

第5章 商品搜索

学习目标

- Elasticsearch安装

```
1 | docker安装Elasticsearch
2 | 系统参数问题-ES消耗资源高
3 | 跨域操作-Kibana/es-head链接ES
```

- IK分词器配置
- Kibana的使用->DSL语句

```
1 | kibana->DSL语句操作->Elasticsearch
```

- ES导入商品搜索数据

```
1 | sku数据导入到Elasticsearch
2 | Map数据类型->Object
```

- 关键词搜索->能够实现搜索流程代码的编写
- 分类统计搜索

1. Elasticsearch 安装

目标

- Elasticsearch使用Docker安装

路径

- Docker安装elasticsearch
- elasticsearch跨域问题处理

讲解

我们之前已经使用过elasticsearch了，这里不再对它进行介绍了，直接下载安装，本章节将采用Docker安装，不过在市面上还有很多采用linux安装，关于linux安装，已经提供了安装手册，这里就不讲了。

(1)docker镜像下载

```
1 | docker pull elasticsearch:5.6.8
```

注意：由于镜像有570MB，所以提供的虚拟机里已经下载好了该镜像，如下图：

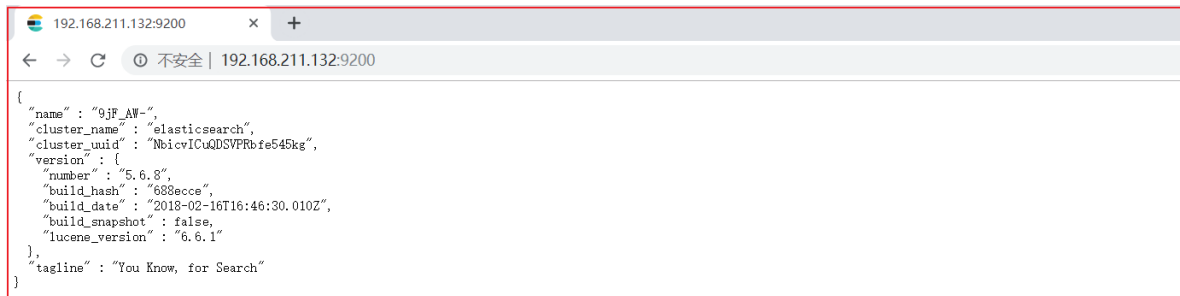
```
[root@localhost ~]# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
docker.io/rabbitmq   latest             80bd4b95a49d       3 weeks ago        198 MB
docker.io/redis       latest             a4fe14ff1981       3 weeks ago        95 MB
docker.io/nginx       latest             53f3fd8007f7       3 weeks ago        109 MB
docker.io/mysql       5.7               7faa3c53e6d6       3 weeks ago        373 MB
docker.io/wurstmeister/kafka latest             c364cbcd5b86       6 weeks ago        421 MB
docker.io/centos      latest             9f38484d220f       2 months ago       202 MB
docker.io/alpine/git   latest             a1d22e4b51ad       3 months ago       27.5 MB
docker.io/lohogames/kibana6.3.2 latest             92815b544b7        10 months ago      771 MB
docker.io/yanliangzhong/elasticsearch6.3.2 latest             96dd1575de9f       10 months ago      826 MB
docker.io/prima/filebeat latest             070a535b6eae       13 months ago      175 MB
docker.io/elasticsearch 5.6.8             6c0bdf761f3b       14 months ago      570 MB
docker.io/styletang/rocketmq-console-ng latest             7df83bbe6e38       21 months ago      702 MB
docker.io/morunchang/fastdfs latest             a729ac95698a       2 years ago        460 MB
docker.io/laoymui/rocketmq latest             6bb10c9d063f       4 years ago        660 MB
docker.io/rossbachp/apache-tomcat8 latest             46366e17bce2       4 years ago        584 MB
[root@localhost ~]#
```

(2)安装es容器

```
1 docker run -di --name=changgou_elasticsearch -p 9200:9200 -p 9300:9300
   elasticsearch:5.6.8
```

9200端口(Web管理平台端口) 9300(服务默认端口)

浏览器输入地址访问: <http://192.168.211.132:9200/>



```
{
  "name" : "9jF_AW-",
  "cluster_name" : "elasticsearch",
  "cluster_uuid" : "NbicvICuQdSVPRbf545kg",
  "version" : {
    "number" : "5.6.8",
    "build_hash" : "688eccc",
    "build_date" : "2018-02-16T16:46:30.010Z",
    "build_snapshot" : false,
    "lucene_version" : "6.6.1"
  },
  "tagline" : "You Know, for Search"
}
```

(3)开启远程连接

上面完成安装后, es并不能正常使用, elasticsearch从5版本以后默认不开启远程连接, 程序直接连接会报如下错误:

```
1 failed to load elasticsearch nodes :
   org.elasticsearch.client.transport.NoNodeAvailableException: None of the
   configured nodes are available: [{#transport#-1}{5ttLpMhkrjKLkvoY7ltUWg}
   {192.168.211.132}{192.168.211.132:9300}]
```

我们需要修改es配置开启远程连接, 代码如下:

登录容器

```
1 docker exec -it changgou_elasticsearch /bin/bash
```

查看目录结构 输入: dir

```
1 root@07f22eb41bb5:/usr/share/elasticsearch# dir
2 NOTICE.txt README.textile bin config data lib logs modules plugins
```

进入config目录

```
1 cd config
```

查看文件

```
1 root@07f22eb41bb5:/usr/share/elasticsearch/config# ls
2 elasticsearch.yml log4j2.properties scripts
```

修改elasticsearch.yml文件

```
1 root@07f22eb41bb5:/usr/share/elasticsearch/config# vi elasticsearch.yml
2 bash: vi: command not found
```

vi命令无法识别，因为docker容器里面没有该命令，我们可以安装该编辑器。

安装vim编辑器

```
1 apt-get update
2 apt-get install vim
```

安装好了后，修改elasticsearch.yml配置，如下图：

```
1 vi elasticsearch.yml
```

修改如下图：



同时添加下面一行代码：

```
1 cluster.name: elasticsearch
```

重启docker

```
1 docker restart changgou_elasticsearch
```

(4)系统参数配置

重启后发现重启启动失败了，这时什么原因呢？这与我们刚才修改的配置有关，因为elasticsearch在启动的时候会进行一些检查，比如最多打开的文件的个数以及虚拟内存区域数量等等，如果你放开了此配置，意味着需要打开更多的文件以及虚拟内存，所以我们还需要系统调优

修改vi /etc/security/limits.conf，追加内容 (nofile是单个进程允许打开的最大文件个数 soft nofile 是软限制 hard nofile是硬限制)

```
1 * soft nofile 65536
2 * hard nofile 65536
```

修改vi /etc/sysctl.conf，追加内容 (限制一个进程可以拥有的VMA(虚拟内存区域)的数量)

```
1 vm.max_map_count=655360
```

执行下面命令 修改内核参数马上生效

```
1 sysctl -p
```

重新启动虚拟机，再次启动容器，发现已经可以启动并远程访问

```
1 | reboot
```

(5)跨域配置

修改elasticsearch/config下的配置文件：elasticsearch.yml，增加以下三句命令，并重启：

```
1 | http.cors.enabled: true
2 | http.cors.allow-origin: "*"
3 | network.host: 127.0.0.1
```

其中：http.cors.enabled: true：此步为允许elasticsearch跨域访问，默认是false。http.cors.allow-origin: "*"：表示跨域访问允许的域名地址（表示任意）。

重启

```
1 | docker restart changgou_elasticsearch
```

小提示：如果想让容器开启重启，可以执行下面命令

```
1 | docker update --restart=always 容器名称或者容器id
```

小提示：Docker中的ES内存不足，无法运行,登录changgou_elasticsearch，修改配置文件 /etc/elasticsearch/jvm.options 文件中，修改内存配置。

总结

2. IK分词器安装

目标

- IK分词器安装

路径

- IK分词器安装
- 自定义扩展词汇
- 自定义停用词汇

讲解

(1)安装ik分词器

IK分词器下载地址<https://github.com/medcl/elasticsearch-analysis-ik/releases>

将ik分词器上传到服务器上，然后解压，并改名字为ik

```
1 | #下载ik分词器
2 | unzip elasticsearch-analysis-ik-5.6.8.zip
3 | #解压
4 | mv elasticsearch ik
```

将ik目录拷贝到docker容器的plugins目录下

```
1 #将ik分词器从虚拟机拷贝到changgou_elasticsearch容器中去
2 docker cp ./ik changgou_elasticsearch:/usr/share/elasticsearch/plugins
3
4 #重启容器,让IK分词器生效
```

(2)IK分词器测试

访问: http://192.168.211.132:9200/_analyze?analyzer=ik_smart&pretty=true&text=我是程序员

```
← → ↻ ① 不安全 | 192.168.211.132:9200/_analyze?analyzer=ik_smart&pretty=true&text=我是程序员

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

访问: http://192.168.211.132:9200/_analyze?analyzer=ik_max_word&pretty=true&text=我是程序员

```
← → ↻ ① 不安全 | 192.168.211.132:9200/_analyze?analyzer=ik_max_word&pretty=true&text=我是程序员

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

作业:

自定义分词器:

修改 `IKAnalyzer.cfg.xml` 配置文件, 添加自定义分词文件。

自定义词: 用户希望那些词语,能被ik分词器所当做一个词语:传智播客 黑马程序员

自定义停用词汇：

修改 `IKAnalyzer.cfg.xml` 配置文件，添加停用词汇文件。

总结

3. Kibana使用-掌握DSL语句

目标

- DSL语句的使用

路径

- 安装Kibana
- 使用DSL语句操作索引库

讲解

我们上面使用的是elasticsearch-head插件实现数据查找的，但是elasticsearch-head的功能比较单一，我们这里需要一个更专业的工具实现对日志的实时分析，也就是我们接下来要讲的kibana。

Kibana 是一款开源的数据分析和可视化平台，它是 Elastic Stack 成员之一，设计用于和 Elasticsearch 协作。您可以使用 Kibana 对 Elasticsearch 索引中的数据进行搜索、查看、交互操作。您可以很方便的利用图表、表格及地图对数据进行多元化的分析和呈现。

Kibana 可以使大数据通俗易懂。它很简单，基于浏览器的界面便于您快速创建和分享动态数据仪表板来追踪 Elasticsearch 的实时数据变化。

搭建 Kibana 非常简单。您可以分分钟完成 Kibana 的安装并开始探索 Elasticsearch 的索引数据 — 没有代码、不需要额外的基础设施。

3.1 Kibana下载安装

我们项目中不再使用linux，直接使用Docker，所有这里就不演示在windows的下载安装了。

(1)镜像下载

```
1 | docker pull docker.io/kibana:5.6.8
```

为了节省时间，虚拟机中已经存在该版本的镜像了。

(2)安装kibana容器

执行如下命令，开始安装kibana容器

```
1 | docker run -it -d -e ELASTICSEARCH_URL=http://192.168.211.132:9200 --name kibana --restart=always -p 5601:5601 kibana:5.6.8
```

