

# 前面的话

本笔记出自狂神说：<https://www.bilibili.com/video/BV17a4y1x7zq?p=13>

## 前言

1998年9月4日，Google公司在美国硅谷成立。正如大家所知，它是一家做搜索引擎起家的公司。

无独有偶，一位名叫 **Doug Cutting** 的美国工程师，也迷上了搜索引擎。他做了一个用于文本搜索的函数库（姑且理解为软件的功能组件），命名为 **Lucene**。

Lucene是用JAVA写成的，目标是为各种中小型应用软件加入 **全文检索** 功能。因为好用而且 **开源**，非常受程序员们的欢迎。

Lucene 是一套信息检索工具包！jar包！ **不包含 搜索引擎系统**

包含的：索引结构，读写索引的工具，排序，搜索规则.... 工具类

### Lucene 和 Elasticsearch 关系

- Elasticsearch 是基于 Lucene 做了一些封装和增强（我们上手是十分简单！）

## ElasticSearch概述

Elasticsearch，简称为es，es是一个 **开源** 的 **高扩展的分布式全文检索引擎**，它可以 **近乎实时的存储、检索数据**；本身扩展性很好，可以扩展到上百台服务器，处理PB级别（大数据时代）的数据。es 也使用

Java开发并使用Lucene作为其核心来实现所有索引和搜索的功能，但是它的目的是通过简单的RESTfulAPI来隐藏Lucene的复杂性，从而让全文搜索变得简单。

据国际权威的数据库产品评测机构DB Engines的统计，在2016年1月，ElasticSearch已超过Solr等，成为排名第一的搜索引擎类应用

## ES和Solr的比较

# Elasticsearch简介

---

Elasticsearch是一个 **实时分布式搜索和分析引擎**。它让你以前所未有的速度处理大数据成为可能。

它用于 **全文搜索、结构化搜索、分析** 以及将这三者混合使用

通过简单的 **RESTful API** 来隐藏Lucene的复杂性，从而让全文搜索变得简单

## Solr简介

---

Solr 是 **Apache下的一个顶级开源项目**，采用Java开发，它是基于Lucene的全文搜索服务器。

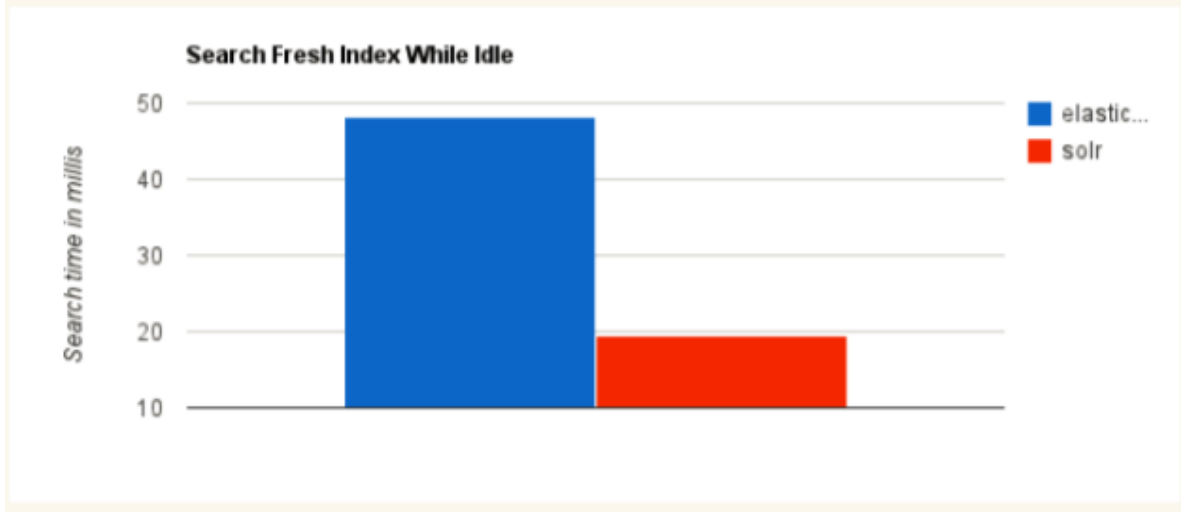
Solr提供了比Lucene更为丰富的查询语言，同时实现了 **可配置、可扩展，并对索引、搜索性能进行了优化**

## 两者对比

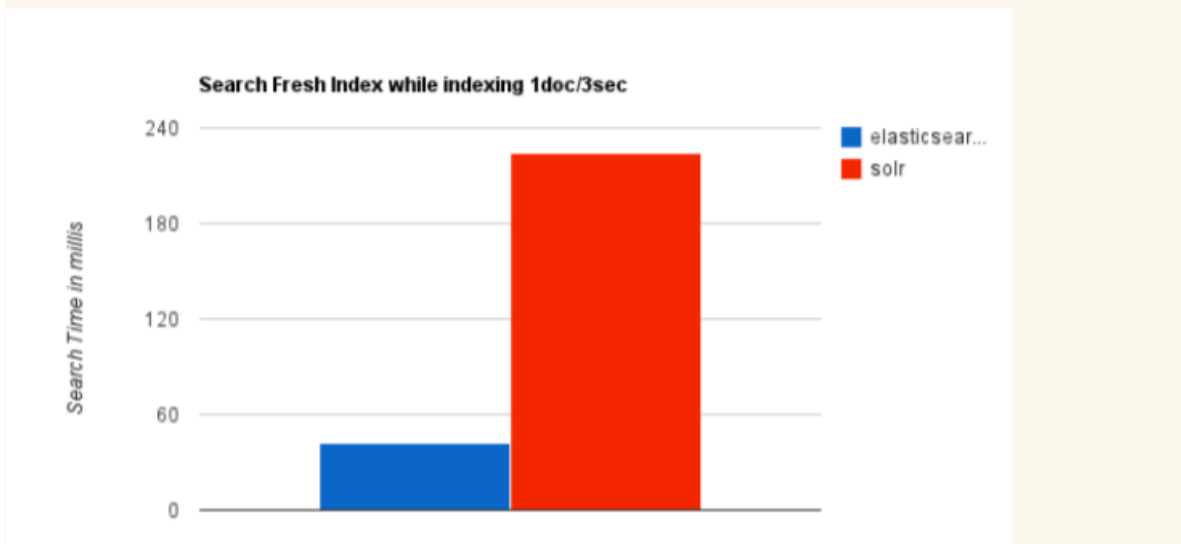
---

Solr	ES
支持Json, Xml, CS	仅支持Json
安装稍微复杂	开箱即用
利用Zookeeper进行分布式管理	自身带有分布式协调管理功能
提供的功能更多	更注重核心功能，更多功能以第三方插件提供
查询快，更新索引慢	建立索引快，查询慢（实时性查询快）
基于传统搜索应用的有力解决方案	新兴的实时搜索
比较成熟，用户更多，开发者社区更大	相对开发维护者较少，更新太快，学习成本较高

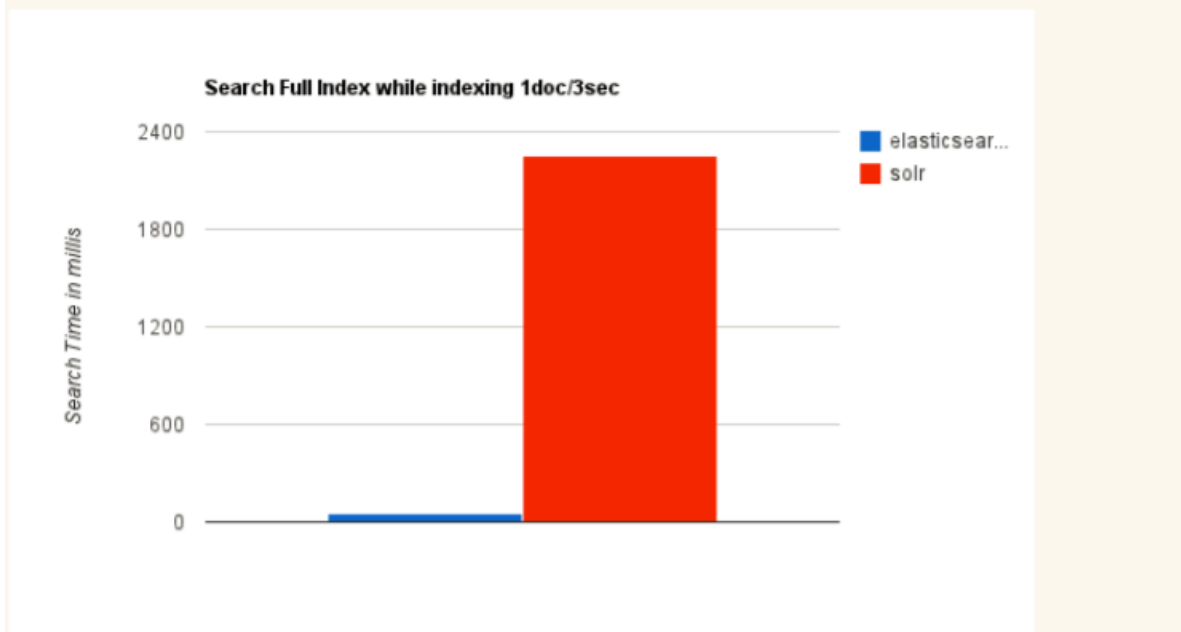
当单纯的对已有数据进行搜索时，Solr更快。



当实时建立索引时，Solr会产生io阻塞，查询性能较差，Elasticsearch具有明显的优势。



随着数据量的增加，Solr的搜索效率会变得更低，而Elasticsearch却没有明显的变化。



# ELK

ELK是 **Elasticsearch**、**Logstash**、**Kibana** 三大开源框架首字母大写简称。市面上也被成为 **Elastic Stack**

## Logstash是ELK的中央数据流引擎

用于从不同目标（文件/数据存储/MQ）收集的不同格式数据，经过过滤后支持输出到不同目的地：

- 文件
- MQ
- redis
- elasticsearch
- kafka

## Kibana

可以将elasticsearch的数据通过友好的页面展示出来，提供实时分析的功能。

市面上很多开发只要提到ELK能够一致说出它是一个日志分析架构技术栈总称。

但实际上ELK不仅仅适用于日志分析，它还可以支持其它任何数据分析和收集的场景，日志分析和收集只是更具有代表性。

# ES安装

官网：<https://www.elastic.co/>

## windows下：解压即用

### 1. 文件目录

```
1 bin 启动文件
2 config 配置文件
3     log4j2 日志配置文件
4     jvm.options java 虚拟机相关的配置
5     elasticsearch.yml elasticsearch 的配置文件！ 默认 9200 端口！ 跨域！
6 lib 相关jar包
7 logs 日志！
8 modules 功能模块
9 plugins 插件！
```

