第3章 搜索解决方案-1 elasticsearch

学习目标：

- 理解elasticsearch索引结构和数据类型，掌握IK分词器的使用
- 掌握索引的常用操作（使用Kibana工具）
- 掌握javaRest高级api
- 完成数据批量导入

1 走进ElasticSearch

1.1 全文检索

1.1.1 为什么要使用全文检索

用户访问我们的首页，一般都会直接搜索来寻找自己想要购买的商品。

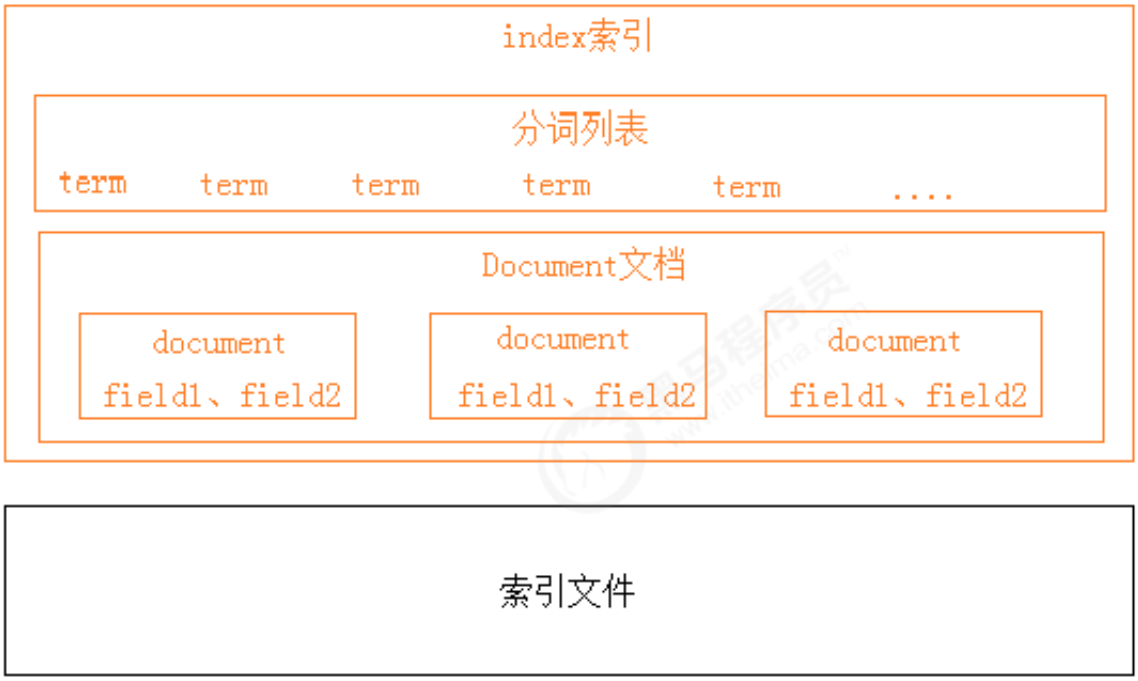
而商品的数量非常多，而且分类繁杂。如果能正确的显示出用户想要的商品，并进行合理的过滤，尽快促成交易，是搜索系统要研究的核心。

面对这样复杂的搜索业务和数据量，使用传统数据库搜索就显得力不从心，一般我们都会使用全文检索技术。

常见的全文检索技术有 Lucene、solr 、elasticsearch 等。

1.1.2 理解索引结构

下图是索引结构，下边黑色部分是物理结构，上边黄色部分是逻辑结构，逻辑结构也是为了更好的去描述工作原理及去使用物理结构中的索引文件。



逻辑结构部分是一个倒排索引表：

1、将要搜索的文档内容分词，所有不重复的词组成分词列表。

2、将搜索的文档最终以Document方式存储起来。

3、每个词和docment都有关联。

如下：

Term	Doc_1	Doc_2
Quick		X
The	X	
brown	X	X
dog	X	
dogs		X
fox	X	
foxes		X
in		X
jumped	X	
lazy	X	X
leap		X
over	X	X
quick	X	
summer		X
the	X	

现在，如果我们想搜索 `quick brown` ，我们只需要查找包含每个词条的文档：

```
Term      Doc_1  Doc_2
------------------------
brown   |   X   |  X
quick   |   X   |
------------------------
Total   |   2   |  1
```

两个文档都匹配，但是第一个文档比第二个匹配度更高。如果我们使用仅计算匹配词条数量的简单 相似性算法 ，那么，我们可以说，对于我们查询的相关性来讲，第一个文档比第二个文档更佳。

1.2 Elasticsearch

1.2.1 Elasticsearch简介

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是用Java开发的，并作为Apache许可条款下的开放源码发布，是当前流行的企业级搜索引擎。设计用于云计算中，能够达到实时搜索，稳定，可靠，快速，安装使用方便。

