ES学习笔记

# 什么是搜索

百度是对搜索的第一印象，比如搜索一部喜欢的电影，或者一本书，但搜索=百度，这是不对的。

垂直搜索：（站内搜索）

互联网搜索：电商网站，招聘网站，新闻网站。

IT系统的搜索：OA软件，办公室管理，人员管理，项目管理，会议管理。

BI系统（Business intelligence）

搜索：就是在任何场景下，找寻你想要的信息，这时会输入一段关键字，然后期望返回你想要的信息。

# 传统数据库搜索的弊端

2.1 传统数据实现搜索是一条一条去匹配的，如搜索的字段是一段很长的文本，每次都要对每条记录的所有进行扫描.

2.2 不能将搜索词分开来，尽可能去搜索更多符合你要求的记录，如你想搜索“生化危机” 而错输入“生化机”就会匹配不到你想要的记录。

所以数据库实现搜索性能是很低的。

# 什么是ElasticSearch（ES）

ElasticSearch是一个基于Lucene的分布式搜索服务器。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 java 接口，及其他语言的接口。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](https://baike.baidu.com/item/%E4%BA%91%E8%AE%A1%E7%AE%97/9969353)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（对海量数据）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。

分布式：ES可以将海量数据分散的多台服务器存储和检索

海量数据的存储：分布式以后，就可以采用大量的服务器存储数据。

近实时：在秒级别范围内分析和搜索。



# ES的应用场景

4.1. 维基百科 （类似于百度百科） 例如牙膏的百度百科

4.2. the Guardian（国外新闻网站） 类似于搜狐新闻 用户行为日志（点击，浏览，收藏，评论等）

4.3 Github （开源代码管理）搜索上亿行代码。

4.4 BI系统（business inteligence） 商业智能 例如大型商场用户消费的趋势。

4.5 电商，IT系统分析

# ES的特点

4.1.自动维护数据分布到多个节点的索引，还有搜索请求分布到多个节点的执行

4.2.自动维护数据的冗余副本，保证宕机后数据不丢失。

4.3. 全文检索引擎，ES是建立在Lucene上的开源搜索引擎，可以用来全文检索，地理信息搜索（例如搜索当前距离我1公里内的烤肉店）

4.4.文档存储和查询，可以向Nosql那样存储任意格式的文档，并能根据条件搜索文档

4.5.大数据分析，ES号称能准确实时的进行大数据分析，数据量从TB到PB,国内外很多大公司都用ES做大数据分析

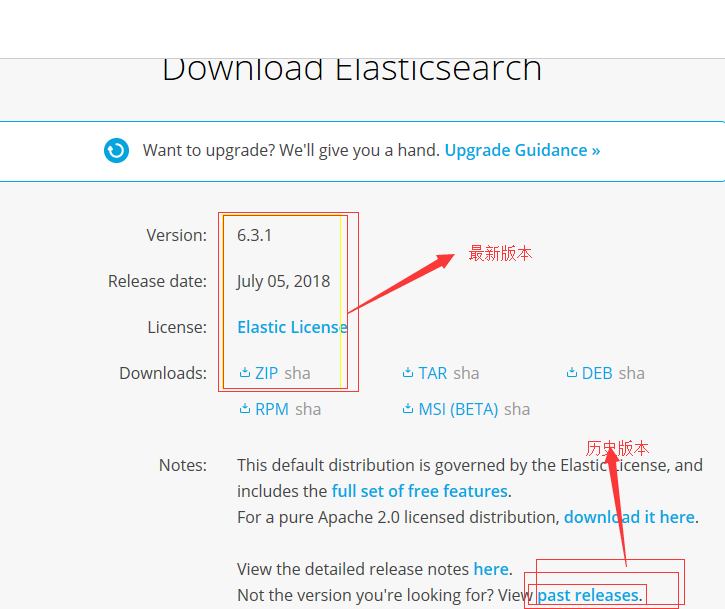
4.6.ES提供了REST API,用来简化ES的操作，因此可以使用任何语言的客户端，同时也提供了java API，springBoot也对REST API进行了封装，简化了开发。

