Aggregation）
#node.data: true

#ip 地址
network.host: localhost
#http 端口
http.port: 9203
#tcp 监听端口
transport.host: localhost
transport.port: 9303

#候选主节点的地址，在开启服务后可以被选为主节点
discovery.seed_hosts: ["localhost:9301", "localhost:9302"]
#discovery.zen.fd.ping_timeout: 1m
#discovery.zen.fd.ping_retries: 5

#集群内的可以被选为主节点的节点列表
cluster.initial_master_nodes: ["node-9201", "node-9202","node-9203"]

#跨域配置
#action.destructive_requires_name: true
http.cors.enabled: true
http.cors.allow-origin: "*"
```

```
xpack.security.enabled: false
xpack.security.enrollment.enabled: true
xpack.security.http.ssl:
  enabled: false
  keystore.path: certs/http.p12

xpack.security.transport.ssl:
  enabled: true
  verification_mode: certificate
  keystore.path: certs/transport.p12
  truststore.path: certs/transport.p12
```

4. 启动集群

依次双击执行elasticsearch1 ,elasticsearch2,elasticsearch3的 bin/elasticsearch.bat, 启动节点服务器，启动后，会自动加入指定名称的集群

9201节点：

```
warning: ignoring JAVA_HOME=C:\Users\mao\.jdks\openjdk-16.0.2; using bundled JDK
warning: ignoring JAVA_HOME=C:\Users\mao\.jdks\openjdk-16.0.2; using
ES_JAVA_HOME
[2022-06-01T22:41:01,088][INFO ][o.e.n.Node               ] [node-9201]
version[8.1.3], pid[6816],
build[default/zip/39afaa3c0fe7db4869a161985e240bd7182d7a07/2022-04-
19T08:13:25.444693396Z], OS[Windows 10/10.0/amd64], JVM[Eclipse Adoptium/OpenJDK
64-Bit Server VM/18/18+36]
[2022-06-01T22:41:01,093][INFO ][o.e.n.Node               ] [node-9201] JVM home
[H:\opensoft\elasticsearch-cluster\elasticsearch1\jdk], using bundled JDK [true]
[2022-06-01T22:41:01,094][INFO ][o.e.n.Node               ] [node-9201] JVM
arguments [-Des.networkaddress.cache.ttl=60, -
Des.networkaddress.cache.negative.ttl=10, -Djava.security.manager=allow, -
XX:+AlwaysPreTouch, -Xss1m, -Djava.awt.headless=true, -Dfile.encoding=UTF-8, -
Djna.nosys=true, -XX:-OmitStackTraceInFastThrow, -
XX:+ShowCodeDetailsInExceptionMessages, -Dio.netty.noUnsafe=true, -
Dio.netty.noKeySetOptimization=true, -Dio.netty.recycler.maxCapacityPerThread=0,
-Dio.netty.allocator.numDirectArenas=0, -Dlog4j.shutdownHookEnabled=false, -
Dlog4j2.disable.jmx=true, -Dlog4j2.formatMsgNoLookups=true, -
Djava.locale.providers=SPI,COMPAT, --add-opens=java.base/java.io=ALL-UNNAMED, -
Xms1g, -Xmx8g, -XX:+UseG1GC, -
Djava.io.tmpdir=C:\Users\mao\AppData\Local\Temp\elasticsearch, -
XX:+HeapDumpOnOutOfMemoryError, -XX:+ExitOnOutOfMemoryError, -
XX:HeapDumpPath=data, -XX:ErrorFile=logs/hs_err_pid%p.log, -
Xlog:gc*,gc+age=trace,safepoint:file=logs/gc.log:utctime,pid,tags:filecount=32,f
ilesize=64m, -XX:MaxDirectMemorySize=4294967296, -
XX:InitiatingHeapOccupancyPercent=30, -XX:G1ReservePercent=25, -Delasticsearch,
-Des.path.home=H:\opensoft\elasticsearch-cluster\elasticsearch1, -
Des.path.conf=H:\opensoft\elasticsearch-cluster\elasticsearch1\config, -
Des.distribution.flavor=default, -Des.distribution.type=zip, -
Des.bundled_jdk=true]
```

```
[2022-06-01T22:41:04,106][INFO ][o.e.p.PluginsService     ] [node-9201] loaded
module [aggs-matrix-stats]
[2022-06-01T22:41:04,107][INFO ][o.e.p.PluginsService     ] [node-9201] loaded
module [analysis-common]
[2022-06-01T22:41:04,107][INFO ][o.e.p.PluginsService     ] [node-9201] loaded
module [constant-keyword]
[2022-06-01T22:41:04,107][INFO ][o.e.p.PluginsService     ] [node-9201] loaded
module [data-streams]
[2022-06-01T22:41:04,108][INFO ][o.e.p.PluginsService     ] [node-9201] loaded
module [frozen-indices]
[2022-06-01T22:41:04,108][INFO ][o.e.p.PluginsService     ] [node-9201] loaded
module [ingest-common]
[2022-06-01T22:41:04,108][INFO ][o.e.p.PluginsService     ] [node-9201] loaded
module [ingest-geoip]
[2022-06-01T22:41:04,108][INFO ][o.e.p.PluginsService     ] [node-9201] loaded
module [ingest-user-agent]
[2022-06-01T22:41:04,109][INFO ][o.e.p.PluginsService     ] [node-9201] loaded
module [kibana]
[2022-06-01T22:41:04,109][INFO ][o.e.p.PluginsService     ] [node-9201] loaded
module [lang-expression]
[2022-06-01T22:41:04,109][INFO ][o.e.p.PluginsService     ] [node-9201] loaded
module [lang-mustache]
[2022-06-01T22:41:04,109][INFO ][o.e.p.PluginsService     ] [node-9201] loaded
module [lang-painless]
[2022-06-01T22:41:04,110][INFO ][o.e.p.PluginsService     ] [node-9201] loaded
module [legacy-geo]
[2022-06-01T22:41:04,110][INFO ][o.e.p.PluginsService     ] [node-9201] loaded
module [mapper-extras]
[2022-06-01T22:41:04,110][INFO ][o.e.p.PluginsService     ] [node-9201] loaded
module [mapper-version]
[2022-06-01T22:41:04,110][INFO ][o.e.p.PluginsService     ] [node-9201] loaded
module [old-lucene-versions]
[2022-06-01T22:41:04,111][INFO ][o.e.p.PluginsService     ] [node-9201] loaded
module [parent-join]
[2022-06-01T22:41:04,111][INFO ][o.e.p.PluginsService     ] [node-9201] loaded
module [percolator]
[2022-06-01T22:41:04,111][INFO ][o.e.p.PluginsService     ] [node-9201] loaded
module [rank-eval]
[2022-06-01T22:41:04,111][INFO ][o.e.p.PluginsService     ] [node-9201] loaded
module [reindex]
[2022-06-01T22:41:04,112][INFO ][o.e.p.PluginsService     ] [node-9201] loaded
module [repositories-metering-api]
[2022-06-01T22:41:04,112][INFO ][o.e.p.PluginsService     ] [node-9201] loaded
module [repository-azure]
[2022-06-01T22:41:04,112][INFO ][o.e.p.PluginsService     ] [node-9201] loaded
module [repository-encrypted]
[2022-06-01T22:41:04,112][INFO ][o.e.p.PluginsService     ] [node-9201] loaded
module [repository-gcs]
[2022-06-01T22:41:04,112][INFO ][o.e.p.PluginsService     ] [node-9201] loaded
module [repository-s3]
[2022-06-01T22:41:04,113][INFO ][o.e.p.PluginsService     ] [node-9201] loaded
module [repository-url]
[2022-06-01T22:41:04,113][INFO ][o.e.p.PluginsService     ] [node-9201] loaded
module [runtime-fields-common]
[2022-06-01T22:41:04,113][INFO ][o.e.p.PluginsService     ] [node-9201] loaded
module [search-business-rules]
[2022-06-01T22:41:04,114][INFO ][o.e.p.PluginsService     ] [node-9201] loaded
module [searchable-snapshots]
```