我们建立一个网站或应用程序，并要添加搜索功能，但是想要完成搜索工作的创建是非常困难的。我们希望搜索解决方案要运行速度快，我们希望能有一个零配置和一个完全免费的搜索模式，我们希望能够简单地使用JSON通过HTTP来索引数据，我们希望我们的搜索服务器始终可用，我们希望能够从一台开始并扩展到数百台，我们要实时搜索，我们要简单的多租户，我们希望建立一个云的解决方案。因此我们利用Elasticsearch来解决所有这些问题及可能出现的更多其它问题。

官方网址：https://www.elastic.co/cn/products/elasticsearch

Github：https://github.com/elastic/elasticsearch

优点：

（1）可以作为一个大型分布式集群（数百台服务器）技术，处理PB级数据，服务大公司；也可以运行在单机上

（2）将全文检索、数据分析以及分布式技术，合并在了一起，才形成了独一无二的ES；

（3）开箱即用的，部署简单

（4）全文检索，同义词处理，相关度排名，复杂数据分析，海量数据的近实时处理

下表是Elasticsearch与MySQL数据库逻辑结构概念的对比

Elasticsearch	关系型数据库**Mysql**
索引(index)	数据库(databases)
类型(type)	表(table)
文档(document)	行(row)

1.2.2 安装与启动

下载ElasticSearch 6.5.2版本

https://www.elastic.co/downloads/past-releases/elasticsearch-6.5.2

资源\配套软件中也提供了安装包

无需安装，解压安装包后即可使用

在命令提示符下，进入ElasticSearch安装目录下的bin目录,执行命令

```
elasticsearch
```

即可启动。

我们打开浏览器，在地址栏输入http://127.0.0.1:9200/ 即可看到输出结果

```
{
  "name" : "uxxprtP",
  "cluster_name" : "elasticsearch",
  "cluster_uuid" : "busVHkVASnW5x2TAwYKnkw",
  "version" : {
    "number" : "6.5.2",
    "build_flavor" : "default",
    "build_type" : "zip",
    "build_hash" : "9434bed",
    "build_date" : "2018-11-29T23:58:20.891072Z",
    "build_snapshot" : false,
    "lucene_version" : "7.5.0",
    "minimum_wire_compatibility_version" : "5.6.0",
    "minimum_index_compatibility_version" : "5.0.0"
  },
  "tagline" : "You Know, for Search"
}
```

1.3 使用Postman操作索引库

1.3.1 新建文档

使用postman测试：以post方式提交 http://127.0.0.1:9200/testindex/doc

body:

```
{
    "name":"测试商品",
    "price":123
}
```

返回结果如下：

```
{
    "_index": "testindex",
    "_type": "doc",
    "_id": "ctP3GmkBaXmTUn7r0YOA",
    "_version": 1,
    "result": "created",
    "_shards": {
        "total": 2,
        "successful": 1,
        "failed": 0
    },
    "_seq_no": 0,
    "_primary_term": 1
}
```


_id是由系统自动生成的。 为了方便之后的演示，我们再次录入几条测试数据。

1.3.2 查询文档

查询某索引某类型的全部数据，以get方式请求

http://127.0.0.1:9200/testindex/doc/_search 返回结果如下：

```
{
    "took": 58,
    "timed_out": false,
    "_shards": {
        "total": 5,
        "successful": 5,
        "skipped": 0,
        "failed": 0
    },
    "hits": {
        "total": 2,
        "max_score": 1,
        "hits": [
            {
                "_index": "testindex",
                "_type": "doc",
                "_id": "ctP3GmkBaXmTUn7r0YOA",
                "_score": 1,
                "_source": {
                    "name": "测试商品",
                    "price": 123
                }
            },
            {
                "_index": "testindex",
                "_type": "doc",
                "_id": "c9P4GmkBaXmTUn7rYoPS",
                "_score": 1,
                "_source": {
                    "name": "华为手机",
                    "price": 123
                }
            }
        ]
    }
}
```


1.4 映射与数据类型

映射(Mapping)相当于数据表的表结构。ElasticSearch中的映射（Mapping）用来定义一个文档，可以定义所包含的字段以及字段的类型、分词器及属性等等。

映射可以分为动态映射和静态映射。

动态映射 （dynamic mapping）：在关系数据库中，需要事先创建数据库，然后在该数据库实例下创建数据表，然后才能在该数据表中插入数据。而ElasticSearch中不需要事先定义映射（Mapping），文档写入ElasticSearch时，会根据文档字段自动识别类型，这种机制称之为动态映射。

静态映射 ：在ElasticSearch中也可以事先定义好映射，包含文档的各个字段及其类型等，这种方式称之为静态映射。

常用类型如下：

1.4.1 字符串类型

类型	描述
text	当一个字段是要被全文搜索的，比如Email内容、产品描述，应该使用text类型。设置text类型以后，字段内容会被分析，在生成倒排索引以前，字符串会被分析器分成一个一个词项。text类型的字段不用于排序，很少用于聚合。
keyword	keyword类型适用于索引结构化的字段，比如email地址、主机名、状态码和标签。如果字段需要进行过滤(比如查找已发布博客中status属性为published的文章)、排序、聚合。keyword类型的字段只能通过精确值搜索到。

