常用数据类型：<https://www.cnblogs.com/chy18883701161/p/12723658.html>

参考博客：<https://www.jianshu.com/p/1fbfde2aefa5>

elastisearch中的dynamic的用法：<https://www.cnblogs.com/Pythonzrq/p/11982645.html>

springboot集成：

springdata官网

<https://www.cnblogs.com/cnbai/p/12827868.html>

搜索：

有两个操作bean，搜索这两个bean怎么操作数据。

|  |
| --- |
| @Autowired **private** ArticleRepository **articleRepository**;  @Autowired **private** ElasticsearchTemplate **elasticsearchTemplate**; |

关键词：

pretty=true表示格式化输出

### 后台启动方式

es: ./elasticsearch -d

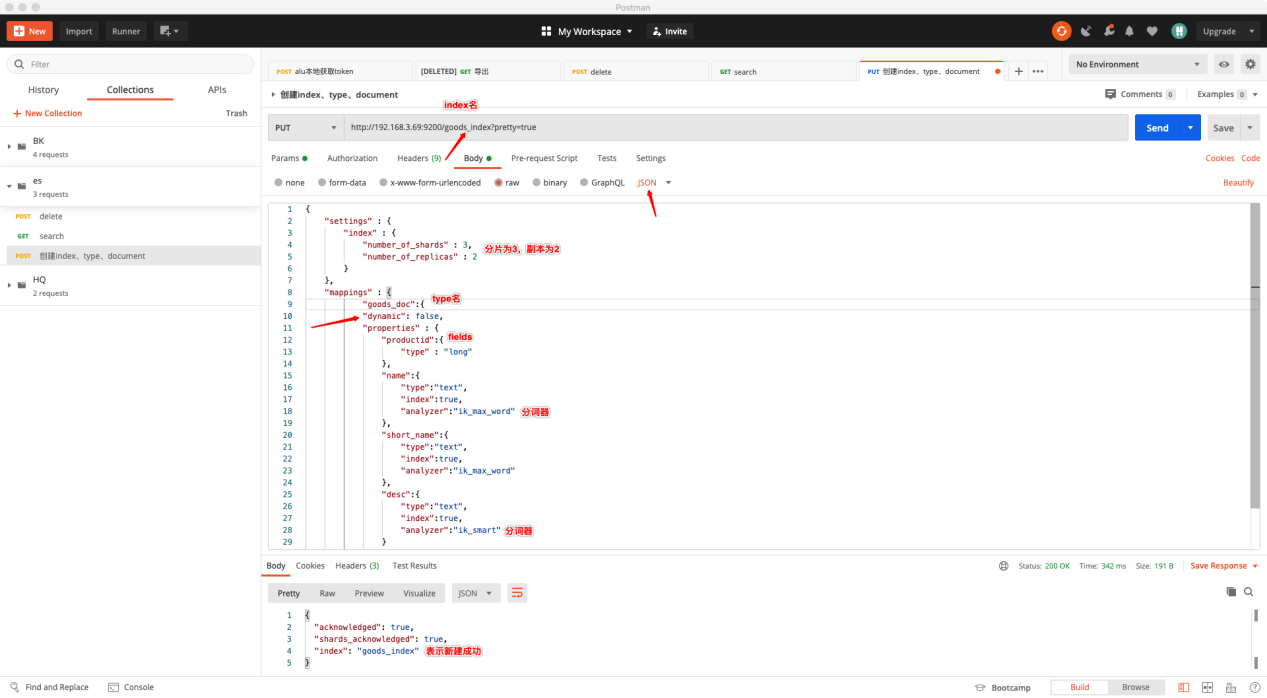
kibana: nohup /usr/local/kibana/bin/kibana &

## 与DB对比

|  |  |
| --- | --- |
| es | db |
| index | database |
| type | table |
| document | row |
| field | colums |

## 创建index、type、field

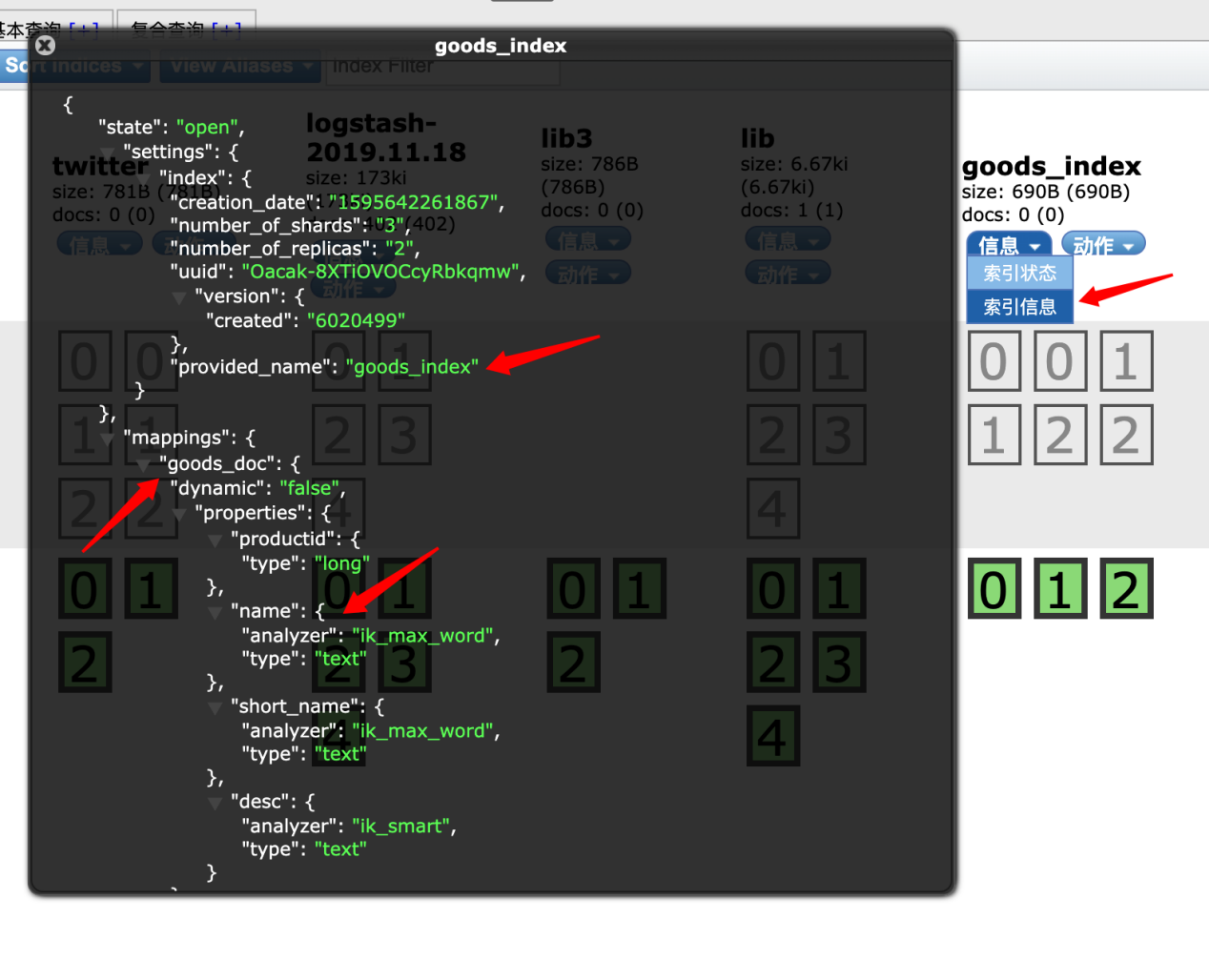
ElasticSearch6.0以上的版本，不支持在同一个Index中拥有不同的Type