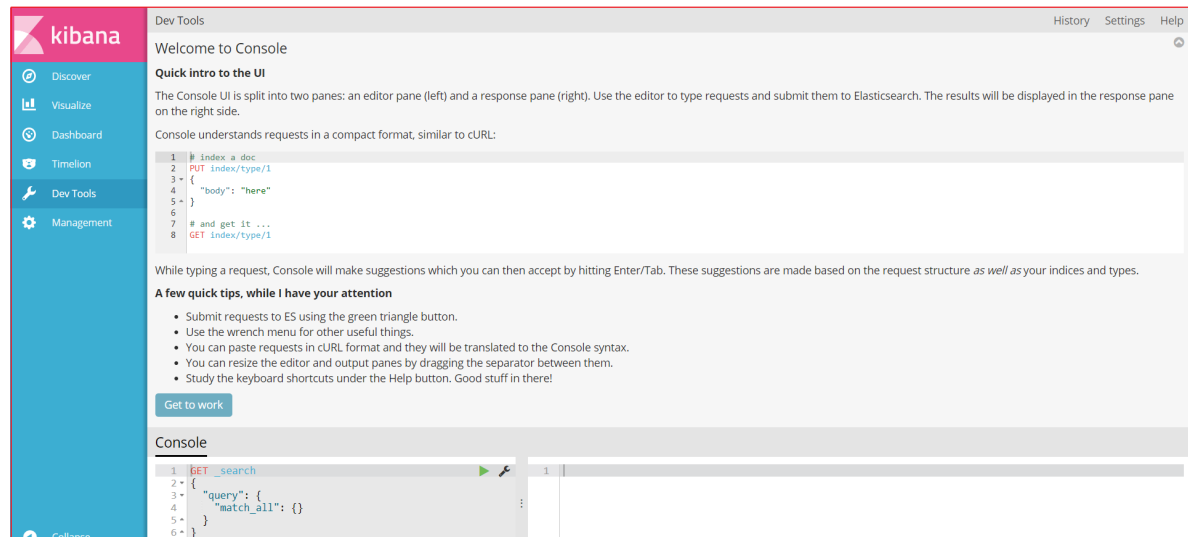
ELASTICSEARCH_URL=<http://192.168.211.132:9200>：是指链接的ES地址

restart=always:每次服务都会重启，也就是开启启动

5601:5601:端口号

(3)访问测试

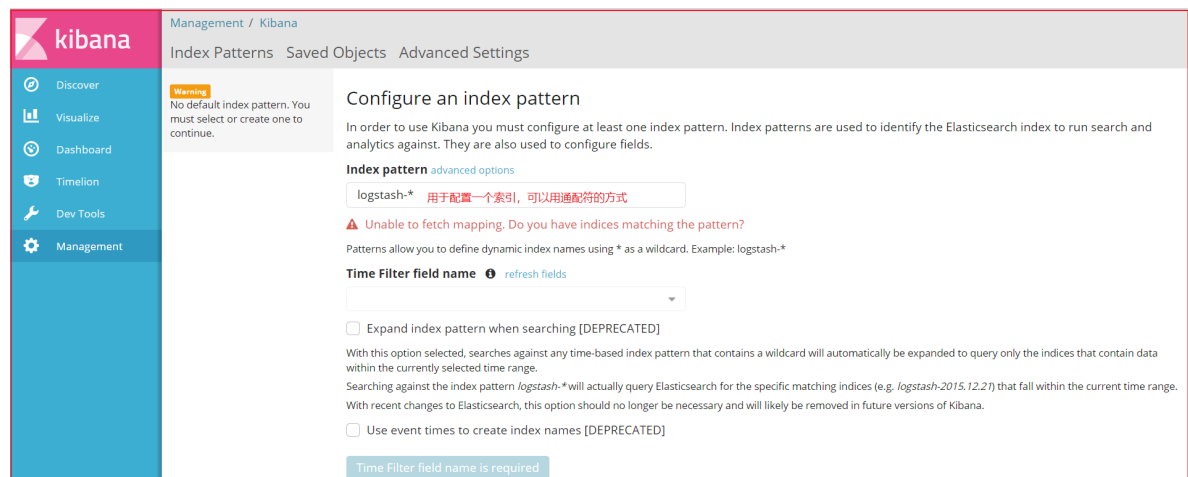
访问 `http://192.168.211.132:5601` 如下：



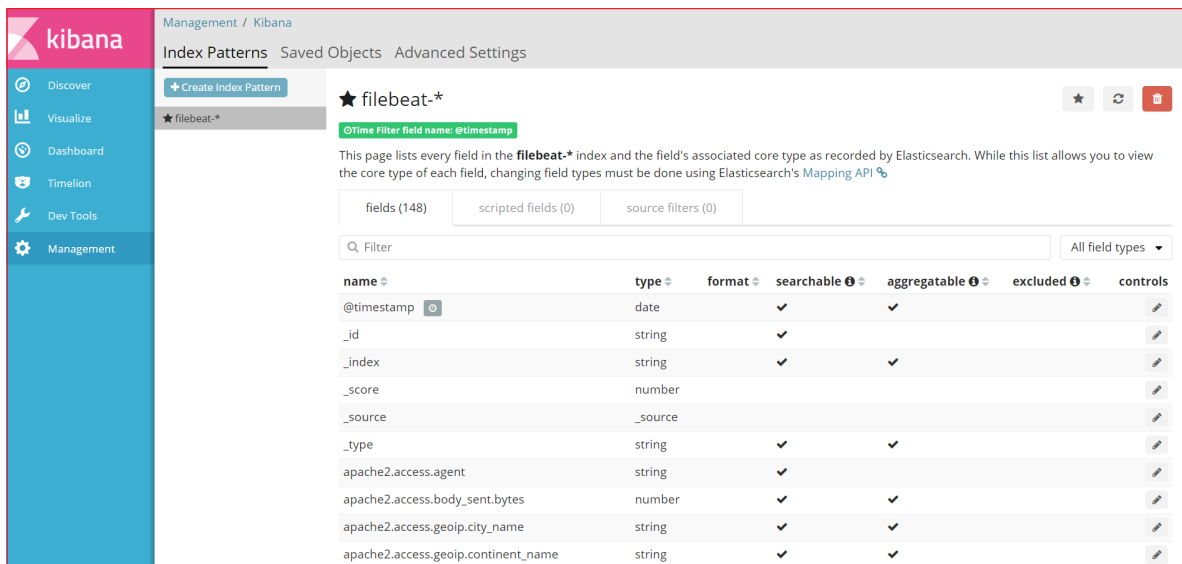
3.2 Kibana使用

3.2.1 配置索引

要使用Kibana，您必须至少配置一个索引。索引用于标识Elasticsearch索引以运行搜索和分析。它们还用于配置字段。



我们修改索引名称的匹配方式即可，下面2个选项不用勾选。点击create，会展示出当前配置的索引的域信息，如下图：

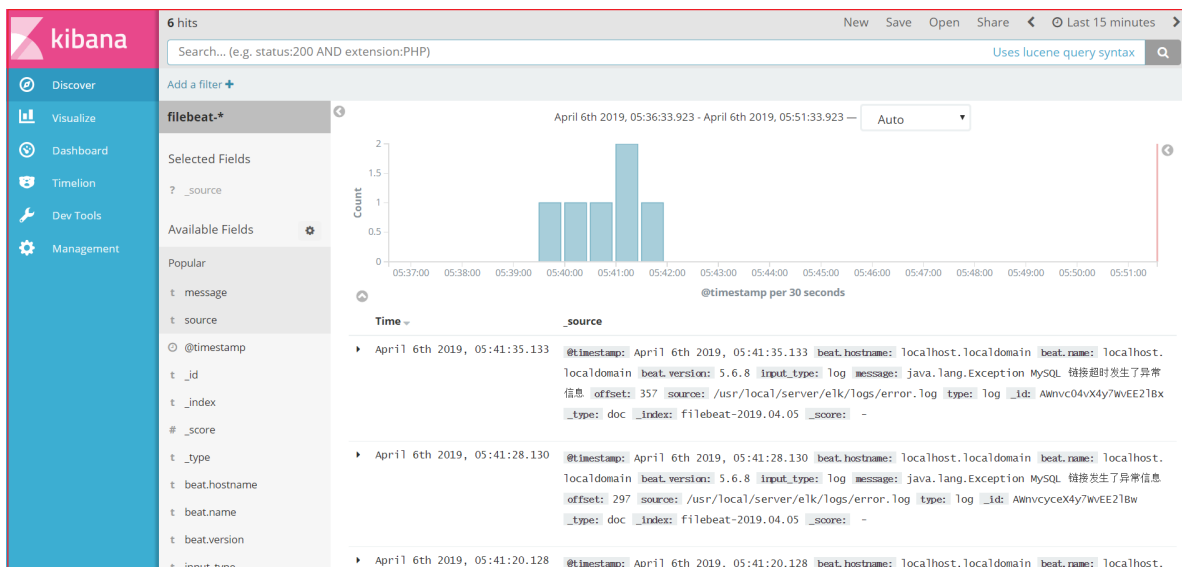


域每个标题选项分别代表如下意思：

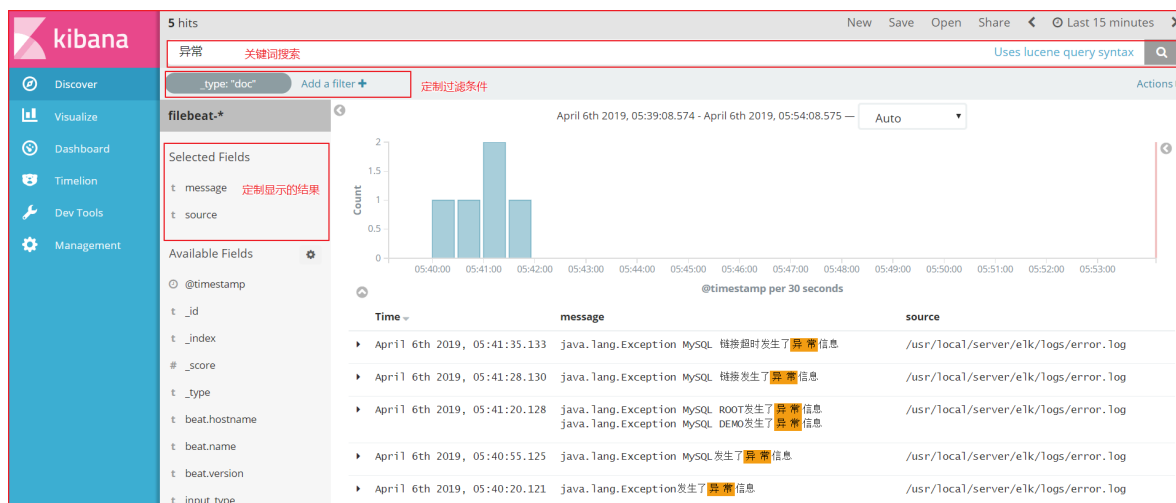
name	type	format	searchable	aggregatable	excluded	controls
名称	类型	格式	搜索	可聚合	排除	控制

3.2.2 数据搜索

Discover为数据搜索部分，可以对日志信息进行搜索操作。



可以使用Discover实现数据搜索过滤和搜索条件显示以及关键词搜索，如下图：



3.2.3 DSL语句使用

3.2.3.1 Query DSL结构化查询介绍

Query DSL是一个Java开源框架用于构建类型安全的SQL查询语句。采用API代替传统的拼接字符串来构造查询语句。目前Querydsl支持的平台包括JPA,JDO, SQL, Java Collections, RDF, Lucene, Hibernate Search。elasticsearch提供了一整套基于JSON的查询DSL语言来定义查询。Query DSL当作一系列的抽象的查询表达式树(AST)特定查询能够包含其它的查询, (如 bool), 有些查询能够包含过滤器(如 constant_score), 还有的可以同时包含查询和过滤器 (如 filtered). 都能够从ES支持查询集合里面选择任意一个查询或者是从过滤器集合里面挑选出任意一个过滤器, 这样的话, 我们就可以构造出任意复杂 (maybe 非常有趣) 的查询了。

3.2.3.2 索引操作

(1)查询所有索引

```
1 GET /_cat/indices?v
```

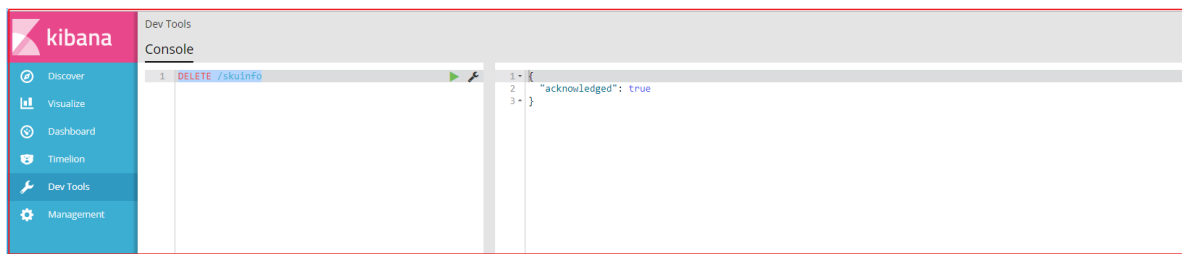
结果如下:

health	status	index	uuid	pri	rep	docs.count	docs.deleted	store.size	pri.store.size
yellow	open	.kibana	1t462b1TSS0dF7T88aCc6Q	1	1	1	0	3.2kb	3.2kb
yellow	open	.kibana	341s17xSVy0oQPSY4ontlg	5	1	31185	0	17.4mb	17.4mb