1.4.2 整数类型

类型	取值范围
byte	-128~127
short	-32768~32767
integer	-2^{31} ~ 2^{31}-1
long	-2^{63}~2^{63}-1

1.4.3 浮点类型

类型	取值范围
doule	64位双精度浮点类型
float	32位单精度浮点类型
half_float	16位半精度浮点类型
scaled_float	缩放类型的的浮点数

1.4.4 date类型

日期类型表示格式可以是以下几种：
（1）日期格式的字符串，比如 “2018-01-13” 或 “2018-01-13 12:10:30”
（2）long类型的毫秒数(milliseconds-since-the-epoch，epoch就是指UNIX诞生的UTC时间1970年1月1日0时0分0秒)
（3）integer的秒数(seconds-since-the-epoch)

1.4.5 boolean类型

逻辑类型（布尔类型）可以接受true/false

1.4.6 binary类型

二进制字段是指用base64来表示索引中存储的二进制数据，可用来存储二进制形式的数据，例如图像。默认情况下，该类型的字段只存储不索引。二进制类型只支持index_name属性。

1.4.7 array类型

在ElasticSearch中，没有专门的数组（Array）数据类型，但是，在默认情况下，任意一个字段都可以包含0或多个值，这意味着每个字段默认都是数组类型，只不过，数组类型的各个元素值的数据类型必须相同。在ElasticSearch中，数组是开箱即用的（out of box），不需要进行任何配置，就可以直接使用。

在同一个数组中，数组元素的数据类型是相同的，ElasticSearch不支持元素为多个数据类型：[10, “some string”]，常用的数组类型是：

（1）字符数组: [“one”, “two”]
（2）整数数组: productid:[1, 2]
（3）对象（文档）数组: “user”:[{ “name”: “Mary”, “age”: 12 }, { “name”: “John”, “age”: 10 }]，ElasticSearch内部把对象数组展开为 {“user.name”: [“Mary”, “John”], “user.age”: [12,10]}

1.4.8 object类型

JSON天生具有层级关系，文档会包含嵌套的对象

1.5 IK分词器

1.5.1 什么是IK分词器

使用postman测试 post方式提交 http://127.0.0.1:9200/testindex/_analyze

```
{"analyzer": "chinese", "text": "黑马程序员" }
```

浏览器返回效果如下

```
{
    "tokens": [
        {
            "token": "黑",
            "start_offset": 0,
            "end_offset": 1,
            "type": "<IDEOGRAPHIC>",
            "position": 0
        },
        {
            "token": "马",
            "start_offset": 1,
            "end_offset": 2,
            "type": "<IDEOGRAPHIC>",
            "position": 1
        },
        {
            "token": "程",
            "start_offset": 2,
            "end_offset": 3,
            "type": "<IDEOGRAPHIC>",
            "position": 2
        },
        {
            "token": "序",
            "start_offset": 3,
            "end_offset": 4,
            "type": "<IDEOGRAPHIC>",
            "position": 3
        },
        {
            "token": "员",
            "start_offset": 4,
            "end_offset": 5,
            "type": "<IDEOGRAPHIC>",
            "position": 4
        }
    ]
}
```

默认的中文分词是将每个字看成一个词，这显然是不符合要求的，所以我们需要安装中文分词器来解决这个问题。

IK分词是一款国人开发的相对简单的中文分词器。虽然开发者自2012年之后就不在维护了，但在工程应用中IK算是比较流行的一款！我们今天就介绍一下IK中文分词器的使用。

1.5.2 IK分词器安装

下载地址：https://github.com/medcl/elasticsearch-analysis-ik/releases 下载6.5.2版本 课程配套资源也提供了： 资源\配套软件\elasticsearch\elasticsearch-analysis-ik-6.5.2.zip

（1）先将其解压，将解压后的elasticsearch文件夹重命名文件夹为ik

（2）将ik文件夹拷贝到elasticsearch/plugins 目录下。

（3）重新启动，即可加载IK分词器

1.5.3 IK分词器测试

IK提供了两个分词算法ik_smart 和 ik_max_word

其中 ik_smart 为最少切分，ik_max_word为最细粒度划分

我们分别来试一下

（1）最小切分：

使用postman测试 post方式提交 http://127.0.0.1:9200/testindex/_analyze

```
{"analyzer": "ik_smart", "text": "黑马程序员" }
```

输出的结果为：

```
{
    "tokens": [
        {
            "token": "黑马",
            "start_offset": 0,
            "end_offset": 2,
            "type": "CN_WORD",
            "position": 0
        },
        {
            "token": "程序员",
            "start_offset": 2,
            "end_offset": 5,
            "type": "CN_WORD",
            "position": 1
        }
    ]
}
```


（2）最细切分：

使用postman测试 post方式提交 http://127.0.0.1:9200/testindex/_analyze

```
{"analyzer": "ik_max_word", "text": "黑马程序员" }
```