4.7.ES常常配合传统数据库一起使用，ES用来负责大数据的查询，搜索和统计分析。

4.8对用户而言 开箱即用（操作简单，3分钟部署ES 拿来就能立即前代码）

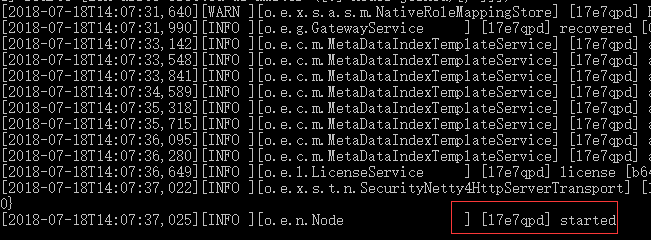
# ES安装

第一步：下载ES 地址：<https://www.elastic.co/downloads/elasticsearch>



第二步：解压到D://

第三步：进入D:\ElasticSearch\elasticsearch-6.3.1\bin 双击elasticsearch.bat进行安装

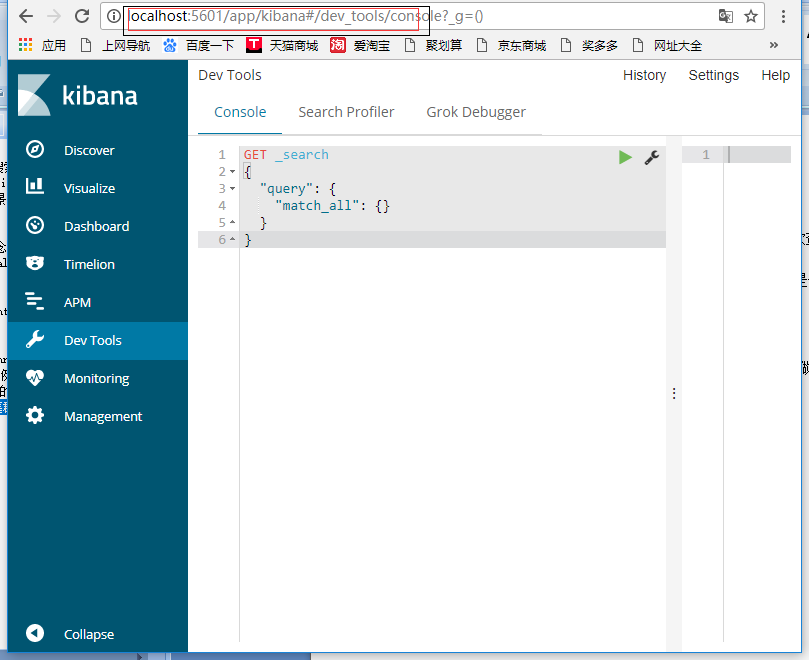


第四步：在浏览器输入<http://9200> 若出现下面页面则表示安装成功



第五步：下载kibana 官方：<https://www.elastic.co/downloads/kibana>

第六步：解压



# ES的基本概念

7.1 Near RealTime （近实时）

4.1．Index

Index是文档（document）的集合，在index下包含了Type，用于对文档进一步分类，可以理解为index为数据库，而type相当于数据库的表，ES可以轻易的联合Index和type来搜索数据。

4.2．Type

进一步组织Document，一个index可以有多个Type

4.3. Document

文档是ES能够存储和搜索的基本信息，类似于数据库的表行数据，Document为JSON格式，文档属于Type

4.4.Node

节点是集群里的一台ES Server，用于文档的存储和查询，应用可以只有一个节点，也可以由上百个节点组成集群来存储和搜索数据，每个节点都有一个节点名字，以及所属集群的名字。

4.5．集群

同样集群名的节点将组合成ES集群，用来联合完成数据的存储和搜索，默认的集群名是elasticsearch.

4.6 分区（shrads）和复制（Replicas）

每个index理论上都可以包含大量的数据，超过了单个节点的存储限制，而且，单个节点的处理庞大的数据，将明显的限制存储和搜索性能，为了解决这个问题，ES会进一步将index在物理上细分为多个区，而这些区会按照配置复制到多个节点，Index的分区称为主分区，复制的分区称为复制分区，这样的好处是既保证数据不会丢失 ，又能提高查询性能。每个分区是一个单独的工作单元，可以完成存储和搜索功能，每个分区能存储2147483519个文档。

ES集群

INDEX 3T

Shard 1T

Shard 1T

Shard 1T

Index会被分为多个每个shard都会存放index的一部分内容。这些shard都会分散在多个服务器上

好处：横向扩展，当数据量增大时就会加shard

减少单机服务器的压力

Replicas：放了防止宕机丢失数据，每个shard都会建立一个或多个副本。数据量和对应的shard数据量一样多。（即高可用）

4.7 ES和数据库的对比

|  |  |
| --- | --- |
| ES | 数据库 |
| Index | 库 |
| Type | 表 |
| Document | 行数据 |

# 8.电商网站的案例

8.1Document的数据格式（json）

面向文档的搜索分析引擎

8.1.1 应用系统的数据结构是面向对象的，复杂的。

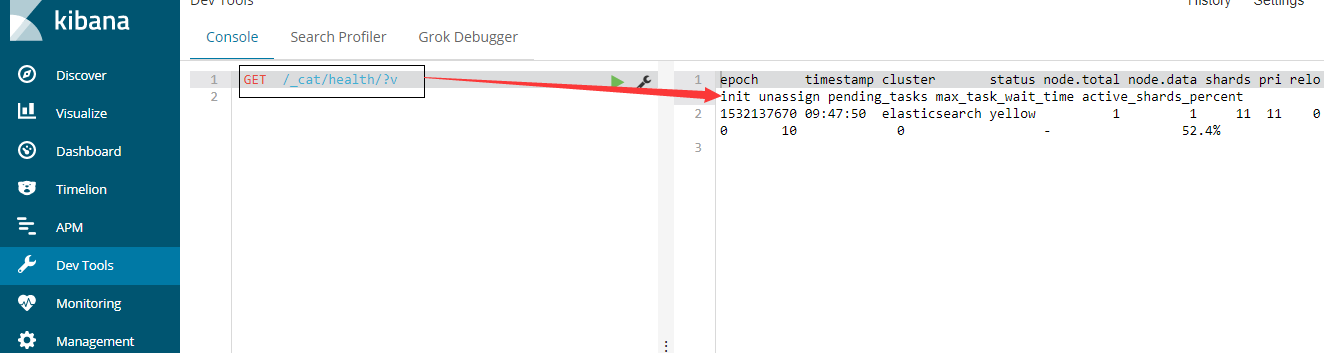
8.1.2 对象存储在数据库只能拆开来，变为多张表，每次查询都会还原对象格式，相当麻烦