### 2. 启动，访问9200

# Windows下安装可视化界面

1. 需要前端环境，比如nodejs等
2. 下载地址：<https://github.com/mobz/elasticsearch-head/>

```
1 npm install
```

3. 启动，访问9100

```
1 npm run start
```

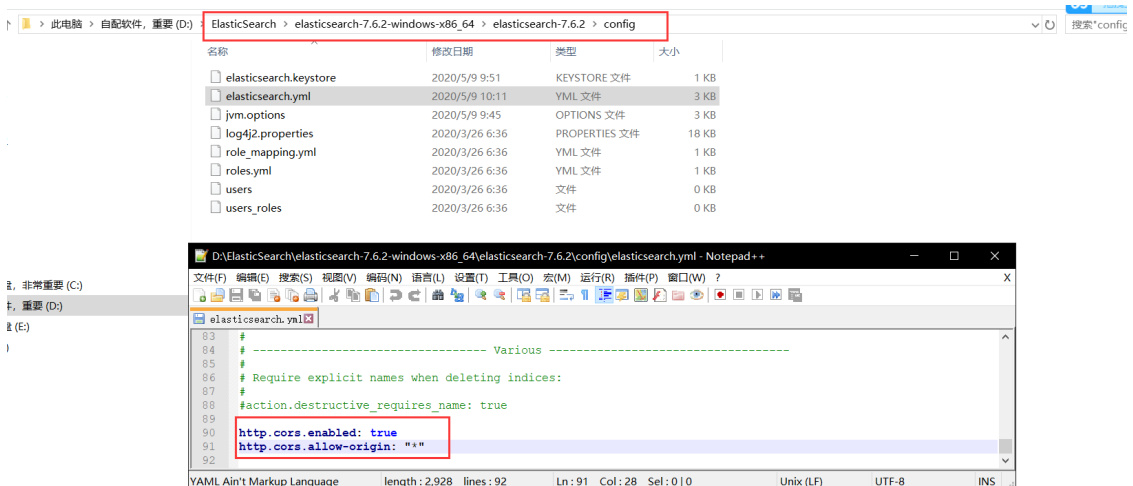
```
D:\ElasticSearch\elasticsearch-head-master>npm run start

> elasticsearch-head@0.0.0 start D:\ElasticSearch\elasticsearch-head-master
> grunt server

Running "connect:server" (connect) task
Waiting forever...
Started connect web server on http://localhost:9100
```

4. 连接测试，发现有一个跨域问题，访问不到ES，那么我们配置ES

```
1 http.cors.enabled: true
2 http.cors.allow-origin: "*"
```



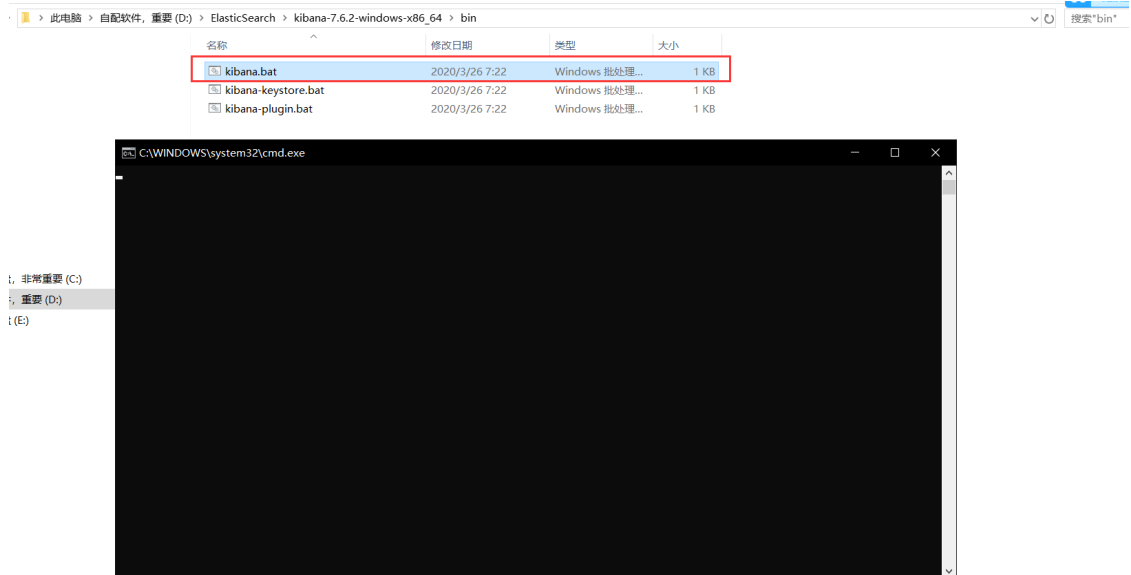
5. 重启ES服务，测试连接



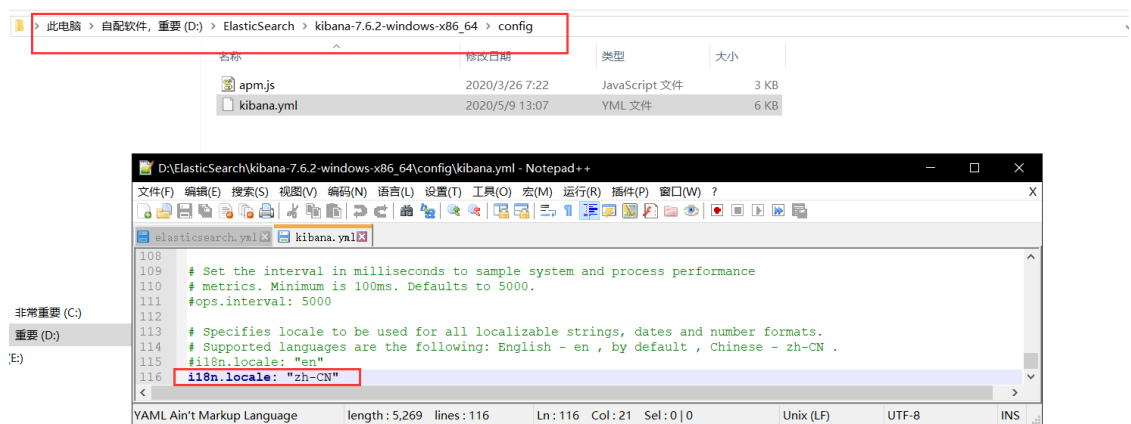
# 安装Kibana

**注意，Kibana要和ES的版本一致，否则会出大问题**

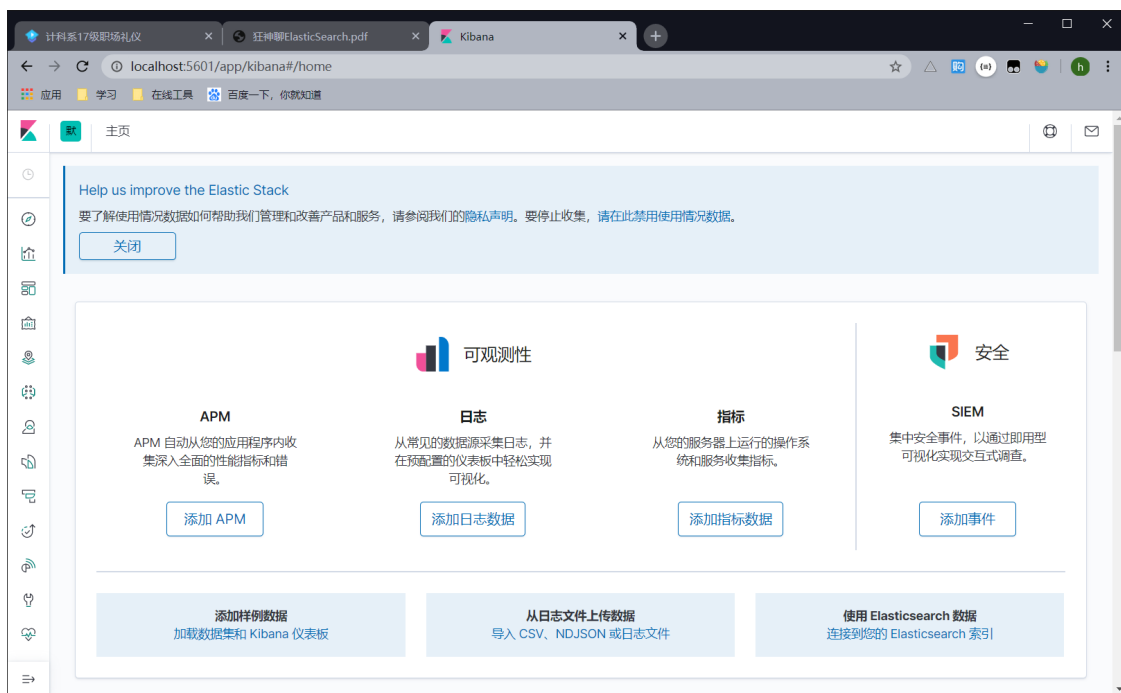
1. 官网：<https://www.elastic.co/cn/kibana>
2. 好消息是解压即可，坏消息是解压要半小时
3. 启动测试



4. 访问5601，出现了界面
5. 汉化：默认是英文版的，但是Kibana有中文版的，只需要配置即可：zh-CN



6. 配置完成之后再次重启，进入5601



## ES核心概念

1. 索引：数据库，非常多文档的集合
2. 字段类型：整型，浮点型，.....
3. 文档：一条条的数据，Json格式

### 物理设计：

ES在后台将 **每一个索引划分为多个分片**，**每一个分片可以在集群中的不同服务器之间迁移**。

如果没有集群配置，那么一个人就是一个集群，默认集群名字是elasticsearch

### 逻辑设计：

我们寻找文档，可以通过：索引-->类型-->文档ID来找到它

### 倒排索引

假设现在有两个文档：

- 1 Study every day, good good up to forever # 文档1包含的内容
- 2 To forever, study every day, good good up # 文档2包含的内容

现在将每一个词条抽出来，然后创建一个不重复的排序列表，这就是倒排索引

term	doc_1	doc_2
Study	✓	x
To	✗	✓
every	✓	✓
forever	✓	✓
day	✓	✓
student	✗	✓
good	✓	✓
to	✓	✗
up	✓	✓

那么我们现在想要根据倒排索引查询：to forever

term	doc_1	doc_2
to	✓	✗
forever	✓	✓

那么两个文档都能匹配到，但是doc\_1匹配度（权重）更高，所以默认按照权重排序的话，doc\_1在前面。

再看另外的实例：

博客文章(原始数据)		索引列表(倒排索引)	
博客文章ID	标签	标签	博客文章ID
1	python	python	1, 2, 3
2	python	linux	3, 4
3	linux, python		
4	linux		