输出的结果为：

```
{
    "tokens": [
        {
            "token": "黑马",
            "start_offset": 0,
            "end_offset": 2,
            "type": "CN_WORD",
            "position": 0
        },
        {
            "token": "程序员",
            "start_offset": 2,
            "end_offset": 5,
            "type": "CN_WORD",
            "position": 1
        },
        {
            "token": "程序",
            "start_offset": 2,
            "end_offset": 4,
            "type": "CN_WORD",
            "position": 2
        },
        {
            "token": "员",
            "start_offset": 4,
            "end_offset": 5,
            "type": "CN_CHAR",
            "position": 3
        }
    ]
}
```


1.5.4 自定义词库

我们现在测试"传智播客"，结果如下：

```
{
  "tokens" : [
    {
      "token" : "传",
      "start_offset" : 0,
      "end_offset" : 1,
      "type" : "CN_CHAR",
      "position" : 0
    },
    {
      "token" : "智",
      "start_offset" : 1,
      "end_offset" : 2,
      "type" : "CN_CHAR",
      "position" : 1
    },
    {
      "token" : "播",
      "start_offset" : 2,
      "end_offset" : 3,
      "type" : "CN_CHAR",
      "position" : 2
    },
    {
      "token" : "客",
      "start_offset" : 3,
      "end_offset" : 4,
      "type" : "CN_CHAR",
      "position" : 3
    }
  ]
}
```


默认的分词并没有识别“传智播客”是一个词。如果我们想让系统识别“传智播客”是一个词，需要编辑自定义词库。

步骤：

（1）进入elasticsearch/plugins/ik/config目录

（2）新建一个my.dic文件，编辑内容：

```
传智播客
```

修改IKAnalyzer.cfg.xml（在ik/config目录下）

```
<properties>
    <comment>IK Analyzer 扩展配置</comment>
    <!--用户可以在这里配置自己的扩展字典 -->
    <entry key="ext_dict">my.dic</entry>
     <!--用户可以在这里配置自己的扩展停止词字典-->
    <entry key="ext_stopwords"></entry>
</properties>
```

重新启动elasticsearch，通过浏览器测试分词效果

```
{
    "tokens": [
        {
            "token": "传智播客",
            "start_offset": 0,
            "end_offset": 4,
            "type": "CN_WORD",
            "position": 0
        }
    ]
}
```

1.6 Kibana

1.6.1 Kibana简介

Kibana 是一个开源的分析和可视化平台，旨在与 Elasticsearch 合作。Kibana 提供搜索、查看和与存储在 Elasticsearch 索引中的数据进行交互的功能。开发者或运维人员可以轻松地执行高级数据分析，并在各种图表、表格和地图中可视化数据。

1.6.2 Kibana安装与启动

（1）解压 资料\配套软件\elasticsearch\kibana-6.5.2-windows-x86_64.zip

（2）如果Kibana远程连接Elasticsearch ， 可以修改config\kibana.yml

（3）执行bin\kibana.bat

（4）打开浏览器，键入http://localhost:5601 访问Kibana

我们这里使用Kibana进行索引操作，Kibana与Postman相比省略了服务地址，并且有语法提示，非常便捷。

2 索引操作

2.1 创建索引与映射字段

> 语法

请求方式依然是PUT

```
PUT /索引库名
{
   "mappings": {
       "类型名称":{
          "properties": {
                "字段名": {
                  "type": "类型",
                  "index": true,
                  "store": true,
                  "analyzer": "分词器"
           }
       }
   }
}
```

- 类型名称：就是前面将的type的概念，类似于数据库中的不同表
  字段名：任意填写，可以指定许多属性，例如：
  - type：类型，可以是text、long、short、date、integer、object等
  - index：是否索引，默认为true
  - store：是否单独存储，默认为false ，一般内容比较多的字段设置成true，可提升查询性能
  - analyzer：分词器

> 示例

发起请求：

```
#创建索引结构
PUT sku
{
  "mappings": {
    "doc":{
      "properties":{
        "name":{
          "type":"text",
          "analyzer":"ik_smart"
        },
        "price":{
          "type":"integer"
        },
        "image":{
          "type":"text"
        },
        "createTime":{
          "type":"date"
        },
        "spuId":{
          "type":"text"
        },
        "categoryName":{
          "type":"keyword"
        },
        "brandName":{
          "type":"keyword"
        },
        "spec":{
          "type":"object"
        },
        "saleNum":{
          "type":"integer"
        },
        "commentNum":{
          "type":"integer"
        }
      }
    }

  }
```

```
}
```

响应结果：

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

2.2 文档增加与修改

2.2.1 增加文档自动生成ID

通过POST请求，可以向一个已经存在的索引库中添加数据。

> 语法：

```
POST 索引库名/类型名
{
    "key":"value"
}
```

> 示例：

```
POST sku/doc
{
  "name":"小米手机",
  "price":1000,
  "spuId":"101",
  "createTime":"2019-03-01",
  "categoryName":"手机",
  "brandName":"小米",
  "saleNum":10102,
  "commentNum":1331,
  "spec":{
    "网络制式":"移动4G",
    "屏幕尺寸":"4.5"
  }
}
```