8.1.3 ES是面向文档的，文档存储的结构和对象结构是一样的。

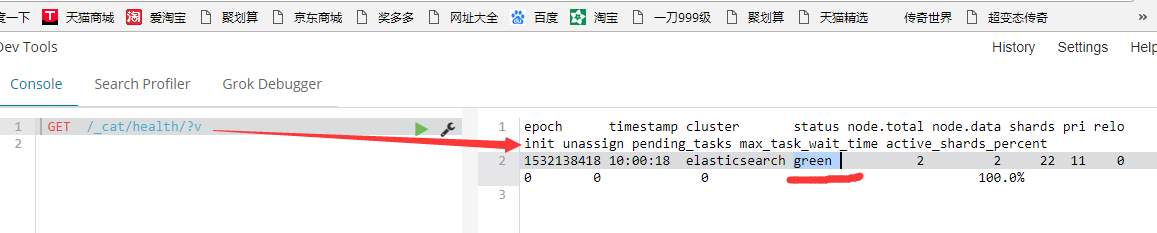
8.1.4 ES的document是用json数据格式来表达。

8.2 简单的集群管理

8.2.1 快速检查集群的健康状况 ES提供一套api 叫做cat api 可以查看ES中各种各样的数据 GET /\_cat/healthy/?v



当重新再打开一个ES服务器时候状态就会发生变化，会自动进行集群



Status:代表健康状况

Green 每个索引的primary shard 和 replica shard都是active状态。

Yellow每个索引的primary shard都是active装，但部分replica shard不是active状态。

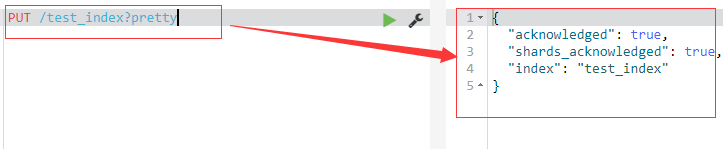
Red 不是所有的primary shard 都是active的。 部分索引数据丢失。

8.3如何快速查询集群中有哪些索引

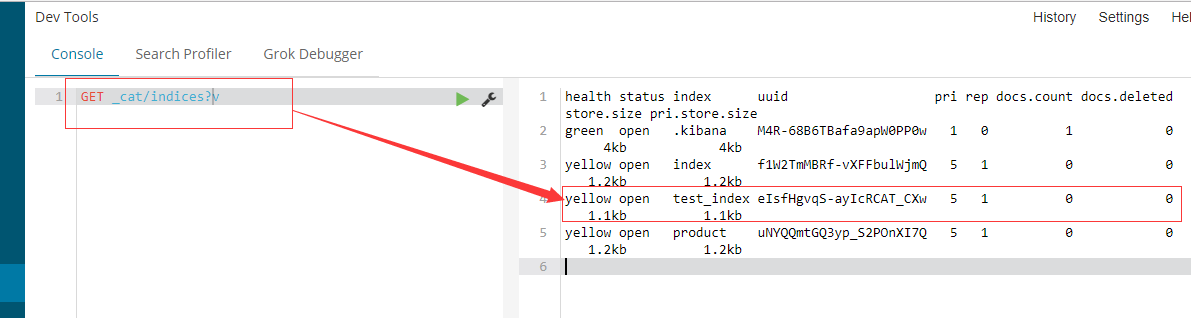
Get \_cat/indices?v

8.4索引管理

8.4.1 创建索引



8.4.2查询索引



8.4.3 删除索引

DELETE test\_index

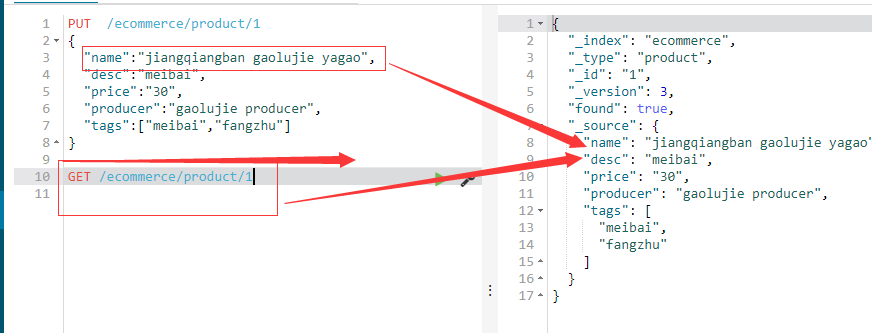
8.5商品的（CRUD操作）

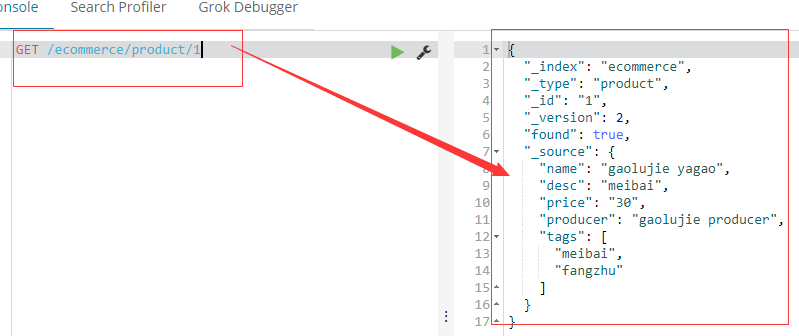
8.5.1新增商品，新增文档，建立索引

|  |
| --- |
| PUT /ecommerce/product/2  {  "name":"jiajieshi yagao",  "desc":"fangzhu",  "price":"31",  "producer":"jiajieshi producer",  "tags":["fangzhu"]  }  PUT /ecommerce/product/3  {  "name":"zhaonghua yagao",  "desc":"caoben zhiwu",  "price":"26",  "producer":"zhonghua producer",  "tags":["qingxin"]  } |