假如我要查找python，不会到文档4去查找，因为倒排索引文档中根本就没有。

索引

当我们想要创建索引时：





在刚才的物理设计中我们说过，一个索引有多个分片，每一个分片都可以在集群内的不同服务器中进行转移。

现在引入另外一个概念：节点。

一个节点是一个ES进程，节点可以有多个索引。

默认的，如果你创建索引，那么 **索引将会有五个分片**（如上图），这五个分片又称主分片。

**每一个主分片都会有一个副本**，副本又称复制分片。

但是主分片和复制分片不会在同一个节点内，这样做的原因是：

假如有一个节点挂掉了，数据也不会丢失。

**事实上，一个分片是一个Lucene索引，包含倒排索引的文件目录**



上图就是一个集群中的多节点。我们可以看到索引的主分片和复制分片不在同一个节点内。

---

## IK分词器插件

### 什么是IK分词器

---

分词器有很多种，ES内置的有很多，但是专为中文分词还真没有几个。

IK分词器，专为中文分词。

## 安装

1. 下载: <https://github.com/medcl/elasticsearch-analysis-ik>
2. 下载完成之后，放到ES插件中即可，会自动读取



3. 重启观察ES，发现被加载了

```
[2020-05-29T15:56:04.540][INFO ][o.e.p.PluginsService] [BEAN] loaded module [x-pack-watcher]
[2020-05-29T15:56:04.541][INFO ][o.e.p.PluginsService] [BEAN] loaded plugin [analysis-ik]
```

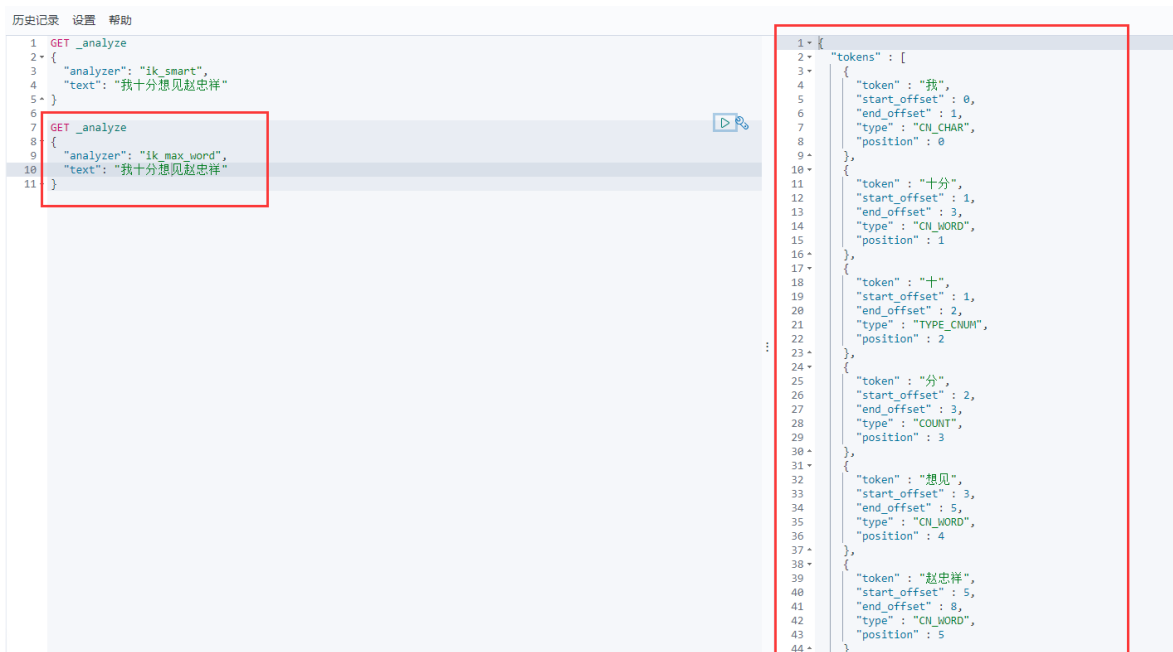
## 使用

IK分词器有两种使用方式：

### 最小切分



### 最细粒度划分

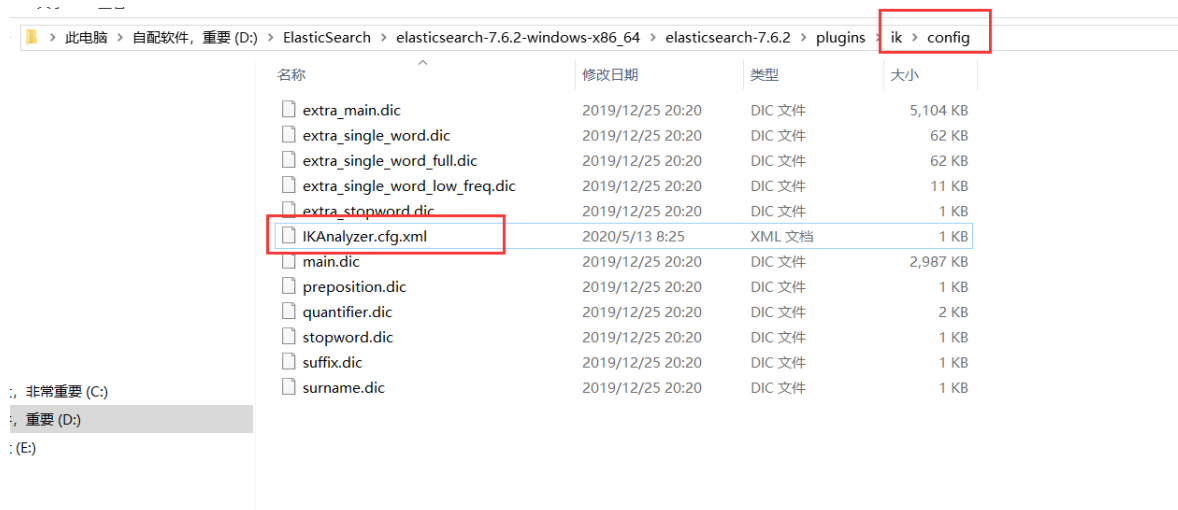


这么一看可能还不太清楚，那就说清楚点：

- 最小切分：按照一个词一个词地拆分，拆成最小的 **词语**。
- 最细粒度划分：**不管是词还是字**，只要在字典里的，那就拆。

## 配置字典

经过上面的基本使用，那么问题来了：词典在哪呢？



这就是词典的路径，如果要添加词典，那么就要打开这个文件，添加自己的词典

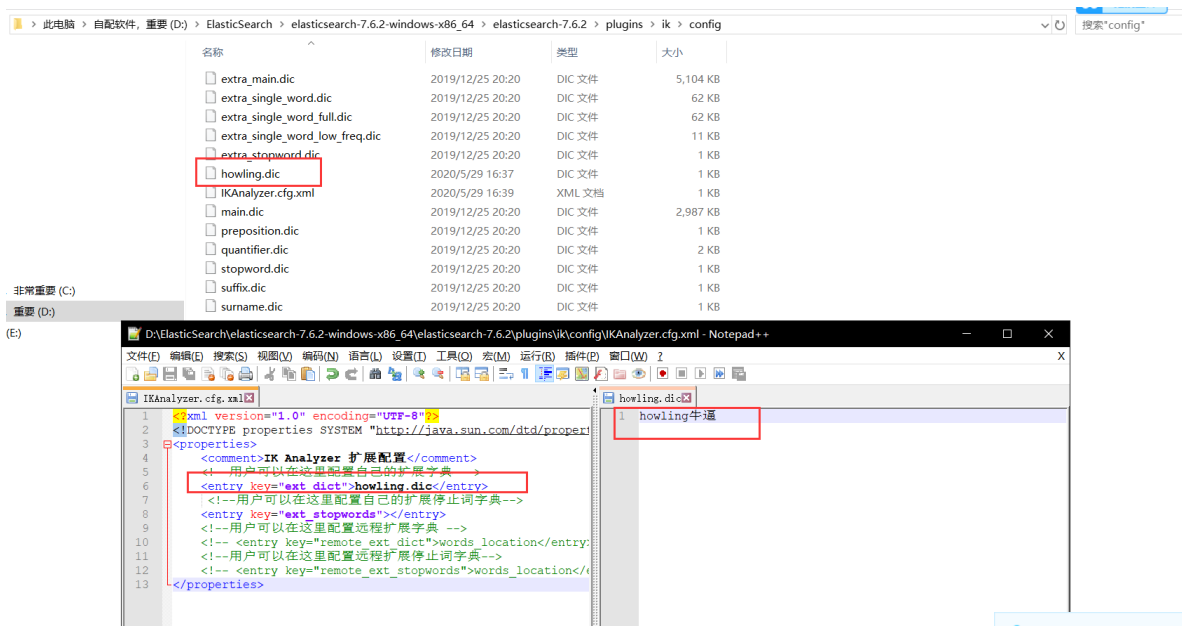
```
IKAnalyzer.cfg.xml
1  <?xml version="1.0" encoding="UTF-8"?>
2  <!DOCTYPE properties SYSTEM "http://java.sun.com/dtd/properties.dtd">
3  <properties>
4      <comment>IK Analyzer 扩展配置</comment>
5      <!--用户可以在这里配置自己的扩展字典 -->
6      <entry key="ext_dict"></entry>
7      <!--用户可以在这里配置自己的扩展停止词字典-->
8      <entry key="ext_stopwords"></entry>
9      <!--用户可以在这里配置远程扩展字典 -->
10     <!-- <entry key="remote_ext_dict">words_location</entry> -->
11     <!--用户可以在这里配置远程扩展停止词字典-->
12     <!-- <entry key="remote_ext_stopwords">words_location</entry> -->
13 </properties>
```

举个例子，我输入 `howling牛逼`，但是并没有这个词

```
历史记录 设置 帮助
1 GET _analyze
2 {
3   "analyzer": "ik_smart",
4   "text": "howling牛逼"
5 }
6

1 {
2   "tokens": [
3     {
4       "token": "howling",
5       "start_offset": 0,
6       "end_offset": 7,
7       "type": "ENGLISH",
8       "position": 0
9     },
10    {
11      "token": "牛",
12      "start_offset": 7,
13      "end_offset": 8,
14      "type": "CN_CHAR",
15      "position": 1
16    },
17    {
18      "token": "逼",
19      "start_offset": 8,
20      "end_offset": 9,
21      "type": "CN_CHAR",
22      "position": 2
23    }
24  ]
25 }
26 }
```