响应：

```
{
  "_index": "sku",
  "_type": "doc",
  "_id": "hyjXKWkBtgZXp--BshX_",
  "_version": 1,
  "result": "created",
  "_shards": {
    "total": 2,
    "successful": 1,
    "failed": 0
  },
  "_seq_no": 0,
  "_primary_term": 1
}
```

通过以下命令查询sku索引的数据

```
GET sku/_search
```

响应结果如下：

```
{
  "took": 2,
  "timed_out": false,
  "_shards": {
    "total": 5,
    "successful": 5,
    "skipped": 0,
    "failed": 0
  },
  "hits": {
    "total": 1,
    "max_score": 1,
    "hits": [
      {
        "_index": "sku",
        "_type": "doc",
        "_id": "hyjXKWkBtgZXp--BshX_",
        "_score": 1,
        "_source": {
          "name": "小米手机",
          "price": 1000,
          "spuId": 1010101011,
          "createTime": "2019-03-01",
          "categoryName": "手机",
          "brandName": "小米",
          "saleNum": 10102,
          "commentNum": 1331,
          "spec": {
            "网络制式": "移动4G",
            "屏幕尺寸": "4.5"
          }
        }
      }
    ]
  }
}
```


2.2.2 新增文档指定ID

如果我们想要自己新增的时候指定id，可以这么做：

> 语法

```
PUT /索引库名/类型/id值
{
    ...
}
```

> 示例：

```
PUT sku/doc/1
{
  "name":"小米电视",
  "price":1000,
  "spuId":1010101011,
  "createTime":"2019-03-01",
  "categoryName":"电视",
  "brandName":"小米",
  "saleNum":10102,
  "commentNum":1331,
  "spec":{
    "网络制式":"移动4G",
    "屏幕尺寸":"4.5"
  }
}
```

> 响应：

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

再次查询，观察返回的结果。

2.3 索引查询

> 基本语法

```
GET /索引库名/_search
{
    "query":{
        "查询类型":{
            "查询条件":"查询条件值"
        }
    }
}
```

这里的query代表一个查询对象，里面可以有不同的查询属性

- 查询类型：
  - 例如：`match_all`，`match`，`term`，`range` 等等
- 查询条件：查询条件会根据类型的不同，写法也有差异，后面详细讲解

  为了看到清晰的测试效果，我们再录入两条数据：三星手机和三星电视

2.3.1 查询所有数据（match_all）

> 示例：

```
#查询所有
GET /sku/_search
{
    "query":{
        "match_all": {}
    }
}
```

- `query`：代表查询对象
- `match_all`：代表查询所有

> 结果：

```
{
  "took": 2,
  "timed_out": false,
  "_shards": {
    "total": 5,
    "successful": 5,
    "skipped": 0,
    "failed": 0
  },
  "hits": {
    "total": 4,
    "max_score": 1,
    "hits": [
      {
        "_index": "sku",
        "_type": "doc",
        "_id": "hyjXKWkBtgZXp--BshX_",
        "_score": 1,
        "_source": {
          "name": "小米手机",
          "price": 1000,
          "spuId": 1010101011,
          "createTime": "2019-03-01",
          "categoryName": "手机",
          "brandName": "小米",
          "saleNum": 10102,
          "commentNum": 1331,
          "spec": {
            "网络制式": "移动4G",
            "屏幕尺寸": "4.5"
          }
        }
      },
      {
        "_index": "sku",
        "_type": "doc",
        "_id": "1",
        "_score": 1,
        "_source": {

          "name": "小米电视",
```

```
        "price": 1000,
        "spuId": 1010101011,
        "createTime": "2019-03-01",
        "categoryName": "电视",
        "brandName": "小米",
        "saleNum": 10102,
        "commentNum": 1331,
        "spec": {
          "网络制式": "移动4G",
          "屏幕尺寸": "4.5"
        }
      }
    },
    {
      "_index": "sku",
      "_type": "doc",
      "_id": "iCjkKWkBtgZXp--BtBWX",
      "_score": 1,
      "_source": {
        "name": "三星手机",
        "price": 1000,
        "spuId": 1010101011,
        "createTime": "2019-03-01",
        "categoryName": "手机",
        "brandName": "三星",
        "saleNum": 10102,
        "commentNum": 1331,
        "spec": {
          "网络制式": "移动4G",
          "屏幕尺寸": "4.5"
        }
      }
    },
    {
      "_index": "sku",
      "_type": "doc",
      "_id": "iSjlKWkBtgZXp--BDRUU",
      "_score": 1,
      "_source": {
        "name": "三星电视",
        "price": 1000,
```

```
            "spuId": 1010101011,
            "createTime": "2019-03-01",
            "categoryName": "电视",
            "brandName": "三星",
            "saleNum": 10102,
            "commentNum": 1331,
            "spec": {
              "网络制式": "移动4G",
              "屏幕尺寸": "4.5"
            }
          }
        }
      ]
    }
}
```