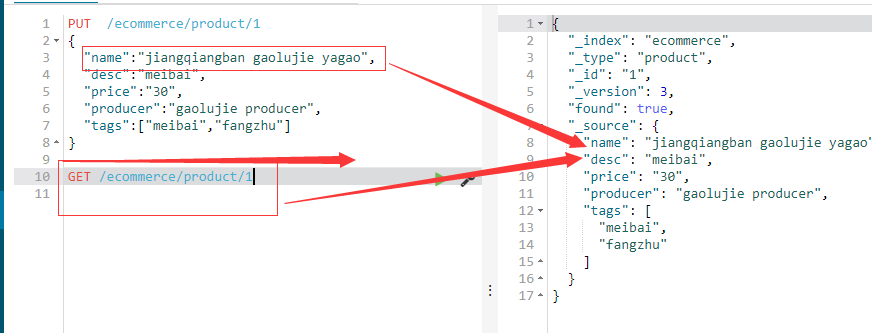
8.5.2 查询商品 检索文档

GET /index/type/id



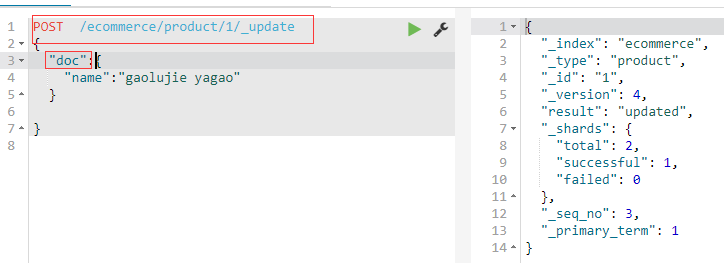


8.5.3修改商品 替换



替换有一个不好的地方 必须带上所有的field

8.5.4修改商品 更新



8.5.5 删除

Delete / ecommerce/product/1

8.6电商产品多种搜索方式

8.6.1 query string search（query string search的由来，因为search参数都是以http请求的query string来附带的）

|  |
| --- |
| #搜索商品名称中包含yagao的商品，而且按照售价降序排序：  GET /ecommerce/product/\_search?q=name:yagao&sort=price:desc  适用于临时的在命令行使用一些工具，比如curl，快速的发出请求，来检索想要的信息；但是如果查询请求很复杂，是很难去构建的  在生产环境中，几乎很少使用query string search |

搜索全部商品：GET /ecommerce/product/\_search

8.6.2 query DSL （Domain Specified Language，特定领域的语言）

http request body：请求体，可以用json的格式来构建查询语法，比较方便，可以构建各种复杂的语法，比query string search肯定强大多了

|  |
| --- |
| #查询所有商品  GET /ecommerce/product/\_search  {  "query": {  "match\_all": {}  }  }  #返回值与 GET /ecommerce/product/\_search 相同  #查询名称包含yagao的商品，同时按照价格降序排序 GET /ecommerce/product/\_search  {  "query": {  "match": { "name":"yagao" }  },  "sort": [  { "price": "desc" }  ]  }  #fenye查询分页  GET ecommerce/product/\_search  {  "query": {  "match\_all": {}  }  , "from": 1 //从第几条数据开始  , "size": 2 //查询多少条数据  }  #指定要查询出来商品的名称和价格字段就可以  GET /ecommerce/product/\_search  {  "query": {  "match\_all": {}  },  "\_source": ["name","price"]  } |

8.6.3 query filter（对数据进行过滤）

|  |
| --- |
| #搜索商品名称包含yagao，而且售价大于25元的商品（这里是有两个查询条件）  GET ecommerce/product/\_search  {  "query": {  "bool": {  "must": [  {  "match": {  "name": "yagao"  }  }  ]  , "filter": {  "range": {  "price": {"gt":25}  }  }  }  }  } |

8.6.4 全文检索（full-text search）

|  |
| --- |
| PUT ecommerce/product/4  {  "name":"special yagao",  "desc":"special meibai",  "price":50,  "producer":"special yagao producer",  "tags":["meibai"]  }  GET /ecommerce/product/\_search  {  "query": {  "match": {  "producer": "yagao producer"  }  }  }  #过程  {  "took": 20,  "timed\_out": false,  "\_shards": {  "total": 5,  "successful": 5,  "skipped": 0,  "failed": 0  },  "hits": {  "total": 3,  "max\_score": 0.8092568,  "hits": [  {  "\_index": "ecommerce",  "\_type": "product",  "\_id": "4",  "\_score": 0.8092568,  "\_source": {  "name": "special yagao",  "desc": "special meibai",  "price": 50,  "producer": "special yagao producer",  "tags": [  "meibai"  ]  }  },  {  "\_index": "ecommerce",  "\_type": "product",  "\_id": "3",  "\_score": 0.2876821,  "\_source": {  "name": "zhaonghua yagao",  "desc": "caoben zhiwu",  "price": "26",  "producer": "zhonghua producer",  "tags": [  "qingxin"  ]  }  },  {  "\_index": "ecommerce",  "\_type": "product",  "\_id": "2",  "\_score": 0.19856805,  "\_source": {  "name": "jiajieshi yagao",  "desc": "fangzhu",  "price": "31",  "producer": "jiajieshi producer",  "tags": [  "fangzhu"  ]  }  }  ]  }  }  #结果  {  "took": 20,  "timed\_out": false,  "\_shards": {  "total": 5,  "successful": 5,  "skipped": 0,  "failed": 0  },  "hits": {  "total": 3,  "max\_score": 0.8092568,  "hits": [  {  "\_index": "ecommerce",  "\_type": "product",  "\_id": "4",  "\_score": 0.8092568,  "\_source": {  "name": "special yagao",  "desc": "special meibai",  "price": 50,  "producer": "special yagao producer",  "tags": [  "meibai"  ]  }  },  {  "\_index": "ecommerce",  "\_type": "product",  "\_id": "3",  "\_score": 0.2876821,  "\_source": {  "name": "zhaonghua yagao",  "desc": "caoben zhiwu",  "price": "26",  "producer": "zhonghua producer",  "tags": [  "qingxin"  ]  }  },  {  "\_index": "ecommerce",  "\_type": "product",  "\_id": "2",  "\_score": 0.19856805,  "\_source": {  "name": "jiajieshi yagao",  "desc": "fangzhu",  "price": "31",  "producer": "jiajieshi producer",  "tags": [  "fangzhu"  ]  }  }  ]  }  } |

