ElasticSearch整理

# 一、ElasticSearch介绍

1、什么是elasticsearch?

elasticsearch是基于lucene的全文检索服务器，对外提供restful接口

2、elasticsearch原理

正排索引：查字典时从第一页开始找，直到找到关键字为止（CTRL+F）

倒排索引：查字典时通过目录查找

逻辑结构：一个倒排索引表，由三部分组成

document

term

term----关联----document

# 二、ES安装

a、安装

1、设置虚拟机内存>1.5G

2、创建用户

创建elk 用户组groupadd elk

创建用户admin

useradd admin

passwd admin

将admin用户添加到elk组 usermod -G elk admin

为用户分配权限

#chown将指定文件的拥有者改为指定的用户或组 -R处理指定目录以及其子目录下的所有文件

chown -R admin:elk /usr/upload

chown -R admin:elk /usr/java

切换用户：su admin

3、安装

解压即安装：tar -zxvf elasticsearch-6.2.3.tar.gz -C /usr/java

配置elasticsearch.yml

|  |
| --- |
| cluster.name: usian  node.name: usian\_node\_1  network.host: 0.0.0.0  http.port: 9200  transport.tcp.port: 9300  node.master: true  node.data: true  discovery.zen.ping.unicast.hosts: ["0.0.0.0:9300", "0.0.0.0:9301"]  bootstrap.memory\_lock: false  path.data: /usr/java/elasticsearch-6.2.3/data  path.logs: /usr/java/elasticsearch-6.2.3/logs  http.cors.enabled: true  http.cors.allow-origin: /.\*/ |

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。

默认内存占用太多了，我们调小一些：

-Xms512m

-Xmx512m

2.3.3.log4j2.properties

日志文件设置，ES使用log4j，注意日志级别的配置。

4、解决内核问题

修改elasticsearch.yml文件，在最下面添加如下配置：

bootstrap.system\_call\_filter: false

5、解决文件创建权限问题

使用root用户修改配置文件:

vim /etc/security/limits.conf

添加下面的内容：

\* soft nofile 65536

\* hard nofile 65536

6、决绝线程开启限制问题

使用root用户修改配置：

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

修改下面的内容：

\* soft nproc 1024

改为：

\* soft nproc 4096

7、解决虚拟机内存问题

vim /etc/sysctl.conf

追加下面内容：

vm.max\_map\_count=655360 #限制一个进程可以拥有的VMA(虚拟内存区域)的数量

然后执行命令，让sysctl.conf配置生效：

sysctl -p

2.5.测试

b、启动和关闭

启动：

./bin/elasticsearch

./elasticsearch -d

关闭：

kill -9 pid

# 三.安装Kibana

## 3.1.什么是Kibana

Kibana是ES提供的一个基于Node.js的基于Node.js的管理控制台, 可以很容易实现高级的数据分析和可视化，以图标的形式展现出来。

kibana可以用来编辑请求语句的，方便学习操作es的语法。有时在进行编写程序，写到查询语句时，往往我会使用kibana进行书写，然后再粘贴到程序中。（不容易出错）

## 3.2.下载

ElasticSearch官网：<https://www.elastic.co/cn/>

## 3.3.安装

在window中安装Kibana很方便，解压即安装

## 3.3.修改配置

修改config/kibana.yml配置：

server.port:5601  
server.host:"0.0.0.0" #允许来自远程用户的连接  
elasticsearch.url:http://192.168.233.134:9200 #Elasticsearch实例的URL

## 3.4.启动

./bin/kibana

## 3.5.测试

访问：<http://127.0.0.1:5601>

# 四.安装head

## 4.1.什么是head

head插件是ES的一个可视化管理插件，用来监视ES的状态，并通过head客户端和ES服务进行交互，比如创建映射、创建索引等。从ES6.0开始，head插件支持使得node.js运行。