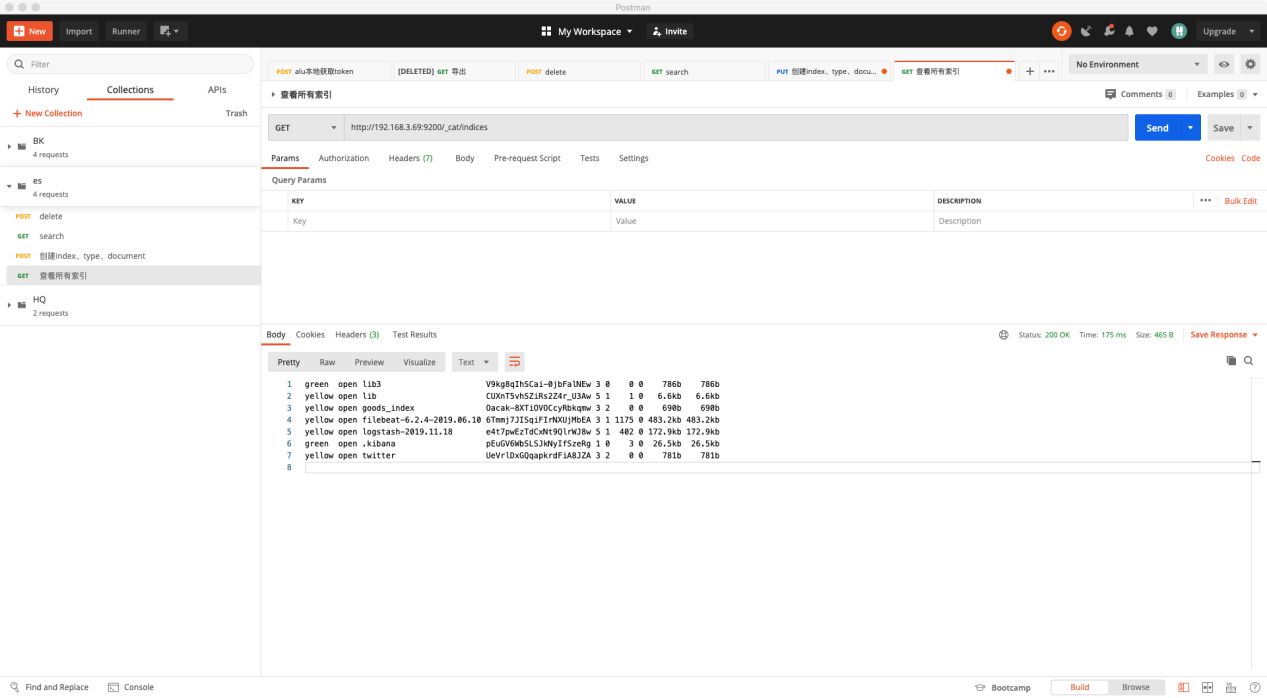


## 通过head查看index信息

有点类似查看表信息一样，可以查看数据库名、表名、字段名

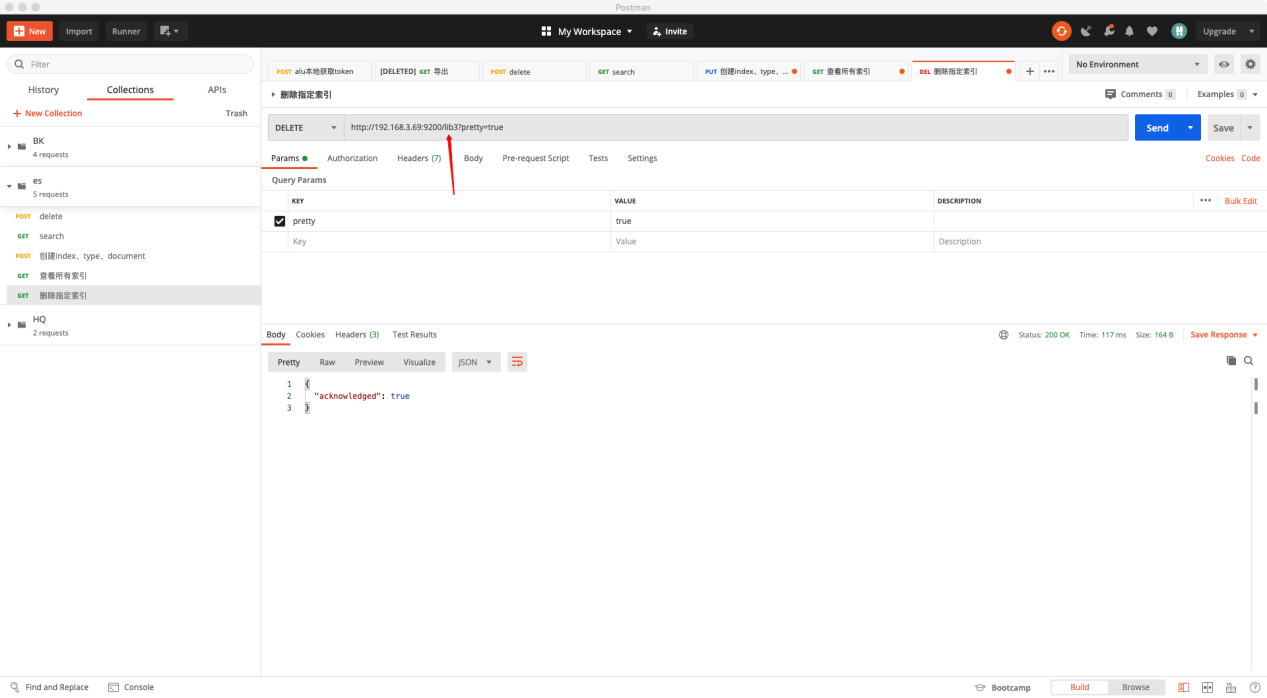


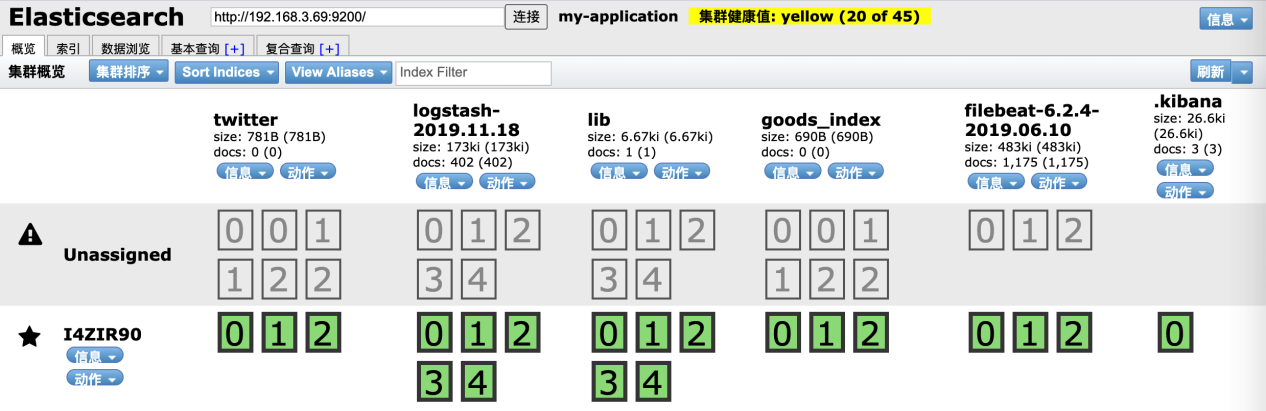