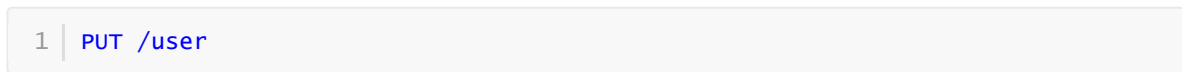
(2)删除某个索引

```
1 DELETE /skuinfo
```

效果如下:



(3)新增索引



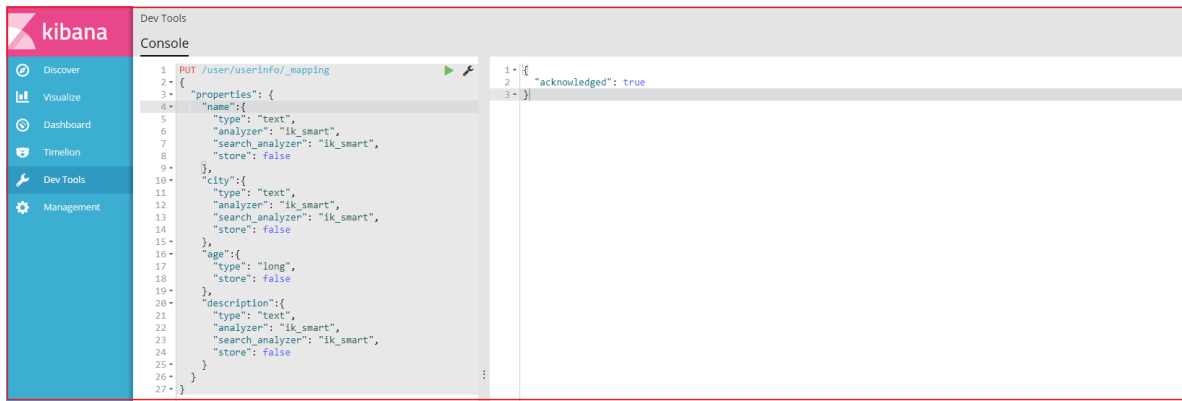
效果如下:



(4)创建映射



效果如下:



(5)新增文档数据

```

1 PUT /user/userinfo/1
2 {
3   "name": "李四",
4   "age": 22,
5   "city": "深圳",
6   "description": "李四来自湖北武汉！"
7 }

```

效果如下：



我们再增加3条记录：

```

1 #新增文档数据 id=2
2 PUT /user/userinfo/2
3 {
4   "name": "王五",
5   "age": 35,
6   "city": "深圳",
7   "description": "王五家住在深圳！"
8 }
9
10 #新增文档数据 id=3
11 PUT /user/userinfo/3
12 {
13   "name": "张三",
14   "age": 19,
15   "city": "深圳",
16   "description": "在深圳打工，来自湖北武汉"
17 }
18
19 #新增文档数据 id=4
20 PUT /user/userinfo/4
21 {
22   "name": "张三丰",

```

```

23     "age":66,
24     "city":"武汉",
25     "description":"在武汉读书，家在武汉！ "
26 }
27
28 #新增文档数据 id=5
29 PUT /user/userinfo/5
30 {
31     "name":"赵子龙",
32     "age":77,
33     "city":"广州",
34     "description":"赵子龙来自深圳宝安，但是在广州工作！ ",
35     "address":"广东省茂名市"
36 }
37
38 #新增文档数据 id=6
39 PUT /user/userinfo/6
40 {
41     "name":"赵毅",
42     "age":55,
43     "city":"广州",
44     "description":"赵毅来自广州白云区，从事电子商务8年！ "
45 }
46
47 #新增文档数据 id=7
48 PUT /user/userinfo/7
49 {
50     "name":"赵哈哈",
51     "age":57,
52     "city":"武汉",
53     "description":"武汉赵哈哈，在深圳打工已有半年了，月薪7500！ "
54 }

```

(6)修改数据

a.替换操作

更新数据可以使用之前的增加操作,这种操作会将整个数据替换掉，代码如下：

```

1  #更新数据,id=4
2  PUT /user/userinfo/4
3  {
4      "name":"张三丰",
5      "description":"在武汉读书，家在武汉！ 在深圳工作！ "
6  }

```

效果如下：

The screenshot shows the Kibana Dev Tools interface. On the left is a sidebar with navigation links: Discover, Visualize, Dashboard, Timeline, Dev Tools (selected), and Management. The main area is titled 'Dev Tools' and contains a 'Console' tab. The console shows a REST client request and its response. The request is a PUT to '/user/userinfo/4' with a JSON body: {'name': '张三丰', 'description': '在武汉读书，家在武汉！ 在深圳工作！ '}. The response is a 200 OK status with a JSON body: {'_index': 'user', '_type': 'userinfo', '_id': '4', '_version': 5, 'result': 'updated', '_shards': {'total': 2, 'successful': 1, 'failed': 0}, 'created': false}.

```

75 #更新数据,id=4
76 PUT /user/userinfo/4
77 {
78     "name":"张三丰",
79     "description":"在武汉读书，家在武汉！ 在深圳工作！ "
80 }
81
82
83
84
85
86
87
88

```

```

1 {
2   "_index": "user",
3   "_type": "userinfo",
4   "_id": "4",
5   "_version": 5,
6   "result": "updated",
7   "_shards": {
8     "total": 2,
9     "successful": 1,
10    "failed": 0
11  },
12   "created": false
13 }

```

使用GET命令查看：

```
1 #根据ID查询
2 GET /user/userinfo/4
```

效果如下：



b.更新操作

我们先使用下面命令恢复数据：

```
1 #恢复文档数据 id=4
2 PUT /user/userinfo/4
3 {
4   "name": "张三丰",
5   "age": 66,
6   "city": "武汉",
7   "description": "在武汉读书，家在武汉！"
8 }
```

使用POST更新某个列的数据

```
1 #使用POST更新某个域的数据
2 POST /user/userinfo/4/_update
3 {
4   "doc": {
5     "name": "张三丰",
6     "description": "在武汉读书，家在武汉！在深圳工作！"
7   }
8 }
```

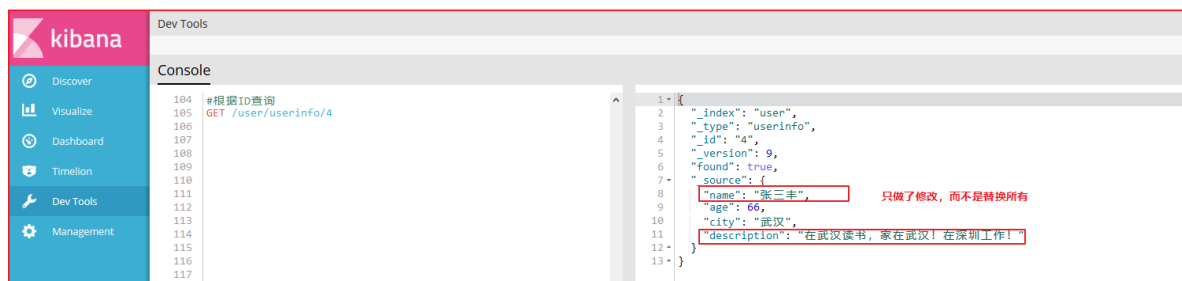
效果如下：



使用GET命令查看：

```
1 #根据ID查询
2 GET /user/userinfo/4
```

效果如下：



(7)删除Document

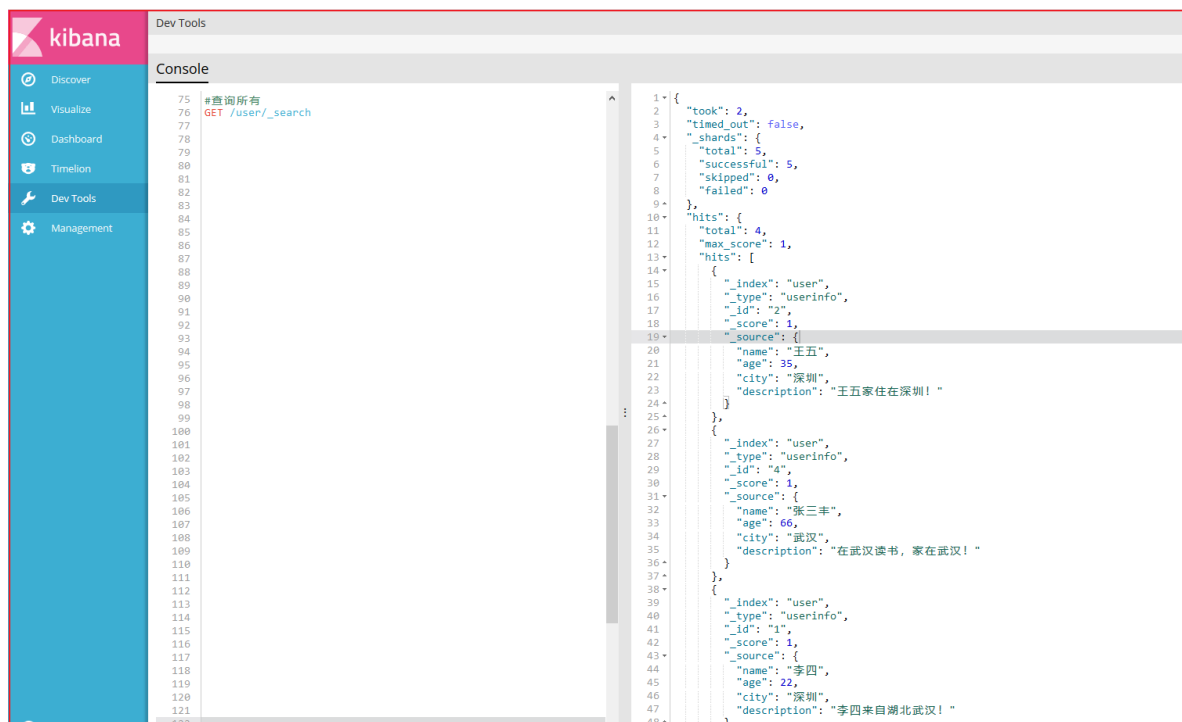
```
1 #删除数据
2 DELETE user/userinfo/7
```