```
[2022-06-01T22:41:04,114][INFO ][o.e.p.PluginsService     ] [node-9201] loaded
module [snapshot-based-recoveries]
[2022-06-01T22:41:04,114][INFO ][o.e.p.PluginsService     ] [node-9201] loaded
module [snapshot-repo-test-kit]
[2022-06-01T22:41:04,114][INFO ][o.e.p.PluginsService     ] [node-9201] loaded
module [spatial]
[2022-06-01T22:41:04,115][INFO ][o.e.p.PluginsService     ] [node-9201] loaded
module [transform]
[2022-06-01T22:41:04,115][INFO ][o.e.p.PluginsService     ] [node-9201] loaded
module [transport-netty4]
[2022-06-01T22:41:04,115][INFO ][o.e.p.PluginsService     ] [node-9201] loaded
module [unsigned-long]
[2022-06-01T22:41:04,115][INFO ][o.e.p.PluginsService     ] [node-9201] loaded
module [vector-tile]
[2022-06-01T22:41:04,115][INFO ][o.e.p.PluginsService     ] [node-9201] loaded
module [vectors]
[2022-06-01T22:41:04,116][INFO ][o.e.p.PluginsService     ] [node-9201] loaded
module [wildcard]
[2022-06-01T22:41:04,116][INFO ][o.e.p.PluginsService     ] [node-9201] loaded
module [x-pack-aggregate-metric]
[2022-06-01T22:41:04,116][INFO ][o.e.p.PluginsService     ] [node-9201] loaded
module [x-pack-analytics]
[2022-06-01T22:41:04,116][INFO ][o.e.p.PluginsService     ] [node-9201] loaded
module [x-pack-async]
[2022-06-01T22:41:04,116][INFO ][o.e.p.PluginsService     ] [node-9201] loaded
module [x-pack-async-search]
[2022-06-01T22:41:04,117][INFO ][o.e.p.PluginsService     ] [node-9201] loaded
module [x-pack-autoscaling]
[2022-06-01T22:41:04,117][INFO ][o.e.p.PluginsService     ] [node-9201] loaded
module [x-pack-ccr]
[2022-06-01T22:41:04,117][INFO ][o.e.p.PluginsService     ] [node-9201] loaded
module [x-pack-core]
[2022-06-01T22:41:04,117][INFO ][o.e.p.PluginsService     ] [node-9201] loaded
module [x-pack-deprecation]
[2022-06-01T22:41:04,117][INFO ][o.e.p.PluginsService     ] [node-9201] loaded
module [x-pack-enrich]
[2022-06-01T22:41:04,118][INFO ][o.e.p.PluginsService     ] [node-9201] loaded
module [x-pack-eql]
[2022-06-01T22:41:04,118][INFO ][o.e.p.PluginsService     ] [node-9201] loaded
module [x-pack-fleet]
[2022-06-01T22:41:04,118][INFO ][o.e.p.PluginsService     ] [node-9201] loaded
module [x-pack-graph]
[2022-06-01T22:41:04,119][INFO ][o.e.p.PluginsService     ] [node-9201] loaded
module [x-pack-identity-provider]
[2022-06-01T22:41:04,119][INFO ][o.e.p.PluginsService     ] [node-9201] loaded
module [x-pack-ilm]
[2022-06-01T22:41:04,119][INFO ][o.e.p.PluginsService     ] [node-9201] loaded
module [x-pack-logstash]
[2022-06-01T22:41:04,119][INFO ][o.e.p.PluginsService     ] [node-9201] loaded
module [x-pack-ml]
[2022-06-01T22:41:04,119][INFO ][o.e.p.PluginsService     ] [node-9201] loaded
module [x-pack-monitoring]
[2022-06-01T22:41:04,119][INFO ][o.e.p.PluginsService     ] [node-9201] loaded
module [x-pack-ql]
[2022-06-01T22:41:04,120][INFO ][o.e.p.PluginsService     ] [node-9201] loaded
module [x-pack-rollup]
[2022-06-01T22:41:04,120][INFO ][o.e.p.PluginsService     ] [node-9201] loaded
module [x-pack-security]
```

```
[2022-06-01T22:41:04,120][INFO ][o.e.p.PluginsService     ] [node-9201] loaded
module [x-pack-shutdown]
[2022-06-01T22:41:04,120][INFO ][o.e.p.PluginsService     ] [node-9201] loaded
module [x-pack-sql]
[2022-06-01T22:41:04,121][INFO ][o.e.p.PluginsService     ] [node-9201] loaded
module [x-pack-stack]
[2022-06-01T22:41:04,121][INFO ][o.e.p.PluginsService     ] [node-9201] loaded
module [x-pack-text-structure]
[2022-06-01T22:41:04,121][INFO ][o.e.p.PluginsService     ] [node-9201] loaded
module [x-pack-voting-only-node]
[2022-06-01T22:41:04,121][INFO ][o.e.p.PluginsService     ] [node-9201] loaded
module [x-pack-watcher]
[2022-06-01T22:41:04,122][INFO ][o.e.p.PluginsService     ] [node-9201] no
plugins loaded
[2022-06-01T22:41:04,689][INFO ][o.e.e.NodeEnvironment    ] [node-9201] using [1]
data paths, mounts [[鏂囨欢 鏃?(H:)]], net usable_space [76.7gb], net total_space
[195.3gb], types [NTFS]

[2022-06-01T22:41:04,690][INFO ][o.e.e.NodeEnvironment    ] [node-9201] heap size
[8gb], compressed ordinary object pointers [true]
[2022-06-01T22:41:04,743][INFO ][o.e.n.Node               ] [node-9201] node
name [node-9201], node ID [Q3EYFIhCR3K2BHnvunZ7Eg], cluster name [my-
elasticsearch], roles [ml, data_hot, transform, data_content, data_warm, master,
remote_cluster_client, data, data_cold, ingest, data_frozen]
[2022-06-01T22:41:09,077][INFO ][o.e.x.m.p.l.CppLogMessageHandler] [node-9201]
[controller/5832] [Main.cc@123] controller (64 bit): Version 8.1.3 (Build
92d8267e6ebfb7) Copyright (c) 2022 Elasticsearch BV
[2022-06-01T22:41:09,363][INFO ][o.e.x.s.Security         ] [node-9201] Security
is disabled
[2022-06-01T22:41:10,213][INFO ][o.e.t.n.NettyAllocator   ] [node-9201] creating
NettyAllocator with the following configs: [name=elasticsearch_configured,
chunk_size=1mb, suggested_max_allocation_size=1mb, factors=
{es.unsafe.use_netty_default_chunk_and_page_size=false, g1gc_enabled=true,
g1gc_region_size=4mb}]
[2022-06-01T22:41:10,236][INFO ][o.e.i.r.RecoverySettings ] [node-9201] using
rate limit [40mb] with [default=40mb, read=0b, write=0b, max=0b]
[2022-06-01T22:41:10,269][INFO ][o.e.d.DiscoveryModule    ] [node-9201] using
discovery type [multi-node] and seed hosts providers [settings]
[2022-06-01T22:41:11,233][INFO ][o.e.n.Node               ] [node-9201]
initialized
[2022-06-01T22:41:11,234][INFO ][o.e.n.Node               ] [node-9201] starting
...
[2022-06-01T22:41:11,303][INFO ][o.e.x.s.c.f.PersistentCache] [node-9201]
persistent cache index loaded
[2022-06-01T22:41:11,305][INFO ][o.e.x.d.l.DeprecationIndexingComponent] [node-
9201] deprecation component started
[2022-06-01T22:41:11,455][INFO ][o.e.t.TransportService   ] [node-9201]
publish_address {localhost/127.0.0.1:9301}, bound_addresses {127.0.0.1:9301},
{[::1]:9301}
[2022-06-01T22:41:12,241][WARN ][o.e.b.BootstrapChecks    ] [node-9201] initial
heap size [1073741824] not equal to maximum heap size [8589934592]; this can
cause resize pauses
[2022-06-01T22:41:12,423][INFO ][o.e.c.c.Coordinator      ] [node-9201] setting
initial configuration to
VotingConfiguration{Q3EYFIhCR3K2BHnvunZ7Eg,1Ctmvjk6SASdqV7w78PQyg,kJet5gFIQ3SpxO
CB7mOGlg}
```