## 查看所有的索引index



## 删除指定索引index

根据索引名删除

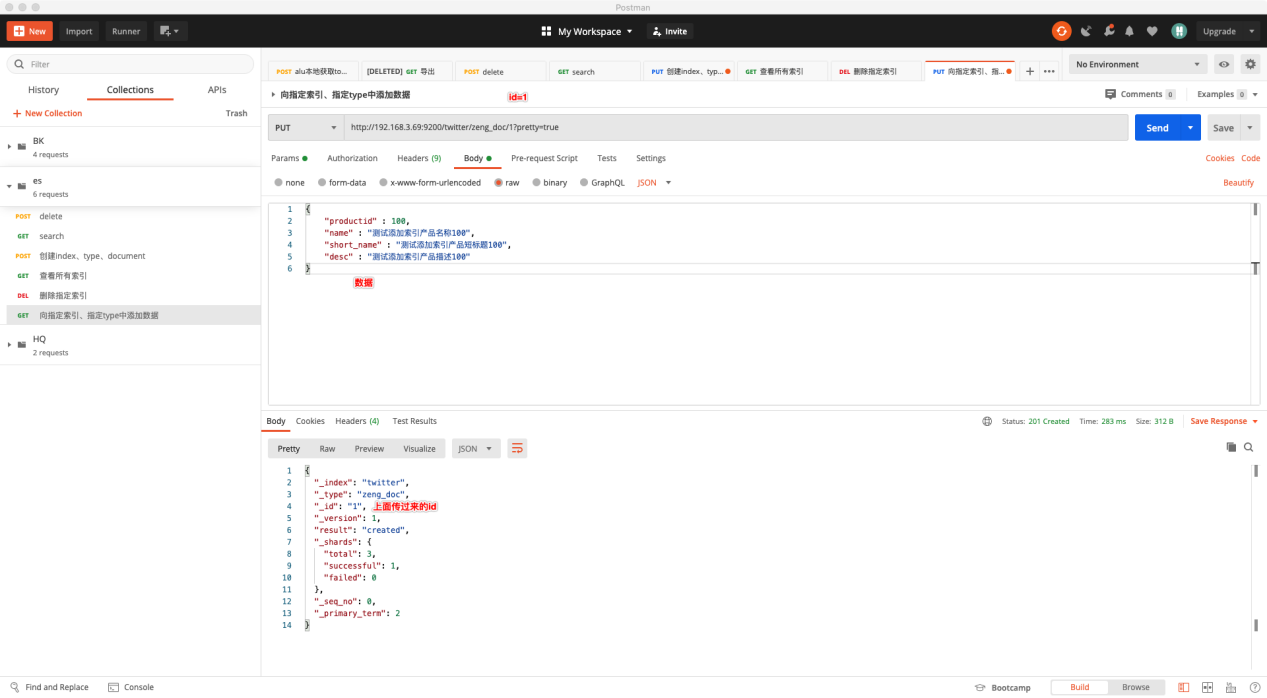




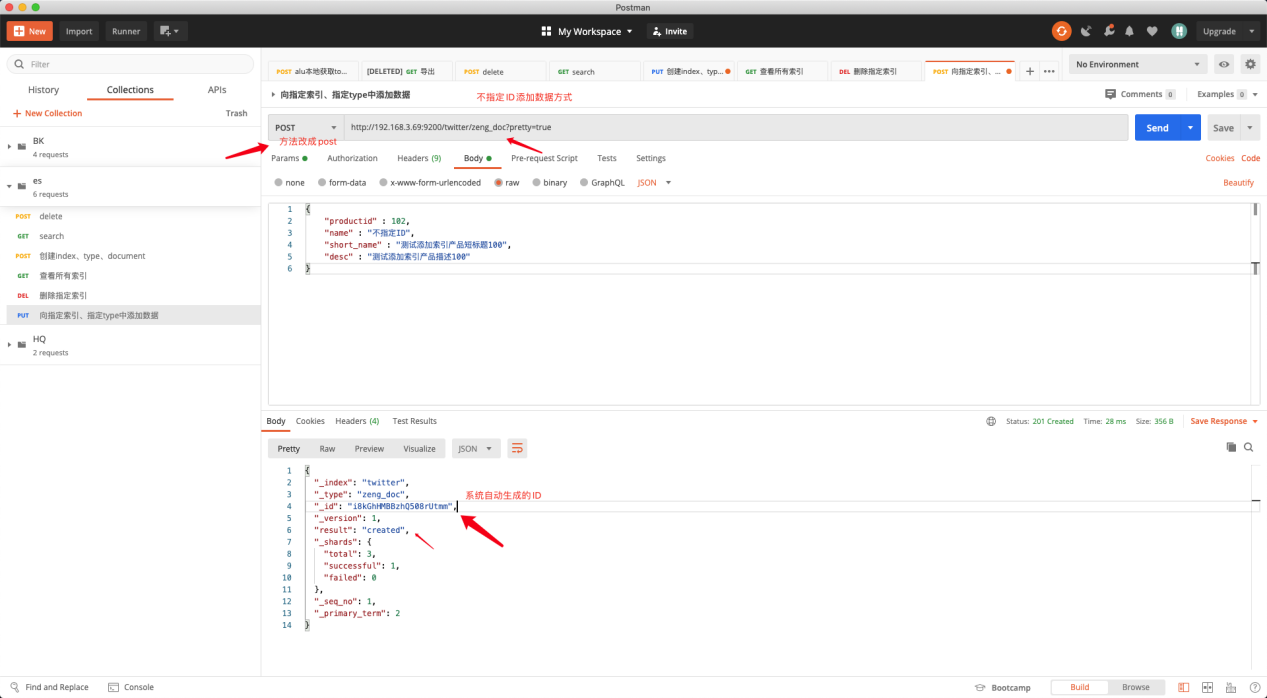
名为lib3的索引被删除