## 4.2.安装

1、下载head

下载地址：<https://github.com/mobz/elasticsearch-head>

2、安装依赖

npm install

3、运行

npm run start

## 4.3.测试

浏览器访问：<http://127.0.0.1:9100/>

# 五.ES快速入门

ES作为一个索引及搜索服务，对外提供丰富的REST接口，快速入门部分的实例使用kibana来测试，目的是对ES的使用方法及流程有个初步的认识。

## 5.1.index管理

### 5.1.1.创建index

索引。包含若干相似结构的 Document 数据，相当于数据库的database。

语法：

PUT /java1906  
{  
 "settings": {  
   "number\_of\_shards": 2,  
   "number\_of\_replicas": 0  
}  
}

**number\_of\_shards** - 是表示一个索引库将拆分成多片分别存储不同的结点，提高了ES的处理能力和高可用性

**number\_of\_replicas**- 是为每个 primary shard分配的replica shard数，**如果只有一台机器，设置为0**

效果：

### 5.1.2.修改index

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

PUT /java1906/\_settings  
{  
 "number\_of\_replicas" : 1  
}

ES 中对 shard 的分布是有要求的，有其内置的特殊算法：

Replica shard 会保证不和他的那个 primary shard 分配在同一个节点上；如过只有一个节点，则此案例执行后索引的状态一定是yellow。

### 5.1.3.删除index

DELETE /java1906 [, other\_index]

## 5.2.mapping管理

映射，创建映射就是向索引库中创建field（类型、是否索引、是否存储等特性）的过程，下边是document和field与关系数据库的概念的类比：

索引库（indices）-----------------------Databases数据库

类型（type）-----------------------------Table数据表

文档（Document）---------------------Row 行

字段（Field）----------------------------Columns 列

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，ES6.x 版本之后，type概念被弱化ES官方将在ES7.0版本中彻底删除type。

### 5.2.1 创建mapping

语法：POST index\_name/type\_name/\_mapping

如：

POST /java1906/course/\_mapping  
{  
 "properties": {  
    "name": {  
       "type": "text"  
    },  
    "description": {  
       "type": "text"  
    },  
    "studymodel": {  
       "type": "keyword"  
    }  
}  
}

效果：

### 5.2.2.查询mapping

查询所有索引的映射：

GET /java1906/course/\_mapping

### 5.2.3.更新mapping

映射创建成功可以添加新字段，已有字段不允许更新。

### 5.2.4.删除mapping

通过删除索引来删除映射。

## 5.3.document管理

### 5.3.1.创建document

ES中的文档相当于MySQL数据库表中的记录。

#### 5.3.1.1.POST语法

此操作为 ES 自动生成 id 的新增 Document 方式。

语法：POST/index\_name/type\_name{fieldname:fieldvalue}

如：

POST /java1906/course/1  
{  
 "name":"python从入门到放弃",  
 "description":"人生苦短，我用Python",  
 "studymodel":"201002"  
}  
​  
POST /java1906/course  
{  
 "name":".net从入门到放弃",  
 "description":".net程序员谁都不服",  
 "studymodel":"201003"  
}

#### 5.3.1.2.PUT语法

此操作为手工指定 id 的 Document 新增方式。

语法：PUT/index\_name/type\_name/id{field\_name:field\_value}

如：

PUT /java1906/course/2  
{  
 "name":"php从入门到放弃",  
 "description":"php是世界上最好的语言",  
 "studymodel":"201001"  
}

结果：

{  
 "\_index": "test\_index", 新增的 document 在什么 index 中，  
 "\_type": "my\_type", 新增的 document 在 index 中的哪一个 type 中。  
 "\_id": "1", 指定的 id 是多少  
 "\_version": 1, document 的版本是多少，版本从 1 开始递增，每次写操作都会+1  
 "result": "created", 本次操作的结果，created 创建，updated 修改，deleted 删除  
 "\_shards": { 分片信息  
     "total": 2, 分片数量只提示 primary shard  
     "successful": 1, 数据 document 一定只存放在 index 中的某一个 primary shard 中  
     "failed": 0  
},  
 "\_seq\_no": 0,   
 "\_primary\_term": 1  
}