3.2.3.3 数据查询

(1)查询所有数据

```
1 #查询所有
2 GET /user/_search
```

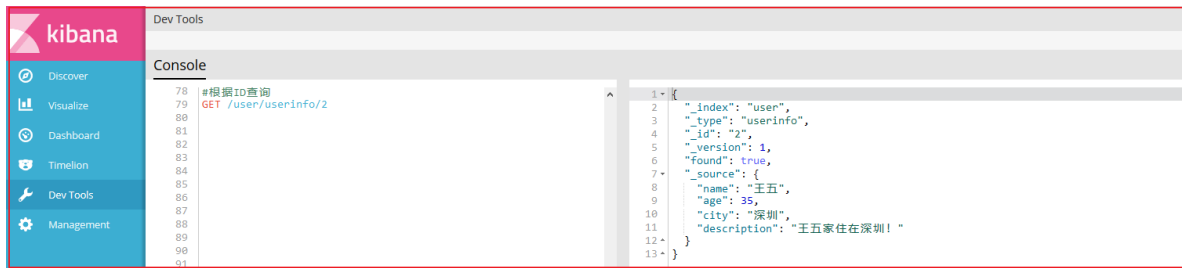
效果如下：



(2)根据ID查询

```
1 #根据ID查询
2 GET /user/userinfo/2
```

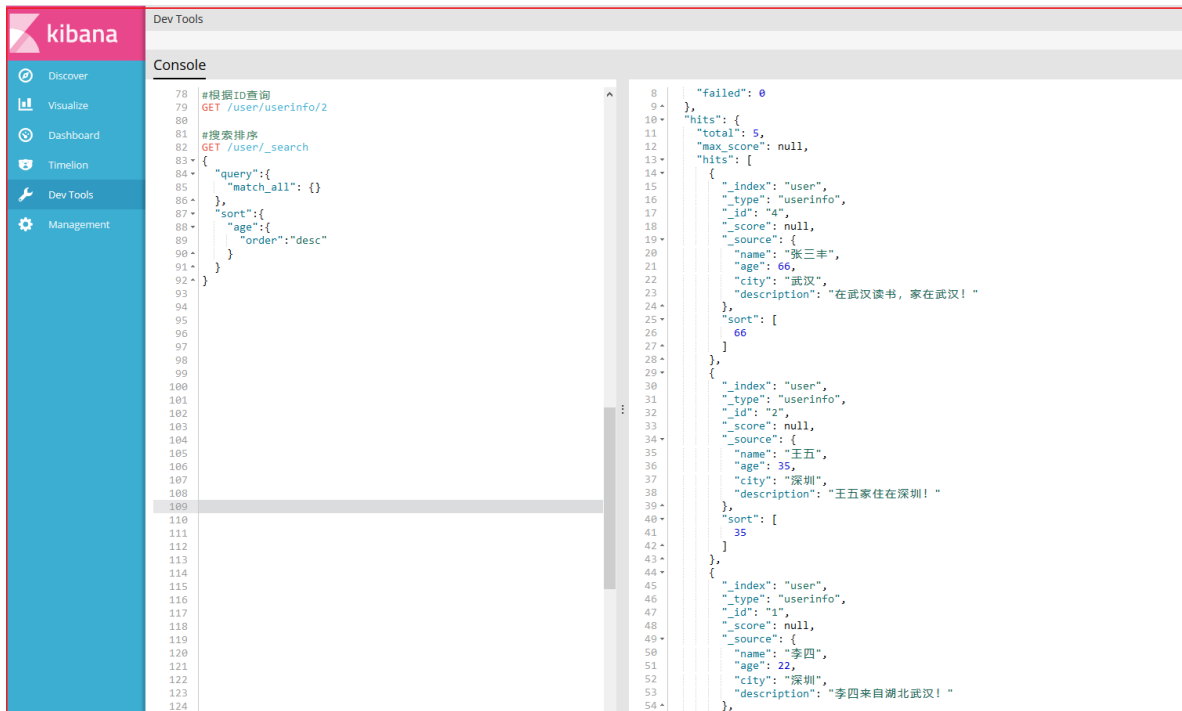
效果如下：



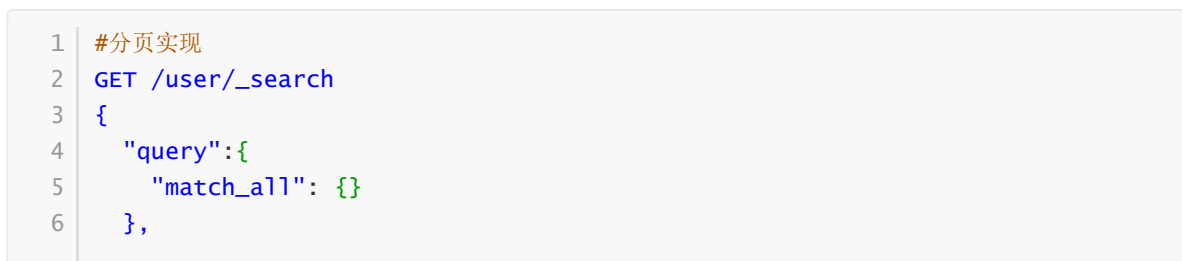
(3)Sort排序



效果如下：



(4)分页



```

7   "sort":{
8     "age":{
9       "order":"desc"
10    }
11  },
12  "from": 0, #从第几页开始,页码是从0页开始
13  "size": 2  #每页显示多少条数据
14 }

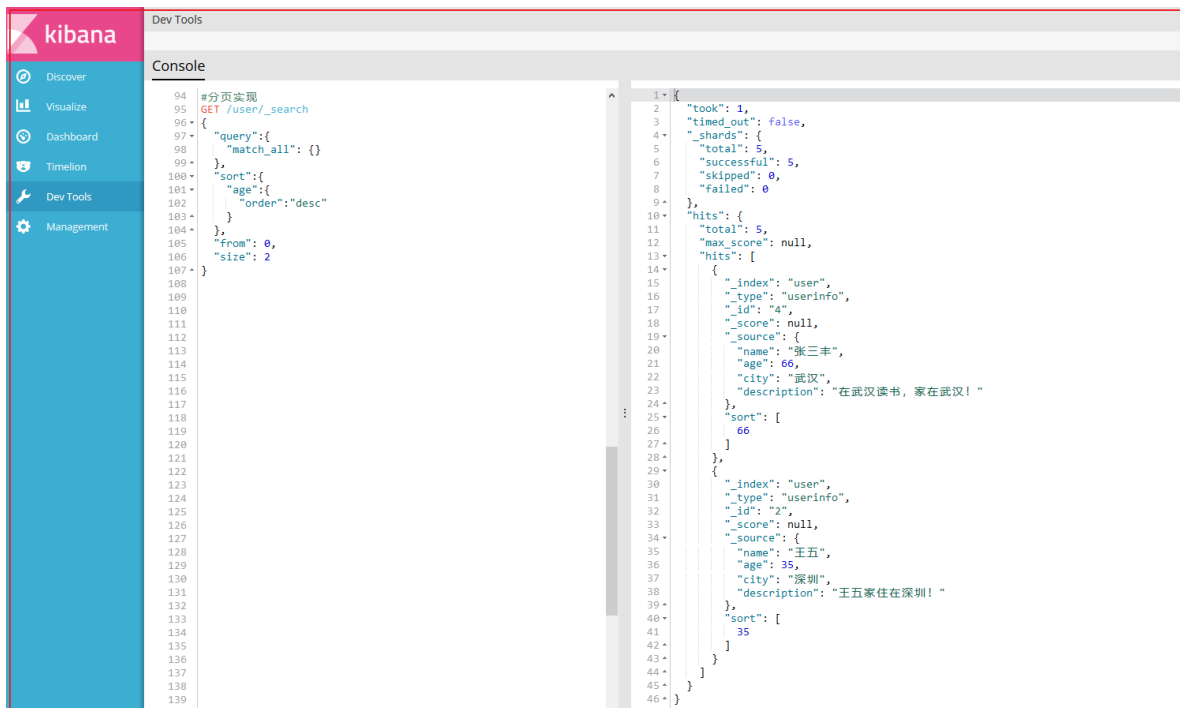
```

解释:

from:从下N的记录开始查询

size:每页显示条数

效果如下:



3.2.3.4 过滤查询

(1)term过滤

term主要用于分词精确匹配，如字符串、数值、日期等（不适合情况：1.列中除英文字符外有其它值 2.字符串值中有冒号或中文 3.系统自带属性如_version）

如下案例:

```

1  #过滤查询-term
2  GET _search
3  {
4    "query":{
5      "term":{
6        "city":"武汉"
7      }
8    }
9  }

```

效果如下:



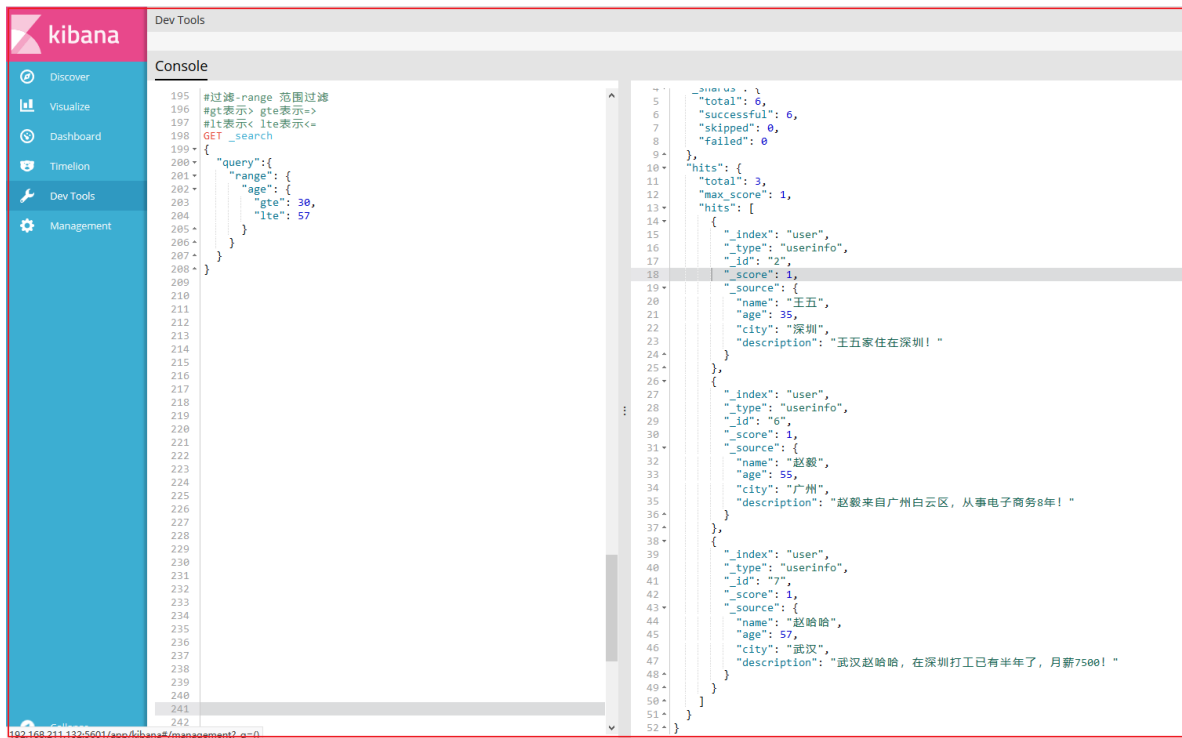
(2)terms 过滤

terms 跟 term 有点类似，但 terms 允许指定多个匹配条件。如果某个字段指定了多个值，那么文档需要一起去做匹配。

案例如下：

```
1 #过滤查询-terms 允许多个Term
2 GET _search
3 {
4   "query":{
5     "terms":{
6       "city":
7         [
8           "武汉",
9           "广州"
10        ]
11     }
12   }
13 }
```

果如下：



(4)exists过滤

exists 过滤可以用于查找拥有某个域的数据

案例如下：

```
1 #过滤搜索 exists: 是指包含某个域的数据检索
2 GET _search
3 {
4   "query": {
5     "exists":{
6       "field":"address"
7     }
8   }
9 }
```

效果如下：



(5) bool 过滤

bool 过滤可以用来合并多个过滤条件查询结果的布尔逻辑，它包含一下操作符：

- must : 多个查询条件的完全匹配,相当于 and。
- must_not : 多个查询条件的相反匹配，相当于 not。
- should : 至少有一个查询条件匹配, 相当于 or。

这些参数可以分别继承一个过滤条件或者一个过滤条件的数组：

案例如下：

```
1  #过滤搜索 bool
2  #must : 多个查询条件的完全匹配,相当于 and。
3  #must_not : 多个查询条件的相反匹配，相当于 not。
4  #should : 至少有一个查询条件匹配，相当于 or。
5  GET _search
6  {
7    "query": {
8      "bool": {
9        "must": [
10         {
11           "term": {
12             "city": {
13               "value": "深圳"
14             }
15           }
16         },
17         {
18           "range": {
19             "age": {
20               "gte": 20,
21               "lte": 99
22             }
23           }
24         }
25       ]
26     }
27   }
28 }
```

效果如下：



(6) match_all 查询

可以查询到所有文档，是没有查询条件下的默认语句。

案例如下：

```
1 #查询所有 match_all
2 GET _search
3 {
4   "query": {
5     "match_all": {}
6   }
7 }
```