- took：查询花费时间，单位是毫秒
- time_out：是否超时
- _shards：分片信息
- hits：搜索结果总览对象
  - total：搜索到的总条数
  - max_score：所有结果中文档得分的最高分
  - hits：搜索结果的文档对象数组，每个元素是一条搜索到的文档信息
    - _index：索引库
    - _type：文档类型
    - _id：文档id
    - _score：文档得分
    - _source：文档的源数据

2.3.2 匹配查询（match）

> 示例：查询名称包含手机的记录

```
GET /sku/doc/_search
{
   "query": {
     "match":{"name":"手机"}
   }
}
```

> 结果：查询出三星手机和小米手机两条结果

如果查询名称包含电视的，结果为三星电视和小米电视两条结果

如果我们查询“小米电视”会有几条记录被查询出来呢？你可能说会有一条，但我们测试一下会看到结果为小米电视、三星电视、小米手机三条结果，这是为什么呢？这是因为在查询时，会先将搜索关键字进行分词，对分词后的字符串进行查询，只要是包含这些字符串的都是要被查询出来的，多个词之间是`or`的关系。

返回的结果中_score是对这条记录的评分，评分代表这条记录与搜索关键字的匹配度，查询结果按评分进行降序排序。 比如我们刚才搜索“小米电视”，那小米电视这条记录的评分是最高的，排列在最前面。

如果我们需要精确查找，也就是同时包含小米和电视两个词的才可以查询出来，这就需要将操作改为`and`关系:

```
GET /sku/doc/_search
{
   "query": {
     "match":{
       "name":{
         "query": "小米电视",
         "operator":"and"
       }
     }
   }
}
```

查询结果为小米电视一条记录了。

2.3.3 多字段查询（multi_match）

multi_match`与`match类似，不同的是它可以在多个字段中查询

```
GET /sku/_search
{
    "query":{
        "multi_match": {
            "query":    "小米",
            "fields":   [ "name", "brandName","categoryName"]
        }
    }
}
```

本例中，我们会在name、brandName 、categoryName字段中查询`小米`这个词

2.3.4 词条匹配(term)

term查询被用于精确值 匹配，这些精确值可能是数字、时间、布尔或者那些未分词的字符串

```
GET /sku/_search
{
    "query":{
        "term":{
            "price":1000
        }
    }
}
```

2.3.5 多词条匹配(terms)

terms 查询和 term 查询一样，但它允许你指定多值进行匹配。如果这个字段包含了指定值中的任何一个值，那么这个文档满足条件：

```
GET /sku/_search
{
    "query":{
        "terms":{
            "price":[1000,2000,3000]
        }
    }
}
```

2.3.6 布尔组合（bool）

`bool`把各种其它查询通过`must`（与）、`must_not`（非）、`should`（或）的方式进行组合

> 示例：查询名称包含手机的，并且品牌为小米的。

```
GET /sku/_search
{
    "query":{
        "bool":{
            "must": [
                { "match": { "name": "手机" }} ,
                { "term":  {"brandName": "小米"}}
            ]
        }
    }
}
```

> 示例：查询名称包含手机的，或者品牌为小米的。

```
GET /sku/_search
{
    "query":{
        "bool":{
            "should": [
                { "match": { "name": "手机" }} ,
                { "term":  {"brandName": "小米"}}
            ]
        }
    }
}
```

2.3.7 过滤查询

过滤是针对搜索的结果进行过滤，过滤器主要判断的是文档是否匹配，不去计算和判断文档的匹配度得分，所以过滤器性能比查询要高，且方便缓存，推荐尽量使用过滤器去实现查询或者过滤器和查询共同使用。

> 示例：过滤品牌为小米的记录

```
GET /sku/_search
{
    "query":{
        "bool":{
            "filter": [
              {"match":{"brandName": "小米"}}
            ]
        }
    }
}
```

2.3.8 分组查询

> 示例：按分组名称聚合查询，统计每个分组的数量

```
GET /sku/_search
{
    "size" : 0,
    "aggs" : {
        "sku_category" : {
            "terms" : {
              "field" : "categoryName"
            }
        }
    }
}
```

size为0 不会将数据查询出来，目的是让查询更快。

```
{
  "took": 6,
  "timed_out": false,
  "_shards": {
    "total": 5,
    "successful": 5,
    "skipped": 0,
    "failed": 0
  },
  "hits": {
    "total": 5,
    "max_score": 0,
    "hits": []
  },
  "aggregations": {
    "sku_category": {
      "doc_count_error_upper_bound": 0,
      "sum_other_doc_count": 0,
      "buckets": [
        {
          "key": "手机",
          "doc_count": 3
        },
        {
          "key": "电视",
          "doc_count": 2
        }
      ]
    }
  }
}
```


我们也可以一次查询两种分组统计结果：

```
#分组查询
GET sku/_search
{
  "size":0,
  "aggs": {
    "sku_category": {
      "terms": {
        "field": "categoryName"
      }
    },
    "sku_brand": {
      "terms": {
        "field": "brandName"
      }
    }
  }
}
```