通过head查询数据：

通过luke查看分词列表：

### 5.3.2.查询document

语法：

GET/index\_name/type\_name/id

或

GET/index\_name/type\_name/\_search?q=field\_name:field\_value

如：根据课程id查询文档

GET /java1906/course/1

如：查询所有记录

GET /java1906/course/\_search

如：查询名称中包括php 关键字的的记录

GET /java1906/course/\_search?q=name:门

结果：

{  
 "took": 1, # 执行的时长。单位毫秒  
 "timed\_out": false, # 是否超时  
 "\_shards": { # shard 相关数据  
   "total": 1, # 总计多少个 shard  
   "successful": 1, # 成功返回结果的 shard 数量  
   "skipped": 0,  
   "failed": 0  
},  
 "hits": { # 搜索结果相关数据  
   "total": 3, # 总计多少数据，符合搜索条件的数据数量  
   "max\_score": 1, # 最大相关度分数，和搜索条件的匹配度  
   "hits": [# 具体的搜索结果  
    {  
       "\_index": "java1906",# 索引名称  
       "\_type": "course", # 类型名称  
       "\_id": "1",# id 值  
       "\_score": 1, # 匹配度分数，本条数据匹配度分数  
       "\_source": { # 具体的数据内容  
         "name": "php从入门到放弃",  
         "description": "php是世界上最好的语言",  
         "studymodel": "201001"  
      }, {  
"\_index": "java1906",  
"\_type": "course",  
"\_id": "2",  
"\_score": 0.13353139,  
"\_source": {  
"name": "php从入门到放弃",  
"description": "php是世界上最好的语言",  
"studymodel": "201001"  
}  
}, {  
"\_index": "java1906",  
"\_type": "course",  
"\_id": "6ljFCnIBp91f7uS8FkjS",  
"\_score": 0.13353139,  
"\_source": {  
"name": ".net从入门到放弃",  
"description": ".net程序员谁都不服",  
"studymodel": "201003"  
}  
}  
]  
}  
}

### 5.3.3.删除Document

ES 中执行删除操作时，ES先标记Document为deleted状态，而不是直接物理删除。当ES 存储空间不足或工作空闲时，才会执行物理删除操作，标记为deleted状态的数据不会被查询搜索到（ES 中删除 index ，也是标记。后续才会执行物理删除。所有的标记动作都是为了NRT（近实时）实现）

语法：DELETE/index\_name/type\_name/id

如：

DELETE /java1906/course/3

结果：

{  
 "\_index": "java1906",  
 "\_type": "course",  
 "\_id": "2",  
 "\_version": 2,  
 "result": "deleted",  
 "\_shards": {  
   "total": 1,  
   "successful": 1,  
   "failed": 0  
},  
 "\_seq\_no": 3,  
 "\_primary\_term": 1  
}

## 5.4.ES读写原理

### 5.4.1.documnet routing（数据路由）

当客户端创建document的时候，es需要确定这个document放在该index哪个shard上，这个过程就是document routing。

路由过程：

　　　　路由算法：shard = hash(routing) %number\_of\_primary\_shards

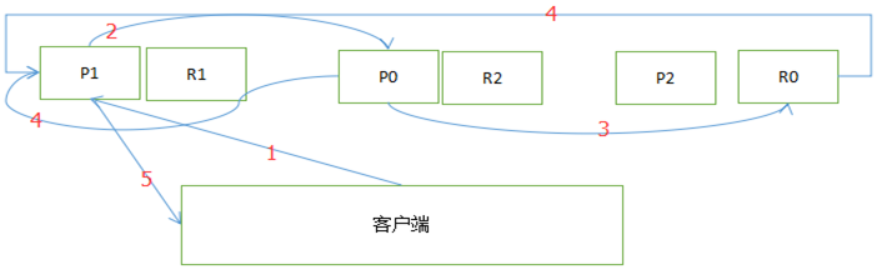
　　　　routing：document的\_id，可能是手动指定，也可能是自动生成，决定一个document在哪个shard上