```
[2022-06-01T22:41:14,108][INFO ][o.e.c.s.ClusterApplierService] [node-9201]
master node changed {previous [], current [{node-9202}{1Ctmvjk6SASdqV7W78PQyg}
{mkH_V71VRemz3r7W4MjE2A}{localhost}{127.0.0.1:9302}{cdfhilmrstw}]}, added {{node-
9202}{1Ctmvjk6SASdqV7W78PQyg}{mkH_V71VRemz3r7W4MjE2A}{localhost}{127.0.0.1:9302}
{cdfhilmrstw}, {node-9203}{kJet5gFIQ3SpxOCB7mOGlg}{PAvvs6hMSM673GrsDYgKig}
{localhost}{127.0.0.1:9303}{cdfhilmrstw}}, term: 4, version: 58, reason:
ApplyCommitRequest{term=4, version=58, sourceNode={node-9202}
{1Ctmvjk6SASdqV7W78PQyg}{mkH_V71VRemz3r7W4MjE2A}{localhost}{127.0.0.1:9302}
{cdfhilmrstw}{ml.machine_memory=17113276416, ml.max_jvm_size=8589934592,
xpack.installed=true}}
[2022-06-01T22:41:14,190][INFO ][o.e.i.g.DatabaseNodeService] [node-9201]
retrieve geoip database [GeoLite2-ASN.mmdb] from [.geoip_databases] to
[C:\Users\mao\AppData\Local\Temp\elasticsearch\geoip-
databases\Q3EYFIhCR3K2BHnvunZ7Eg\GeoLite2-ASN.mmdb.tmp.gz]
[2022-06-01T22:41:14,407][INFO ][o.e.l.LicenseService     ] [node-9201] license
[c779e3df-021a-4d03-a7e5-2c46eb3118ce] mode [basic] - valid
[2022-06-01T22:41:14,535][INFO ][o.e.h.AbstractHttpServerTransport] [node-9201]
publish_address {localhost/127.0.0.1:9201}, bound_addresses {127.0.0.1:9201},
{[::1]:9201}
[2022-06-01T22:41:14,536][INFO ][o.e.n.Node               ] [node-9201] started
[2022-06-01T22:41:15,318][INFO ][o.e.i.g.DatabaseNodeService] [node-9201]
successfully loaded geoip database file [GeoLite2-ASN.mmdb]
[2022-06-01T22:41:17,219][INFO ][o.e.i.g.DatabaseNodeService] [node-9201]
retrieve geoip database [GeoLite2-City.mmdb] from [.geoip_databases] to
[C:\Users\mao\AppData\Local\Temp\elasticsearch\geoip-
databases\Q3EYFIhCR3K2BHnvunZ7Eg\GeoLite2-City.mmdb.tmp.gz]
[2022-06-01T22:41:18,177][INFO ][o.e.i.g.DatabaseNodeService] [node-9201]
successfully loaded geoip database file [GeoLite2-City.mmdb]
[2022-06-01T22:41:20,117][INFO ][o.e.i.g.DatabaseNodeService] [node-9201]
retrieve geoip database [GeoLite2-Country.mmdb] from [.geoip_databases] to
[C:\Users\mao\AppData\Local\Temp\elasticsearch\geoip-
databases\Q3EYFIhCR3K2BHnvunZ7Eg\GeoLite2-Country.mmdb.tmp.gz]
[2022-06-01T22:41:20,214][INFO ][o.e.i.g.DatabaseNodeService] [node-9201]
successfully loaded geoip database file [GeoLite2-Country.mmdb]
```

9202节点：

```
warning: ignoring JAVA_HOME=C:\Users\mao\.jdks\openjdk-16.0.2; using bundled JDK
warning: ignoring JAVA_HOME=C:\Users\mao\.jdks\openjdk-16.0.2; using
ES_JAVA_HOME
[2022-06-01T22:40:12,555][INFO ][o.e.n.Node               ] [node-9202]
version[8.1.3], pid[3028],
build[default/zip/39afaa3c0fe7db4869a161985e240bd7182d7a07/2022-04-
19T08:13:25.444693396Z], OS[Windows 10/10.0/amd64], JVM[Eclipse Adoptium/OpenJDK
64-Bit Server VM/18/18+36]
[2022-06-01T22:40:12,560][INFO ][o.e.n.Node               ] [node-9202] JVM home
[H:\opensoft\elasticsearch-cluster\elasticsearch2\jdk], using bundled JDK [true]
```

```
[2022-06-01T22:40:12,560][INFO ][o.e.n.Node               ] [node-9202] JVM
arguments [-Des.networkaddress.cache.ttl=60, -
Des.networkaddress.cache.negative.ttl=10, -Djava.security.manager=allow, -
XX:+AlwaysPreTouch, -Xss1m, -Djava.awt.headless=true, -Dfile.encoding=UTF-8, -
Djna.nosys=true, -XX:-OmitStackTraceInFastThrow, -
XX:+ShowCodeDetailsInExceptionMessages, -Dio.netty.noUnsafe=true, -
Dio.netty.noKeySetOptimization=true, -Dio.netty.recycler.maxCapacityPerThread=0,
-Dio.netty.allocator.numDirectArenas=0, -Dlog4j.shutdownHookEnabled=false, -
Dlog4j2.disable.jmx=true, -Dlog4j2.formatMsgNoLookups=true, -
Djava.locale.providers=SPI,COMPAT, --add-opens=java.base/java.io=ALL-UNNAMED, -
Xms1g, -Xmx8g, -XX:+UseG1GC, -
Djava.io.tmpdir=C:\Users\mao\AppData\Local\Temp\elasticsearch, -
XX:+HeapDumpOnOutOfMemoryError, -XX:+ExitOnOutOfMemoryError, -
XX:HeapDumpPath=data, -XX:ErrorFile=logs/hs_err_pid%p.log, -
Xlog:gc*,gc+age=trace,safepoint:file=logs/gc.log:utctime,pid,tags:filecount=32,f
ilesize=64m, -XX:MaxDirectMemorySize=4294967296, -
XX:InitiatingHeapOccupancyPercent=30, -XX:G1ReservePercent=25, -Delasticsearch,
-Des.path.home=H:\opensoft\elasticsearch-cluster\elasticsearch2, -
Des.path.conf=H:\opensoft\elasticsearch-cluster\elasticsearch2\config, -
Des.distribution.flavor=default, -Des.distribution.type=zip, -
Des.bundled_jdk=true]
[2022-06-01T22:40:15,284][INFO ][o.e.p.PluginsService     ] [node-9202] loaded
module [aggs-matrix-stats]
[2022-06-01T22:40:15,285][INFO ][o.e.p.PluginsService     ] [node-9202] loaded
module [analysis-common]
[2022-06-01T22:40:15,285][INFO ][o.e.p.PluginsService     ] [node-9202] loaded
module [constant-keyword]
[2022-06-01T22:40:15,285][INFO ][o.e.p.PluginsService     ] [node-9202] loaded
module [data-streams]
[2022-06-01T22:40:15,285][INFO ][o.e.p.PluginsService     ] [node-9202] loaded
module [frozen-indices]
[2022-06-01T22:40:15,285][INFO ][o.e.p.PluginsService     ] [node-9202] loaded
module [ingest-common]
[2022-06-01T22:40:15,286][INFO ][o.e.p.PluginsService     ] [node-9202] loaded
module [ingest-geoip]
[2022-06-01T22:40:15,286][INFO ][o.e.p.PluginsService     ] [node-9202] loaded
module [ingest-user-agent]
[2022-06-01T22:40:15,286][INFO ][o.e.p.PluginsService     ] [node-9202] loaded
module [kibana]
[2022-06-01T22:40:15,286][INFO ][o.e.p.PluginsService     ] [node-9202] loaded
module [lang-expression]
[2022-06-01T22:40:15,287][INFO ][o.e.p.PluginsService     ] [node-9202] loaded
module [lang-mustache]
[2022-06-01T22:40:15,287][INFO ][o.e.p.PluginsService     ] [node-9202] loaded
module [lang-painless]
[2022-06-01T22:40:15,287][INFO ][o.e.p.PluginsService     ] [node-9202] loaded
module [legacy-geo]
[2022-06-01T22:40:15,287][INFO ][o.e.p.PluginsService     ] [node-9202] loaded
module [mapper-extras]
[2022-06-01T22:40:15,288][INFO ][o.e.p.PluginsService     ] [node-9202] loaded
module [mapper-version]
[2022-06-01T22:40:15,288][INFO ][o.e.p.PluginsService     ] [node-9202] loaded
module [old-lucene-versions]
[2022-06-01T22:40:15,288][INFO ][o.e.p.PluginsService     ] [node-9202] loaded
module [parent-join]
[2022-06-01T22:40:15,288][INFO ][o.e.p.PluginsService     ] [node-9202] loaded
module [percolator]
```