(7) match 查询

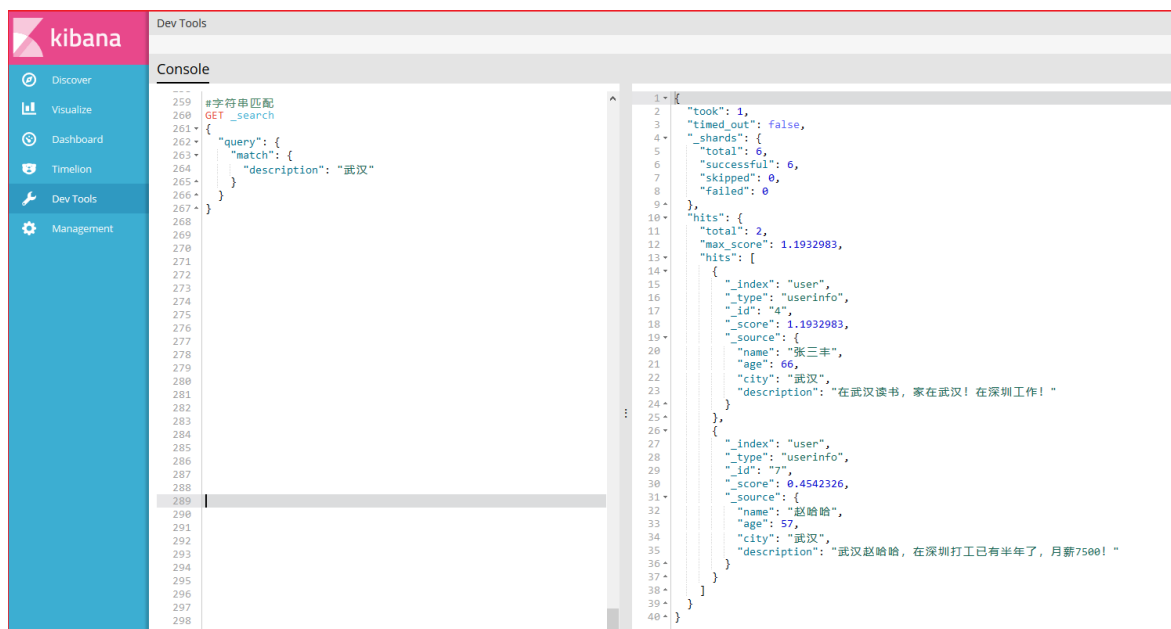
match查询是一个标准查询，不管你需要全文本查询还是精确查询基本上都要用到它。

如果你使用 match 查询一个全文本字段，它会在真正查询之前用分析器先分析match一下查询字符：

案例如下：

```
1 #字符串匹配
2 GET _search
3 {
4   "query": {
5     "match": {
6       "description": "武汉"
7     }
8   }
9 }
```

效果如下：



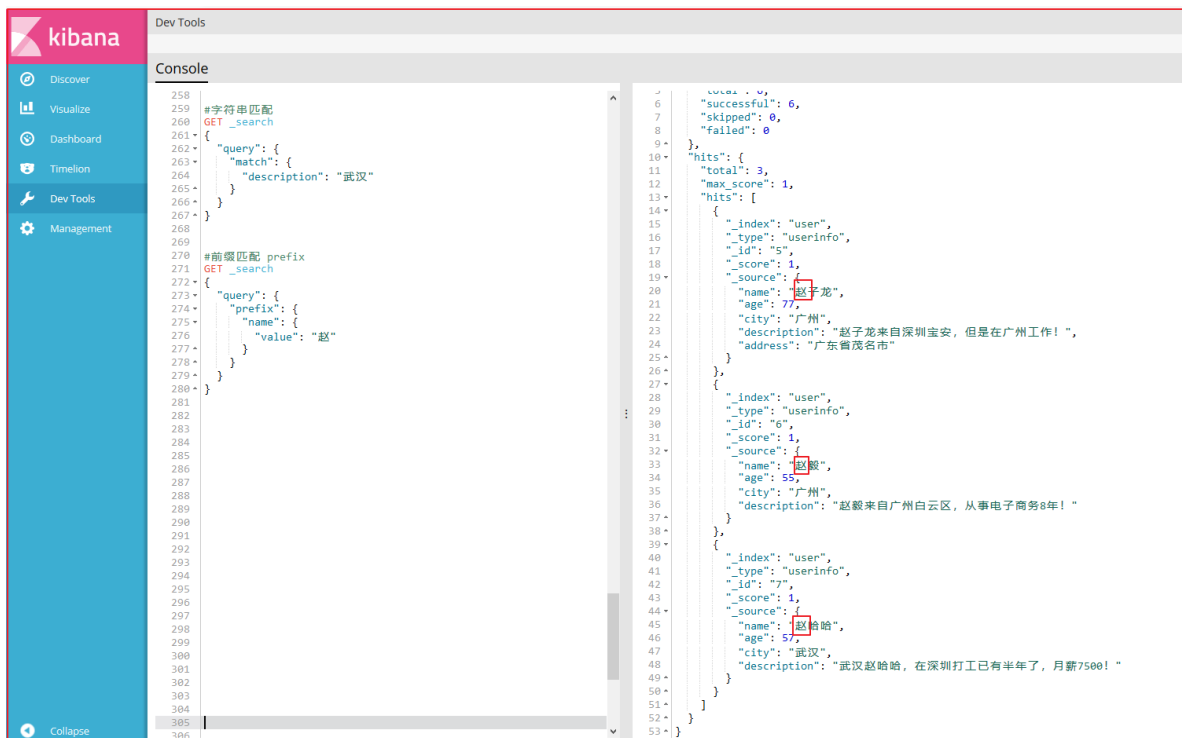
(8)prefix 查询

以什么字符开头的，可以更简单地用 prefix ,例如查询所有以张开始的用户描述 like%国

案例如下：

```
1 #前缀匹配 prefix
2 GET _search
3 {
4   "query": {
5     "prefix": {
6       "name": {
7         "value": "赵"
8       }
9     }
10  }
11 }
```

效果如下：

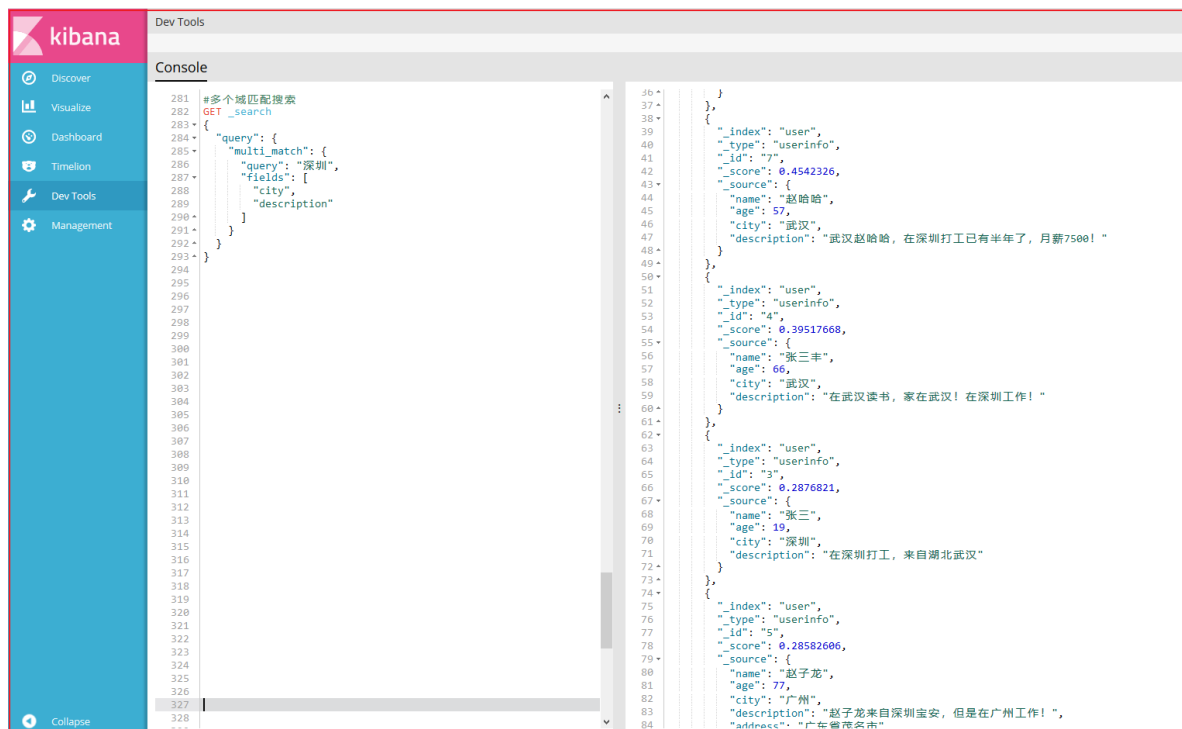


(9)multi_match 查询

multi_match查询允许你做match查询的基础上同时搜索多个字段，在多个字段中同时查一个案例如下：

```
1 #多个域匹配搜索
2 GET _search
3 {
4   "query": {
5     "multi_match": {
6       "query": "深圳",
7       "fields": [
8         "city",
9         "description"
10      ]
11     }
12   }
13 }
```

效果如下：



3.2.3.5 完整DSL语句代码

```
1  #查看所有索引
2  GET /_cat/indices?v
3
4  #删除某个索引
5  DELETE /skuinfo
6
7  #新增索引
8  PUT /user
9
10 #创建映射
11 PUT /user/userinfo/_mapping
12 {
13   "properties": {
14     "name":{
15       "type": "text",
16       "analyzer": "ik_smart",
17       "search_analyzer": "ik_smart",
18       "store": false
19     },
20     "city":{
21       "type": "text",
22       "analyzer": "ik_smart",
23       "search_analyzer": "ik_smart",
24       "store": false
25     },
26     "age":{
27       "type": "long",
28       "store": false
29     },
30     "description":{
31       "type": "text",
```



```
32     "analyzer": "ik_smart",
33     "search_analyzer": "ik_smart",
34     "store": false
35 }
36 }
37 }
38
39 #新增文档数据 id=1
40 PUT /user/userinfo/1
41 {
42     "name": "李四",
43     "age": 22,
44     "city": "深圳",
45     "description": "李四来自湖北武汉！"
46 }
47
48 #新增文档数据 id=2
49 PUT /user/userinfo/2
50 {
51     "name": "王五",
52     "age": 35,
53     "city": "深圳",
54     "description": "王五家住在深圳！"
55 }
56
57 #新增文档数据 id=3
58 PUT /user/userinfo/3
59 {
60     "name": "张三",
61     "age": 19,
62     "city": "深圳",
63     "description": "在深圳打工，来自湖北武汉"
64 }
65
66 #新增文档数据 id=4
67 PUT /user/userinfo/4
68 {
69     "name": "张三丰",
70     "age": 66,
71     "city": "武汉",
72     "description": "在武汉读书，家在武汉！"
73 }
74
75 #新增文档数据 id=5
76 PUT /user/userinfo/5
77 {
78     "name": "赵子龙",
79     "age": 77,
80     "city": "广州",
81     "description": "赵子龙来自深圳宝安，但是在广州工作！",
82     "address": "广东省茂名市"
83 }
84
85 #新增文档数据 id=6
86 PUT /user/userinfo/6
87 {
88     "name": "赵毅",
89     "age": 55,
```

```
90     "city": "广州",
91     "description": "赵毅来自广州白云区，从事电子商务8年！"
92 }
93
94 #新增文档数据 id=7
95 PUT /user/userinfo/7
96 {
97     "name": "赵哈哈",
98     "age": 57,
99     "city": "武汉",
100    "description": "武汉赵哈哈，在深圳打工已有半年了，月薪7500！"
101 }
102
103 #更新数据, id=4
104 PUT /user/userinfo/4
105 {
106     "name": "张三丰",
107     "description": "在武汉读书，家在武汉！在深圳工作！"
108 }
109
110
111 #根据ID查询
112 GET /user/userinfo/4
113
114 #恢复文档数据 id=4
115 PUT /user/userinfo/4
116 {
117     "name": "张三丰",
118     "age": 66,
119     "city": "武汉",
120     "description": "在武汉读书，家在武汉！"
121 }
122
123 #使用POST更新某个域的数据
124 POST /user/userinfo/4/_update
125 {
126     "doc": {
127         "name": "张三丰",
128         "description": "在武汉读书，家在武汉！在深圳工作！"
129     }
130 }
131
132 #根据ID查询
133 GET /user/userinfo/4
134
135 #删除数据
136 DELETE user/userinfo/4
137
138 #查询所有
139 GET /user/_search
140
141 #根据ID查询
142 GET /user/userinfo/2
143
144 #搜索排序
145 GET /user/_search
146 {
147     "query": {
```

```
148     "match_all": {}
149   },
150   "sort":{
151     "age":{
152       "order":"desc"
153     }
154   }
155 }
156
157 #分页实现
158 GET /user/_search
159 {
160   "query":{
161     "match_all": {}
162   },
163   "sort":{
164     "age":{
165       "order":"desc"
166     }
167   },
168   "from": 0,
169   "size": 2
170 }
171
172 #过滤查询-term
173 GET _search
174 {
175   "query":{
176     "term":{
177       "city":"武汉"
178     }
179   }
180 }
181
182 #过滤查询-terms 允许多个Term
183 GET _search
184 {
185   "query":{
186     "terms":{
187       "city":
188         [
189           "武汉",
190           "广州"
191         ]
192     }
193   }
194 }
195
196 #过滤-range 范围过滤
197 #gt表示> gte表示>=
198 #lt表示< lte表示<=
199 GET _search
200 {
201   "query":{
202     "range": {
203       "age": {
204         "gte": 30,
205         "lte": 57
```

```
206     }
207   }
208 }
209 }
210
211
212 #过滤搜索 exists: 是指包含某个域的数据检索
213 GET _search
214 {
215   "query": {
216     "exists":{
217       "field":"address"
218     }
219   }
220 }
221
222 #过滤搜索 bool
223 #must : 多个查询条件的完全匹配,相当于 and。
224 #must_not : 多个查询条件的相反匹配,相当于 not。
225 #should : 至少有一个查询条件匹配, 相当于 or。
226 GET _search
227 {
228   "query": {
229     "bool": {
230       "must": [
231         {
232           "term": {
233             "city": {
234               "value": "深圳"
235             }
236           }
237         },
238         {
239           "range":{
240             "age":{
241               "gte":20,
242               "lte":99
243             }
244           }
245         }
246       ]
247     }
248   }
249 }
250
251 #查询所有 match_all
252 GET _search
253 {
254   "query": {
255     "match_all": {}
256   }
257 }
258
259 #字符串匹配
260 GET _search
261 {
262   "query": {
263     "match": {
```

```
264         "description": "武汉"
265     }
266 }
267 }
268
269 #前缀匹配 prefix
270 GET _search
271 {
272     "query": {
273         "prefix": {
274             "name": {
275                 "value": "赵"
276             }
277         }
278     }
279 }
280
281 #多个域匹配搜索
282 GET _search
283 {
284     "query": {
285         "multi_match": {
286             "query": "深圳",
287             "fields": [
288                 "city",
289                 "description"
290             ]
291         }
292     }
293 }
```