　　　　number\_of\_primary\_shards*：*主分片。

### 5.4.2.ES document写操作原理

ES增删改的处理流程：增删改的请求一定作用在主分片上。

假如我们es集群有3个node，每个node上一个主分片一个复制分片,

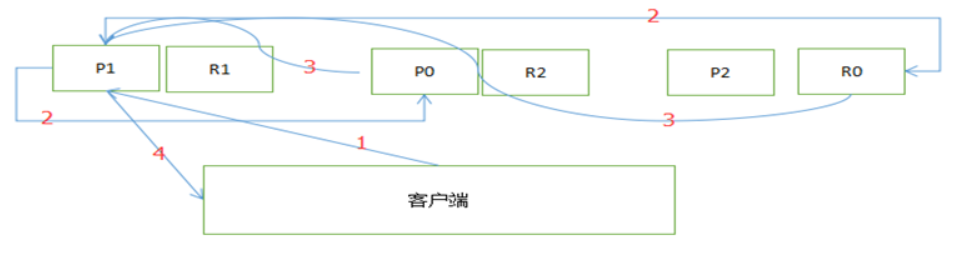


1、第一步 客户端发起一个PUT请求，假如该请求被发送到第一个node节点，那么该节点将成为协调节点  
  (coordinatingnode)，如图P1所在的节点就是协调节点。他将根据该请求的路由信息计算，该document将被存储   到哪个分片。  
2、第二步 通过计算发现该document被存储到p0分片，那么就将请求转发到node2节点。  
3、第三步P0根据请求信息创建document，和相应的索引信息，创建完毕后将信息同步到自己的副本节点R0上。  
4、第四步P0和R0将通知我们的协调节点，任务完成情况。  
5、第五部 协调节点响应客户端最终的处理结果。

### 5.4.3.ES document读操作原理

假如我们es集群有3个node，每个node上一个主分片一个复制分片,

如下图：



1、第一步 客户端发送读器请求到协调节点(coordinate node)。  
2、第二步 协调节点(coordinate node)根据请求信息对document进行路由计算，将请求转发到对应的node，node2  
  或者node3，此时会使用round-robin随机轮询算法，在primary shard以及其所有replica(副本)中随机选择一  
  个让读请求负载均衡。  
3、第三步 相应接收到请求的节点(node2或者node3)将处理结果返回给协调节点(coordinate node)。  
4、第四步 协调节点将最终的结果反馈给客户端。

### 5.4.4.为什么primary shard数量不可变？

原因：假如我们的集群在初始化的时候有5个primary shard，我们往里边加入一个document id=5，假如hash(5)=23,这时该document 将被加入 (shard=23%5=3)P3这个分片上。如果随后我们给es集群添加一个primary shard ，此时就有6个primary shard，当我们GET id=5 ，这条数据的时候，es会计算该请求的路由信息找到存储他的 primary shard（shard=23%6=5） ，根据计算结果定位到P5分片上。而我们的数据在P3上。所以es集群无法添加primary shard，但是可以扩展replicas shard。

# 6、IK分词器

## 6.1、安装

解压到/usr/java/elasticsearch-6.2.3/plugs目录下，并重命名为ik

## 6.2、自定义词库

IkAnalyzer.cfg.xml：配置扩展词典和停用词典

main.dic：扩展词典

stopwords.dic：停用词典

## 6.3、两种分词模式

ik\_smart：粗粒度拆分

ik\_max\_word：细粒度拆分

# 7、field详细介绍

## 7.1 field的属性介绍

### 7.1.1.type

通过type属性指定field的类型。

|  |
| --- |
| "name":{  "type":"text"  } |

### 7.1.2.analyzer：

通过analyzer属性指定分词模式。

|  |
| --- |
| "name": {  "type": "text",  analyzer":"ik\_max\_word"  } |

上边指定了analyzer是指在索引和搜索都使用ik\_max\_word，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。 对于ik分词器建议是索引时使用ik\_max\_word将搜索内容进行细粒度分词，搜索时使用ik\_smart提高搜索精确性。

|  |
| --- |
| "name": {  "type": "text",  "analyzer":"ik\_max\_word",  "search\_analyzer":"ik\_smart"  } |