下面我自己定义一个词典，然后重新启动ES



在启动的过程中，发现IK分词器插入了一个自己配置的字典：

```
[2020-05-29T16:42:00.075][INFO ][o.e.g.GatewayService] [BEAN] recovered [5] indices into cluster state
[2020-05-29T16:42:00.264][INFO ][o.e.h.AbstractHttpServerTransport] [BEAN] publish_address {127.0.0.1:9200}, bound_addresses {[::1]:9200}
[2020-05-29T16:42:00.279][INFO ][o.e.n.Node] [BEAN] started
[2020-05-29T16:42:00.557][INFO ][o.w.a.d.Monitor] [BEAN] try load config from D:\ElasticSearch\elasticsearch-7.6.2-windows-x86_64\elasticsearch-7.6.2\config\analysis-ik\IKAnalyzer.cfg.xml
[2020-05-29T16:42:00.567][INFO ][o.w.a.d.Monitor] [BEAN] try load config from D:\ElasticSearch\elasticsearch-7.6.2-windows-x86_64\elasticsearch-7.6.2\plugins\ik\config\IKAnalyzer.cfg.xml
[2020-05-29T16:42:00.876][INFO ][o.w.a.d.Monitor] [BEAN] [Dict Loading] D:\ElasticSearch\elasticsearch-7.6.2-windows-x86_64\elasticsearch-7.6.2\plugins\ik\config\howling.dic
[2020-05-29T16:42:05.699][INFO ][o.e.c.r.a.AllocationService] [BEAN] Cluster health status changed from [RED] to [YELLOW] (reason: [shards started [{bean}0]]).
```

下面再次请求

历史记录 设置 帮助

```
1 GET _analyze
2 {
3   "analyzer": "ik_smart",
4   "text": "howling牛逼"
5 }
6
```

```
1 {
2   "tokens" : [
3     {
4       "token" : "howling牛逼",
5       "start_offset" : 0,
6       "end_offset" : 9,
7       "type" : "CN_WORD",
8       "position" : 0
9     }
10  ]
11 }
12
```

有了

## ES的Rest请求说明

method	例子	描述
PUT	索引名字/类型名字/文档id	创建文档（指定id）
POST	索引名字/类型名字	创建文档（随机指定id）
POST	索引名字/类型名字/文档id/_update	修改文档
DELETE	索引名字/类型名字/文档id	删除文档
GET	索引名字/类型名字/文档id	通过文档id查询文档
POST	索引名字/类型名字/_search	查询所有数据

**注意，REST全部都要大写**

### PUT

PUT创建

```
1 PUT /test2/_doc/1
2 {
3   "name": "howling",
4   "age": 3
5 }
```

```
历史记录 设置 帮助
1 PUT test1/type1/1
2 {
3   "name": "howling",
4   "age": 3
5 }

1 #! Deprecation: [types removal] Specifying types in document index requests is
2 deprecated, use the typeless endpoints instead (/index/_doc/{id}, /index/_doc, or
3 /index/_create/{id}).
4 {
5   "index": "test1",
6   "type": "type1",
7   "id": "1",
8   "version": 1,
9   "result": "created",
10  "shards": {
11    "total": 2,
12    "successful": 1,
13    "failed": 0
14  },
15  "seq_no": 0,
16  "primary_term": 1
17 }
```

看这里，警告说不推荐使用类型，这是因为类型在以后的ES中将会废弃，以后的默认类型就是 `_doc`，我们听官方的

```
历史记录 设置 帮助
1 PUT /test2/_doc/1
2 {
3   "name": "howling",
4   "age": 3
5 }

1 {
2   "index": "test2",
3   "type": "_doc",
4   "id": "1",
5   "version": 1,
6   "result": "created",
7   "shards": {
8     "total": 2,
9     "successful": 1,
10    "failed": 0
11  },
12  "seq_no": 0,
13  "primary_term": 1
14 }
15
```

## POST

### POST创建

```
1 POST /test1/_doc
2 {
3   "name": "howling",
4   "age": 3
5 }

1 {
2   "index": "test1",
3   "type": "_doc",
4   "id": "w16mX31B%wMBNnUPxx 8",
5   "version": 1,
6   "result": "created",
7   "shards": {
8     "total": 2,
9     "successful": 1,
10    "failed": 0
11  },
12  "seq_no": 0,
13  "primary_term": 1
14 }
15
```

### POST修改

```
1 POST /test1/_doc/w16mX3IBMWmBNnUPxx_8/_update
2 {
3   "doc": {
4     "name": "bean",
5     "age": 113
6   }
7 }
```

历史记录 设置 帮助

```
1 POST /test1/_doc/w16mX3IBMWmBNnUPxx_8/_update
2 {
3   "doc": {
4     "name": "bean",
5     "age": 113
6   }
7 }
```

```
1 #! Deprecation: [types removal] Specifying types in document update requests is
  deprecated, use the endpoint /{index}/_update/{id} instead.
2 {
3   "index": "test1",
4   "type": "_doc",
5   "id": "w16mX3IBMWmBNnUPxx_8",
6   "version": 2,
7   "result": "noop",
8   "shards": {
9     "total": 0,
10    "successful": 0,
11    "failed": 0
12  },
13   "seq_no": 1,
14   "_primary_term": 1
15 }
16
```

## POST查询

```
1 POST test4/_doc/_search
```

历史记录 设置 帮助

```
1 POST test4/_doc/_search
```

```
1 #! Deprecation: [types removal] Specifying types in search requests is
  deprecated.
2 {
3   "took": 0,
4   "timed_out": false,
5   "shards": {
6     "total": 1,
7     "successful": 1,
8     "skipped": 0,
9     "failed": 0
10  },
11   "hits": {
12     "total": {
13       "value": 1,
14       "relation": "eq"
15     },
16     "max_score": 1.0,
17     "hits": [
18       {
19         "_index": "test4",
20         "_type": "doc",
21         "_id": "1",
22         "_score": 1.0,
23         "_source": {
24           "name": "howling",
25           "age": 3
26         }
27       }
28     ]
29   }
30 }
31
```

## DELETE

### DELETE删除文档

```
1 DELETE test1/_doc/w16mX3IBMWmBNnUPxx_8
```

控制台Search ProfilerGrok Debugger

历史记录 设置 帮助

1DELETEtest1/\_doc/w16mX31BMWmBNnUPxx\_8

1- {

"\_index" : "test1",

"\_type" : "\_doc",

"\_id" : "w16mX31BMWmBNnUPxx\_8",

"\_version" : 3,

"result" : "deleted",

"shards" : {

"total" : 2,

"successful" : 1,

"failed" : 0

},

"\_seq\_no" : 3,

"\_primary\_term" : 1

14^ }

15

## GET

GET获取文档

1GET /test4/\_doc/1

历史记录 设置 帮助

1GET /test4/\_doc/1

1- {

"\_index" : "test4",

"\_type" : "\_doc",

"\_id" : "1",

"\_version" : 1,

"\_seq\_no" : 0,

"\_primary\_term" : 1,

"found" : true,

"source" : {

"name" : "howling",

"age" : 3

}

12^ }

13^ }

14

## 修改使用POST和PUT的比较

未修改之前：

查询 1 个分片中用的 1 个. 1 命中. 耗时 0.000 秒

_index	_type	_id	_score ▲	name	age	
test4	_doc	1	1	howling	3	

原来的方法就是使用PUT来修改

```
1PUT test4/_doc/1
2{
3  "name": "bean"
4}
```



查询 1 个分片中用的 1 个. 1 命中. 耗时 0.000 秒

_index	_type	_id	_score ▲	name
test4	_doc	1	1	bean

age 给改没了

现在的方法是使用POST来修改

复原数据，再来一次

```
1 POST test4/_doc/1/_update
2 {
3   "doc": {
4     "name": "bean"
5   }
6 }
```

查询 1 个分片中用的 1 个. 1 命中. 耗时 0.000 秒

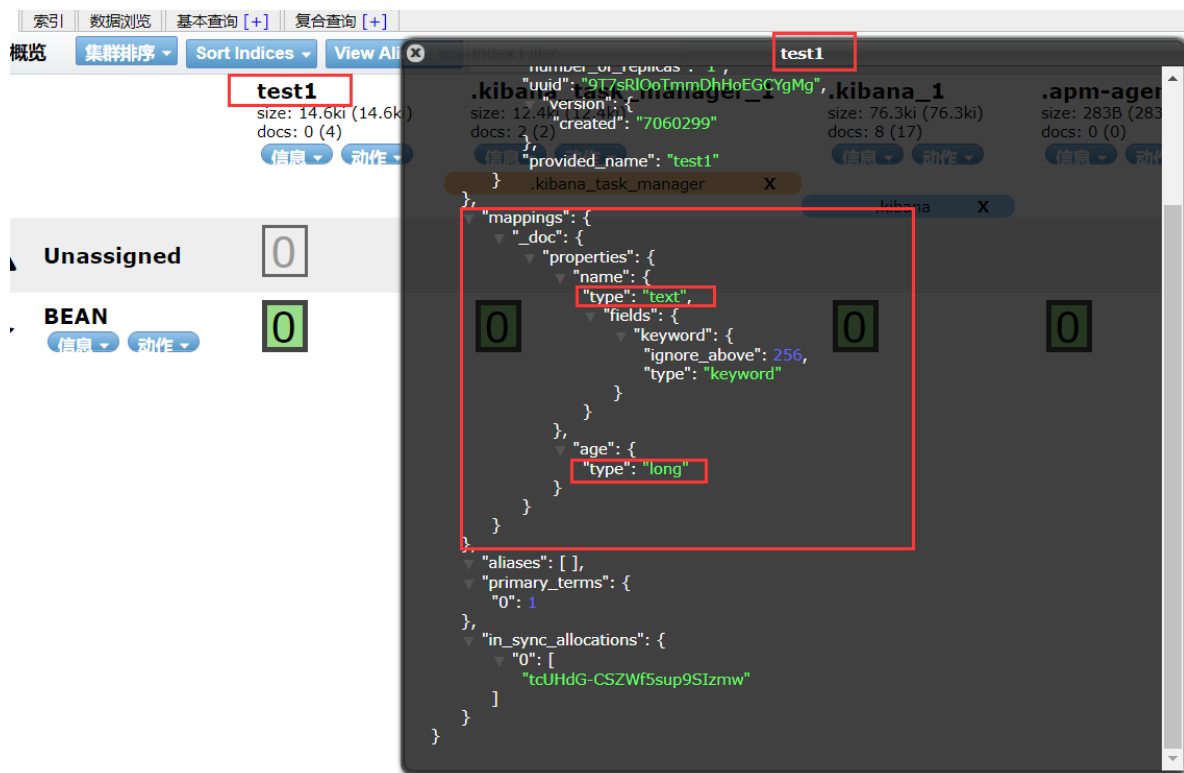
_index	_type	_id	_score ▲	name	age
test4	_doc	1	1	bean	3

发现 age 没有变

所以现在我们通常使用POST来进行修改，虽然写法上有些复杂，但是数据不会丢失

## ES创建索引规则

在上面，索引规则是自动添加的，可以截个图看一下：



从这张图可以看出来，在我们没有指定索引的情况下，ES可以根据我们输入的数据自动推算

## 索引类型

- 字符串: text, keyword
- 数值: byte, short, integer, long, double, float, half\_float, scaled\_float
- 日期: date
- 布尔: boolean
- 二进制: binary
- 等

其中有一些类型的区别我们需要探讨一下，后面会讲到

## 我们也可以手动指定索引类型

我们可以看到，在上面的图中，有一个mappings，指定了类型，这里也是mappings

```
1 PUT /test2
2 {
3   "mappings": {
4     "properties": {
5       "name": {
6         "type": "text"
7       },
8       "age": {
9         "type": "long"
10      },
11      "birthday": {
12        "type": "date"
13      }
14    }
15  }
```

注意，我只是指定到了索引的类型，而没有指定到类型或者是某一个文档

The screenshot shows the Kibana Index Filter interface. The 'test2' index is selected. The 'mappings' section is highlighted with a red box, showing the mapping for '\_doc' with properties: 'birthday' (date), 'name' (text), and 'age' (long). The 'settings' section is also visible, showing the index configuration.