```
[2022-06-01T22:40:15,288][INFO ][o.e.p.PluginsService     ] [node-9202] loaded
module [rank-eval]
[2022-06-01T22:40:15,289][INFO ][o.e.p.PluginsService     ] [node-9202] loaded
module [reindex]
[2022-06-01T22:40:15,289][INFO ][o.e.p.PluginsService     ] [node-9202] loaded
module [repositories-metering-api]
[2022-06-01T22:40:15,289][INFO ][o.e.p.PluginsService     ] [node-9202] loaded
module [repository-azure]
[2022-06-01T22:40:15,289][INFO ][o.e.p.PluginsService     ] [node-9202] loaded
module [repository-encrypted]
[2022-06-01T22:40:15,290][INFO ][o.e.p.PluginsService     ] [node-9202] loaded
module [repository-gcs]
[2022-06-01T22:40:15,290][INFO ][o.e.p.PluginsService     ] [node-9202] loaded
module [repository-s3]
[2022-06-01T22:40:15,290][INFO ][o.e.p.PluginsService     ] [node-9202] loaded
module [repository-url]
[2022-06-01T22:40:15,290][INFO ][o.e.p.PluginsService     ] [node-9202] loaded
module [runtime-fields-common]
[2022-06-01T22:40:15,290][INFO ][o.e.p.PluginsService     ] [node-9202] loaded
module [search-business-rules]
[2022-06-01T22:40:15,291][INFO ][o.e.p.PluginsService     ] [node-9202] loaded
module [searchable-snapshots]
[2022-06-01T22:40:15,291][INFO ][o.e.p.PluginsService     ] [node-9202] loaded
module [snapshot-based-recoveries]
[2022-06-01T22:40:15,291][INFO ][o.e.p.PluginsService     ] [node-9202] loaded
module [snapshot-repo-test-kit]
[2022-06-01T22:40:15,292][INFO ][o.e.p.PluginsService     ] [node-9202] loaded
module [spatial]
[2022-06-01T22:40:15,292][INFO ][o.e.p.PluginsService     ] [node-9202] loaded
module [transform]
[2022-06-01T22:40:15,292][INFO ][o.e.p.PluginsService     ] [node-9202] loaded
module [transport-netty4]
[2022-06-01T22:40:15,292][INFO ][o.e.p.PluginsService     ] [node-9202] loaded
module [unsigned-long]
[2022-06-01T22:40:15,292][INFO ][o.e.p.PluginsService     ] [node-9202] loaded
module [vector-tile]
[2022-06-01T22:40:15,293][INFO ][o.e.p.PluginsService     ] [node-9202] loaded
module [vectors]
[2022-06-01T22:40:15,293][INFO ][o.e.p.PluginsService     ] [node-9202] loaded
module [wildcard]
[2022-06-01T22:40:15,293][INFO ][o.e.p.PluginsService     ] [node-9202] loaded
module [x-pack-aggregate-metric]
[2022-06-01T22:40:15,293][INFO ][o.e.p.PluginsService     ] [node-9202] loaded
module [x-pack-analytics]
[2022-06-01T22:40:15,293][INFO ][o.e.p.PluginsService     ] [node-9202] loaded
module [x-pack-async]
[2022-06-01T22:40:15,294][INFO ][o.e.p.PluginsService     ] [node-9202] loaded
module [x-pack-async-search]
[2022-06-01T22:40:15,294][INFO ][o.e.p.PluginsService     ] [node-9202] loaded
module [x-pack-autoscaling]
[2022-06-01T22:40:15,294][INFO ][o.e.p.PluginsService     ] [node-9202] loaded
module [x-pack-ccr]
[2022-06-01T22:40:15,294][INFO ][o.e.p.PluginsService     ] [node-9202] loaded
module [x-pack-core]
[2022-06-01T22:40:15,295][INFO ][o.e.p.PluginsService     ] [node-9202] loaded
module [x-pack-deprecation]
[2022-06-01T22:40:15,295][INFO ][o.e.p.PluginsService     ] [node-9202] loaded
module [x-pack-enrich]
```

```
[2022-06-01T22:40:15,295][INFO ][o.e.p.PluginsService     ] [node-9202] loaded
module [x-pack-eql]
[2022-06-01T22:40:15,295][INFO ][o.e.p.PluginsService     ] [node-9202] loaded
module [x-pack-fleet]
[2022-06-01T22:40:15,295][INFO ][o.e.p.PluginsService     ] [node-9202] loaded
module [x-pack-graph]
[2022-06-01T22:40:15,296][INFO ][o.e.p.PluginsService     ] [node-9202] loaded
module [x-pack-identity-provider]
[2022-06-01T22:40:15,296][INFO ][o.e.p.PluginsService     ] [node-9202] loaded
module [x-pack-ilm]
[2022-06-01T22:40:15,296][INFO ][o.e.p.PluginsService     ] [node-9202] loaded
module [x-pack-logstash]
[2022-06-01T22:40:15,296][INFO ][o.e.p.PluginsService     ] [node-9202] loaded
module [x-pack-ml]
[2022-06-01T22:40:15,296][INFO ][o.e.p.PluginsService     ] [node-9202] loaded
module [x-pack-monitoring]
[2022-06-01T22:40:15,297][INFO ][o.e.p.PluginsService     ] [node-9202] loaded
module [x-pack-ql]
[2022-06-01T22:40:15,298][INFO ][o.e.p.PluginsService     ] [node-9202] loaded
module [x-pack-rollup]
[2022-06-01T22:40:15,298][INFO ][o.e.p.PluginsService     ] [node-9202] loaded
module [x-pack-security]
[2022-06-01T22:40:15,298][INFO ][o.e.p.PluginsService     ] [node-9202] loaded
module [x-pack-shutdown]
[2022-06-01T22:40:15,299][INFO ][o.e.p.PluginsService     ] [node-9202] loaded
module [x-pack-sql]
[2022-06-01T22:40:15,299][INFO ][o.e.p.PluginsService     ] [node-9202] loaded
module [x-pack-stack]
[2022-06-01T22:40:15,299][INFO ][o.e.p.PluginsService     ] [node-9202] loaded
module [x-pack-text-structure]
[2022-06-01T22:40:15,299][INFO ][o.e.p.PluginsService     ] [node-9202] loaded
module [x-pack-voting-only-node]
[2022-06-01T22:40:15,300][INFO ][o.e.p.PluginsService     ] [node-9202] loaded
module [x-pack-watcher]
[2022-06-01T22:40:15,300][INFO ][o.e.p.PluginsService     ] [node-9202] no
plugins loaded
[2022-06-01T22:40:15,775][INFO ][o.e.e.NodeEnvironment    ] [node-9202] using [1]
data paths, mounts [[鏂囨欢 鏃?(H:)]], net usable_space [76.8gb], net total_space
[195.3gb], types [NTFS]

[2022-06-01T22:40:15,776][INFO ][o.e.e.NodeEnvironment    ] [node-9202] heap size
[8gb], compressed ordinary object pointers [true]
[2022-06-01T22:40:15,836][INFO ][o.e.n.Node               ] [node-9202] node
name [node-9202], node ID [1Ctmvjk6SASdqV7W78PQyg], cluster name [my-
elasticsearch], roles [remote_cluster_client, master, data_warm, data_content,
transform, data_hot, ml, data_frozen, ingest, data_cold, data]
[2022-06-01T22:40:20,099][INFO ][o.e.x.m.p.l.CppLogMessageHandler] [node-9202]
[controller/20312] [Main.cc@123] controller (64 bit): Version 8.1.3 (Build
92d8267e6ebfb7) Copyright (c) 2022 Elasticsearch BV
[2022-06-01T22:40:20,397][INFO ][o.e.x.s.Security         ] [node-9202] Security
is disabled
[2022-06-01T22:40:21,167][INFO ][o.e.t.n.NettyAllocator   ] [node-9202] creating
NettyAllocator with the following configs: [name=elasticsearch_configured,
chunk_size=1mb, suggested_max_allocation_size=1mb, factors=
{es.unsafe.use_netty_default_chunk_and_page_size=false, g1gc_enabled=true,
g1gc_region_size=4mb}]
[2022-06-01T22:40:21,190][INFO ][o.e.i.r.RecoverySettings ] [node-9202] using
rate limit [40mb] with [default=40mb, read=0b, write=0b, max=0b]
```