## 添加数据

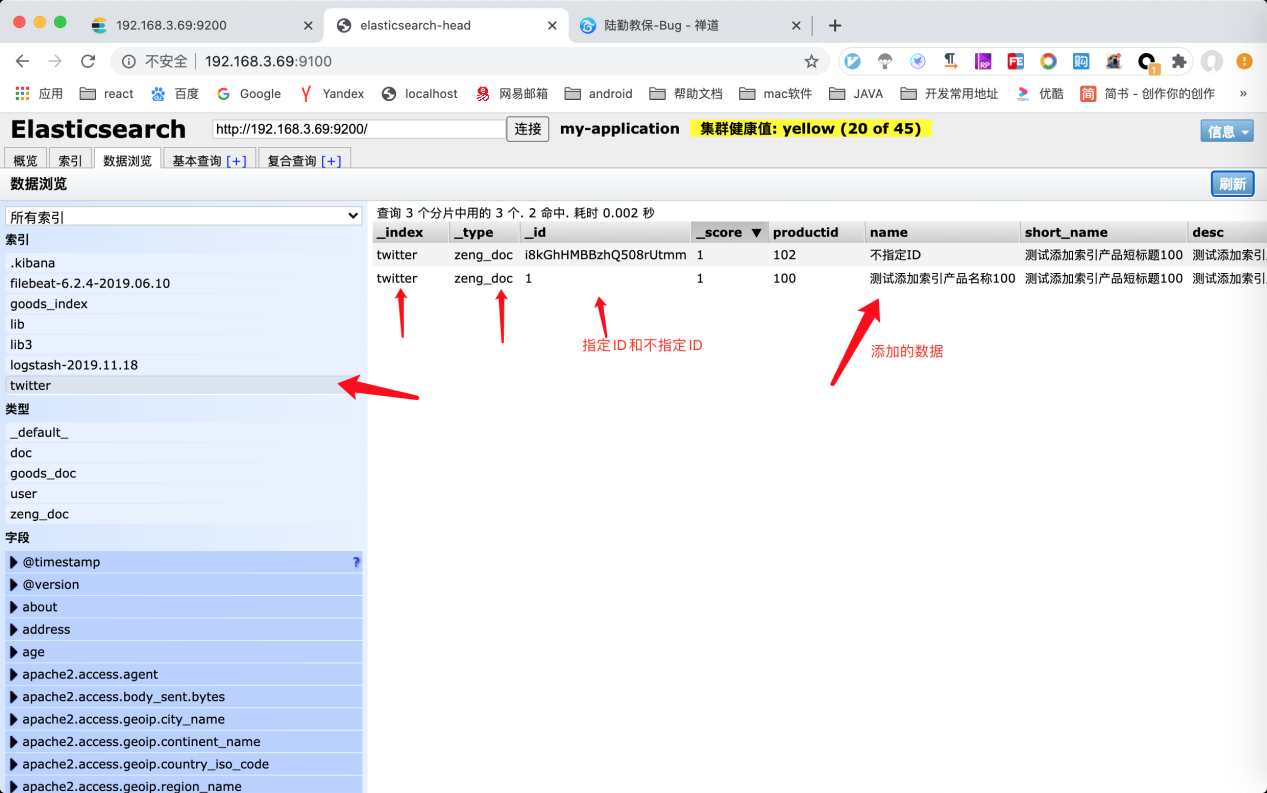
指定ID：



不指定ID：



通过head查看数据：

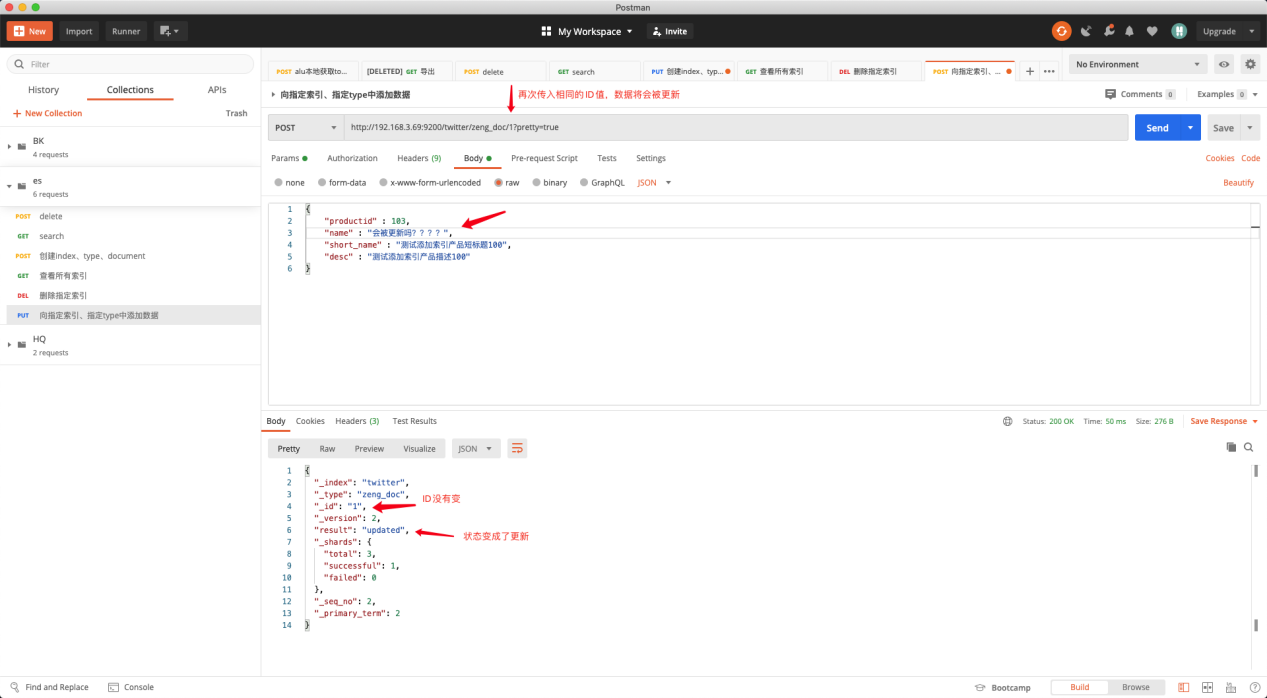


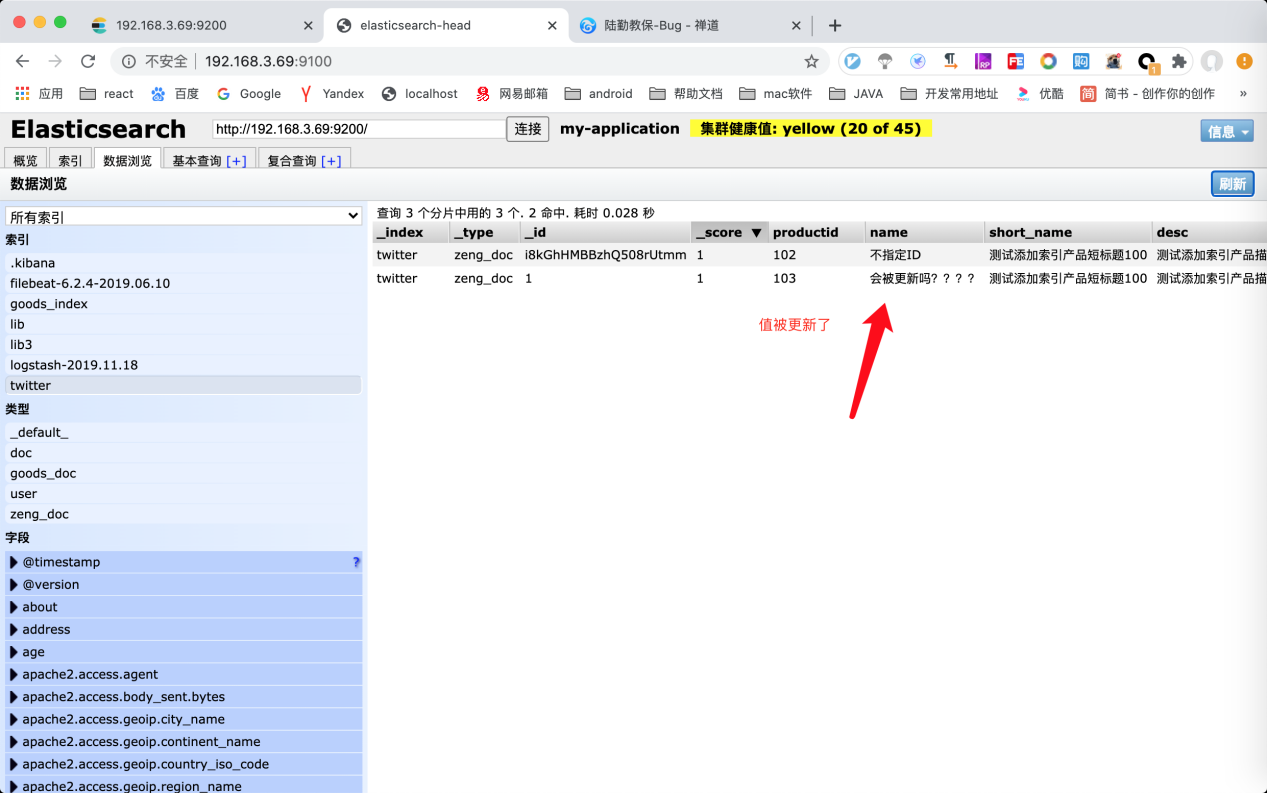