可以通过GET命令查看

The screenshot shows the Kibana console with the command `GET /test2` entered. The output is a JSON object representing the index configuration. The 'mappings' section is highlighted by a red box, showing the mapping for '\_doc' with properties: 'birthday' (date), 'name' (text), and 'age' (long). The 'settings' section is also visible, showing the index configuration.

## 复杂搜索

# 前言

我们之前在REST请求的时候曾经有过搜索，根据ID的最简单搜索，但是我们当然不限于此。我们知道，其实大部分时间都在和查询打交道，所以查询是重点。

接下来请看查询的骚操作

- 排序查询
- 分页查询
- 高亮查询
- 模糊查询
- 精准查询
- 多条件查询
- 多参数匹配查询

## 查询

### term和match，keyword和text的分析对比

在开始之前，我先在文档里面放入四个数据：

查询 1 个分片中用的 1 个. 4 命中. 耗时 0.000 秒							
_index	_type	_id	_score ▲	name	age	tags	
test1	_doc	1	1	bean	13	a	
test1	_doc	2	1	howling	24	b	
test1	_doc	3	1	bean howling	13	a b	
test1	_doc	4	1	howling bean	24	b a	

下面使用这四条数据来进行两组的说明

#### term和match

- term：用于精确匹配
- match：用于分词匹配

这样说可能有点不太好理解，那就用通俗点的说：

term只能查一个词，match能查多个词。

原因在于：term是直接进行倒排索引的方式查询的，而match是通过分词的方式查询的

倒排索引的方式一下只能查询一个词语。

分词查询是先分词，然后在查询

理论很清晰了，下面来看例子

term

```

1 GET /test1/_search
2 {
3   "query": {
4     "term": {
5       "name": "howling"
6     }
7   }
8 }

```

历史记录 设置 帮助

```

1 GET /test1/_search
2 {
3   "query": {
4     "term": {
5       "name": "howling"
6     }
7   }
8 }

```

```

13 {
14   "relation": "eq",
15   "max_score": 0.41299206,
16   "hits": [
17     {
18       "index": "test1",
19       "type": "doc",
20       "id": "2",
21       "score": 0.41299206,
22       "source": {
23         "name": "howling",
24         "age": 24,
25         "tags": "b"
26       }
27     },
28     {
29       "index": "test1",
30       "type": "doc",
31       "id": "3",
32       "score": 0.31387398,
33       "source": {
34         "name": "bean howling",
35         "age": 13,
36         "tags": "a b"
37       }
38     },
39     {
40       "index": "test1",
41       "type": "doc",
42       "id": "4",
43       "score": 0.31387398,
44       "source": {
45         "name": "howling bean",
46         "age": 24,
47         "tags": "a b"
48       }
49     }
50   ]
51 }

```

查询单个词还不错

```

1 GET /test1/_search
2 {
3   "query": {
4     "term": {
5       "name": "howling bean"
6     }
7   }
8 }

```

历史记录 设置 帮助

```

1 GET /test1/_search
2 {
3   "query": {
4     "term": {
5       "name": "howling bean"
6     }
7   }
8 }

```

```

1 {
2   "took": 0,
3   "timed_out": false,
4   "shards": {
5     "total": 1,
6     "successful": 1,
7     "skipped": 0,
8     "failed": 0
9   },
10  "hits": {
11    "total": {
12      "value": 0,
13      "relation": "eq"
14    },
15    "max_score": null,
16    "hits": [ ]
17  }
18 }
19

```

查多个词查不到，原因已经讲过了，倒排索引一次只能查询一个

match

```

1 GET /test1/_search
2 {
3   "query": {
4     "match": {
5       "name": "howling bean"
6     }
7   }
8 }

```

控制台 Search Profiler Grok Debugger

历史记录 设置 帮助

```

1 GET /test1/_search
2 {
3   "query": {
4     "match": {
5       "name": "howling bean"
6     }
7   }
8 }

```

```

24     "age" : 13,
25     "tags" : "a b"
26   },
27   {
28     {
29       "index" : "test1",
30       "type" : "doc",
31       "id" : "4",
32       "score" : 0.62774795,
33       "source" : {
34         "name" : "howling bean",
35         "age" : 24,
36         "tags" : "b a"
37       }
38     },
39     {
40       "index" : "test1",
41       "type" : "doc",
42       "id" : "1",
43       "score" : 0.41299206,
44       "source" : {
45         "name" : "bean",
46         "age" : 13,
47         "tags" : "a"
48       }
49     },
50     {
51       "index" : "test1",
52       "type" : "doc",
53       "id" : "2",
54       "score" : 0.41299206,
55       "source" : {
56         "name" : "howling",
57         "age" : 24,
58         "tags" : "b"

```

四个都查出来了

```

1 GET /test1/_search
2 {
3   "query": {
4     "match": {
5       "name": "howling"
6     }
7   }
8 }

```

历史记录 设置 帮助

```

1 GET /test1/_search
2 {
3   "query": {
4     "match": {
5       "name": "howling"
6     }
7   }
8 }

```

```

18     "index" : "test1",
19     "type" : "doc",
20     "id" : "2",
21     "score" : 0.41299206,
22     "source" : {
23       "name" : "howling",
24       "age" : 24,
25       "tags" : "b"
26     },
27   },
28   {
29     {
30       "index" : "test1",
31       "type" : "doc",
32       "id" : "3",
33       "score" : 0.31387398,
34       "source" : {
35         "name" : "bean howling",
36         "age" : 13,
37         "tags" : "a b"
38       }
39     },
40     {
41       "index" : "test1",
42       "type" : "doc",
43       "id" : "4",
44       "score" : 0.31387398,
45       "source" : {
46         "name" : "howling bean",
47         "age" : 24,
48         "tags" : "b a"
49       }
50     },
51   }

```

查询单个更是不在话下

## keyword和text

- keyword：不能被拆分
- text：可以被拆分

这么说吧，keyword可以被分词器拆分，而text不可以被分词器拆分

在细致点：

- keyword可以看成是一个字
- 而text可以看成是一句话

所以keyword不可以被拆分的意思是，只要有了keyword属性，那么这个属性就相当于一个字，一个字还想怎么被分词器解析呢？

text可以看成一句话，所以一句话可以被分词器解析的

那么就又印出来一个新的操作：

keyword既然可以被看成一个字，那么在term上就可以解析了，因为倒排索引显然可以查询一个字

下面来看一组对比：

### term

```
1 GET /test1/_search
2 {
3   "query": {
4     "term": {
5       "tags": "a b"
6     }
7   }
8 }
```

历史记录 设置 帮助

```
1 GET /test1/_search
2 {
3   "query": {
4     "term": {
5       "tags": "a b"
6     }
7   }
8 }
```

```
1 {
2   "took": 0,
3   "timed_out": false,
4   "_shards": {
5     "total": 1,
6     "successful": 1,
7     "skipped": 0,
8     "failed": 0
9   },
10  "hits": {
11    "total": {
12      "value": 1,
13      "relation": "eq"
14    },
15    "max_score": 1.2039728,
16    "hits": [
17      {
18        "_index": "test1",
19        "_type": "doc",
20        "_id": "3",
21        "_score": 1.2039728,
22        "_source": {
23          "name": "bean howling",
24          "age": 13,
25          "tags": "a b"
26        }
27      }
28    ]
29  }
30 }
31
```

### match

```

1 GET /test1/_search
2 {
3   "query": {
4     "match": {
5       "tags": "a b"
6     }
7   }
8 }

```

历史记录 设置 帮助

```

1 GET /test1/_search
2 {
3   "query": {
4     "match": {
5       "tags": "a b"
6     }
7   }
8 }

```

```

1 {
2   "took" : 0,
3   "timed_out" : false,
4   "shards" : {
5     "total" : 1,
6     "successful" : 1,
7     "skipped" : 0,
8     "failed" : 0
9   },
10  "hits" : {
11    "total" : {
12      "value" : 1,
13      "relation" : "eq"
14    },
15    "max_score" : 1.2039728,
16    "hits" : [
17      {
18        "_index" : "test1",
19        "_type" : "doc",
20        "_id" : "3",
21        "_score" : 1.2039728,
22        "_source" : {
23          "name" : "bean howling",
24          "age" : 13,
25          "tags" : "a b"
26        }
27      }
28    ]
29  }
30 }

```

注意match，假如是使用了分词器解析的时候，tags明显可以被分成 **a**，**b**，**ab** 等词语  
但是现在只是查询出了一个，所以说明没有被分词器解析到

## 我非要让term查询多个词语

这个说明就像标题，我非要让term查询多个词语怎么办呢？

没有办法，但是我们可以进行多次查询。

使用 **terms**，我们可以指定多个词语来进行多次倒排索引的查询，结果一起返回

```

1 GET /test1/_search
2 {
3   "query": {
4     "terms": {
5       "name": ["howling", "bean"]
6     }
7   }
8 }

```



```
历史记录 设置 帮助
1 GET /test1/_search
2 {
3   "query": {
4     "terms": {
5       "name": ["howling", "bean"]
6     }
7   }
8 }

1 {
2   "took" : 1,
3   "timed_out" : false,
4   "shards" : {
5     "total" : 1,
6     "successful" : 1,
7     "skipped" : 0,
8     "failed" : 0
9   },
10  "hits" : {
11    "total" : {
12      "value" : 4,
13      "relation" : "eq"
14    },
15    "max_score" : 1.0,
16    "hits" : [
17      {
18        "_index" : "test1",
19        "_type" : "_doc",
20        "_id" : "1",
21        "score" : 1.0,
22        "source" : {
23          "name" : "bean",
24          "age" : 13,
25          "tags" : "a"
26        }
27      },
28      {
29        "index" : "test1",
```

## 布尔值查询（多条件匹配）

### 前言

bool查询：

- must：等同于and，所有条件都要符合
- should：等同于or，只要符合一个条件即可
- must\_not：等同于not，取反
- filter：过滤器
- gte：大于等于
- lte：小于等于
- gt：大于
- lt：小于