```
[2022-06-01T22:40:21,221][INFO ][o.e.d.DiscoveryModule    ] [node-9202] using
discovery type [multi-node] and seed hosts providers [settings]
[2022-06-01T22:40:22,169][INFO ][o.e.n.Node               ] [node-9202]
initialized
[2022-06-01T22:40:22,169][INFO ][o.e.n.Node               ] [node-9202] starting
...
[2022-06-01T22:40:22,205][INFO ][o.e.x.s.c.f.PersistentCache] [node-9202]
persistent cache index loaded
[2022-06-01T22:40:22,206][INFO ][o.e.x.d.l.DeprecationIndexingComponent] [node-
9202] deprecation component started
[2022-06-01T22:40:22,364][INFO ][o.e.t.TransportService   ] [node-9202]
publish_address {localhost/127.0.0.1:9302}, bound_addresses {127.0.0.1:9302},
{[::1]:9302}
[2022-06-01T22:40:22,651][WARN ][o.e.b.BootstrapChecks    ] [node-9202] initial
heap size [1073741824] not equal to maximum heap size [8589934592]; this can
cause resize pauses
[2022-06-01T22:40:22,652][INFO ][o.e.c.c.Coordinator      ] [node-9202] cluster
UUID [ww8YZfIsRwe6ngSzu3Bi1Q]
[2022-06-01T22:40:32,667][WARN ][o.e.c.c.ClusterFormationFailureHelper] [node-
9202] master not discovered or elected yet, an election requires 2 nodes with
ids [1Ctmvjk6SASdqV7W78PQyg, kJet5gFIQ3SpxOCB7mOGlg], have only discovered non-
quorum [{node-9202}{1Ctmvjk6SASdqV7W78PQyg}{mkH_V71VRemz3r7W4MjE2A}{localhost}
{127.0.0.1:9302}{cdfhilmrstw}]; discovery will continue using [127.0.0.1:9301,
[::1]:9301, 127.0.0.1:9303, [::1]:9303] from hosts providers and [{node-9202}
{1Ctmvjk6SASdqV7W78PQyg}{mkH_V71VRemz3r7W4MjE2A}{localhost}{127.0.0.1:9302}
{cdfhilmrstw}] from last-known cluster state; node term 2, last-accepted version
49 in term 2
[2022-06-01T22:40:42,676][WARN ][o.e.c.c.ClusterFormationFailureHelper] [node-
9202] master not discovered or elected yet, an election requires 2 nodes with
ids [1Ctmvjk6SASdqV7W78PQyg, kJet5gFIQ3SpxOCB7mOGlg], have only discovered non-
quorum [{node-9202}{1Ctmvjk6SASdqV7W78PQyg}{mkH_V71VRemz3r7W4MjE2A}{localhost}
{127.0.0.1:9302}{cdfhilmrstw}]; discovery will continue using [127.0.0.1:9301,
[::1]:9301, 127.0.0.1:9303, [::1]:9303] from hosts providers and [{node-9202}
{1Ctmvjk6SASdqV7W78PQyg}{mkH_V71VRemz3r7W4MjE2A}{localhost}{127.0.0.1:9302}
{cdfhilmrstw}] from last-known cluster state; node term 2, last-accepted version
49 in term 2
[2022-06-01T22:40:52,675][WARN ][o.e.n.Node               ] [node-9202] timed out
while waiting for initial discovery state - timeout: 30s
[2022-06-01T22:40:52,683][INFO ][o.e.h.AbstractHttpServerTransport] [node-9202]
publish_address {localhost/127.0.0.1:9202}, bound_addresses {127.0.0.1:9202},
{[::1]:9202}
[2022-06-01T22:40:52,684][INFO ][o.e.n.Node               ] [node-9202] started
[2022-06-01T22:40:52,691][WARN ][o.e.c.c.ClusterFormationFailureHelper] [node-
9202] master not discovered or elected yet, an election requires 2 nodes with
ids [1Ctmvjk6SASdqV7W78PQyg, kJet5gFIQ3SpxOCB7mOGlg], have only discovered non-
quorum [{node-9202}{1Ctmvjk6SASdqV7W78PQyg}{mkH_V71VRemz3r7W4MjE2A}{localhost}
{127.0.0.1:9302}{cdfhilmrstw}]; discovery will continue using [127.0.0.1:9301,
[::1]:9301, 127.0.0.1:9303, [::1]:9303] from hosts providers and [{node-9202}
{1Ctmvjk6SASdqV7W78PQyg}{mkH_V71VRemz3r7W4MjE2A}{localhost}{127.0.0.1:9302}
{cdfhilmrstw}] from last-known cluster state; node term 2, last-accepted version
49 in term 2
```

```
[2022-06-01T22:40:56,791][INFO ][o.e.c.s.MasterService    ] [node-9202] elected-
as-master ([2] nodes joined)[{node-9203}{kJet5gFIQ3SpxOCB7mOGlg}
{PAvvs6hMSM673GrsDYgKig}{localhost}{127.0.0.1:9303}{cdfhilmrstw} completing
election, {node-9202}{1Ctmvjk6SASdqV7W78PQyg}{mkH_V71VRemz3r7W4MjE2A}{localhost}
{127.0.0.1:9302}{cdfhilmrstw} completing election, _BECOME_MASTER_TASK_,
_FINISH_ELECTION_], term: 4, version: 50, delta: master node changed {previous
[], current [{node-9202}{1Ctmvjk6SASdqV7W78PQyg}{mkH_V71VRemz3r7W4MjE2A}
{localhost}{127.0.0.1:9302}{cdfhilmrstw}]}, added {{node-9203}
{kJet5gFIQ3SpxOCB7mOGlg}{PAvvs6hMSM673GrsDYgKig}{localhost}{127.0.0.1:9303}
{cdfhilmrstw}}
[2022-06-01T22:40:57,074][INFO ][o.e.c.s.ClusterApplierService] [node-9202]
master node changed {previous [], current [{node-9202}{1Ctmvjk6SASdqV7W78PQyg}
{mkH_V71VRemz3r7W4MjE2A}{localhost}{127.0.0.1:9302}{cdfhilmrstw}]}, added {{node-
9203}{kJet5gFIQ3SpxOCB7mOGlg}{PAvvs6hMSM673GrsDYgKig}{localhost}{127.0.0.1:9303}
{cdfhilmrstw}}, term: 4, version: 50, reason: Publication{term=4, version=50}
[2022-06-01T22:40:57,120][INFO ][o.e.c.r.a.DiskThresholdMonitor] [node-9202]
skipping monitor as a check is already in progress
[2022-06-01T22:40:57,297][INFO ][o.e.l.LicenseService     ] [node-9202] license
[c779e3df-021a-4d03-a7e5-2c46eb3118ce] mode [basic] - valid
[2022-06-01T22:40:57,301][INFO ][o.e.g.GatewayService     ] [node-9202] recovered
[0] indices into cluster_state
[2022-06-01T22:41:00,260][INFO ][o.e.c.m.MetadataCreateIndexService] [node-9202]
[.geoip_databases] creating index, cause [auto(bulk api)], templates [], shards
[1]/[0]
[2022-06-01T22:41:00,305][INFO ][o.e.c.r.a.AllocationService] [node-9202]
updating number_of_replicas to [1] for indices [.geoip_databases]
[2022-06-01T22:41:04,276][INFO ][o.e.i.g.DatabaseNodeService] [node-9202]
retrieve geoip database [GeoLite2-ASN.mmdb] from [.geoip_databases] to
[C:\Users\mao\AppData\Local\Temp\elasticsearch\geoip-
databases\1Ctmvjk6SASdqV7W78PQyg\GeoLite2-ASN.mmdb.tmp.gz]
[2022-06-01T22:41:04,322][INFO ][o.e.c.r.a.AllocationService] [node-9202]
current.health="GREEN" message="Cluster health status changed from [YELLOW] to
[GREEN] (reason: [shards started [[.geoip_databases][0]]])."
previous.health="YELLOW" reason="shards started [[.geoip_databases][0]]"
[2022-06-01T22:41:04,581][INFO ][o.e.i.g.DatabaseNodeService] [node-9202]
successfully loaded geoip database file [GeoLite2-ASN.mmdb]
[2022-06-01T22:41:13,622][INFO ][o.e.c.s.MasterService    ] [node-9202] node-
join[{node-9201}{Q3EYFIhCR3K2BHnvunZ7Eg}{BdUROmGdRkKaMfjTRBMdIg}{localhost}
{127.0.0.1:9301}{cdfhilmrstw} joining], term: 4, version: 58, delta: added
{{node-9201}{Q3EYFIhCR3K2BHnvunZ7Eg}{BdUROmGdRkKaMfjTRBMdIg}{localhost}
{127.0.0.1:9301}{cdfhilmrstw}}
[2022-06-01T22:41:14,531][INFO ][o.e.c.s.ClusterApplierService] [node-9202] added
{{node-9201}{Q3EYFIhCR3K2BHnvunZ7Eg}{BdUROmGdRkKaMfjTRBMdIg}{localhost}
{127.0.0.1:9301}{cdfhilmrstw}}, term: 4, version: 58, reason:
Publication{term=4, version=58}
[2022-06-01T22:41:17,223][INFO ][o.e.i.g.DatabaseNodeService] [node-9202]
retrieve geoip database [GeoLite2-City.mmdb] from [.geoip_databases] to
[C:\Users\mao\AppData\Local\Temp\elasticsearch\geoip-
databases\1Ctmvjk6SASdqV7W78PQyg\GeoLite2-City.mmdb.tmp.gz]
[2022-06-01T22:41:18,124][INFO ][o.e.i.g.DatabaseNodeService] [node-9202]
successfully loaded geoip database file [GeoLite2-City.mmdb]
[2022-06-01T22:41:20,126][INFO ][o.e.i.g.DatabaseNodeService] [node-9202]
retrieve geoip database [GeoLite2-Country.mmdb] from [.geoip_databases] to
[C:\Users\mao\AppData\Local\Temp\elasticsearch\geoip-
databases\1Ctmvjk6SASdqV7W78PQyg\GeoLite2-Country.mmdb.tmp.gz]
[2022-06-01T22:41:20,216][INFO ][o.e.i.g.DatabaseNodeService] [node-9202]
successfully loaded geoip database file [GeoLite2-Country.mmdb]
```

9203节点：