总结

4. 数据导入ES

目标

- 商品数据导入索引库

路径

- 创建数据导入工程集成SpringDataES
- 数据导入Elasticsearch

讲解

4.1 SpringData Elasticsearch介绍

4.1.1 SpringData介绍

Spring Data是一个用于简化数据库访问，并支持云服务的开源框架。其主要目标是使得对数据的访问变得方便快捷，并支持map-reduce框架和云计算数据服务。Spring Data可以极大的简化JPA的写法，可以在几乎不用写实现的情况下，实现对数据的访问和操作。除了CRUD外，还包括如分页、排序等一些常用的功能。

Spring Data的官网：<http://projects.spring.io/spring-data/>

4.1.2 SpringData ES介绍

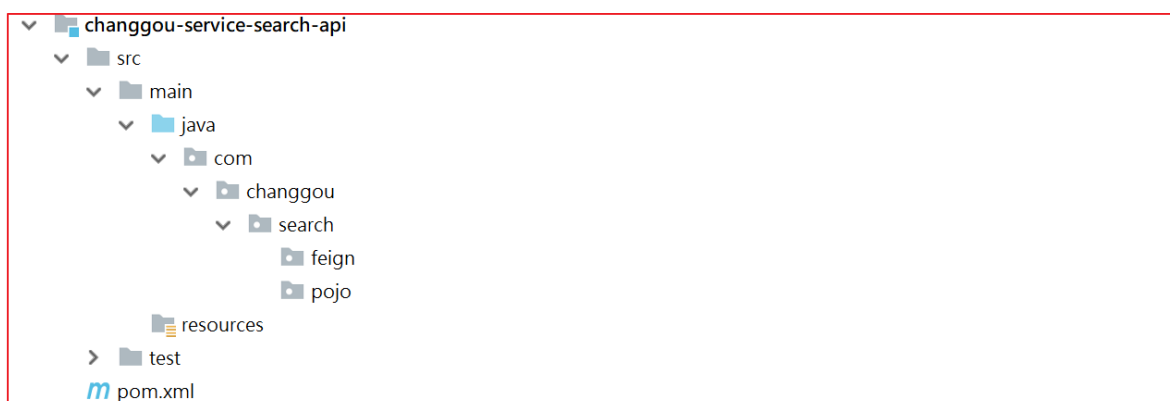
Spring Data ElasticSearch 基于 spring data API 简化 elasticSearch操作，将原始操作ElasticSearch的客户端API 进行封装。Spring Data为Elasticsearch项目提供集成搜索引擎。Spring Data Elasticsearch POJO的关键功能区域为中心的模型与Elasticsearch交互文档和轻松地编写一个存储库数据访问层。官方网站：<http://projects.spring.io/spring-data-elasticsearch/>

4.2 搜索工程搭建

创建搜索微服务工程，changgou-service-search,该工程主要提供搜索服务以及索引数据的更新操作。

(1)API工程搭建

首先创建search的API工程,在changgou-service-api中创建changgou-service-search-api，如下图：



pom.xml如下：

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0"
3         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
5         http://maven.apache.org/xsd/maven-4.0.0.xsd">
6     <parent>
7         <artifactId>changgou-service-api</artifactId>
8         <groupId>com.changgou</groupId>
9         <version>1.0-SNAPSHOT</version>
10    </parent>
11    <modelVersion>4.0.0</modelVersion>
12    <artifactId>changgou-service-search-api</artifactId>
13    <dependencies>
14        <!--goods API依赖-->
15        <dependency>
16            <groupId>com.changgou</groupId>
17            <artifactId>changgou-service-goods-api</artifactId>
18            <version>1.0-SNAPSHOT</version>
```

```

19         </dependency>
20         <!--SpringDataES依赖-->
21         <dependency>
22             <groupId>org.springframework.boot</groupId>
23             <artifactId>spring-boot-starter-data-elasticsearch</artifactId>
24         </dependency>
25     </dependencies>
26 </project>

```

(2)搜索微服务搭建

在changgou-service中搭建changgou-service-search微服务，并进行相关配置。

pom.xml配置

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0"
3         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
5     <parent>
6         <artifactId>changgou-service</artifactId>
7         <groupId>com.changgou</groupId>
8         <version>1.0-SNAPSHOT</version>
9     </parent>
10    <modelVersion>4.0.0</modelVersion>
11    <artifactId>changgou-service-search</artifactId>
12
13    <dependencies>
14        <!--依赖search api-->
15        <dependency>
16            <groupId>com.changgou</groupId>
17            <artifactId>changgou-service-search-api</artifactId>
18            <version>1.0-SNAPSHOT</version>
19        </dependency>
20    </dependencies>
21
22 </project>

```

application.yml配置

```

1 server:
2     port: 18085
3 spring:
4     application:
5         name: search
6     data:
7         elasticsearch:
8             cluster-name: elasticsearch
9             cluster-nodes: 192.168.211.132:9300
10    eureka:
11        client:

```

```

12     service-url:
13         defaultZone: http://127.0.0.1:7001/eureka
14     instance:
15         prefer-ip-address: true
16     #超时配置
17     ribbon:
18         ReadTimeout: 300000

```

配置说明：

```

1 | ribbon.ReadTimeout: Feign请求读取数据超时时间

```

(3)启动类

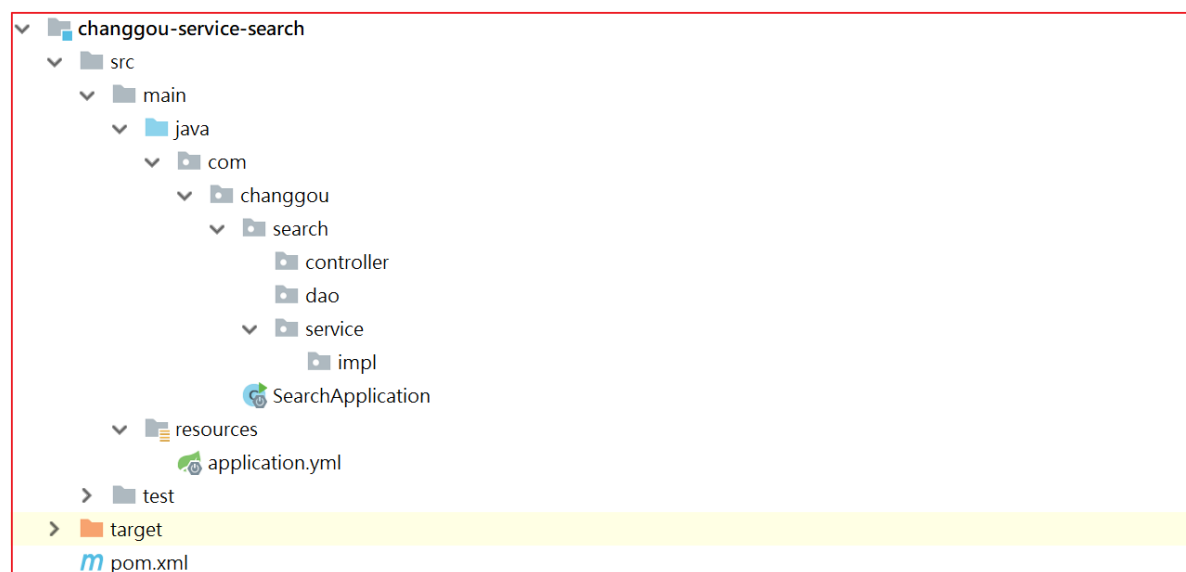
创建SearchApplication作为搜索微服务工程的启动类，代码如下：

```

1  @SpringBootApplication(exclude={DataSourceAutoConfiguration.class})
2  @EnableEurekaClient
3  public class SearchApplication {
4
5      public static void main(String[] args) {
6          /**
7           * Springboot整合Elasticsearch 在项目启动前设置一下的属性，防止报错
8           * 解决netty冲突后初始化client时还会抛出异常
9           * availableProcessors is already set to [12], rejecting [12]
10          ***/
11          //System.setProperty("es.set.netty.runtime.available.processors",
12          "false");
13          SpringApplication.run(SearchApplication.class,args);
14      }
15  }

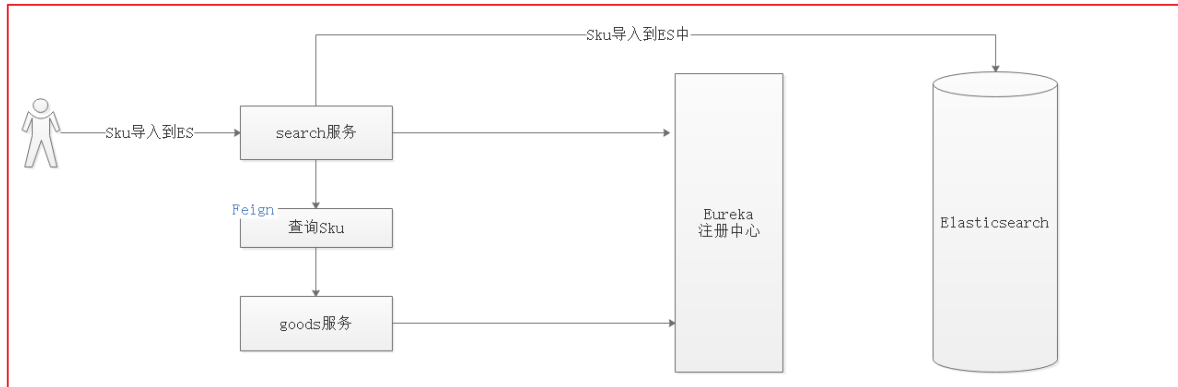
```