8.6.5 phrase search（短语搜索）

跟全文检索相对应，相反，全文检索会将输入的搜索串拆解开来，去倒排索引里面去一一匹配，只要能匹配上任意一个拆解后的单词，就可以作为结果返回

phrase search，要求输入的搜索串，必须在指定的字段文本中，完全包含一模一样的，才可以算匹配，才能作为结果返回

|  |
| --- |
| GET /ecommerce/product/\_search  {  "query": {  "match\_phrase": {  "producer": "yagao producer"  }  }  }  #结果  {  "took": 47,  "timed\_out": false,  "\_shards": {  "total": 5,  "successful": 5,  "skipped": 0,  "failed": 0  },  "hits": {  "total": 1,  "max\_score": 0.8092568,  "hits": [  {  "\_index": "ecommerce",  "\_type": "product",  "\_id": "4",  "\_score": 0.8092568,  "\_source": {  "name": "special yagao",  "desc": "special meibai",  "price": 50,  "producer": "special yagao producer",  "tags": [  "meibai"  ]  }  }  ]  }  } |

8.6.6 highlight search（高亮搜索结果）

|  |
| --- |
| GET /ecommerce/product/\_search  {  "query": {  "match\_phrase":  { "producer": "yagao producer"  }    },  "highlight":  { "fields":  { "producer":{}  }  }  }  #结果  {  "took": 206,  "timed\_out": false,  "\_shards": {  "total": 5,  "successful": 5,  "skipped": 0,  "failed": 0  },  "hits": {  "total": 1,  "max\_score": 0.8092568,  "hits": [  {  "\_index": "ecommerce",  "\_type": "product",  "\_id": "4",  "\_score": 0.8092568,  "\_source": {  "name": "special yagao",  "desc": "special meibai",  "price": 50,  "producer": "special yagao producer",  "tags": [  "meibai"  ]  },  "highlight": {  "producer": [  "special <em>yagao</em> <em>producer</em>"  ]  }  }  ]  }  } |

8.7电商管理：嵌套聚合，下钻分析，聚合分析

8.7.1聚合 分组聚合，求每组的数量

|  |
| --- |
| 需求1：#计算每个tag下的商品数量 分组聚合，求每组的数量  GET ecommerce/product/\_search  {  "aggs": {  "group\_by\_tags": { //aggs的名字  "terms": {  "field": "tags.keyword"  }  }  }  }  #结果  "aggregations": {  "group\_by\_tags": {  "doc\_count\_error\_upper\_bound": 0,  "sum\_other\_doc\_count": 0,  "buckets": [  {  "key": "fangzhu",  "doc\_count": 1  },  {  "key": "meibai",  "doc\_count": 1  },  {  "key": "qingxin",  "doc\_count": 1  }  ]  }  }  }  //修改filedata  PUT /ecommerce/\_mapping/product  {  "properties": {  "tags":{  "type": "text"  , "fielddata":"true"  }  }  }  GET ecommerce/product/\_search  {  "aggs": {  "group\_by\_tags": {  "terms": {  "field": "tags"  }  }  }  }  #需求2 不仅要聚合还要搜索  对名称中包含yagao的商品 计算每个tags下的商品数量  GET ecommerce/product/\_search  {  "size": 0,  "query": {  "match": {  "name": "yagao"  }  },  "aggs": {  "group\_by\_tags": {  "terms": {  "field": "tags"  }  }  }  }  #需求3：先分组 再算每组的平均值 计算每个tags商品的平均价格  GET ecommerce/product/\_search  {  "size": 0  , "aggs": {  "group\_by\_tags": {  "terms": { "field": "tags" }  , "aggs": {  "avg\_price": {  "avg": {  "field": "price"  }  }  }  }  }  }  #需求4：在需求3的基础上进行降序排序  GET ecommerce/product/\_search  {  "size": 0  , "aggs": {  "group\_by\_tags": {  "terms": { "field": "tags" ,"order": {  "avg\_price": "asc" //根据avg\_price进行进行降序排序  }}  , "aggs": {  "avg\_price": {  "avg": {  "field": "price"  }  }  }  }  }  }  #需求5 ：按照指定的价格区间进行分组。然后在每组内按照tags分组，租后在计算每组的价格  GET ecommerce/product/\_search  {  "aggs": {  "group\_by\_price": {  "range": {  "fiel": "price",  "ranges": [  {  "from": 0,  "to": 20  },  {  "from": 20,  "to": 40  },  {  "from": 40,  "to": 60  }  ]  },  "aggs": {  "group\_by\_tags": {  "terms": {  "field": "tags"  },  "aggs": {  "avg\_price": {  "avg": {  "field": "price"  }  }  }  }  }  }  }  } |