```
warning: ignoring JAVA_HOME=C:\Users\mao\.jdks\openjdk-16.0.2; using bundled JDK
warning: ignoring JAVA_HOME=C:\Users\mao\.jdks\openjdk-16.0.2; using
ES_JAVA_HOME
[2022-06-01T22:40:46,068][INFO ][o.e.n.Node               ] [node-9203]
version[8.1.3], pid[13332],
build[default/zip/39afaa3c0fe7db4869a161985e240bd7182d7a07/2022-04-
19T08:13:25.444693396Z], OS[Windows 10/10.0/amd64], JVM[Eclipse Adoptium/OpenJDK
64-Bit Server VM/18/18+36]
[2022-06-01T22:40:46,073][INFO ][o.e.n.Node               ] [node-9203] JVM home
[H:\opensoft\elasticsearch-cluster\elasticsearch3\jdk], using bundled JDK [true]
[2022-06-01T22:40:46,074][INFO ][o.e.n.Node               ] [node-9203] JVM
arguments [-Des.networkaddress.cache.ttl=60, -
Des.networkaddress.cache.negative.ttl=10, -Djava.security.manager=allow, -
XX:+AlwaysPreTouch, -Xss1m, -Djava.awt.headless=true, -Dfile.encoding=UTF-8, -
Djna.nosys=true, -XX:-OmitStackTraceInFastThrow, -
XX:+ShowCodeDetailsInExceptionMessages, -Dio.netty.noUnsafe=true, -
Dio.netty.noKeySetOptimization=true, -Dio.netty.recycler.maxCapacityPerThread=0,
-Dio.netty.allocator.numDirectArenas=0, -Dlog4j.shutdownHookEnabled=false, -
Dlog4j2.disable.jmx=true, -Dlog4j2.formatMsgNoLookups=true, -
Djava.locale.providers=SPI,COMPAT, --add-opens=java.base/java.io=ALL-UNNAMED, -
Xms1g, -Xmx8g, -XX:+UseG1GC, -
Djava.io.tmpdir=C:\Users\mao\AppData\Local\Temp\elasticsearch, -
XX:+HeapDumpOnOutOfMemoryError, -XX:+ExitOnOutOfMemoryError, -
XX:HeapDumpPath=data, -XX:ErrorFile=logs/hs_err_pid%p.log, -
Xlog:gc*,gc+age=trace,safepoint:file=logs/gc.log:utctime,pid,tags:filecount=32,f
ilesize=64m, -XX:MaxDirectMemorySize=4294967296, -
XX:InitiatingHeapOccupancyPercent=30, -XX:G1ReservePercent=25, -Delasticsearch,
-Des.path.home=H:\opensoft\elasticsearch-cluster\elasticsearch3, -
Des.path.conf=H:\opensoft\elasticsearch-cluster\elasticsearch3\config, -
Des.distribution.flavor=default, -Des.distribution.type=zip, -
Des.bundled_jdk=true]
[2022-06-01T22:40:48,937][INFO ][o.e.p.PluginsService     ] [node-9203] loaded
module [aggs-matrix-stats]
[2022-06-01T22:40:48,938][INFO ][o.e.p.PluginsService     ] [node-9203] loaded
module [analysis-common]
[2022-06-01T22:40:48,938][INFO ][o.e.p.PluginsService     ] [node-9203] loaded
module [constant-keyword]
[2022-06-01T22:40:48,938][INFO ][o.e.p.PluginsService     ] [node-9203] loaded
module [data-streams]
[2022-06-01T22:40:48,938][INFO ][o.e.p.PluginsService     ] [node-9203] loaded
module [frozen-indices]
[2022-06-01T22:40:48,938][INFO ][o.e.p.PluginsService     ] [node-9203] loaded
module [ingest-common]
[2022-06-01T22:40:48,939][INFO ][o.e.p.PluginsService     ] [node-9203] loaded
module [ingest-geoip]
[2022-06-01T22:40:48,939][INFO ][o.e.p.PluginsService     ] [node-9203] loaded
module [ingest-user-agent]
[2022-06-01T22:40:48,939][INFO ][o.e.p.PluginsService     ] [node-9203] loaded
module [kibana]
[2022-06-01T22:40:48,939][INFO ][o.e.p.PluginsService     ] [node-9203] loaded
module [lang-expression]
```

```
[2022-06-01T22:40:48,940][INFO ][o.e.p.PluginsService     ] [node-9203] loaded
module [lang-mustache]
[2022-06-01T22:40:48,940][INFO ][o.e.p.PluginsService     ] [node-9203] loaded
module [lang-painless]
[2022-06-01T22:40:48,940][INFO ][o.e.p.PluginsService     ] [node-9203] loaded
module [legacy-geo]
[2022-06-01T22:40:48,940][INFO ][o.e.p.PluginsService     ] [node-9203] loaded
module [mapper-extras]
[2022-06-01T22:40:48,941][INFO ][o.e.p.PluginsService     ] [node-9203] loaded
module [mapper-version]
[2022-06-01T22:40:48,941][INFO ][o.e.p.PluginsService     ] [node-9203] loaded
module [old-lucene-versions]
[2022-06-01T22:40:48,941][INFO ][o.e.p.PluginsService     ] [node-9203] loaded
module [parent-join]
[2022-06-01T22:40:48,941][INFO ][o.e.p.PluginsService     ] [node-9203] loaded
module [percolator]
[2022-06-01T22:40:48,941][INFO ][o.e.p.PluginsService     ] [node-9203] loaded
module [rank-eval]
[2022-06-01T22:40:48,942][INFO ][o.e.p.PluginsService     ] [node-9203] loaded
module [reindex]
[2022-06-01T22:40:48,942][INFO ][o.e.p.PluginsService     ] [node-9203] loaded
module [repositories-metering-api]
[2022-06-01T22:40:48,942][INFO ][o.e.p.PluginsService     ] [node-9203] loaded
module [repository-azure]
[2022-06-01T22:40:48,942][INFO ][o.e.p.PluginsService     ] [node-9203] loaded
module [repository-encrypted]
[2022-06-01T22:40:48,943][INFO ][o.e.p.PluginsService     ] [node-9203] loaded
module [repository-gcs]
[2022-06-01T22:40:48,943][INFO ][o.e.p.PluginsService     ] [node-9203] loaded
module [repository-s3]
[2022-06-01T22:40:48,943][INFO ][o.e.p.PluginsService     ] [node-9203] loaded
module [repository-url]
[2022-06-01T22:40:48,943][INFO ][o.e.p.PluginsService     ] [node-9203] loaded
module [runtime-fields-common]
[2022-06-01T22:40:48,943][INFO ][o.e.p.PluginsService     ] [node-9203] loaded
module [search-business-rules]
[2022-06-01T22:40:48,944][INFO ][o.e.p.PluginsService     ] [node-9203] loaded
module [searchable-snapshots]
[2022-06-01T22:40:48,944][INFO ][o.e.p.PluginsService     ] [node-9203] loaded
module [snapshot-based-recoveries]
[2022-06-01T22:40:48,944][INFO ][o.e.p.PluginsService     ] [node-9203] loaded
module [snapshot-repo-test-kit]
[2022-06-01T22:40:48,944][INFO ][o.e.p.PluginsService     ] [node-9203] loaded
module [spatial]
[2022-06-01T22:40:48,945][INFO ][o.e.p.PluginsService     ] [node-9203] loaded
module [transform]
[2022-06-01T22:40:48,945][INFO ][o.e.p.PluginsService     ] [node-9203] loaded
module [transport-netty4]
[2022-06-01T22:40:48,945][INFO ][o.e.p.PluginsService     ] [node-9203] loaded
module [unsigned-long]
[2022-06-01T22:40:48,945][INFO ][o.e.p.PluginsService     ] [node-9203] loaded
module [vector-tile]
[2022-06-01T22:40:48,945][INFO ][o.e.p.PluginsService     ] [node-9203] loaded
module [vectors]
[2022-06-01T22:40:48,946][INFO ][o.e.p.PluginsService     ] [node-9203] loaded
module [wildcard]
[2022-06-01T22:40:48,946][INFO ][o.e.p.PluginsService     ] [node-9203] loaded
module [x-pack-aggregate-metric]
```