结果：

```
{
  "took" : 2,
  "timed_out" : false,
  "_shards" : {
    "total" : 5,
    "successful" : 5,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : 5,
    "max_score" : 0.0,
    "hits" : [ ]
  },
  "aggregations" : {
    "sku_category" : {
      "doc_count_error_upper_bound" : 0,
      "sum_other_doc_count" : 0,
      "buckets" : [
        {
          "key" : "手机",
          "doc_count" : 3
        },
        {
          "key" : "电视",
          "doc_count" : 2
        }
      ]
    },
    "sku_brand" : {
      "doc_count_error_upper_bound" : 0,
      "sum_other_doc_count" : 0,
      "buckets" : [
        {
          "key" : "三星",
          "doc_count" : 2
        },
        {
          "key" : "小米",

          "doc_count" : 2
```

```
        },
        {
          "key" : "华为",
          "doc_count" : 1
        }
      ]
    }
  }
}
```

3 JavaRest 高级客户端入门

3.1 JavaRest 高级客户端简介

elasticsearch 存在三种Java客户端。

1. Transport Client
2. Java Low Level Rest Client（低级rest客户端）
3. Java High Level REST Client（高级rest客户端）

这三者的区别是：

- TransportClient没有使用RESTful风格的接口，而是二进制的方式传输数据。
- ES官方推出了Java Low Level REST Client,它支持RESTful。但是缺点是因为TransportClient的使用者把代码迁移到Low Level REST Client的工作量比较大。
- ES官方推出Java High Level REST Client,它是基于Java Low Level REST Client的封装，并且API接收参数和返回值和TransportClient是一样的，使得代码迁移变得容易并且支持了RESTful的风格，兼容了这两种客户端的优点。强烈建议ES5及其以后的版本使用Java High Level REST Client。

准备工作：新建工程，引入依赖

```
<dependency>
    <groupId>org.elasticsearch.client</groupId>
    <artifactId>elasticsearch-rest-high-level-client</artifactId>
    <version>6.5.3</version>
</dependency>
```

3.2 快速入门

3.2.1 新增和修改数据

插入单条数据：

HttpHost ： url地址封装

RestClientBuilder： rest客户端构建器

RestHighLevelClient： rest高级客户端

IndexRequest： 新增或修改请求

IndexResponse：新增或修改的响应结果

```
//1.连接rest接口
HttpHost http=new HttpHost("127.0.0.1",9200,"http");
RestClientBuilder builder= RestClient.builder(http);//rest构建器
RestHighLevelClient restHighLevelClient=new
RestHighLevelClient(builder);//高级客户端对象

//2.封装请求对象
IndexRequest indexRequest=new IndexRequest("sku","doc","3");
Map skuMap =new HashMap();
skuMap.put("name","华为p30pro");
skuMap.put("brandName","华为");
skuMap.put("categoryName","手机");
skuMap.put("price",1010221);
skuMap.put("createTime","2019-05-01");
skuMap.put("saleNum",101021);
skuMap.put("commentNum",10102321);
Map spec=new HashMap();
spec.put("网络制式","移动4G");
spec.put("屏幕尺寸","5");
skuMap.put("spec",spec);
indexRequest.source(skuMap);

//3.获取响应结果
IndexResponse response = restHighLevelClient.index(indexRequest,
RequestOptions.DEFAULT);
int status = response.status().getStatus();
System.out.println(status);

restHighLevelClient.close();
```

如果ID不存在则新增，如果ID存在则修改。

批处理请求：

BulkRequest： 批量请求（用于增删改操作）

BulkResponse：批量请求（用于增删改操作）

```
//1.连接rest接口
HttpHost http=new HttpHost("127.0.0.1",9200,"http");
RestClientBuilder builder= RestClient.builder(http);//rest构建器
RestHighLevelClient restHighLevelClient=new
RestHighLevelClient(builder);//高级客户端对象

//2.封装请求对象
BulkRequest bulkRequest=new BulkRequest();
IndexRequest indexRequest=new IndexRequest("sku","doc","4");
Map skuMap =new HashMap();
skuMap.put("name","华为p30pro 火爆上市");
skuMap.put("brandName","华为");
skuMap.put("categoryName","手机");
skuMap.put("price",1010221);
skuMap.put("createTime","2019-05-01");
skuMap.put("saleNum",101021);
skuMap.put("commentNum",10102321);
Map spec=new HashMap();
spec.put("网络制式","移动4G");
spec.put("屏幕尺寸","5");
skuMap.put("spec",spec);

indexRequest.source(skuMap);
bulkRequest.add(indexRequest);//可以多次添加
//3.获取响应结果
BulkResponse response =
restHighLevelClient.bulk(bulkRequest,RequestOptions.DEFAULT);
int status = response.status().getStatus();
System.out.println(status);

String message = response.buildFailureMessage();
System.out.println(message);
restHighLevelClient.close();
```