# 9 .fulltext Search（全文检索）如何 实现精准控制搜索

Match query负责全文检索

9.1 简单匹配or的关系

|  |
| --- |
| GET /forum/article/\_search  {  query":{  "match":{  "title":"java ElasticSearch"  }  }  } |

9.2 and 匹配

|  |
| --- |
| GET /forum/article/\_search  {  query":{  "match":{  title：{  "query":"java ElasticSearch"  “operation”：”and”  }  }  }  } |

9.3指定所搜关键字至少匹配多少关键字，才能作为结果返回

|  |
| --- |
| GET /forum/article/\_search  {  query":{  "match":{  title：{  "query":"java ElasticSearch spark hadoop"  “minimum\_should\_match”：”and”:”75%”  }  }  }  } |

9.4bool组合多个搜索条件，如何relevance score

|  |
| --- |
| GET /forum/article/\_search  {  query":{  "bool":{  “must”:{ “match”:{ “title”:”java”}}  “must\_not”:{“match”:{ “title”:”spark”}}  “should”:{“match”:{“title”:”hadoop”}},  {“match”:{“title”:”elasticSearch” }}  }  }  } |

9.5搜索 java hadop，spark，elasticsearch 至少包含三个关键字

默认情况下should是可以不匹配任何一个的，

但也有例外的情况，如果没有must的话，should必须匹配至少一个才行，比如下面这条记录

|  |
| --- |
| GET /forum/article/\_search  {  query":{  "bool":[  “should”:{“match”:{“title”:”hadoop”}},  {“match”:{“title”:”elasticSearch” }},  {“match”:{“title”:”java” }} ,  {“match”:{“title”:”spark” }},  ]  “minimum\_should\_match”:”3”  }  } |

在诸如match—query进行多值搜索时，es底层会自动将match query转换为bool

# 10搜索条件权重控制

需求：搜索标题中包含java的帖子，而且标题中包含elasticsearch，handoop有限搜索，同时一个包含java handoop，另一个帖子包含java elasticsearch，包含java handoop的帖子要比包含java elasticsearch的帖子有限搜索出来。

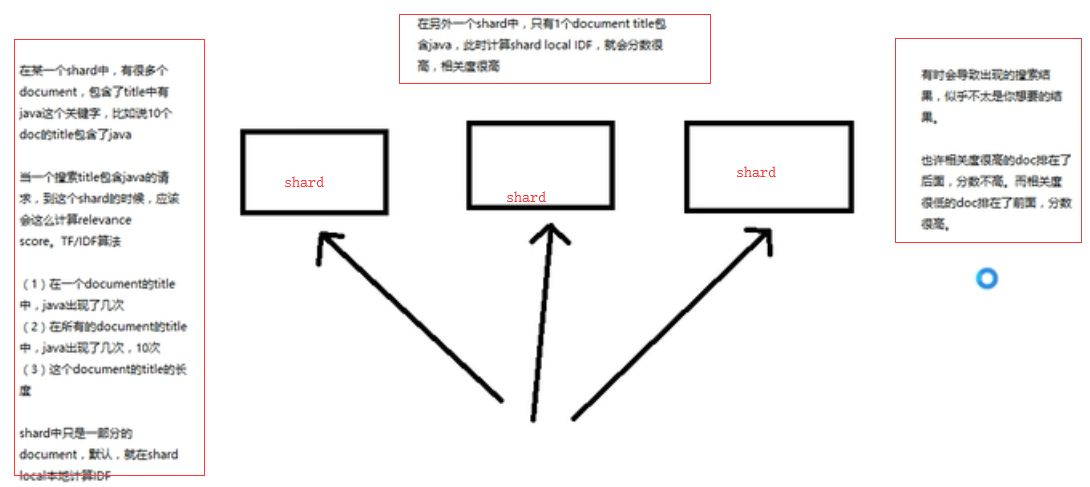
知识点：搜索条件的权重 boost 可以将某个搜索条件的权重加大，此时当匹配这个搜索条件和匹配另一个搜索条件的document，计算Relevance score时，匹配权重更大的那个documen。

默认情况搜索条件的权重都是一样的 ，都是1

|  |
| --- |
| GET /forum/article/\_search  {  "query": {  "bool": {  "must": [  {  "match": {  "title": "java"  }  }  ]  , "should": [  {  "match": {  "title":{  "query": "handoop",  "boost":"3"  }  }  },  {  "match": {  "title": {  "query": "elasticsearch",  "boost":"2"  }  }  }  ]  }  } |

# 11.在多shard场景下relevance score不准确的问题

11.1如果你的一个index有多个shard，你的搜索结果可能不准确，



11.2如何解决这个问题

11.2.1生产环境下，数量数量大尽可能的均匀分配，数据量很大的时候，在概率足额的背景下es都是在多个shard中 均匀路由的，路由根据\_id实现负载均衡

比如：有10个文档 title都包含java 一共有五个shard，如果负载均衡每个shard会包含两个文档，title包含java

11.2.2 测试环环境下，将索引primary shard设置为1 number—shard=1

# 12. \_ElasticSearch dis\_max实现best fields策略进行多字段搜索

12.1 best fields策略 概念

基于多个 field 查询如 title(标题) content 内容.