```
[2022-06-01T22:40:48,946][INFO ][o.e.p.PluginsService     ] [node-9203] loaded
module [x-pack-analytics]
[2022-06-01T22:40:48,946][INFO ][o.e.p.PluginsService     ] [node-9203] loaded
module [x-pack-async]
[2022-06-01T22:40:48,947][INFO ][o.e.p.PluginsService     ] [node-9203] loaded
module [x-pack-async-search]
[2022-06-01T22:40:48,947][INFO ][o.e.p.PluginsService     ] [node-9203] loaded
module [x-pack-autoscaling]
[2022-06-01T22:40:48,947][INFO ][o.e.p.PluginsService     ] [node-9203] loaded
module [x-pack-ccr]
[2022-06-01T22:40:48,947][INFO ][o.e.p.PluginsService     ] [node-9203] loaded
module [x-pack-core]
[2022-06-01T22:40:48,947][INFO ][o.e.p.PluginsService     ] [node-9203] loaded
module [x-pack-deprecation]
[2022-06-01T22:40:48,948][INFO ][o.e.p.PluginsService     ] [node-9203] loaded
module [x-pack-enrich]
[2022-06-01T22:40:48,948][INFO ][o.e.p.PluginsService     ] [node-9203] loaded
module [x-pack-eql]
[2022-06-01T22:40:48,948][INFO ][o.e.p.PluginsService     ] [node-9203] loaded
module [x-pack-fleet]
[2022-06-01T22:40:48,948][INFO ][o.e.p.PluginsService     ] [node-9203] loaded
module [x-pack-graph]
[2022-06-01T22:40:48,948][INFO ][o.e.p.PluginsService     ] [node-9203] loaded
module [x-pack-identity-provider]
[2022-06-01T22:40:48,949][INFO ][o.e.p.PluginsService     ] [node-9203] loaded
module [x-pack-ilm]
[2022-06-01T22:40:48,949][INFO ][o.e.p.PluginsService     ] [node-9203] loaded
module [x-pack-logstash]
[2022-06-01T22:40:48,949][INFO ][o.e.p.PluginsService     ] [node-9203] loaded
module [x-pack-ml]
[2022-06-01T22:40:48,949][INFO ][o.e.p.PluginsService     ] [node-9203] loaded
module [x-pack-monitoring]
[2022-06-01T22:40:48,949][INFO ][o.e.p.PluginsService     ] [node-9203] loaded
module [x-pack-ql]
[2022-06-01T22:40:48,949][INFO ][o.e.p.PluginsService     ] [node-9203] loaded
module [x-pack-rollup]
[2022-06-01T22:40:48,950][INFO ][o.e.p.PluginsService     ] [node-9203] loaded
module [x-pack-security]
[2022-06-01T22:40:48,950][INFO ][o.e.p.PluginsService     ] [node-9203] loaded
module [x-pack-shutdown]
[2022-06-01T22:40:48,950][INFO ][o.e.p.PluginsService     ] [node-9203] loaded
module [x-pack-sql]
[2022-06-01T22:40:48,950][INFO ][o.e.p.PluginsService     ] [node-9203] loaded
module [x-pack-stack]
[2022-06-01T22:40:48,950][INFO ][o.e.p.PluginsService     ] [node-9203] loaded
module [x-pack-text-structure]
[2022-06-01T22:40:48,951][INFO ][o.e.p.PluginsService     ] [node-9203] loaded
module [x-pack-voting-only-node]
[2022-06-01T22:40:48,952][INFO ][o.e.p.PluginsService     ] [node-9203] loaded
module [x-pack-watcher]
[2022-06-01T22:40:48,952][INFO ][o.e.p.PluginsService     ] [node-9203] no
plugins loaded
[2022-06-01T22:40:49,484][INFO ][o.e.e.NodeEnvironment    ] [node-9203] using [1]
data paths, mounts [[鑷囷欢 鐩?(H:)]], net usable_space [76.8gb], net total_space
[195.3gb], types [NTFS]

[2022-06-01T22:40:49,486][INFO ][o.e.e.NodeEnvironment    ] [node-9203] heap size
[8gb], compressed ordinary object pointers [true]
```

```
[2022-06-01T22:40:49,549][INFO ][o.e.n.Node               ] [node-9203] node
name [node-9203], node ID [kJet5gFIQ3SpxOCB7mOGlg], cluster name [my-
elasticsearch], roles [data_content, transform, data_hot, ml, data_frozen,
ingest, data_cold, data, remote_cluster_client, master, data_warm]
[2022-06-01T22:40:53,803][INFO ][o.e.x.m.p.l.CppLogMessageHandler] [node-9203]
[controller/1592] [Main.cc@123] controller (64 bit): Version 8.1.3 (Build
92d8267e6ebfb7) Copyright (c) 2022 Elasticsearch BV
[2022-06-01T22:40:54,075][INFO ][o.e.x.s.Security         ] [node-9203] Security
is disabled
[2022-06-01T22:40:54,861][INFO ][o.e.t.n.NettyAllocator   ] [node-9203] creating
NettyAllocator with the following configs: [name=elasticsearch_configured,
chunk_size=1mb, suggested_max_allocation_size=1mb, factors=
{es.unsafe.use_netty_default_chunk_and_page_size=false, g1gc_enabled=true,
g1gc_region_size=4mb}]
[2022-06-01T22:40:54,884][INFO ][o.e.i.r.RecoverySettings ] [node-9203] using
rate limit [40mb] with [default=40mb, read=0b, write=0b, max=0b]
[2022-06-01T22:40:54,915][INFO ][o.e.d.DiscoveryModule    ] [node-9203] using
discovery type [multi-node] and seed hosts providers [settings]
[2022-06-01T22:40:55,912][INFO ][o.e.n.Node               ] [node-9203]
initialized
[2022-06-01T22:40:55,912][INFO ][o.e.n.Node               ] [node-9203] starting
...
[2022-06-01T22:40:55,936][INFO ][o.e.x.s.c.f.PersistentCache] [node-9203]
persistent cache index loaded
[2022-06-01T22:40:55,936][INFO ][o.e.x.d.l.DeprecationIndexingComponent] [node-
9203] deprecation component started
[2022-06-01T22:40:56,084][INFO ][o.e.t.TransportService   ] [node-9203]
publish_address {localhost/127.0.0.1:9303}, bound_addresses {127.0.0.1:9303},
{[::1]:9303}
[2022-06-01T22:40:56,390][WARN ][o.e.b.BootstrapChecks    ] [node-9203] initial
heap size [1073741824] not equal to maximum heap size [8589934592]; this can
cause resize pauses
[2022-06-01T22:40:56,392][INFO ][o.e.c.c.Coordinator      ] [node-9203] cluster
UUID [ww8YZfIsRwe6ngSzu3Bi1Q]
[2022-06-01T22:40:57,047][INFO ][o.e.c.s.ClusterApplierService] [node-9203]
master node changed {previous [], current [{node-9202}{1Ctmvjk6SASdqV7W78PQyg}
{mkH_V71VRemz3r7W4MjE2A}{localhost}{127.0.0.1:9302}{cdfhilmrstw}]}, added {{node-
9202}{1Ctmvjk6SASdqV7W78PQyg}{mkH_V71VRemz3r7W4MjE2A}{localhost}{127.0.0.1:9302}
{cdfhilmrstw}}, term: 4, version: 50, reason: ApplyCommitRequest{term=4,
version=50, sourceNode={node-9202}{1Ctmvjk6SASdqV7W78PQyg}
{mkH_V71VRemz3r7W4MjE2A}{localhost}{127.0.0.1:9302}{cdfhilmrstw}
{ml.machine_memory=17113276416, ml.max_jvm_size=8589934592,
xpack.installed=true}}
[2022-06-01T22:40:57,069][INFO ][o.e.h.AbstractHttpServerTransport] [node-9203]
publish_address {localhost/127.0.0.1:9203}, bound_addresses {127.0.0.1:9203},
{[::1]:9203}
[2022-06-01T22:40:57,069][INFO ][o.e.n.Node               ] [node-9203] started
[2022-06-01T22:40:57,234][INFO ][o.e.l.LicenseService     ] [node-9203] license
[c779e3df-021a-4d03-a7e5-2c46eb3118ce] mode [basic] - valid
[2022-06-01T22:41:04,271][INFO ][o.e.i.g.DatabaseNodeService] [node-9203]
retrieve geoip database [GeoLite2-ASN.mmdb] from [.geoip_databases] to
[C:\Users\mao\AppData\Local\Temp\elasticsearch\geoip-
databases\kJet5gFIQ3SpxOCB7mOGlg\GeoLite2-ASN.mmdb.tmp.gz]
[2022-06-01T22:41:04,280][INFO ][o.e.i.g.GeoIpDownloader  ] [node-9203]
successfully downloaded geoip database [GeoLite2-ASN.mmdb]
[2022-06-01T22:41:04,523][INFO ][o.e.i.g.DatabaseNodeService] [node-9203]
successfully loaded geoip database file [GeoLite2-ASN.mmdb]
```