## 添加数据，两次的ID值相同，将会被更新

http://localhost:9200/twitter/\_doc/1?op\_type=create&pretty=true

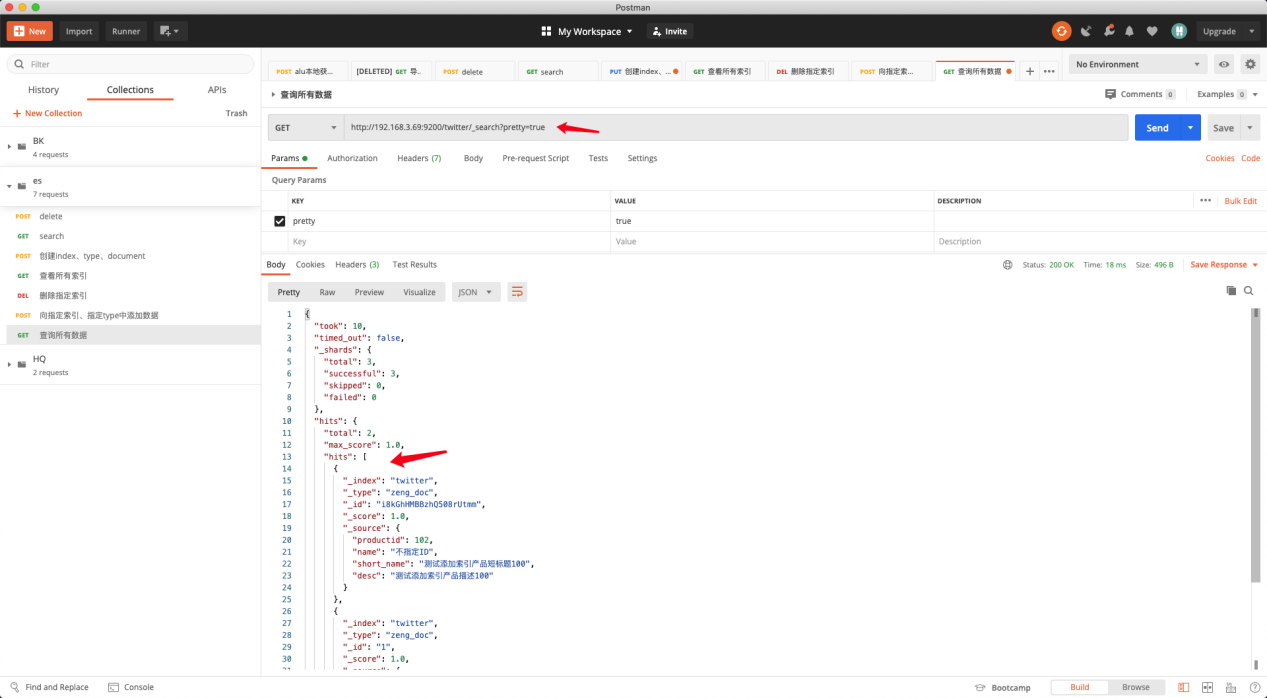
指定id为1，还可以加上参数op\_type=create，这样在创建重复id时会报错导致创建失败，