搜索title或content中包含java或solution的帖子

期望：如果title中包含了java和solution 。或者 content 中保护 java和solution 这样的doc 优先排在前面。

best fields策略，就是说，搜索到的结果，应该是某一个field中匹配到了尽可能多的关键词，被排在前面；而不是尽可能多的field匹配到了少数的关键词，排在了前面。

例子：

|  |
| --- |
| 1、title 字段  POST /forum/article/\_bulk  { "update": { "\_id": "1"} }  { "doc" : {"title" : "this is java and elasticsearch blog"} }  { "update": { "\_id": "2"} }  { "doc" : {"title" : "this is java blog"} }  { "update": { "\_id": "3"} }  { "doc" : {"title" : "this is elasticsearch blog"} }  { "update": { "\_id": "4"} }  { "doc" : {"title" : "this is java, elasticsearch, hadoop blog"} }  { "update": { "\_id": "5"} }  { "doc" : {"title" : "this is spark blog"} }  2.为帖子数据增加content字段  POST /forum/article/\_bulk  { "update": { "\_id": "1"} }  { "doc" : {"content" : "i like to write best elasticsearch article"} }  { "update": { "\_id": "2"} }  { "doc" : {"content" : "i think java is the best programming language"} }  { "update": { "\_id": "3"} }  { "doc" : {"content" : "i am only an elasticsearch beginner"} }  { "update": { "\_id": "4"} }  { "doc" : {"content" : "elasticsearch and hadoop are all very good solution, i am a beginner"} }  { "update": { "\_id": "5"} }  { "doc" : {"content" : "spark is best big data solution based on scala ,an programming language similar to java"} }  3. 搜索title或content中包含java或solution的帖子  下面这个就是multi-field搜索，多字段搜索  GET /forum/article/\_search  {  "query": {  "bool": {  "should": [  { "match": { "title": "java solution" }},  { "match": { "content": "java solution" }}  ]  }  }  }  4.结果分析  期望的是doc5，结果是doc2, doc4排在了前面 (doc 5 中 content字段 中保护了 java 和 solution)  计算每个document的relevance score：每个query的分数，乘以matched query数量，除以总query数量 |

best fields策略，dis\_max

best fields策略，就是说，搜索到的结果，应该是某一个field中匹配到了尽可能多的关键词，被排在前面；而不是尽可能多的field匹配到了少数的关键词，排在了前面

dis\_max语法，直接取多个query中，分数最高的那一个query的分数即可

|  |
| --- |
| GET /forum/article/\_search  {  "query": {  "dis\_max": {  "queries": [  { "match": { "title": "java solution" }},  { "match": { "content": "java solution" }}  ]  }  }  } |

12.2 tie\_breaker参数优化dis\_max搜索效果

dis\_max，只是取分数最高的那个query的分数而已

场景设置：

某个帖子，doc1，title中包含java（1），content不包含java beginner任何一个关键词

某个帖子，doc2，content中包含beginner（1），title中不包含任何一个关键词

某个帖子，doc3，title中包含java（1），content中包含beginner（1）

以上3个doc的最高score都是1所有最终出来的排序不一定是想要的结果

最终搜索，可能出来的结果是，doc1和doc2排在doc3的前面，而不是我们期望的doc3排在最前面

|  |
| --- |
| •搜索title或content中包含java beginner的帖子  GET /forum/article/\_search  {  "query": {  "dis\_max": {  "queries": [  { "match": { "title": "java beginner" }},  { "match": { "body": "java beginner" }}  ]  }  }  }  tie\_breaker 优化 dis\_max  使用tie\_breaker将其他query的分数也考虑进去  •tie\_breaker参数的意义，在于说，将其他query的分数，乘以tie\_breaker，然后综合与最高分数的那个query的分数，综合在一起进行计算  •除了取最高分以外，还会考虑其他的query的分数  •tie\_breaker的值，在0~1之间，是个小数，就ok  GET /forum/article/\_search  {  "query": {  "dis\_max": {  "queries": [  { "match": { "title": "java beginner" }},  { "match": { "body": "java beginner" }}  ],  "tie\_breaker": 0.3  }  }  } |

# 13.Elasticsearch结构化搜索\_在案例中实战使用term filter来搜索数据

根据用户ID、是否隐藏、帖子ID、发帖日期来搜索帖子

13.1插入一些测试帖子数据

|  |
| --- |
| POST /forum/article/\_bulk  { "index": { "\_id": 1 }}  { "articleID" : "XHDK-A-1293-#fJ3", "userID" : 1, "hidden": false, "postDate": "2017-01-01" }  { "index": { "\_id": 2 }}  { "articleID" : "KDKE-B-9947-#kL5", "userID" : 1, "hidden": false, "postDate": "2017-01-02" }  { "index": { "\_id": 3 }}  { "articleID" : "JODL-X-1937-#pV7", "userID" : 2, "hidden": false, "postDate": "2017-01-01" }  { "index": { "\_id": 4 }}  { "articleID" : "QQPX-R-3956-#aD8", "userID" : 2, "hidden": true, "postDate": "2017-01-02" } |