```
[2022-06-01T22:41:14,083][INFO ][o.e.c.s.ClusterApplierService] [node-9203] added
{{node-9201}{Q3EYFIhCR3K2BHnvunZ7Eg}{BdUROmGdRkKaMfjTRBMdIg}{localhost}
{127.0.0.1:9301}{cdfhilmrstw}}, term: 4, version: 58, reason:
ApplyCommitRequest{term=4, version=58, sourceNode={node-9202}
{1Ctmvjk6SASdqV7W78PQyg}{mkH_V71VRemz3r7W4MjE2A}{localhost}{127.0.0.1:9302}
{cdfhilmrstw}{ml.machine_memory=17113276416, ml.max_jvm_size=8589934592,
xpack.installed=true}}
[2022-06-01T22:41:17,199][INFO ][o.e.i.g.DatabaseNodeService] [node-9203]
retrieve geoip database [GeoLite2-City.mmdb] from [.geoip_databases] to
[C:\Users\mao\AppData\Local\Temp\elasticsearch\geoip-
databases\kJet5gFIQ3SpxOCB7mOGlg\GeoLite2-City.mmdb.tmp.gz]
[2022-06-01T22:41:17,225][INFO ][o.e.i.g.GeoIpDownloader  ] [node-9203]
successfully downloaded geoip database [GeoLite2-City.mmdb]
[2022-06-01T22:41:18,093][INFO ][o.e.i.g.DatabaseNodeService] [node-9203]
successfully loaded geoip database file [GeoLite2-City.mmdb]
[2022-06-01T22:41:20,117][INFO ][o.e.i.g.DatabaseNodeService] [node-9203]
retrieve geoip database [GeoLite2-Country.mmdb] from [.geoip_databases] to
[C:\Users\mao\AppData\Local\Temp\elasticsearch\geoip-
databases\kJet5gFIQ3SpxOCB7mOGlg\GeoLite2-Country.mmdb.tmp.gz]
[2022-06-01T22:41:20,129][INFO ][o.e.i.g.GeoIpDownloader  ] [node-9203]
successfully downloaded geoip database [GeoLite2-Country.mmdb]
[2022-06-01T22:41:20,203][INFO ][o.e.i.g.DatabaseNodeService] [node-9203]
successfully loaded geoip database file [GeoLite2-Country.mmdb]
```

5. 测试集群

浏览器输入：

```
http://127.0.0.1:9201/_cluster/health
http://127.0.0.1:9202/_cluster/health
http://127.0.0.1:9203/_cluster/health
```

结果：


```
{"cluster_name":"my-
elasticsearch","status":"green","timed_out":false,"number_of_nodes":3,"number_of
_data_nodes":3,"active_primary_shards":1,"active_shards":2,"relocating_shards":0
,"initializing_shards":0,"unassigned_shards":0,"delayed_unassigned_shards":0,"nu
mber_of_pending_tasks":0,"number_of_in_flight_fetch":0,"task_max_waiting_in_queu
e_millis":0,"active_shards_percent_as_number":100.0}
```

```
{"cluster_name":"my-
elasticsearch","status":"green","timed_out":false,"number_of_nodes":3,"number_of
_data_nodes":3,"active_primary_shards":1,"active_shards":2,"relocating_shards":0
,"initializing_shards":0,"unassigned_shards":0,"delayed_unassigned_shards":0,"nu
mber_of_pending_tasks":0,"number_of_in_flight_fetch":0,"task_max_waiting_in_queu
e_millis":0,"active_shards_percent_as_number":100.0}
```

{"cluster_name":"my-elasticsearch","status":"green","timed_out":false,"number_of_nodes":3,"number_of_data_nodes":3,"active_primary_shards":1,"active_shards":2,"relocating_shards":0,"initializing_shards":0,"unassigned_shards":0,"delayed_unassigned_shards":0,"number_of_pending_tasks":0,"number_of_in_flight_fetch":0,"task_max_waiting_in_queue_millis":0,"active_shards_percent_as_number":100.0}

windows集群一键启动脚本

在elasticsearch-cluster目录下创建一个bat文件，输入以下命令

```
cd  ./elasticsearch1/bin
start "elasticsearch1" elasticsearch.bat
cd ./../../
cd  ./elasticsearch2/bin
start "elasticsearch2"  elasticsearch.bat
cd ./../../
cd  ./elasticsearch3/bin
start "elasticsearch3"  elasticsearch.bat
```

保存。

以后可以双击此文件来启动集群

Docker实现

1. 编写docker-compose文件

```
version: '2.2'
services:
  es01:
    image: elasticsearch:7.12.1
    container_name: es01
    environment:
      - node.name=es01
      - cluster.name=es-docker-cluster
      - discovery.seed_hosts=es02,es03
      - cluster.initial_master_nodes=es01,es02,es03
      - "ES_JAVA_OPTS=-Xms512m -Xmx512m"
    volumes:
      - data01:/usr/share/elasticsearch/data
    ports:
      - 9200:9200
    networks:
      - elastic
  es02:
    image: elasticsearch:7.12.1
```

```
    container_name: es02
    environment:
      - node.name=es02
      - cluster.name=es-docker-cluster
      - discovery.seed_hosts=es01,es03
      - cluster.initial_master_nodes=es01,es02,es03
      - "ES_JAVA_OPTS=-Xms512m -Xmx512m"
    volumes:
      - data02:/usr/share/elasticsearch/data
    ports:
      - 9201:9200
    networks:
      - elastic
  es03:
    image: elasticsearch:7.12.1
    container_name: es03
    environment:
      - node.name=es03
      - cluster.name=es-docker-cluster
      - discovery.seed_hosts=es01,es02
      - cluster.initial_master_nodes=es01,es02,es03
      - "ES_JAVA_OPTS=-Xms512m -Xmx512m"
    volumes:
      - data03:/usr/share/elasticsearch/data
    networks:
      - elastic
    ports:
      - 9202:9200
volumes:
  data01:
    driver: local
  data02:
    driver: local
  data03:
    driver: local

networks:
  elastic:
    driver: bridge
```

2. 修改linux系统权限

修改`/etc/sysctl.conf`文件

vi /etc/sysctl.conf

添加下面的内容：

```
vm.max_map_count=262144
```

然后执行命令，让配置生效：

```
sysctl -p
```

3. 启动集群

```
docker-compose up -d
```

集群脑裂问题

- node.master：备选主节点：主节点可以管理和记录集群状态、决定分片在哪个节点、处理创建和删除索引库的请求
- node.data：数据节点：存储数据、搜索、聚合、CRUD
- node.ingest：数据存储之前的预处理
- coordinating：上面3个参数都为false则为coordinating节点，路由请求到其它节点，合并其它节点处理的结果，返回给用户

默认情况下，集群中的任何一个节点都同时具备上述四种角色。

但是真实的集群一定要将集群职责分离：

- master节点：对CPU要求高，但是内存要求第
- data节点：对CPU和内存要求都高
- coordinating节点：对网络带宽、CPU要求高

职责分离可以让我们根据不同节点的需求分配不同的硬件去部署。而且避免业务之间的互相干扰。

脑裂是因为集群中的节点失联导致的。

例如一个集群中：

- 主节点与其它节点失联：
- 此时，node2和node3认为node1宕机，就会重新选主
- 当node3当选后，集群继续对外提供服务，node2和node3自成集群，node1自成集群，两个集群数据不同步，出现数据差异
- 当网络恢复后，因为集群中有两个master节点，集群状态的不一致，出现脑裂的情况

解决

解决脑裂的方案是，要求选票超过 (eligible节点数量 + 1) / 2 才能当选为主，因此eligible节点数量最好是奇数。对应配置项是discovery.zen.minimum_master_nodes，在es7.0以后，已经成为默认配置，因此一般不会发生脑裂问题

例如：3个节点形成的集群，选票必须超过 （3 + 1） / 2 ，也就是2票。node3得到node2和node3的选票，当选为主。node1只有自己1票，没有当选。集群中依然只有1个主节点，没有出现脑裂。

end

by mao

2022/6/2