### must, should, must\_not

#### must

```
1 GET /test1/_search
2 {
3   "query": {
4     "bool": {
5       "must": [
6         {
7           "match": {"name": "howling"}
8         },
9         {
10          "match": {"tags": "b"}
11        }
12      ]
13    }
14  }
15 }
```

历史记录 设置 帮助

```
1 GET /test1/_search
2 {
3   "query": {
4     "bool": {
5       "must": [
6         {
7           "match": {"name": "howling"}
8         },
9         {
10          "match": {"tags": "b"}
11        }
12      ]
13    }
14  }
15 }
```

```
1 {
2   "took": 0,
3   "timed_out": false,
4   "_shards": {
5     "total": 1,
6     "successful": 1,
7     "skipped": 0,
8     "failed": 0
9   },
10  "hits": {
11    "total": {
12      "value": 1,
13      "relation": "eq"
14    },
15    "max_score": 1.6169648,
16    "hits": [
17      {
18        "_index": "test1",
19        "_type": "doc",
20        "_id": "2",
21        "_score": 1.6169648,
22        "_source": {
23          "name": "howling",
24          "age": 24,
25          "tags": "b"
26        }
27      }
28    ]
29  }
30 }
31
```

should

1 GET /test1/\_search

```
2 {
3   "query": {
4     "bool": {
5       "should": [
6         {
7           "match": {"name": "howling"}
8         },
9         {
10          "match": {"tags": "b"}
11        }
12      ]
13    }
14  }
15 }
```

控制台 Search Profiler Grok Debugger

历史记录 设置 帮助

```
1 GET /test1/_search
2 {
3   "query": {
4     "bool": {
5       "should": [
6         {
7           "match": {"name": "howling"}
8         },
9         {
10          "match": {"tags": "b"}
11        }
12      ]
13    }
14  }
15 }
```

```
18 {
19   "_index": "test1",
20   "_type": "doc",
21   "_id": "2",
22   "_score": 1.6169648,
23   "_source": {
24     "name": "howling",
25     "age": 24,
26     "tags": "b"
27   }
28 },
29 {
30   "_index": "test1",
31   "_type": "doc",
32   "_id": "3",
33   "_score": 0.31387398,
34   "_source": {
35     "name": "bean howling",
36     "age": 13,
37     "tags": "a b"
38   }
39 },
40 {
41   "_index": "test1",
42   "_type": "doc",
43   "_id": "4",
44   "_score": 0.31387398,
45   "_source": {
46     "name": "howling bean",
47     "age": 24,
48     "tags": "b a"
49   }
50 }
51 ]
52 }
```

must\_not

1 GET /test1/\_search

```

2  {
3    "query": {
4      "bool": {
5        "must_not": [
6          {
7            "match": {"name": "howling"}
8          },
9          {
10           "match": {"tags": "b"}
11         }
12       ]
13     }
14   }
15 }

```

控制台 Search Profiler Grok Debugger

历史记录 设置 帮助

```

1 GET /test1/_search
2 {
3   "query": {
4     "bool": {
5       "must_not": [
6         {
7           "match": {"name": "howling"}
8         },
9         {
10          "match": {"tags": "b"}
11        }
12      ]
13    }
14  }
15 }

```

单击可发送请求



```

1 {
2   "took" : 1,
3   "timed_out" : false,
4   "shards" : {
5     "total" : 1,
6     "successful" : 1,
7     "skipped" : 0,
8     "failed" : 0
9   },
10  "hits" : {
11    "total" : {
12      "value" : 1,
13      "relation" : "eq"
14    },
15    "max_score" : 0.0,
16    "hits" : [
17      {
18        "_index" : "test1",
19        "_type" : "doc",
20        "_id" : "1",
21        "_score" : 0.0,
22        "_source" : {
23          "name" : "bean",
24          "age" : 13,
25          "tags" : "a"
26        }
27      }
28    ]
29  }
30 }
31

```

## filter, gt, lt, gte, lte

一个例子足够

```

1 GET /test1/_search
2 {
3   "query": {
4     "bool": {
5       "must": [
6         {
7           "match": {"name": "howling"}
8         }
9       ],
10      "filter": [
11        {
12          "range": {
13            "age": {
14              "gt": 10,
15              "lt": 20
16            }
17          }
18        }
19      ]
20    }
21  }
22 }

```

```
21     }
22 }
```

历史记录 设置 帮助

```
1 GET /test1/_search
2 {
3   "query": {
4     "bool": {
5       "must": [
6         {
7           "match": {"name": "howling"}
8         }
9       ],
10      "filter": [
11        {
12          "range": {
13            "age": {
14              "gt": 10,
15              "lt": 20
16            }
17          }
18        }
19      ]
20    }
21  }
22 }
```

```
1 {
2   "took": 0,
3   "timed_out": false,
4   "shards": {
5     "total": 1,
6     "successful": 1,
7     "skipped": 0,
8     "failed": 0
9   },
10  "hits": {
11    "total": {
12      "value": 1,
13      "relation": "eq"
14    },
15    "max_score": 0.31387398,
16    "hits": [
17      {
18        "_index": "test1",
19        "_type": "_doc",
20        "_id": "3",
21        "_score": 0.31387398,
22        "_source": {
23          "name": "bean howling",
24          "age": 13,
25          "tags": "a b"
26        }
27      }
28    ]
29  }
30 }
31
```

## 一条条件多参数匹配

其实前面已经见过了

```
1 GET /test1/_search
2 {
3   "query": {
4     "match": {
5       "name": "howling bean"
6     }
7   }
8 }
```

控制台 Search Profiler Grok Debugger

历史记录 设置 帮助

```
1 GET /test1/_search
2 {
3   "query": {
4     "match": {
5       "name": "howling bean"
6     }
7   }
8 }
```

空格隔开，代表一个条件的多个参数

```
15 "max_score" : 0.62774795,
16 "hits" : [
17   {
18     "_index": "test1",
19     "_type": "_doc",
20     "_id": "3",
21     "_score": 0.62774795,
22     "_source": {
23       "name": "bean howling",
24       "age": 13,
25       "tags": "a b"
26     }
27   },
28   {
29     "_index": "test1",
30     "_type": "_doc",
31     "_id": "4",
32     "_score": 0.62774795,
33     "_source": {
34       "name": "howling bean",
35       "age": 24,
36       "tags": "b a"
37     }
38   },
39   {
40     "_index": "test1",
41     "_type": "_doc",
42     "_id": "1",
43     "_score": 0.41299206,
44     "_source": {
45       "name": "bean",
46       "age": 13,
47       "tags": "a"
48     }
49   },
50 ]
51
```

空格隔开，代表一个条件多个参数，这个时候可以使用分词器解析

## 排序

**排序警告：原本我们的排序是使用权重排序的，一旦自定义排序之后，权重就失效了**

排序: `order`

```
1 GET /test1/_search
2 {
3   "query": {
4     "match": {
5       "name": "howling"
6     }
7   },
8   "sort": [
9     {
10      "tags": {
11        "order": "asc"
12      }
13    }
14  ]
15 }
```

根据tags来排序，ASC和DESC应该不用说了

历史记录 设置 帮助

```
1 GET /test1/_search
2 {
3   "query": {
4     "match": {
5       "name": "howling"
6     }
7   },
8   "sort": [
9     {
10      "tags": {
11        "order": "asc"
12      }
13    }
14  ]
15 }
```



```
15 "max_score" : null,
16 "hits" : [
17   {
18     "_index" : "test1",
19     "_type" : "doc",
20     "id" : "3",
21     "score" : null,
22     "source" : {
23       "name" : "bean howling",
24       "age" : 13,
25       "tags" : "a b"
26     },
27     "sort" : [
28       "a b"
29     ]
30   },
31   {
32     "_index" : "test1",
33     "_type" : "doc",
34     "id" : "2",
35     "score" : null,
36     "source" : {
37       "name" : "howling",
38       "age" : 24,
39       "tags" : "b"
40     },
41     "sort" : [
42       "b"
43     ]
44   },
45   {
46     "_index" : "test1",
47     "_type" : "doc",
```

## 分页

分页: `from-size`

- from: 从第几个数据开始
- size: 显示几条数据

```
1 GET /test1/_search
2 {
3   "query": {
4     "match": {
5       "name": "howling"
6     }
7   },
8   "from": 0,
9   "size": 1
10 }
```

历史记录 设置 帮助

```
1 GET /test1/_search
2 {
3   "query": {
4     "match": {
5       "name": "howling"
6     }
7   },
8   "from": 0,
9   "size": 1
10 }
```

```
1 {
2   "took" : 0,
3   "timed out" : false,
4   "_shards" : {
5     "total" : 1,
6     "successful" : 1,
7     "skipped" : 0,
8     "failed" : 0
9   },
10  "hits" : {
11    "total" : {
12      "value" : 3,
13      "relation" : "eq"
14    },
15    "max_score" : 0.41299206,
16    "hits" : [
17      {
18        "_index" : "test1",
19        "_type" : "doc",
20        "_id" : "2",
21        "_score" : 0.41299206,
22        "_source" : {
23          "name" : "howling",
24          "age" : 24,
25          "tags" : "b"
26        }
27      }
28    ]
29  }
30 }
```

## 高亮

高亮: highlight

```
1 GET /test1/_search
2 {
3   "query": {
4     "match": {
5       "name": "howling bean"
6     }
7   },
8   "highlight": {
9     "fields": {
10      "name": {}
11    }
12  }
13 }
```

历史记录 设置 帮助

```

1 GET /test1/_search
2 {
3   "query": {
4     "match": {
5       "name": "howling bean"
6     }
7   },
8   "highlight": {
9     "fields": {
10      "name": {}
11    }
12  }
13 }

```

```

44 {
45   "name": {
46     "<em>howling</em> <em>bean</em>"
47   }
48 },
49 {
50   "_index": "test1",
51   "_type": "doc",
52   "_id": "1",
53   "_score": 0.41299206,
54   "_source": {
55     "name": "bean",
56     "age": 13,
57     "tags": "a"
58   },
59   "highlight": {
60     "name": [
61       "<em>bean</em>"
62     ]
63   }
64 },
65 {
66   "_index": "test1",
67   "_type": "doc",
68   "_id": "2",
69   "_score": 0.41299206,
70   "_source": {
71     "name": "howling",
72     "age": 24,
73     "tags": "b"
74   },
75   "highlight": {
76     "name": [
77       "<em>howling</em>"
78     ]
79   }
80 }

```

看到标签了么，这就是给前端加上的高亮

## 自定义高亮

```

1 GET /test1/_search
2 {
3   "query": {
4     "match": {
5       "name": "howling"
6     }
7   },
8   "highlight": {
9     "pre_tags": "<p style='color:red'>",
10    "post_tags": "</p>",
11    "fields": {
12      "name": {}
13    }
14  }
15 }