初步来说，就先搞4个字段，因为整个es是支持json document格式的，所以说扩展性和灵活性非常之好。如果后续随着业务需求的增加，要在document中增加更多的field，那么我们可以很方便的随时添加field。但是如果是在关系型数据库中，比如mysql，我们建立了一个表，现在要给表中新增一些column，那就很坑爹了，必须用复杂的修改表结构的语法去执行。而且可能对系统代码还有一定的影响。

|  |
| --- |
| GET /forum/\_mapping/article  {  "forum": {  "mappings": {  "article": {  "properties": {  "articleID": {  "type": "text",  "fields": {  "keyword": {  "type": "keyword",  "ignore\_above": 256  }  }  },  "hidden": {  "type": "boolean"  },  "postDate": {  "type": "date"  },  "userID": {  "type": "long"  }  }  }  }  }  } |

现在es 5.2版本，type=text，默认会设置两个field，一个是field本身，比如articleID，就是分词的；还有一个的话，就是field.keyword，articleID.keyword，默认不分词，会最多保留256个字符

13..2根据用户ID搜索帖子

|  |
| --- |
| GET /forum/article/\_search  {  "query" : {  "constant\_score" : {  "filter" : {  "term" : {  "userID" : 1  }  }  }  }  } |

term filter/query：对搜索文本不分词，直接拿去倒排索引中匹配，你输入的是什么，就去匹配什么

比如说，如果对搜索文本进行分词的话，“helle world” --> “hello”和“world”，两个词分别去倒排索引中匹配

term，“hello world” --> “hello world”，直接去倒排索引中匹配“hello world”

13.3根据发帖日期搜索帖子

|  |
| --- |
| GET /forum/article/\_search  {  "query" : {  "constant\_score" : {  "filter" : {  "term" : {  "postDate": "2017-01-01"  }  }  }  }  } |

13.4根据帖子ID搜索帖子

|  |
| --- |
| GET /forum/article/\_search  {  "query" : {  "constant\_score" : {  "filter" : {  "term" : {  "articleID" : "XHDK-A-1293-#fJ3"  }  }  }  }  }  {  "took": 1,  "timed\_out": false,  "\_shards": {  "total": 5,  "successful": 5,  "failed": 0  },  "hits": {  "total": 0,  "max\_score": null,  "hits": []  }  }  GET /forum/article/\_search  {  "query" : {  "constant\_score" : {  "filter" : {  "term" : {  "articleID.keyword" : "XHDK-A-1293-#fJ3"  }  }  }  }  }  {  "took": 2,  "timed\_out": false,  "\_shards": {  "total": 5,  "successful": 5,  "failed": 0  },  "hits": {  "total": 1,  "max\_score": 1,  "hits": [  {  "\_index": "forum",  "\_type": "article",  "\_id": "1",  "\_score": 1,  "\_source": {  "articleID": "XHDK-A-1293-#fJ3",  "userID": 1,  "hidden": false,  "postDate": "2017-01-01"  }  }  ]  }  } |

articleID.keyword，是es最新版本内置建立的field，就是不分词的。所以一个articleID过来的时候，会建立两次索引，一次是自己本身，是要分词的，分词后放入倒排索引；另外一次是基于articleID.keyword，不分词，保留256个字符最多，直接一个字符串放入倒排索引中。

所以term filter，对text过滤，可以考虑使用内置的field.keyword来进行匹配。但是有个问题，默认就保留256个字符。所以尽可能还是自己去手动建立索引，指定not\_analyzed吧。在最新版本的es中，不需要指定not\_analyzed也可以，将type=keyword即可。

13.5查看分词

|  |
| --- |
| GET /forum/\_analyze  {  "field": "articleID",  "text": "XHDK-A-1293-#fJ3"  } |

默认是analyzed的text类型的field，建立倒排索引的时候，就会对所有的articleID分词，分词以后，原本的articleID就没有了，只有分词后的各个word存在于倒排索引中。

term，是不对搜索文本分词的，XHDK-A-1293-#fJ3 --> XHDK-A-1293-#fJ3；但是articleI建立索引的时候，XHDK-A-1293-#fJ3 --> xhdk，a，1293，fj3

13.6重建索引

|  |
| --- |
| DELETE /forum  PUT /forum  POST /forum/article/\_bulk  { "index": { "\_id": 1 }}  { "articleID" : "XHDK-A-1293-#fJ3", "userID" : 1, "hidden": false, "postDate": "2017-01-01" }  { "index": { "\_id": 2 }}  { "articleID" : "KDKE-B-9947-#kL5", "userID" : 1, "hidden": false, "postDate": "2017-01-02" }  { "index": { "\_id": 3 }}  { "articleID" : "JODL-X-1937-#pV7", "userID" : 2, "hidden": false, "postDate": "2017-01-01" }  { "index": { "\_id": 4 }}  { "articleID" : "QQPX-R-3956-#aD8", "userID" : 2, "hidden": true, "postDate": "2017-01-02" } |

13.7重新根据帖子ID和发帖日期进行搜索

|  |
| --- |
| GET /forum/article/\_search  {  "query" : {  "constant\_score" : {  "filter" : {  "term" : {  "articleID" : "XHDK-A-1293-#fJ3"  }  }  }  }  } |

13.8总结

term filter：根据exact value进行搜索，数字、boolean、date天然支持

text需要建索引时指定为not\_analyzed，才能用term query

相当于SQL中的单个where条件

select \* from forum.article where articleID='XHDK-A-1293-#fJ3';

# 14.Elasticsearch结构化搜索\_filter执行原理深度剖析（bitset机制与caching机制）

14在倒排索引中查找搜索串，获取document list