### 7.1.3.index

通过index属性指定是否索引。 默认为index=true，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

|  |
| --- |
| "pic": {  "type":"text",  "index":false  } |

### 7.1.4.field索引不存储

如果某个字段内容非常多，业务里面只需要能对该字段进行搜索，比如：商品描述。查看文档内容会再次到mysql或者hbase中取数据，把大字段的内容存在Elasticsearch中只会增大索引，这一点文档数量越大结果越明显，如果一条文档节省几KB，放大到亿万级的量结果也是非常可观的。

如果只想存储某几个字段的原始值到Elasticsearch，可以通过incudes参数来设置，在mapping中的设置如下:

|  |
| --- |
| POST /java1906/course/\_mapping  {  "\_source": {  "includes":["description"]  }  } |

同样，可以通过excludes参数排除某些字段：

|  |
| --- |
| POST /java1906/course/\_mapping  {  "\_source": {  "excludes":["description"]  }  } |

## 7.2. 常用field类型

### 7.2.1.text文本字段

#### 7.2.1.1、创建新映射：

|  |
| --- |
| POST /java1906/course/\_mapping  {  "properties": {  "name": {  "type": "text",  "analyzer":"ik\_max\_word",  "search\_analyzer":"ik\_smart"  },  "description": {  "type": "text",  "analyzer":"ik\_max\_word",  "search\_analyzer":"ik\_smart"  },  "pic":{  "type":"text",  "index":false  },  "studymodel":{  "type":"text"  }  }  } |

#### 7.2.1.2、插入文档：

|  |
| --- |
| POST /java1906/course/1  {  "name":"python从入门到放弃",  "description":"人生苦短，我用Python",  "pic":"250.jpg",  "studymodel":"201002"  } |

#### 7.2.1.3、查询测试：

|  |
| --- |
| GET /java1906/course/\_search?q=name:放弃  GET /java1906/course/\_search?q= description:人生  GET /java1906/course /\_search?q=pic:250.jpg |

结果：name和description支持全文检索，pic不可作为查询条件，不支持全文检索。

### 7.3.1.keyword关键字字段

上边介绍的text文本字段在映射时要设置分词器，keyword字段为关键字字段，通常搜索keyword是按照整体搜索，所以创建keyword字段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。

例如：

#### 7.3.1.1更改映射：

|  |
| --- |
| POST /java1906/course/\_mapping  {  "properties": {  "studymodel":{  "type":"keyword"  },  "description": {  "type": "text",  "analyzer":"ik\_max\_word",  "search\_analyzer":"ik\_smart"  },  "pic":{  "type":"text",  "index":false  },  "name":{  "type":"keyword"  }  }  } |

#### 7.3.1.2插入文档：

|  |
| --- |
| PUT /java1906/course/2  {  "name": "java编程基础",  "description": "java语言是世界第一编程语言",  "pic":"250.jpg",  "studymodel": "201001"  } |

#### 7.3.1.3根据name查询文档：

|  |
| --- |
| GET /java1906/course/\_search?q=name:java编程基础 |

name是keyword类型，所以查询方式是精确查询。

### 7.3.2.date日期类型

日期类型不用设置分词器，通常日期类型的字段用于排序。 1)format 通过format设置日期格式，多个格式使用双竖线||分隔, 每个格式都会被依次尝试, 直到找到匹配的

例如：

#### 7.3.2.1设置允许date字段存储年月日时分秒、年月日及毫秒三种格式。

|  |
| --- |
| POST /java1906/course/\_mapping  {  "properties": {  "timestamp": {  "type": "date",  "format": "yyyy-MM-dd HH:mm:ss||yyyy-MM-dd"  }  }  } |

#### 7.3.2.2插入文档：

|  |
| --- |
| PUT /java1906/course/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",  "pic":"250.jpg",  "timestamp":"2018-07-04 18:28:58"  } |