3.2.2 匹配查询

SearchRequest： 查询请求对象

SearchResponse：查询响应对象

SearchSourceBuilder：查询源构建器

MatchQueryBuilder：匹配查询构建器

> 示例：查询商品名称包含手机的记录。

```
//1.连接rest接口
HttpHost http=new HttpHost("127.0.0.1",9200,"http");
RestClientBuilder builder= RestClient.builder(http);//rest构建器
RestHighLevelClient restHighLevelClient=new
RestHighLevelClient(builder);//高级客户端对象

//2.封装查询请求
SearchRequest searchRequest=new SearchRequest("sku");
searchRequest.types("doc"); //设置查询的类型

SearchSourceBuilder searchSourceBuilder=new SearchSourceBuilder();
MatchQueryBuilder matchQueryBuilder= QueryBuilders.matchQuery("name","手
机");
searchSourceBuilder.query(matchQueryBuilder);
searchRequest.source(searchSourceBuilder);

//3.获取查询结果
SearchResponse searchResponse = restHighLevelClient.search(searchRequest,
RequestOptions.DEFAULT);
SearchHits searchHits = searchResponse.getHits();
long totalHits = searchHits.getTotalHits();
System.out.println("记录数: "+totalHits);
SearchHit[] hits = searchHits.getHits();
for(SearchHit hit:hits){
    String source = hit.getSourceAsString();
    System.out.println(source);
}
restHighLevelClient.close();
```

3.2.3 布尔与词条查询

BoolQueryBuilder：布尔查询构建器

TermQueryBuilder：词条查询构建器

QueryBuilders：查询构建器工厂

示例：查询名称包含手机的，并且品牌为小米的记录

```
//1.连接rest接口  同上......

//2.封装查询请求
SearchRequest searchRequest=new SearchRequest("sku");
searchRequest.types("doc"); //设置查询的类型
SearchSourceBuilder searchSourceBuilder=new SearchSourceBuilder();
BoolQueryBuilder boolQueryBuilder = QueryBuilders.boolQuery();//布尔查询构建器
MatchQueryBuilder matchQueryBuilder= QueryBuilders.matchQuery("name","手机");
boolQueryBuilder.must(matchQueryBuilder);
TermQueryBuilder termQueryBuilder = QueryBuilders.termQuery("brandName","小米");
boolQueryBuilder.must(termQueryBuilder);
searchSourceBuilder.query(boolQueryBuilder);
searchRequest.source(searchSourceBuilder);

//3.获取查询结果  同上......
```

3.2.4 过滤查询

> 示例：筛选品牌为小米的记录

```
//1.连接rest接口  同上.....

//2.封装查询请求
SearchRequest searchRequest=new SearchRequest("sku");
searchRequest.types("doc"); //设置查询的类型
SearchSourceBuilder searchSourceBuilder=new SearchSourceBuilder();
BoolQueryBuilder boolQueryBuilder = QueryBuilders.boolQuery();//布尔查询构建器
TermQueryBuilder termQueryBuilder = QueryBuilders.termQuery("brandName","小米");
boolQueryBuilder.filter(termQueryBuilder);
searchSourceBuilder.query(boolQueryBuilder);
searchRequest.source(searchSourceBuilder);

//3.获取查询结果  同上....
```

3.2.5 分组（聚合）查询

AggregationBuilders：聚合构建器工厂

TermsAggregationBuilder：词条聚合构建器

Aggregations：分组结果封装

Terms.Bucket： 桶

> 示例：按商品分类分组查询，求出每个分类的文档数

```
//1.连接rest接口
HttpHost http=new HttpHost("127.0.0.1",9200,"http");
RestClientBuilder builder= RestClient.builder(http);//rest构建器
RestHighLevelClient restHighLevelClient=new
RestHighLevelClient(builder);//高级客户端对象

//2.封装查询请求
SearchRequest searchRequest=new SearchRequest("sku");
searchRequest.types("doc"); //设置查询的类型
SearchSourceBuilder searchSourceBuilder=new SearchSourceBuilder();

TermsAggregationBuilder termsAggregationBuilder =
AggregationBuilders.terms("sku_category").field("categoryName");
searchSourceBuilder.aggregation(termsAggregationBuilder);
searchSourceBuilder.size(0);
searchRequest.source(searchSourceBuilder);

//3.获取查询结果
SearchResponse searchResponse = restHighLevelClient.search(searchRequest,
RequestOptions.DEFAULT);
Aggregations aggregations = searchResponse.getAggregations();

Map<String, Aggregation> asMap = aggregations.getAsMap();
Terms terms = (Terms) asMap.get("sku_category");
List<? extends Terms.Bucket> buckets = terms.getBuckets();
for(Terms.Bucket bucket:buckets){
    System.out.println(bucket.getKeyAsString()+":"+  bucket.getDocCount()
);
}
restHighLevelClient.close();
```

4 批量数据导入（作业）

4.1 需求分析

单独建立工程，实现青橙商品索引数据的批量导入。

4.2 实现思路

（1）建立工程，引入通用mapper 和elasticsearch依赖

（2）查询sku表数据，将集合数据循环插入到elasticsearch中，参见本章 3.2.1