Es和数据库的对比操作,可以把es理解成一个数据库

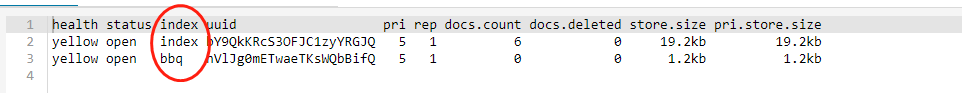
数据库中最重要的几个概念:库,表,字段,主键,增删改查,然后对比着数据库来进行操作,引出es的一些概念

因为es是支持restful风格的操作,所以es的一些命令在理解起来也不是很困难

**名词概念:索引.对比数据库来说,索引有点类似于数据库中的库.比如说咱们做过的项目一般都要连接什么什么库,对于es来说咱们要查询的就是什么什么索引.**

1.简单的索引操作

通过命令**GET /\_cat/indices?v**来查询当前的es中有哪些索引,就是查下咱们有哪些表,一个意思,查询结果如下:



在这里我们可以看到当前的es中包含两个索引,一个是index,是一个bbq,里面的数据是一些文档的状况,有多少条文档啊,删除了多少条文档啊,等等

通过命令**PUT /test\_index**可以添加一条索引

**注意:重复添加索引会报错,就跟数据库重复创建库也会报错一样**

通过命令: **DELETE /test\_index**可以删除一条索引

**注意:重复操作也会报错**

2.商品文档的基础操作(CRUD)

2.1新增商品

标准语法:

put /index/type/id

{

k-v类型的数据

}

**put /goods/fruit/1**

**{**

**"name":"xiangjiao",**

**"describe":"haochi tian",**

**"price":40,**

**"producer":"feilvbin",**

**"tags":["xiangjiao","haochi"]**

**}**

**put /goods/fruit/2**

**{**

**"name":"pingguo",**

**"describe":" cui",**

**"price":60,**

**"producer":"zhongguo",**

**"tags":["haokan","xiang"]**

**}**

其中 goods表示要创建的索引名称,fruit表示要创建的数据类别,1表示插入的数据的主键,这里需要明确一个概念就是倒排索引

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

2.2查询商品

标准语法:

GET /index /type/id

查询一条数据

**GET /goods/fruit/1**

查询结果如下:

{

"\_index" : "goods",

"\_type" : "fruit",

"\_id" : "1",

"\_version" : 1,----------------表示当前的操作版本号

"found" : true,

"\_source" : {------------------表示元数据,内部信息就是当前的这条记录信息

"name" : "xiangjiao",

"describe" : "haochi tian",

"price" : 40,

"producer" : "feilvbin",

"tags" : [

"xiangjiao",

"haochi"

]

}

}

**注意:带下划线的都是es自带的属性**

2.3 修改商品

标准语法:

POST /index/type/id/\_update

{

"doc": {

"field":"vaule"

}

}

**POST /goods/fruit/1/\_update**

**{**

**"doc": {**

**"name":"yutou"**

**}**

**}**

可以同时修改多条属性的值,值与值之间用”,”号间隔,还有一种非主流的写法也是可以的

**PUT /goods/fruit/1**

**{**

**"name":"mei"**

**}**

操作结果如下:

{

"\_index" : "goods",

"\_type" : "fruit",

"\_id" : "1",

"\_version" : 6,

"result" : "updated",------这里指明是修改操作

"\_shards" : {

"total" : 2,

"successful" : 1,

"failed" : 0

},

"\_seq\_no" : 5,

"\_primary\_term" : 1

}

**重点:当再次执行put操作的时候,必须加上所有的field(属性,字段)才可以.如果不加上所有的属性,则只更新提供的属性,其他的属性都清空,如果加上所有的属性,则可以完成修改操作,这个操作实际上是替换操作.把新文档替换掉旧文档,切记!!!**

2.4删除操作

标准语法:

DELETE /index/type/id

**DELETE /goods/fruit/1**

操作结果如下:

{

"\_index" : "goods",

"\_type" : "fruit",

"\_id" : "1",

"\_version" : 14,

"result" : "deleted",------这指明了是已经删除了

"\_shards" : {

"total" : 2,

"successful" : 1,

"failed" : 0

},

"\_seq\_no" : 13,

"\_primary\_term" : 1

}

3.查询方式

3.1 全部检索

标准语法:

GET /index/type/\_search

**GET /goods/fruit/\_search**

查询结果如下:

{

"took" : 5,

"timed\_out" : false,

"\_shards" : {

"total" : 5,

"successful" : 5,

"skipped" : 0,

"failed" : 0

},

"hits" : {

"total" : 1,

"max\_score" : 1.0,

"hits" : [

{

"\_index" : "goods",

"\_type" : "fruit",

"\_id" : "2",

"\_score" : 1.0,

"\_source" : {

"name" : "taozi",

"describe" : "haochi",

"price" : 60,

"producer" : "zhongguo",

"tags" : [

"haokan",

"xiang"

]

}

}

]

}

}

took：耗费了几毫秒

timed\_out：是否超时，这里是没有

\_shards：数据拆成了5个分片，所以对于搜索请求，会打到所有的primary shard

hits.total：查询结果的数量，多少个document

hits.max\_score：score的含义，就是document对于一个search的相关度的匹配分数，越相关，就越匹配，分数也高

hits.hits：包含了匹配搜索的document的详细数据

**注意:这里还有一种查询方式,但是用的比较少,实际开发中也不会用这样的方式去构建一个完整的查询**

**GET /goods/fruit/\_search?q=name:pingguo**

**实际构建后面会讲到**

3.2 构建查询

**名词解释:DSL语法, Domain Specified Language，特定领域语言**

3.2.1按条件查询和排序:

**重点:这里需要创建一批数据,数据在制作数据文档,第一次制作数据里**

**GET /goods/fruit/\_search**

**{**

**"query": {**

**"match": {---------------------这里指的是匹配,不是数据库中的like也不是”=”**

**"name": "pingguo"**

**}**

**},**

**"sort": [----------------------------这里指的是排序方式**

**{**

**"price": {-----------按照价格**

**"order": "desc"--------------------倒叙排列**

**}**

**}**

**]**

**}**

查询结果如下:

{

"took" : 0,

"timed\_out" : false,

"\_shards" : {

"total" : 5,

"successful" : 5,

"skipped" : 0,

"failed" : 0

},

"hits" : {

"total" : 1,

"max\_score" : 0.2876821,------------------这是检索后的评分,0-1之间,分数越大匹配度越高

"hits" : [

{

"\_index" : "goods",

"\_type" : "fruit",

"\_id" : "2",

"\_score" : 0.2876821,-----------------这里是单独每个词条的匹配度,权重越大越靠前

"\_source" : {

"name" : "pingguo",

"describe" : "haochi",

"price" : 60,

"producer" : "zhongguo",

"tags" : [

"haokan",

"xiang"

]

}

}

]

}

}

3.2.2分页查询:

**GET /goods/fruit/\_search**

**{**

**"query": {**

**"match\_all": {}**

**},**

**"from": 0,**

**"size": 5**

**}**

**注意:这里的检索结果是倒排索引,不是按照id排序的,是按照倒排的方式来进行检索的,再强调下,不是根据id排序.**

3.2.3 指定查询内容:

**GET /goods/fruit/\_search**

**{**

**"query": {**

**"match\_all": {}**

**},**

**"\_source":["name","price"]-----------------这里指的是需要检索的内容**

**}**

检索出来的内容也就只包含了name和price字段的内容

3.2.4 布尔查询:

**GET /goods/fruit/\_search**

**{**

**"query": {**

**"bool": {----------------------------布尔查询**

**"must": [----------------------必须匹配,可以同时匹配多个内容**

**{"match": {----------------------匹配规则**

**"name": "xiangjiao"**

**}}**

**]**

**}**

**}**

**}**

多条件匹配查询

**GET /goods/fruit/\_search**

**{**

**"query": {**

**"bool": {**

**"must": [--------------------------这里是数组,已经提示你可以匹配多条内容**

**{"match": {**

**"name": "xiangjiao"**

**}},**

**{**

**"match": {**

**"price": 60**

**}**

**}**

**]**

**}**

**}**

**}**

**注意:**

**Must是必须匹配,相当于and**

**Must\_not是不匹配,相当于not**

**Should是可以匹配,相当于or**

3.2.5 过滤查询:

**GET /goods/fruit/\_search**

**{**

**"query": {**

**"bool": {**

**"should": [**

**{"match": {**

**"name": "xiangjiao"**

**}}**

**],**

**"filter": {-----------------------------这里就是要过滤查询的内容**

**"range": {----------------这里指的是检索范围**

**"price": {----------------检索条件**

**"gte": 10-------------大于等于10**

**}**

**}**

**}**

**}**

**}**

**}**

**注意:filter于must/must\_not/should是并列关系,同属于bool的子属性**

**lt:小于,lte:小于等于,gt:大于,gte:大于等于**

3.2.6 全文检索

**GET /goods/fruit/\_search**

**{**

**"query": {**

**"match": {**

**"tags": "tian suan"-------------这里可以同时匹配两个内容**

**}**

**}**

**}**

**注意:如果tag里包含多个内容,只要有tian或者suan的都会被检索出来,可以自己改几个tag测试下效果**

3.2.7短语检索

**GET /goods/fruit/\_search**

**{**

**"query": {**

**"match\_phrase": {**

**"name": "pingguo"**

**}**

**}**

**}**

**注意:短语检索后面在中文分词的时候会详细讲,这里先见一下就好了**

3.2.8 高亮检索

**GET /goods/fruit/\_search**

**{**

**"query": {**

**"match": {**

**"name": "pingguo"------检索内容**

**}**

**},**

**"highlight": {---------高亮显示部分**

**"pre\_tags": ["<b style='color:red'>"],------------前置标签**

**"post\_tags": ["</b>"], ----------------后置标签**

**"fields": {**

**"name": {}-------------需要高亮显示的内容**

**}**

**}**

**}**

查询结果如下:

{

"took" : 35,

"timed\_out" : false,

"\_shards" : {

"total" : 5,

"successful" : 5,

"skipped" : 0,

"failed" : 0

},

"hits" : {

"total" : 1,

"max\_score" : 0.9808292,

"hits" : [

{

"\_index" : "goods",

"\_type" : "fruit",

"\_id" : "2",

"\_score" : 0.9808292,

"\_source" : {

"name" : "pingguo",

"describe" : "cui",

"price" : 60,

"producer" : "zhongguo",

"tags" : [

"haokan",

"xiang"

]

},

"highlight" : {

"name" : [

"<b style='color:red'>pingguo</b>"------------------这里就是要在页面上高亮显示的内容

]

}

}

]

}

}

**重点:高亮显示是检索中非常重要的一块,通过html标签来决定用何种方式来显示.标签的内容可以随便使用**

**Highlight:表示高亮显示**

**Pre\_tags:表示使用的标签的前部分**

**Post\_tags:表示要使用的标签的后部分**

3.2.9聚合函数

**GET /goods/fruit/\_search**

**{**

**"size": 0,---------------------这里是控制输出的那个东西(可选)**

**"aggs": {-----------------------这里指使用聚合函数**

**"jiage": {---------------------别名**

**"min": {---------------------取最小值**

**"field": "price"---------------------需要查询的属性**

**}**

**}**

**}**

**}**

**注意:min:表示最小值,max:表示最大值,avg:表示平均数,sum:表示求和**

**注意:如果不想要上面那一大坨内容,可以使用size:0来控制输出的内容**

在当前数据的基础上再添加一条数据,要求商品的名称一样,价格有所区别,这里添加的是西瓜商品

补充添加的数据:

**put /goods/fruit/7**

**{**

**"name":"xigua",**

**"describe":"pianyi",**

**"price":100,**

**"producer":"zhongguo",**

**"tags":["da","haochi"]**

**}**

分类查询:查询某一类商品,然后再完成分类产品的聚合函数部分

**GET /goods/fruit/\_search**

**{**

**"query": {**

**"match": {**

**"name": "xigua"**

**}**

**},**

**"aggs": {**

**"max\_price":{------------别名**

**"max": {**

**"field": "price"**

**}**

**}**

**}**

**}**

**3.2.10 复杂聚合(待定)**

**名词解释:词频,就是一个单词出现的频率,中英文都有词频的概念**

统计词频:就是每一个词条出现的频率

首先要做一下设置(\_mapping)后面也会详细的讲到\_mapping的作用

**PUT /goods/fruit/\_mapping**

**{**

**"properties": {--------------设置属性**

**"name":{-------------要改变的属性名称**

**"type": "text",--------------设置类型,一般默认都是text,数字型的为Integer**

**"fielddata": true-----------------太长,放下面解释了**

**}**

**}**

**}**

**名词解释:fielddata:当ES进行排序（sort），统计（aggs）时，ES把涉及到的字段数据全部读取到内存（JVM Heap）中进行操作。相当于进行了数据缓存，提升查询效率。  
所以fielddata是延迟加载的，在加载的时候是这个字段所有的字段都要加载。  
ES中利用fielddata这个正排索引，即从文档到item，来加快统计排序等操作，fielddata实际存储方式为列式存储。**

**GET /goods/fruit/\_search**

**{**

**"size": 0,**

**"aggs": {**

**"count\_name": {**

**"terms": {-----------------------------精确匹配搜索多个值,后面会详细讲和match的区别**

**"field": "name"**

**}**

**}**

**}**

**}**

查询结果如下:

{

"took" : 1,

"timed\_out" : false,

"\_shards" : {

"total" : 5,

"successful" : 5,

"skipped" : 0,

"failed" : 0

},

"hits" : {

"total" : 7,

"max\_score" : 0.0,

"hits" : [ ]

},

"aggregations" : {--------------------------------------看这里,聚合函数的结果

"count\_name" : {

"doc\_count\_error\_upper\_bound" : 0,

"sum\_other\_doc\_count" : 0,

"buckets" : [

{

"key" : "xigua",-----------------当前检索到的内容

"doc\_count" : 2-----------------当前文档的统计数量

},

{

"key" : "boluo",

"doc\_count" : 1

},

{

"key" : "lizi",

"doc\_count" : 1

},

{

"key" : "mihoutao",

"doc\_count" : 1

},

{

"key" : "pingguo",

"doc\_count" : 1

},

{

"key" : "xiangjiao",

"doc\_count" : 1

}

]

}

}

}

3.2.11 分类聚合查询

**GET /goods/fruit/\_search**

**{**

**"size": 0,**

**"query": {---------------先分类查询**

**"match": {**

**"name": "xigua"**

**}**

**},**

**"aggs": {----------------后聚合**

**"count\_name": {**

**"terms": {**

**"field": "name"**

**}**

**}**

**}**

**}**

3.2.12 聚合的嵌套查询

**GET /goods/fruit/\_search**

**{**

**"size": 0,**

**"aggs": {------------------第一次聚合查询,先分类**

**"tag": {**

**"terms": {**

**"field": "tags"**

**},**

**"aggs": {-------------第二次聚合查询,查询最小值**

**"min\_price": {**

**"min": {**

**"field": "price"**

**}**

**}**

**}**

**}**

**}**

**}**

3.2.13 初级变态查询

**GET /goods/fruit/\_search**

**{**

**"size": 0,**

**"aggs": {**

**"group\_price": {**

**"range": {-----------------------------设置查询区间**

**"field": "price",--------------------按照何种属性进行设置**

**"ranges": [---------------区间范围**

**{**

**"from": 0,----------------------0到50这一批**

**"to": 50**

**},**

**{**

**"from": 50,--------------------50到100这一批**

**"to": 100**

**}**

**]**

**}**

**}**

**}**

**}**

查询结果如下:

{

"took" : 0,

"timed\_out" : false,

"\_shards" : {

"total" : 5,

"successful" : 5,

"skipped" : 0,

"failed" : 0

},

"hits" : {

"total" : 7,

"max\_score" : 0.0,

"hits" : [ ]

},

"aggregations" : {

"group\_price" : {

"buckets" : [-----------------------------第一部分这里是分区的区间查询结果

{

"key" : "0.0-50.0",

"from" : 0.0,

"to" : 50.0,

"doc\_count" : 3---------------------有3个文档

},

{

"key" : "50.0-100.0",------------------第二部分

"from" : 50.0,

"to" : 100.0,

"doc\_count" : 2---------------查询到2个文档

}

]

}

}

}

3.2.14 高级变态查询

**GET /goods/fruit/\_search**

**{**

**"size": 0,**

**"aggs": {**

**"group\_price": {**

**"range": {**

**"field": "price",**

**"ranges": [**

**{**

**"from": 0,**

**"to": 50**

**},**

**{**

**"from": 50,**

**"to": 100**

**}**

**]**

**},**

**"aggs": {------------------------这里是变态的地方,根据分组以后再查询最小值**

**"avg\_price": {**

**"avg": {**

**"field": "price"**

**}**

**}**

**}**

**}**

**}**

**}**

查询结果如下:

{

"took" : 1,

"timed\_out" : false,

"\_shards" : {

"total" : 5,

"successful" : 5,

"skipped" : 0,

"failed" : 0

},

"hits" : {

"total" : 7,

"max\_score" : 0.0,

"hits" : [ ]

},

"aggregations" : {

"group\_price" : {

"buckets" : [----------------------------这里是变态的查询结果

{

"key" : "0.0-50.0",----------------第一部分

"from" : 0.0,

"to" : 50.0,

"doc\_count" : 3,

"avg\_price" : {

"value" : 31.666666666666668

}

},

{

"key" : "50.0-100.0",--------------第二部分

"from" : 50.0,

"to" : 100.0,

"doc\_count" : 2,

"avg\_price" : {

"value" : 67.0

}

}

]

}

}

}