### 7.3.3.Numeric类型

下图是ES支持的数值类型：尽可能选择范围小的数据类型,字段的长度越短, 索引和搜索的效率越高;



例如：

#### 7.3.3.1更新已有映射：

|  |
| --- |
| POST /java1906/course/\_mapping  {  "properties": {  "price": {  "type": "double"  }  }  } |

#### 7.3.3.2插入文档

|  |
| --- |
| PUT /java1906/course/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",  "pic":"250.jpg",  "timestamp":"2018-07-04 18:28:58",  "price":38.6  } |

# 8、总结

field属性的设置标准：

|  |  |
| --- | --- |
|  | **标准** |
| 分词 | 是否有意义 |
| 索引 | 是否搜索 |
| 存储 | 是否展示 |

# 9、SpringBoot整合ElasicSearch

## 9.1.ES客户端

ES提供多种不同的客户端：

1、TransportClient

ES提供的传统客户端，官方计划8.0版本删除此客户端。

2、RestClient

RestClient是官方推荐使用的，它包括两种：Java Low Level RESTClient和 Java HighLevel REST Client。ES在6.0之后提供 JavaHigh Level REST Client， 两种客户端官方更推荐使用 Java High Level RESTClient，不过当前它还处于完善中，有些功能还没有。

## 9.2.搭建工程

1. **pom.xml所需依赖**

|  |
| --- |
| <dependencies>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-web</artifactId>  </dependency>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-web</artifactId>  </dependency>  <dependency>  <groupId>org.elasticsearch.client</groupId>  <artifactId>elasticsearch-rest-high-level-client</artifactId>  </dependency>  <dependency>  <groupId>org.elasticsearch</groupId>  <artifactId>elasticsearch</artifactId>  </dependency>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-test</artifactId>  </dependency>  </dependencies> |

1. **application.yml 配置文件**

|  |
| --- |
| spring:  data:  elasticsearch:  cluster-nodes: 192.168.233.134:9200 |

1. **Config**

|  |
| --- |
| package com.usian.config;  import org.apache.http.HttpHost;  import org.elasticsearch.client.RestClient;  import org.elasticsearch.client.RestHighLevelClient;  import org.springframework.boot.autoconfigure.data.elasticsearch.ElasticsearchProperties;  import org.springframework.context.annotation.Bean;  import org.springframework.context.annotation.Configuration;  @Configuration  public class ElasticsearchConfig extends ElasticsearchProperties{  @Bean  public RestHighLevelClient getRestHighLevelClient() {  String[] hosts = getClusterNodes().split(",");  HttpHost[] httpHosts = new HttpHost[hosts.length];  for (int i = 0; i < httpHosts.length; i++) {  String h = hosts[i];  httpHosts[i] = new HttpHost(h.split(":")[0],Integer.parseInt(h.split(":")[1]));  }  return new RestHighLevelClient(RestClient.builder(httpHosts));  }  } |

1. **启动类**

|  |
| --- |
| package com.usian;  import org.springframework.boot.SpringApplication;  import org.springframework.boot.autoconfigure.SpringBootApplication;  @SpringBootApplication  public class ElasticsearchApp {  public static void main(String[] args) {  SpringApplication.run(ElasticsearchApp.class, args);  }  } |

## 9.3.索引管理

#### 创建索引库

|  |
| --- |
| CreateIndexRequest createIndexRequest = new CreateIndexRequest("java1906"); restHighLevelClient.indices().create(createIndexRequest,RequestOptions.DEFAULT) |

#### 删除索引库

|  |
| --- |
| DeleteIndexRequest deleteIndexRequest = new DeleteIndexRequest("java1906");  restHighLevelClient.indices().delete(deleteIndexRequest,RequestOptions.DEFAULT); |

#### 添加文档

|  |
| --- |
| IndexRequest indexRequest = new IndexRequest("java1906", "course", "1");  restHighLevelClient.index(indexRequest,RequestOptions.DEFAULT); |