```

历史记录 设置 帮助

```

1 GET /test1/_search
2 {
3   "query": {
4     "match": {
5       "name": "howling"
6     }
7   },
8   "highlight": {
9     "pre_tags": "<p style='color:red'>",
10    "post_tags": "</p>",
11    "fields": {
12      "name": {}
13    }
14  }
15 }

```

```

33 {
34   "_index": "test1",
35   "_type": "doc",
36   "_id": "3",
37   "_score": 0.31387398,
38   "_source": {
39     "name": "bean howling",
40     "age": 13,
41     "tags": "a b"
42   },
43   "highlight": {
44     "name": [
45       "bean <p style='color:red'>howling</p>"
46     ]
47   }
48 },
49 {
50   "_index": "test1",
51   "_type": "doc",
52   "_id": "4",
53   "_score": 0.31387398,
54   "_source": {
55     "name": "howling bean",
56     "age": 24,
57     "tags": "b a"
58   },
59   "highlight": {
60     "name": [
61       "<p style='color:red'>howling</p> bean"
62     ]
63   }
64 },
65 }
66 }

```

这是自定义高亮

# SpringBoot集成ES

一切从官方文档开始：<https://www.elastic.co/guide/index.html>

- [Elastic Common Schema \(ECS\) Reference \[1.5\]](#) — other versions
- [Azure Marketplace and Resource Manager \(ARM\) template \[7.7\]](#) — other versions

☐ 一律翻译英

Google Translate

## Elasticsearch: Store, Search, and Analyze

- [Elasticsearch Reference \[7.7\]](#) — other versions
- [Elasticsearch Resiliency Status](#)
- [Painless Scripting Language \[7.7\]](#) — other versions
- [Plugins and Integrations \[7.7\]](#) — other versions
- [Elasticsearch Clients](#) ES客户端
- [Elasticsearch for Apache Hadoop and Spark \[7.7\]](#) — other versions
- [Curator Index Management \[5.8\]](#) — other versions

## Cloud: Provision, Manage and Monitor the Elastic Stack

- [Elasticsearch Service - Hosted Elastic Stack](#)
- [Elasticsearch Add-On for Heroku - Hosted Elasticsearch and Kibana for Heroku Users](#)
- [Elastic Cloud Enterprise - Elastic Cloud on your Infrastructure \[2.5\]](#) — other versions

- [Java REST Client \[7.7\]](#) — other versions

推荐使用RESTFUL

- [Java API \[7.7\]](#) — other versions

Java原生API也支持

- [JavaScript API \[7.x\]](#) — other versions
- [Ruby API \[7.x\]](#) — other versions
- [Go API](#)
- [.NET API \[7.x\]](#) — [other versions](#)
- [PHP API \[7.x\]](#) — other versions
- [Perl API](#)
- [Python API](#)
- [eland Client](#)
- [Rust API](#)
- [Community Contributed Clients](#)



# Java REST Client

+ Java REST Client: 7.7 (current) ▼

Overview

+ Java Low Level REST Client

+ Java High Level REST Client

一般使用高版本客户端

Over

## 1. 找到依赖

```
1 <dependency>
2   <groupId>org.elasticsearch.client</groupId>
3   <artifactId>elasticsearch-rest-high-level-client</artifactId>
4   <version>7.6.2</version>
5 </dependency>
```

## 2. 找对象

```
RestHighLevelClient client = new RestHighLevelClient(
    RestClient.builder(
        new HttpHost("localhost", 9200, "http"),
        new HttpHost("localhost", 9201, "http"))));
```

```
client.close();
```

用完别忘了关闭

## 3. 保证依赖一致

```

1>https://
loper</ro
timezone>
o.com/spr
t:git://gi
on>scm:git
ion>
stem>
.spring.id
am>

```

- kuangshen-es-api
  - Lifecycle
  - Plugins
  - Dependencies
    - org.springframework.boot:spring-boot-starter-data-elasticsearch:2.2.6.RELEASE
      - org.springframework.boot:spring-boot-starter:2.2.6.RELEASE
      - org.springframework.boot:spring-boot:2.2.6.RELEASE (omitted for duplicate)
      - org.springframework.boot:spring-boot-autoconfigure:2.2.6.RELEASE (omitted for duplicate)
      - org.springframework.boot:spring-boot-starter-logging:2.2.6.RELEASE
      - jakarta.annotation:jakarta.annotation-api:1.3.5
      - org.springframework:spring-core:5.2.5.RELEASE (omitted for duplicate)
      - org.yaml:snakeyaml:1.25
    - org.springframework.data:spring-data-elasticsearch:3.2.6.RELEASE
      - org.springframework:spring-context:5.2.5.RELEASE
      - org.springframework:spring-tx:5.2.5.RELEASE
      - org.springframework.data:spring-data-commons:2.2.6.RELEASE
      - joda-time:joda-time:2.10.5
      - org.elasticsearch.client:transport:6.8.7
      - org.elasticsearch.plugin:transport-netty4-client:6.8.7
      - org.elasticsearch.client:elasticsearch-rest-high-level-client:6.8.7
      - com.fasterxml.jackson.core:jackson-core:2.10.3
      - com.fasterxml.jackson.core:jackson-databind:2.10.3
      - org.slf4j:slf4j-api:1.7.30
    - org.springframework.boot:spring-boot-starter-web:2.2.6.RELEASE

默认的版本和我们使用的版本不一致

源码中提供对象！

The screenshot shows an IDE with two panes. The left pane displays the project structure under 'Project'.

- Project Structure:**
  - src/main/resources
    - data
    - cassandra
    - couchbase
    - elasticsearch
      - ElasticsearchAutoConfiguration
      - ElasticsearchDataAutoConfiguration
      - ElasticsearchProperties
      - ElasticsearchRepositoriesAutoConfiguration
      - ElasticsearchRepositoriesRegistrar
      - ReactiveElasticsearchRepositoriesAutoConfiguration
      - ReactiveElasticsearchRepositoriesRegistrar
      - RestClientAutoConfiguration
      - RestClientProperties
  - src/main/java
    - org.springframework.boot.autoconfigure.elasticsearch
      - RestClientAutoConfiguration
      - RestClientBuilderCustomizer
      - RestClientConfigurations
      - RestClientProperties

The right pane shows the source code for `RestClientAutoConfiguration.java`.

```
1 //...
16 package org.springframework.boot.autoconfigure.elasticsearch.rest;
17
18 import ...
19
20 /**
21  * {@link EnableAutoConfiguration Auto-configuration} for Elasticsearch REST clients.
22  *
23  * @author Brian Clozel
24  * @author Stephane Nicoll
25  * @since 2.1.0
26  */
27 @Configuration(proxyBeanMethods = false)
28 @ConditionalOnClass(RestClient.class)
29 @EnableConfigurationProperties(RestClientProperties.class)
30 @Import({ RestClientConfigurations.RestClientBuilderConfiguration.class,
31         RestClientConfigurations.RestHighLevelClientConfiguration.class,
32         RestClientConfigurations.RestClientFallbackConfiguration.class })
33 public class RestClientAutoConfiguration {
34
35     }
36 }
```

A red arrow points from the `RestClientAutoConfiguration` class in the project structure to its definition in the source code.

## 1. 依赖

```
1      <properties>
2          <java.version>1.8</java.version>
3
4          <!--手动更改es版本-->
5          <elasticsearch.version>7.6.1</elasticsearch.version>
6      </properties>
```

```
1      <!-- https://mvnrepository.com/artifact/com.alibaba/fastjson -->
2      <dependency>
3          <groupId>com.alibaba</groupId>
4          <artifactId>fastjson</artifactId>
5          <version>1.2.68</version>
6      </dependency>
7      <dependency>
8          <groupId>org.springframework.boot</groupId>
9          <artifactId>spring-boot-starter-data-elasticsearch</artifactId>
10     </dependency>
```

## 2. 配置

```
1  package com.bean.config;
2
3
4  import org.apache.http.HttpHost;
5  import org.elasticsearch.client.RestClient;
6  import org.elasticsearch.client.RestHighLevelClient;
7  import org.springframework.context.annotation.Bean;
8  import org.springframework.context.annotation.Configuration;
9
10 @Configuration
11 public class ElasticSearchClientConfig {
12
13     @Bean
14     public RestHighLevelClient restHighLevelClient() {
15         RestHighLevelClient client = new RestHighLevelClient(
16             RestClient.builder(
17                 //localhost, es的端口, 连接方式
18                 new HttpHost("localhost", 9200, "http")
19             )
20         );
21         return client;
22     }
23
24 }
```

## 3. 弄一个简单的对象

```
1  package com.bean.pojo;
2
3
4  import lombok.AllArgsConstructor;
5  import lombok.Data;
6  import lombok.NoArgsConstructor;
```

```

7     import lombok.experimental.Accessors;
8     import org.springframework.stereotype.Component;
9
10    @AllArgsConstructor
11    @NoArgsConstructor
12    @Data
13    @Accessors(chain = true)
14    @Component
15    public class User {
16
17        private String name;
18        private int age;
19    }

```

#### 4. 尝试API

```

1     package com.bean;
2
3
4     import com.alibaba.fastjson.JSON;
5     import com.bean.pojo.User;
6     import org.apache.lucene.util.QueryBuilder;
7     import org.elasticsearch.action.admin.indices.delete.DeleteIndexRequest;
8     import org.elasticsearch.action.bulk.BulkRequest;
9     import org.elasticsearch.action.bulk.BulkResponse;
10    import org.elasticsearch.action.delete.DeleteRequest;
11    import org.elasticsearch.action.delete.DeleteResponse;
12    import org.elasticsearch.action.get.GetRequest;
13    import org.elasticsearch.action.get.GetResponse;
14    import org.elasticsearch.action.index.IndexRequest;
15    import org.elasticsearch.action.index.IndexResponse;
16    import org.elasticsearch.action.search.SearchRequest;
17    import org.elasticsearch.action.search.SearchResponse;
18    import org.elasticsearch.action.support.master.AcknowledgedResponse;
19    import org.elasticsearch.action.update.UpdateRequest;
20    import org.elasticsearch.action.update.UpdateResponse;
21    import org.elasticsearch.client.IndicesClient;
22    import org.elasticsearch.client.RequestOptions;
23    import org.elasticsearch.client.RestHighLevelClient;
24    import org.elasticsearch.client.indices.CreateIndexRequest;
25    import org.elasticsearch.client.indices.CreateIndexResponse;
26    import org.elasticsearch.client.indices.GetIndexRequest;
27    import org.elasticsearch.common.unit.TimeValue;
28    import org.elasticsearch.common.xcontent.XContentType;
29    import org.elasticsearch.index.get.GetResult;
30    import org.elasticsearch.index.query.MatchAllQueryBuilder;
31    import org.elasticsearch.index.query.QueryBuilders;
32    import org.elasticsearch.index.query.TermQueryBuilder;
33    import org.elasticsearch.search.SearchHit;
34    import org.elasticsearch.search.builder.SearchSourceBuilder;
35    import org.junit.jupiter.api.Test;
36    import org.springframework.beans.factory.annotation.Autowired;
37    import org.springframework.beans.factory.annotation.Qualifier;
38    import org.springframework.boot.test.context.SpringBootTest;
39
40    import java.io.IOException;
41    import java.util.ArrayList;
42    import java.util.List;