分别创建对应的包，dao、service、controller，如下图：



4.3 数据导入

现在需要将数据从数据库中查询出来，然后将数据导入到ES中。



数据导入流程如下：

1. 请求search服务, 调用数据导入地址
2. 根据注册中心中的注册的goods服务的地址, 使用Feign方式查询所有已经审核的Sku
3. 使用SpringData Es将查询到的Sku集合导入到ES中

实现过程:

1. 创建一个JavaBean, 在JavaBean(SkuInfo)中添加索引库映射配置
2. 创建Feign, 实现查询所有Sku集合
3. 在搜索微服务中调用Feign, 查询所有Sku集合, 并且将Sku集合转换成SkuInfo的集合
4. Controller->Service->调用Dao(继承ElasticsearchRepository)实现数据导入到ES中

4.3.1 文档映射Bean创建

搜索商品的时候, 会根据如下属性搜索数据, 并且不是所有的属性都需要分词搜索, 我们创建JavaBean, 将JavaBean数据存入到ES中要以搜索条件和搜索展示结果为依据, 部分关键搜索条件分析如下:

1. 可能会根据商品名称搜索, 而且可以搜索商品名称中的任意一个词语, 所以需要分词
2. 可能会根据商品分类搜索, 商品分类不需要分词
3. 可能会根据商品品牌搜索, 商品品牌不需要分词
4. 可能会根据商品商家搜索, 商品商家不需要分词
5. 可能根据规格进行搜索, 规格是一个键值对结构, 用Map

根据上面的分析, 我们可以在changgou-service-search-api工程中创建com.changgou.search.pojo.SkuInfo, 如下

```
1 @Document(indexName = "skuindex", type = "skuinfo")
2 public class SkuInfo implements Serializable {
3
4     //商品id, 同时也是商品编号
5     @Id //导的包是 org.springframework.data.annotation.Id包
6     private String id;
7
8     //SKU名称
9     @Field(type = FieldType.Text, analyzer = "ik_smart")
10    private String name;
```

```

11
12 //商品价格, 单位为: 元
13 @Field(type = FieldType.Double)
14 private Long price;
15
16 //库存数量
17 private Integer num;
18
19 //商品图片
20 private String image;
21
22 //商品状态, 1-正常, 2-下架, 3-删除
23 private String status;
24
25 //创建时间
26 private Date createTime;
27
28 //更新时间
29 private Date updateTime;
30
31 //是否默认
32 private String isDefault;
33
34 //SPUID
35 private String spuId;
36
37 //类目ID
38 private Long categoryId;
39
40 //类目名称
41 @Field(type = FieldType.Keyword)
42 private String categoryName;
43
44 //品牌名称
45 @Field(type = FieldType.Keyword)
46 private String brandName;
47
48 //规格
49 private String spec;
50
51 //规格参数
52 private Map<String, Object> specMap;
53
54 //...略
55 }

```

4.3.2 搜索审核通过Sku

修改changgou-service-goods微服务, 添加搜索审核通过的Sku, 供search微服务调用。下面都是针对goods微服务的操作。

修改SkuService接口, 添加根据状态查询Sku方法, 代码如下:

```

1  /**
2   * 根据状态查询SKU列表
3   */
4  List<Sku> findByStatus(String status);

```

修改SkuServiceImpl，添加根据状态查询Sku实现方法，代码如下：

```

1  /**
2   * 根据状态查询SKU列表
3   * @return
4   */
5  @Override
6  public List<Sku> findByStatus(String status) {
7      Sku sku = new Sku();
8      sku.setStatus(status);
9      return skuMapper.select(sku);
10 }

```

修改com.changgou.goods.controller.SkuController，添加根据审核状态查询Sku方法，代码如下：

```

1  /**
2   * 根据审核状态查询Sku
3   * @param status
4   * @return
5   */
6  @GetMapping("/status/{status}")
7  public Result<List<Sku>> findByStatus(@PathVariable String status){
8      List<Sku> list = skuService.findByStatus(status);
9      return new Result<List<Sku>>(true, StatusCode.OK, "查询成功", list);
10 }

```

4.3.3 Sku导入ES实现

(1) Feign配置

修改changgou-service-goods-api工程，在com.changgou.goods.feign.SkuFeign上添加findSkuList方法，代码如下：

```

1  @FeignClient(name="goods")
2  @RequestMapping(value = "/sku")
3  public interface SkuFeign {
4
5      /**
6       * 根据审核状态查询Sku
7       * @param status
8       * @return
9       */
10     @GetMapping("/status/{status}")
11     Result<List<Sku>> findByStatus(@PathVariable String status);
12 }

```

(2) Dao创建

修改changgou-service-search工程，创建com.changgou.search.dao.SkuEsMapper,该接口主要用于索引数据操作，主要使用它来实现将数据导入到ES索引库中，代码如下：

```

1  @Repository
2  public interface SkuEsMapper extends ElasticsearchRepository<SkuInfo,String>
3  {
4
5  }

```

(3) 服务层创建

修改changgou-service-search工程，创建com.changgou.search.service.SkuService,代码如下：

```

1  public interface SkuService {
2
3      /**
4       * 导入SKU数据
5       */
6      void importSku();
7  }

```

修改changgou-service-search工程，创建com.changgou.search.service.impl.SkuServiceImpl,实现Sku数据导入到ES中，代码如下：

```

1  @Service
2  public class SkuServiceImpl implements SkuService {
3
4      @Autowired
5      private SkuFeign skuFeign;
6
7      @Autowired
8      private SkuEsMapper skuEsMapper;
9
10     /**
11      * 导入SKU数据
12      */

```

```

13     @Override
14     public void importSku() {
15         //调用changgou-service-goods,真实场景我们建议用分页查询,课堂为了节约时间,
我们一次性查询所有数据
16         Result<List<Sku>> skuResults = skuFeign.findByStatus("1");
17
18         //将数据转成SkuInfo
19         List<SkuInfo> skuInfos =
20
JSON.parseArray(JSON.toJSONString(skuResults.getData()),SkuInfo.class);
21
22         //循环将所有的规格spec转成SpecMap对象
23         for (SkuInfo skuInfo : skuInfos) {
24             //获取Spec
25             String spec = skuInfo.getSpec();
26             //将spec转成Map
27             Map<String, Object> specMap = JSON.parseObject(spec);
28             skuInfo.setSpecMap(specMap);
29         }
30
31         //将所有数据存入到ES中
32         skuEsMapper.saveAll(skuInfos);
33     }
34 }

```

(4)控制层配置

修改changgou-service-search工程,在com.changgou.search.controller.SkuController类中添加如下方法调用上述导入方法,代码如下:

```

1  @RestController
2  @RequestMapping(value = "/search")
3  @CrossOrigin
4  public class SkuController {
5
6      @Autowired
7      private SkuService skuService;
8
9      /**
10       * 导入数据
11       * @return
12       */
13      @GetMapping("/import")
14      public Result importData(){
15          skuService.importSku();
16          return new Result(true, StatusCode.OK, "导入数据到索引库中成功!");
17      }
18  }

```

(5)修改启动类

启动类中需要开启Feign客户端,并且需要添加ES包扫描,代码如下:

```
1 @SpringBootApplication(exclude={DataSourceAutoConfiguration.class})
2 @EnableEurekaClient
3 @EnableFeignClients(basePackages = "com.changgou.goods.feign")
4 @EnableElasticsearchRepositories(basePackages = "com.changgou.search.dao")
5 public class SearchApplication {
6
7     public static void main(String[] args) {
8         /**
9          * Springboot整合Elasticsearch 在项目启动前设置一下的属性，防止报错
10          * 解决netty冲突后初始化client时还会抛出异常
11          * java.lang.IllegalStateException: availableProcessors is already
12          set to [12], rejecting [12]
13          */
14         System.setProperty("es.set.netty.runtime.available.processors",
15 "false");
16         SpringApplication.run(SearchApplication.class,args);
17     }
18 }
```

(6)测试

调用<http://localhost:18085/search/import>进行测试

打开es-head可以看到如下数据：

Elasticsearch http://192.168.211.132:9200/ 连接 my-application 集群健康值: yellow (6 of 12) 信息									
数据浏览 索引 数据浏览 基本查询 复合查询 刷新									
查询 6 个分片中用的 6 个, 195 命中, 耗时 0.005 秒									
所有索引	_index	_type	_id	_score	buildNum	id	name	price	num
.kibana	config		5.6.8	1	15616				
skuinfo	docs		0	1		0		0	0
类型	skuinfo	docs	1999999	1	1999999		华为移动4G对讲机	260	999
default	skuinfo	docs	983253	1	983253		诺基亚(NOKIA) 107 (RM-961) 黑色 移动联通2G手机 双卡双待	168	999
config	skuinfo	docs	1089950	1	1089950		诺基亚(NOKIA) 220 (RM-969) 黑色 移动联通2G手机 双卡双待	269	999
dashboard	skuinfo	docs	1103652	1	1103652		OPPO R1S(R8007)白色 移动4G手机	2499	999
docs	skuinfo	docs	1105661	1	1105661		HTC Desire (816w) 时尚版 联通3G手机 双卡双待	1799	999
graph-workspace	skuinfo	docs	1140358	1	1140358		OPPO Find 7(X9077)标准版 黑色 移动4G手机	2999	999
index-pattern	skuinfo	docs	1143136	1	1143136		OPPO R827S(R6007)白色 移动4G手机	1599	999
search	skuinfo	docs	1182718	1	1182718		诺基亚(NOKIA) Lumia 530 (RM-1019) 灰色 联通3G手机 双卡双待	399	999
server	skuinfo	docs	1199181	1	1199181		OPPO R830S 白色 联通4G手机	1099	999
timelion-sheet	skuinfo	docs	1199729	1	1199729		诺基亚(NOKIA) Lumia 930 (RM-1087) 橙色 联通3G手机	2699	999
url	skuinfo	docs	1226356	1	1226356		诺基亚(NOKIA) 130 (RM-1035) 白色 移动联通2G手机 双卡双待	189	999
visualization	skuinfo	docs	1226359	1	1226359		诺基亚(NOKIA) 130 (RM-1035) 黄色 移动联通2G手机 双卡双待	189	999
字段	skuinfo	docs	1284978	1	1284978		OPPO 1107 白色 移动4G手机 双卡双待	1099	999
▶ accessCount	skuinfo	docs	1322564	1	1322564		OPPO 3005 白色 电信4G手机 双卡双待	1599	999
▶ accessDate	skuinfo	docs	1330216	1	1330216		【联通4G手机】 HTC Desire 820mu 德国时尚白 移动联通4G手机 双卡双待	1149	999
▶ brandName	skuinfo	docs	1340284	1	1340284		OPPO R11C(R8205) 冰晶白 电信4G 双卡双待	2599	999
▶ buildNum	skuinfo	docs	1087918019889467392	1	1087918019889467400		这个是商品名称带图	600000	100
▶ categoryId	skuinfo	docs	1087967575956131840	1	1087967575956131800		新增！！ 红 64G	900000	100
▶ categoryName	skuinfo	docs	1088253802911502336	1	1088253802911502300		新增！！ 红 64G	900000	100
▶ columns	skuinfo	docs	1088253873229008896	1	1088253873229008900		新增！！ 红 64G	900000	100

总结

5. 关键字搜索

目标

- 关键词搜索

路径

- 关键词搜索分析
- 关键搜索实现

讲解



我们先使用SpringDataElasticsearch实现一个简单的搜索功能，先实现根据关键字搜索，从上面搜索图片可以看得到，每次搜索的时候，除了关键字外，还有可能有品牌、分类、规格等，后台接收搜索条件使用Map接收比较合适。

5.1 服务层实现

修改search服务的com.changgou.search.service.SkuService,添加搜索方法，代码如下：

```
1  /**
2   * 搜索
3   * @param searchMap
4   * @return
5   */
6  Map search(Map<String, String> searchMap);
```