#### 批量添加文档

|  |
| --- |
| bulkRequest.add(new IndexRequest("java1906", "course","2").source("{\"name\":\"php实战\",\"description\":\"php谁都不服\",\"studymodel\":\"201001\",\"price\":\"5.6\"}", XContentType.JSON));  bulkRequest.add(new IndexRequest("java1906", "course","3").source("{\"name\":\"net实战\",\"description\":\"net从入门到放弃\",\"studymodel\":\"201001\",\"price\":\"7.6\"}", XContentType.JSON));  restHighLevelClient.bulk(bulkRequest,RequestOptions.DEFAULT); |

#### 修改文档

|  |
| --- |
| UpdateRequest updateRequest = new UpdateRequest("java1906", "course", "1");  restHighLevelClient.update(updateRequest,RequestOptions.DEFAULT); |

#### 删除文档

|  |
| --- |
| DeleteRequest deleteRequest = new DeleteRequest("java1906","course","1");  restHighLevelClient.delete(deleteRequest,RequestOptions.DEFAULT); |

#### 查询文档

|  |
| --- |
| GetRequest getRequest = new GetRequest("java1906","course","1");  restHighLevelClient.get(getRequest,RequestOptions.DEFAULT); |

#### DSL文档

1. **match\_all查询**

|  |
| --- |
| SearchRequest searchRequest = new SearchRequest("java1906");  searchRequest.types("course");  SearchSourceBuilder searchSourceBuilder = new SearchSourceBuilder();  searchSourceBuilder.query(QueryBuilders.matchAllQuery());  searchRequest.search(searchSourceBuilder)  restHighLevelClient.search(searchRequest,RequestOptions.DEFAULT); |

1. **分页查询**

|  |
| --- |
| SearchSourceBuilder searchSourceBuilder = new SearchSourceBuilder();  searchSourceBuilder.query(QueryBuilders.matchAllQuery());  searchSourceBuilder.form(1);  searchSourceBuilder.size(2);  searchSourceBuilder.sort("price",SortOrder.DESC); |

1. **match查询**

|  |
| --- |
| SearchSourceBuilder searchSourceBuilder = new SearchSourceBuilder();  searchSourceBuilder.query(QueryBuilders.matchQuery("name", "spring开发").operator(Operator.AND));  searchRequest.source(searchSourceBuilder);  searchResponse = restHighLevelClient.search(searchRequest,RequestOptions.DEFAULT); |

1. **multi\_match查询**

|  |
| --- |
| SearchSourceBuilder searchSourceBuilder = new SearchSourceBuilder();  searchSourceBuilder.query(QueryBuilders.multiMatchQuery("开发","name","description"));  searchRequest.source(searchSourceBuilder);  searchResponse = restHighLevelClient.search(searchRequest,RequestOptions.DEFAULT); |

1. **bool查询**

|  |
| --- |
| SearchSourceBuilder searchSourceBuilder = new SearchSourceBuilder();  BoolQueryBuilder boolQueryBuilder = QueryBuilders.boolQuery();  boolQueryBuilder.must(QueryBuilders.matchQuery("name", "开发"));  boolQueryBuilder.must(QueryBuilders.matchQuery("description","开发"));  searchSourceBuilder.query(boolQueryBuilder);  searchRequest.source(searchSourceBuilder);  searchResponse = restHighLevelClient.search(searchRequest,RequestOptions.DEFAULT); |

1. **filter查询**

|  |
| --- |
| SearchSourceBuilder searchSourceBuilder = new SearchSourceBuilder();  BoolQueryBuilder boolQueryBuilder = QueryBuilders.boolQuery();  boolQueryBuilder.must(QueryBuilders.matchQuery("name","开发"));  boolQueryBuilder.filter(QueryBuilders.rangeQuery("price").gte(10).lte(100))  searchSourceBuilder.query(boolQueryBuilder);  searchRequest.source(searchSourceBuilder);  searchResponse = restHighLevelClient.search(searchRequest,RequestOptions.DEFAULT); |