```

```

43     import java.util.concurrent.TimeUnit;
44
45     @SpringBootTest
46     class EsApplicationTests {
47
48         @Autowired
49         @Qualifier("restHighLevelClient")
50         private RestHighLevelClient client;
51
52         //创建索引
53         @Test
54         void createIndex() throws IOException {
55             //创建一个请求，请求包含创建的索引
56             CreateIndexRequest request = new CreateIndexRequest("bean_test");
57
58             //这个indicesClient包含了所有东西
59             IndicesClient indices = client.indices();
60
61             //创建索引，把请求给他，请求设置就使用默认的就好
62             CreateIndexResponse createIndexResponse = indices.create(request,
RequestOptions.DEFAULT);
63
64             //打印一下看看什么东
西:org.elasticsearch.client.indices.CreateIndexResponse@8a7f4660
65             System.out.println(createIndexResponse);
66         }
67
68         //查看索引是否存在
69         @Test
70         void isExistIndex() throws IOException {
71             //创建一个请求，请求包含创建的索引
72
73             GetIndexRequest request = new GetIndexRequest("bean_test");
74
75             IndicesClient indices = client.indices();
76
77             boolean exists = indices.exists(request, RequestOptions.DEFAULT);
78
79             System.out.println(exists);
80         }
81
82         //删除索引
83         @Test
84         void deleteIndex() throws IOException {
85             DeleteIndexRequest request = new DeleteIndexRequest("bean_test");
86
87             AcknowledgedResponse delete = client.indices().delete(request,
RequestOptions.DEFAULT);
88
89             //查看是否删除成功
90             System.out.println(delete.isAcknowledged());
91         }
92
93         //插入文档
94         @Test
95         void indexDocument() throws IOException {
96             User user = new User("bean", 10);
97

```

```

98         //规则: put/bean_test/1
99         IndexRequest request = new IndexRequest("bean_test");
100        request.id("1");
101        //设置超时时间为1s
102        request.timeout(TimeValue.timeValueSeconds(1));
103
104        //将数据放入json
105        request.source(JSON.toJSONString(user), XContentType.JSON);
106
107        //发送数据,获得相应
108        IndexResponse response = client.index(request, RequestOptions.DEFAULT);
109
110        System.out.println(request.toString()); //返回索引信息: index {[bean_test]
[_doc][1], source[{age:10,"name":"bean"}]}
111
112        System.out.println(response.status()); //返回当前的状态: CREATED
113
114    }
115
116
117    //文档是否存在
118    @Test
119    void existDocument() throws IOException {
120        GetRequest request = new GetRequest("bean_test", "1");
121
122        boolean exists = client.exists(request, RequestOptions.DEFAULT);
123
124        System.out.println(exists);
125    }
126
127    //获得文档的具体信息
128    @Test
129    void getDocument() throws IOException {
130        GetRequest request = new GetRequest("bean_test", "1");
131
132        GetResponse getResponse = client.get(request, RequestOptions.DEFAULT);
133
134        //打印文档内容: {age:10,"name":"bean"}
135        System.out.println(getResponse.getSourceAsString());
136
137        //返回的全部内容:
{"_index":"bean_test","_type":"_doc","_id":"1","_version":2,"_seq_no":1,"_primary_term":1,"found":true,"_source":{"age":10,"name":"bean"}}
138        System.out.println(getResponse);
139    }
140
141    //更新文档
142    @Test
143    void updateDocument() throws IOException {
144        UpdateRequest request = new UpdateRequest("bean_test", "1");
145
146        User user = new User("豌豆", 22);
147
148        UpdateRequest doc = request.doc(JSON.toJSONString(user),
XContentType.JSON);
149
150        UpdateResponse updateResponse = client.update(request,
RequestOptions.DEFAULT);

```

```

151
152         //OK
153         System.out.println(updateResponse.status());
154
155     }
156
157     //删除文档
158     @Test
159     void deleteDocument() throws IOException {
160
161         DeleteRequest request = new DeleteRequest("bean_test", "1");
162
163         DeleteResponse deleteResponse = client.delete(request,
RequestOptions.DEFAULT);
164
165         //OK
166         System.out.println(deleteResponse.status());
167
168     }
169
170     //批量请求
171     @Test
172     void testBulkRequest() throws IOException {
173         BulkRequest bulkRequest = new BulkRequest();
174
175         List<User> list = new ArrayList<>();
176         list.add(new User("bean1", 3));
177         list.add(new User("bean2", 3));
178         list.add(new User("bean3", 3));
179         list.add(new User("bean4", 3));
180         list.add(new User("bean5", 3));
181         list.add(new User("bean6", 3));
182
183         for (int i = 0; i < list.size(); i++) {
184             bulkRequest.add(
185                 new IndexRequest("bean_test")
186                     .id("" + (i + 1))
187                     .source(JSON.toJSONString(list.get(i)),
XContentType.JSON)
188             );
189         }
190
191         BulkResponse bulk = client.bulk(bulkRequest, RequestOptions.DEFAULT);
192         System.out.println(bulk.hasFailures()); //false, 但是false代表的是成功
193     }
194
195
196     //查询
197     @Test
198     void search() throws IOException {
199         SearchRequest request = new SearchRequest("bean_test");
200
201         //利用这个搜索构建器可以构建出所有的查询条件，我们的原生所有的搜索条件都在这里
202         SearchSourceBuilder builder = new SearchSourceBuilder();
203
204
205         //比如我要使用高亮: builder.highlighter()
206         //比如说我要分页: builder.from(), builder.size()

```

```

207         //排序: builder.sort()
208
209         //虽然所有的条件都在这里, 但是有些需要一些构造器, 比如query就需要一个query的builder
210         //下面使用query作为一个例子: 可以使用queryBuilders这个工具类进行快速匹配
211
212
213         //      MatchAllQueryBuilder queryBuilder = QueryBuilders.matchAllQuery();
214         TermQueryBuilder queryBuilder = QueryBuilders.termQuery("name", "bean1");
215         //查询
216         builder.query(queryBuilder);
217
218         //设置超时时间
219         builder.timeout(new TimeValue(60, TimeUnit.SECONDS));
220
221         //将搜索构建起放到请求里面
222         request.source(builder);
223         //执行
224         SearchResponse search = client.search(request, RequestOptions.DEFAULT);
225
226         //在之前我们就说过, hits里面都是具体的信息
227         /*
228             {
229                 "fragment":true,
230                 "hits":
231                     [
232                         {
233                             "fields":{},
234                             "fragment":false,
235                             "highlightFields":{},
236                             "id":"1",
237                             "matchedQueries":[],
238                             "primaryTerm":0,
239                             "rawSortValues":[],
240                             "score":1.540445,
241                             "seqNo":-2,
242                             "sortValues":[],
243                             "sourceAsMap":{"name":"bean1","age":3},
244                             "sourceAsString":{"age":3,"name":"bean1"},
245                             "sourceRef":{"fragment":true},
246                             "type":"_doc",
247                             "version":-1
248                         }
249                     ],
250                 "maxScore":1.540445,
251                 "totalHits":{"relation":"EQUAL_TO","value":1}
252             }
253         */
254         System.out.println(JSON.toJSONString(search.getHits()));
255
256         System.out.println("=====");
257
258         ////{name=bean1, age=3}
259         for (SearchHit hit : search.getHits().getHits()) {
260             System.out.println(hit.getSourceAsMap());
261         }
262     }
263 }

```



# 爬虫练习

找京东的页面

## 1. 依赖

```
1      <dependency>
2          <groupId>org.springframework.boot</groupId>
3          <artifactId>spring-boot-starter-data-elasticsearch</artifactId>
4      </dependency>
5
6
7      <!-- https://mvnrepository.com/artifact/org.jsoup/jsoup -->
8      <dependency>
9          <groupId>org.jsoup</groupId>
10         <artifactId>jsoup</artifactId>
11         <version>1.11.3</version>
12     </dependency>
```

## 2. pojo, 对应名字, 价格, 图片

```
1     package com.bean.pojo;
2
3
4     import lombok.AllArgsConstructor;
5     import lombok.Data;
6     import lombok.NoArgsConstructor;
7     import lombok.experimental.Accessors;
8
9     @Data
10    @AllArgsConstructor
11    @NoArgsConstructor
12    @Accessors(chain = true)
13    public class Content {
14
15        private String name;
16        private String price;
17        private String img;
18    }
```

## 3. Util

```
1     package com.bean.utils;
2
3
4     import com.bean.pojo.Content;
5     import org.jsoup.Jsoup;
6     import org.jsoup.nodes.Document;
7     import org.jsoup.nodes.Element;
8     import org.jsoup.select.Elements;
9
10    import java.io.IOException;
11    import java.net.URL;
12    import java.util.ArrayList;
13    import java.util.List;
```

```

13
14     public class JsoupUtil {
15
16         public static List<Content> getMessage(String keyword) throws IOException {
17
18             String url = "https://search.jd.com/Search?keyword="+keyword+"&enc=utf-8";
19
20             List<Content> contents = new ArrayList<>();
21
22
23             //返回的json
24             Document document = Jsoup.parse(new URL(url), 3000);
25
26             //获取所有的产品列表
27             Element elements = document.getElementById("J_goodsList");
28             //获取所有的li
29             Elements li = elements.getElementsByTag("li");
30
31             for (Element element : li) {
32                 //获取对应的图片
33                 String img = element.getElementsByTag("img").get(0).attr("src");
34                 //获取对应的价格
35                 String price = element.getElementsByClass("p-price").get(0).text();
36                 //获取对应的名称
37                 String name = element.getElementsByClass("p-
name").get(0).getElementsByTag("em").text();
38
39                 contents.add(new Content(name,price,img));
40             }
41             return contents;
42         }
43     }

```

#### 4. 测试

```

1     package com.bean;
2
3     import com.bean.pojo.Content;
4     import com.bean.utils.JsoupUtil;
5     import org.jsoup.Jsoup;
6     import org.jsoup.nodes.Document;
7     import org.jsoup.nodes.Element;
8     import org.jsoup.select.Elements;
9     import org.junit.jupiter.api.Test;
10    import org.springframework.boot.test.context.SpringBootTest;
11
12    import java.io.IOException;
13    import java.net.MalformedURLException;
14    import java.net.URL;
15    import java.util.List;
16
17    @SpringBootTest
18    class EsjdApplicationTests {
19
20        @Test
21        void contextLoads() throws IOException {
22
23            List<Content> message = JsoupUtil.getMessage("你好");

```

```
24
25     message.forEach(System.out::println);
26     }
27
28 }
```