修改search服务的com.changgou.search.service.impl.SkuServiceImpl,添加搜索实现方法,代码如下：

```
1  @Autowired
2  private ElasticsearchTemplate esTemplate;
3
4  /**
5   * 搜索数据
6   * @param searchMap
7   * @return
8   */
9  @Override
10 public Map search(Map<String, String> searchMap) {
11     //1. 条件构建
12     NativeSearchQueryBuilder builder = buildBasicQuery(searchMap);
13
14     //2. 搜索列表
15     Map resultMap = searchList(builder);
16
17     return resultMap;
```

```

18 }
19
20
21 /**
22  * 数据搜索
23  * @param builder
24  * @return
25  */
26 private Map searchList(NativeSearchQueryBuilder builder){
27     Map resultMap=new HashMap();//返回结果
28     //查询解析器
29     NativeSearchQuery searchQuery = builder.build();
30     Page<SkuInfo> skuPage =
31     esTemplate.queryForPage(searchQuery,SkuInfo.class);
32
33     //存储对应数据
34     resultMap.put("rows",skuPage.getContent());
35     resultMap.put("totalPages",skuPage.getTotalPages());
36     return resultMap;
37 }
38
39 /**
40  * 构建基本查询
41  * @param searchMap
42  * @return
43  */
44 private NativeSearchQueryBuilder buildBasicQuery(Map<String,String>
45 searchMap) {
46     // 查询构建器
47     NativeSearchQueryBuilder nativeSearchQueryBuilder = new
48 NativeSearchQueryBuilder();
49     if(searchMap!=null){
50         //1.关键字查询
51         if(!StringUtils.isEmpty(searchMap.get("keywords"))){
52             nativeSearchQueryBuilder.withQuery(QueryBuilders.matchQuery("name",searchMa
53 p.get("keywords"))));
54         }
55     }
56     return nativeSearchQueryBuilder;
57 }

```

为了让搜索更清晰，我们将每个步骤封装成独立的方法了。

5.2 控制层实现

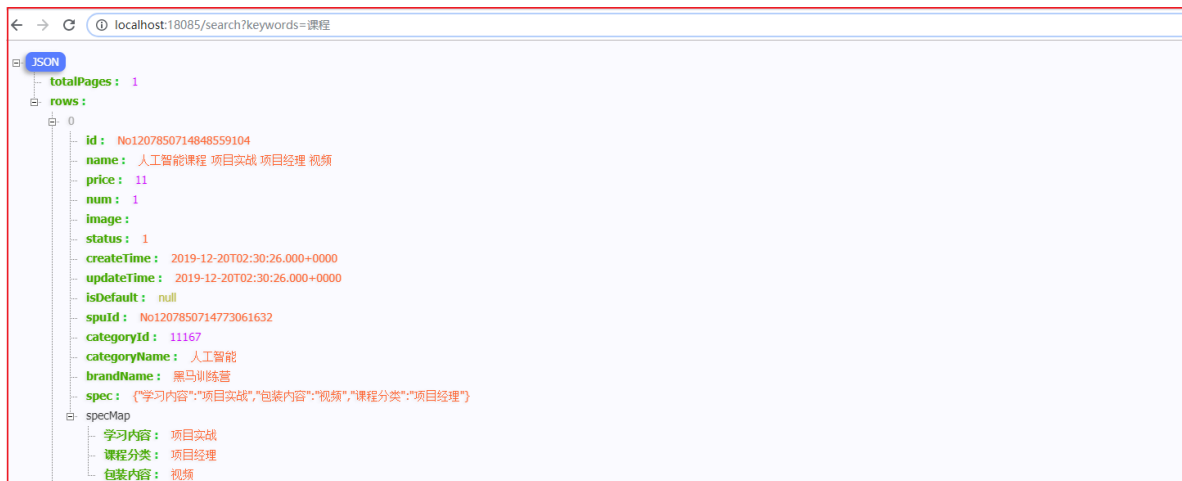
修改com.changgou.search.controller.SkuController，在控制层调用Service层即可，代码如下：


```
1  /**
2   * 调用搜索实现
3   */
4  @GetMapping
5  public Map search(@RequestParam(required = false) Map<String,String>
6    searchMap) throws Exception{
7      return skuService.search(searchMap);
8  }
```

5.3 测试

请求: `http://localhost:18085/search?keywords=课程`

效果如下:



总结

6. 分类统计

目标

- 分类统计实现

路径

- 分类统计需求分析
- 分类统计数据获取实现

讲解

6.1 分类统计分析

看下面的SQL语句，我们在执行搜索的时候，第1条SQL语句是执行搜，第2条语句是根据分类名字分组查看有多少分类，大概执行了2个步骤就可以获取数据结果以及分类统计，我们可以发现他们的搜索条件完全一样。

```
1  -- 查询所有
2  SELECT * FROM tb_sku WHERE name LIKE '%手机%';
3  -- 根据分类名字分组查询
4  SELECT category_name FROM tb_sku WHERE name LIKE '%手机%' GROUP BY
   category_name;
```

手机、数码、配件		分类显示										多选		更多														
品牌	<div>索尼 (SONY)</div>												<div>TCL</div>		<div>长虹 (CHAN...</div>		<div>飞利浦 (PHIL...</div>		<div>风行电视</div>		<div>ESR 亿色</div>		<div>ROCK</div>		<div>EXCO</div>		<div>BASEUS</div>	
	<div>LENTION 雷迅</div>												<div>PISEN 品胜</div>		<div>stiger 斯迪格</div>		<div>stiger 斯迪格</div>		<div>MOMAX 摩米士</div>		<div>Apple</div>		<div>ESR 亿色</div>		<div>ROCK</div>		<div>SAMSUNG</div>	
网络制式	GSM (移动/联通2G)		电信2G		电信3G		移动3G		联通3G		联通4G		电信3G		移动3G		联通3G		联通4G									
显示屏尺寸	4.0-4.9英寸		4.0-4.9英寸																									
摄像头像素	1200万以上		800-1199万		1200-1599万		1600万以上		无摄像头																			
价格	0-500元		500-1000元		1000-1500元		1500-2000元		2000-3000元		3000元以上																	
更多筛选项	特点		系统		手机内存		单卡双卡		其他																			
综合	销量		新品		评价		价格																					

我们每次执行搜索的时候，需要显示商品分类名称，这里要显示的分类名称其实就是符合搜索条件的所有商品的分类集合，我们可以按照上面的实现思路，使用ES根据分组名称做一次分组查询即可实现。

6.2 分类分组统计实现

修改search微服务的com.changgou.search.service.impl.SkuServiceImpl类，添加一个分类分组搜索，代码如下：

```
1  /**
2   * 搜索分类分组数据
3   */
4  public List<String> searchCategoryList(NativeSearchQueryBuilder builder){
5      /**
6       * 指定分类域，并根据分类域配置聚合查询
7       * 1:给分组查询取别名
8       * 2:指定分组查询的域
9       */
10
11     builder.addAggregation(AggregationBuilders.terms("skuCategory").field("categoryName"));
12
13     //执行搜索
14     AggregatedPage<SkuInfo> skuPage =
15     esTemplate.queryForPage(builder.build(), SkuInfo.class);
16
17     //获取所有分组查询的数据
18     Aggregations aggregations = skuPage.getAggregations();
19     //从所有数据中获取别名为skuCategory的数据
20     StringTerms terms = aggregations.get("skuCategory");
21
22     //分装List集合，将搜索结果存入到List集合中
23     List<String> categoryList = new ArrayList<String>();
24     for (StringTerms.Bucket bucket : terms.getBuckets()) {
```

```

23         categoryList.add(bucket.getKeyAsString());
24     }
25     return categoryList;
26 }

```

搜索方法中调用上面分类分组搜索，代码如下：

```

/**
 * 商品搜索实现
 * @param searchMap
 * @return
 */
@Override
public Map search(Map<String, String> searchMap) {
    //构建条件
    NativeSearchQueryBuilder builder = buildBasicQuery(searchMap);

    //搜索实现
    Map resultMap = searchList(builder);

    //分组搜索
    List<String> categoryList = searchCategoryList(builder);
    resultMap.put("categoryList", categoryList);
    return resultMap;
}

```

上图代码如下：

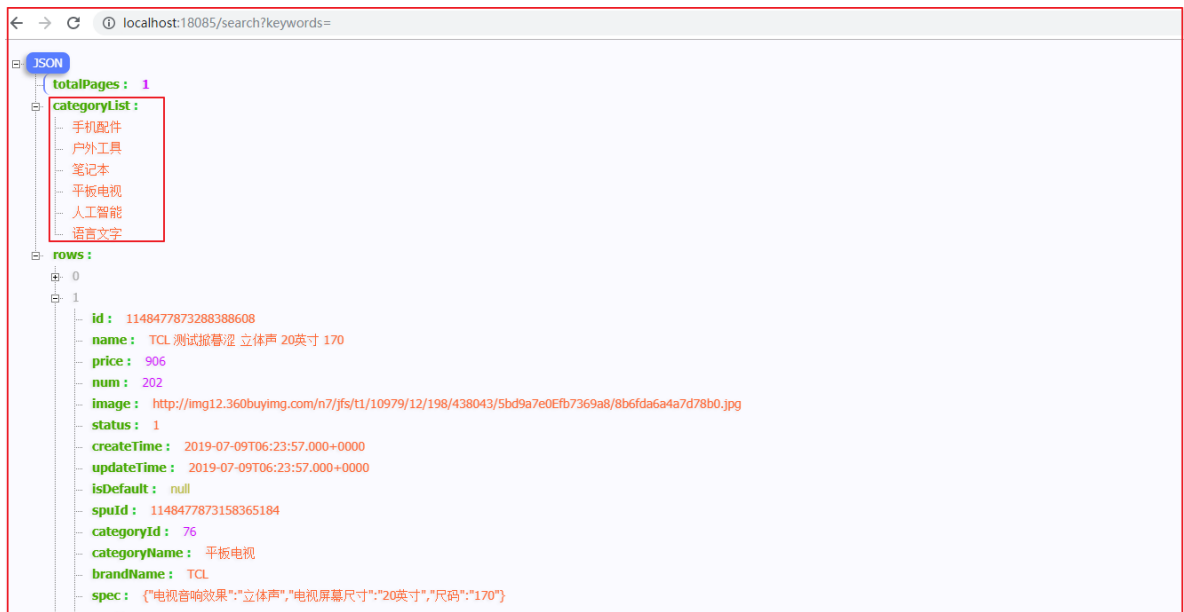
```

1 //分组搜索
2 List<String> categoryList = searchCategoryList(builder);
3 resultMap.put("categoryList", categoryList);

```

6.3 测试

请求<http://localhost:18085/search>



```

{
  "totalPages": 1,
  "categoryList": [
    "手机配件",
    "户外工具",
    "笔记本",
    "平板电视",
    "人工智能",
    "语言文字"
  ],
  "rows": [
    {
      "id": 1148477873288388608,
      "name": "TCL 测试掀幕湿 立体声 20英寸 170",
      "price": 906,
      "num": 202,
      "image": "http://img12.360buyimg.com/n7/jfs/t1/10979/12/198/438043/5bd9a7e0Efb7369a8/8b6fda6a4a7d78b0.jpg",
      "status": 1,
      "createTime": "2019-07-09T06:23:57.000+0000",
      "updateTime": "2019-07-09T06:23:57.000+0000",
      "isDefault": null,
      "spuId": 1148477873158365184,
      "categoryId": 76,
      "categoryName": "平板电视",
      "brandName": "TCL",
      "spec": "{\"电视音响效果\":\"立体声\",\"电视屏幕尺寸\":\"20英寸\",\"尺码\":\"170\"}"
    }
  ]
}

```

思考：

- 1 | 分类可以通过查询数据统计实现，那么规格和品牌呢？该如何实现呢？同学们思考一下，我们明天来实现。

总结

课后要求:

1.在kibana中尝试一下DSL语句,将效果全部实现一遍(DSL语句不用记住)

2.完成ES的数据导入:知道数据导入的一个流程:先通过状态查询出所有正常的商品列表,然后将商品保存到es中去

3.实现关键字搜索,通过关键字能正常的查询到数据:理解关键字搜索的实现的流程

4.完成分类的集合查询,查询出符合条件的所有的类别的名字,并且出重

5.预习第六天的内容