这样可以避免被更新【因为更新就调用更新的接口，不要混用】





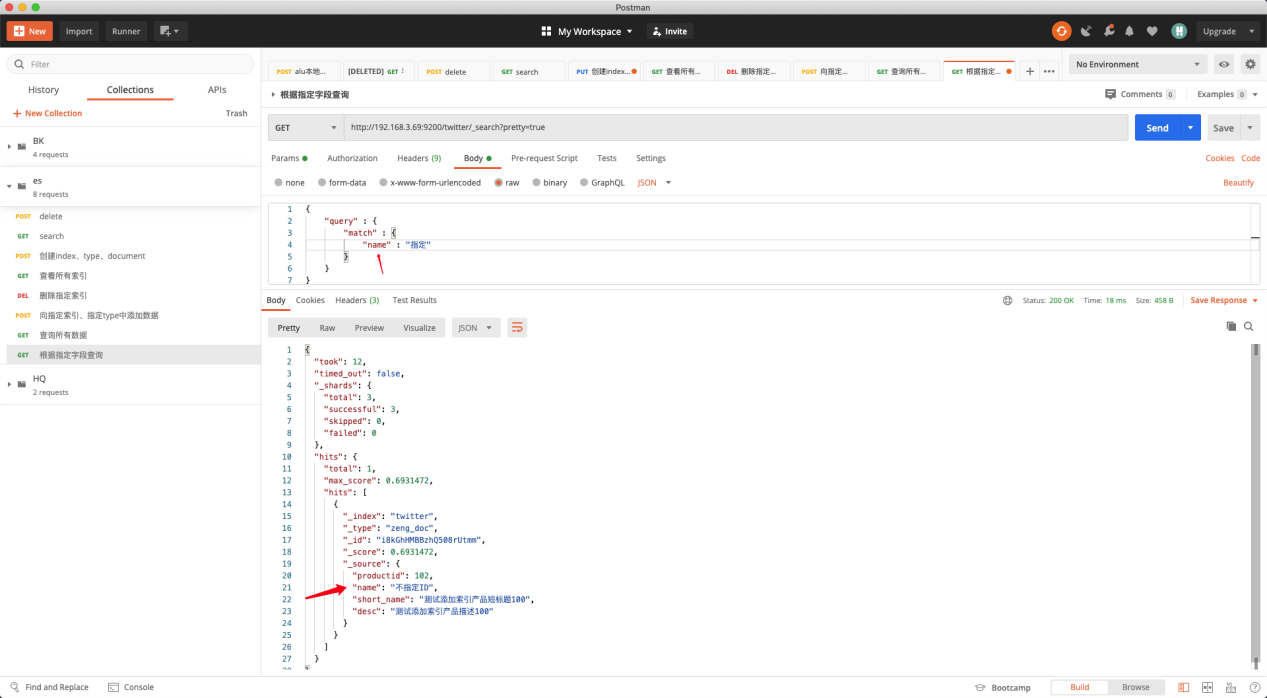
## 查询所有数据



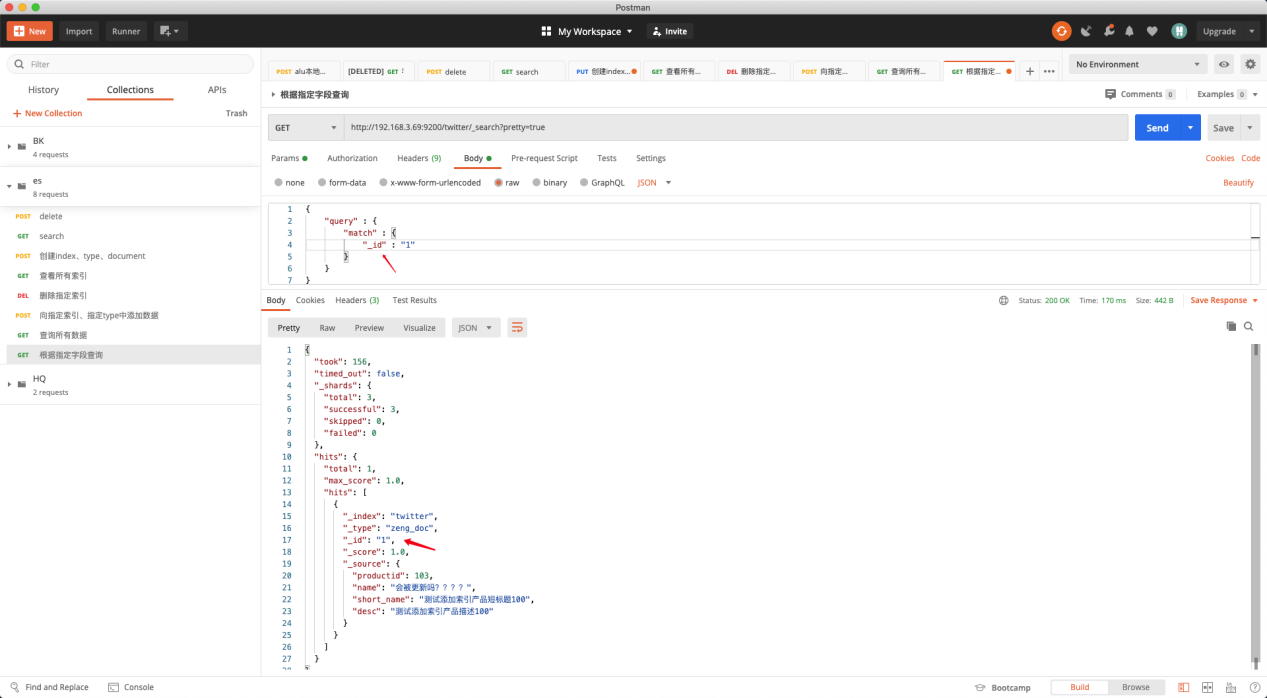
## 根据指定字段查询

条件查询会涉及到精确词查询、匹配查询、多条件查询、聚合查询四种，分别为"term"、"match"、"multi\_match"、"multi\_match"。

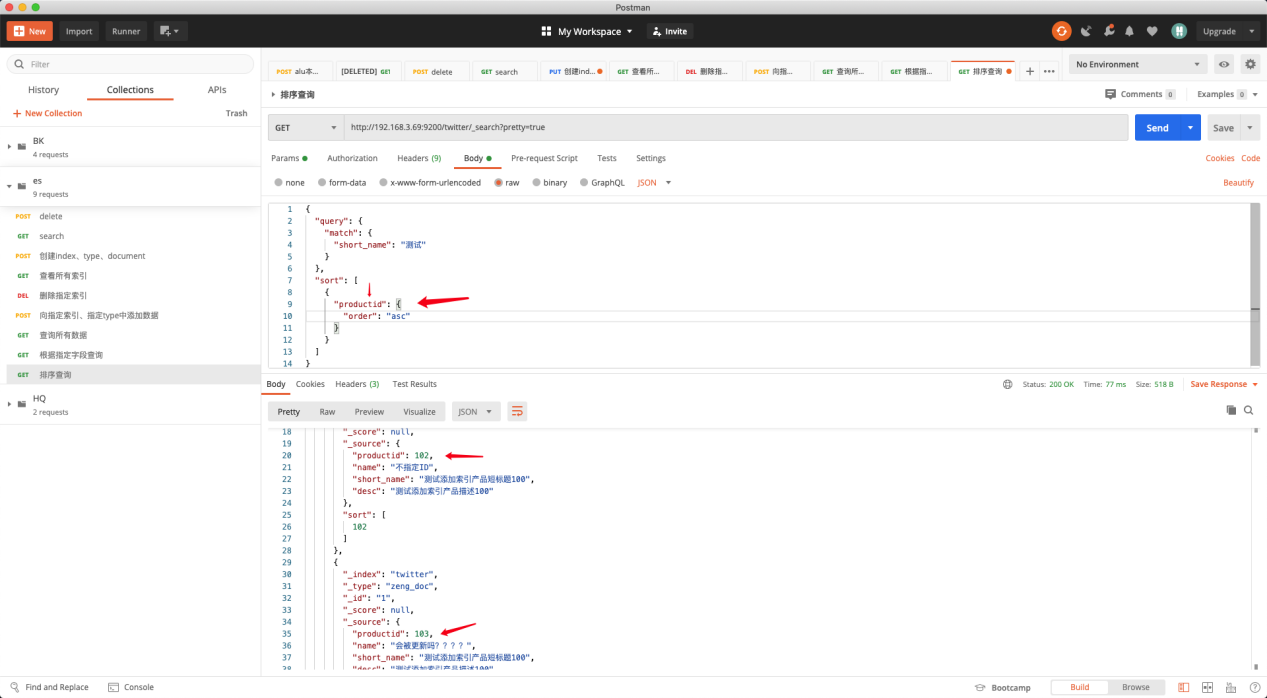
根据字段field查询：



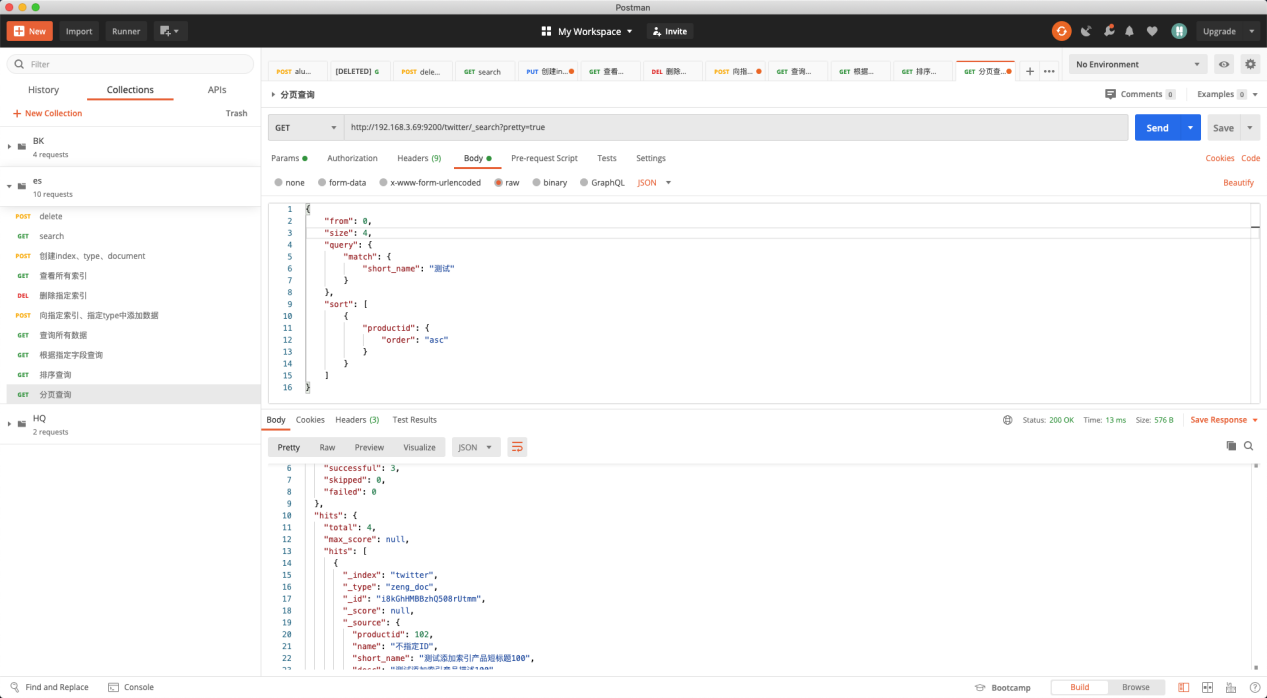
根据ID查询： 为 \_id



## 排序查询



## 分页查询

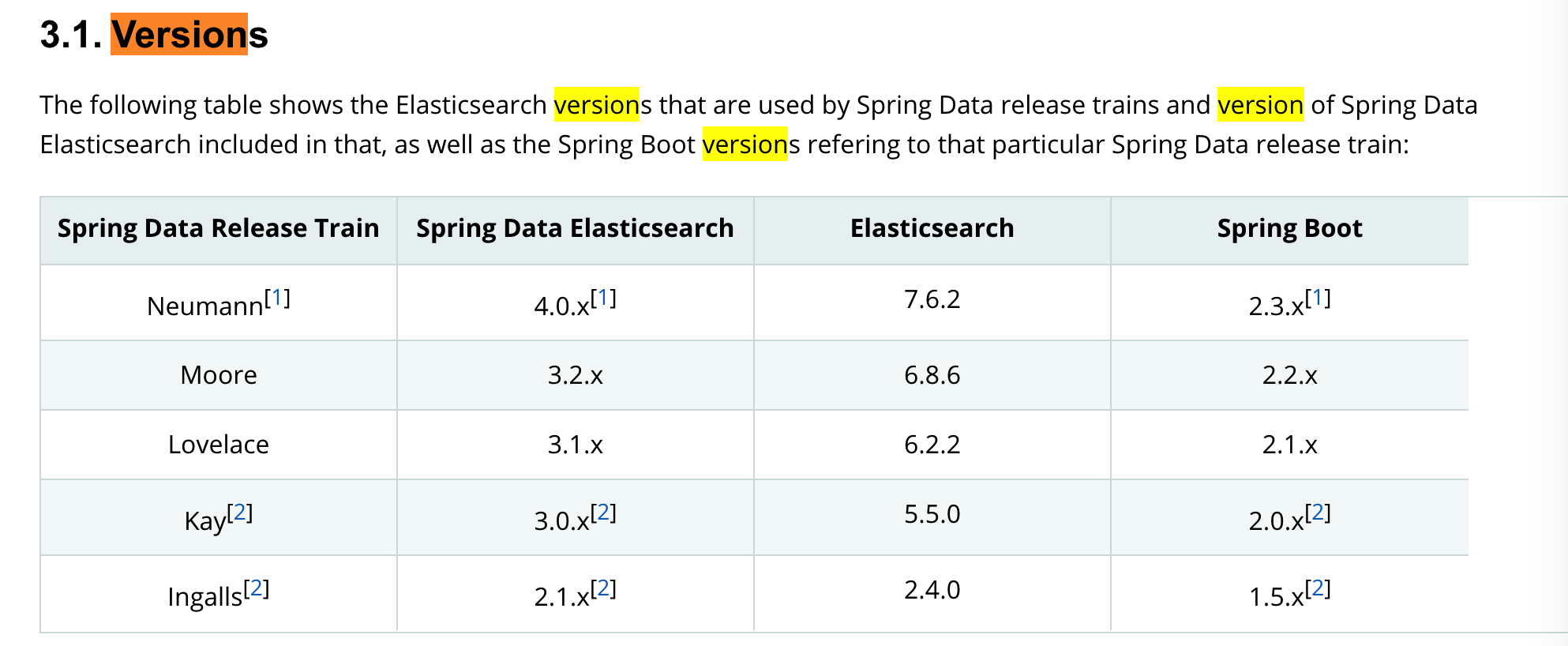


|  |
| --- |
| {  "took": 5,  "timed\_out": false,  "\_shards": {  "total": 3,  "successful": 3,  "skipped": 0,  "failed": 0  },  "hits": {  "total": 4,  "max\_score": null,  "hits": [  {  "\_index": "twitter",  "\_type": "zeng\_doc",  "\_id": "i8kGhHMBBzhQ508rUtmm",  "\_score": null,  "\_source": {  "productid": 102,  "name": "不指定ID",  "short\_name": "测试添加索引产品短标题100",  "desc": "测试添加索引产品描述100"  },  "sort": [  102  ]  },  {  "\_index": "twitter",  "\_type": "zeng\_doc",  "\_id": "1",  "\_score": null,  "\_source": {  "productid": 103,  "name": "会被更新吗？？？？",  "short\_name": "测试添加索引产品短标题100",  "desc": "测试添加索引产品描述100"  },  "sort": [  103  ]  },  {  "\_index": "twitter",  "\_type": "zeng\_doc",  "\_id": "2",  "\_score": null,  "\_source": {  "productid": 1102,  "name": "会被更新吗1102",  "short\_name": "测试添加索引产品短标题1102",  "desc": "测试添加索引产品描述1102"  },  "sort": [  1102  ]  },  {  "\_index": "twitter",  "\_type": "zeng\_doc",  "\_id": "3",  "\_score": null,  "\_source": {  "productid": 1103,  "name": "会被更新吗1103",  "short\_name": "测试添加索引产品短标题1103",  "desc": "测试添加索引产品描述1103"  },  "sort": [  1103  ]  }  ]  }  } |

结果分页成功

# springboot集成es

## 版本参考



## pom导入

|  |
| --- |
| <**parent**>  <**groupId**>org.springframework.boot</**groupId**>  <**artifactId**>spring-boot-starter-parent</**artifactId**>  <**version**>2.1.3.RELEASE</**version**>  <**relativePath**/> *<!-- lookup parent from repository -->* </**parent**> |
| <**dependency**>  <**groupId**>org.springframework.data</**groupId**>  <**artifactId**>spring-data-elasticsearch</**artifactId**>  <**version**>3.2.1.RELEASE</**version**> </**dependency**> |

## 配置文件

|  |
| --- |
| *#Elasticsearch配置* **elasticsearch**:  **host**: 192.168.3.70  **port**: 9300  **clustername**: my-application  **search**:  **pool**:  **size**: 5 |

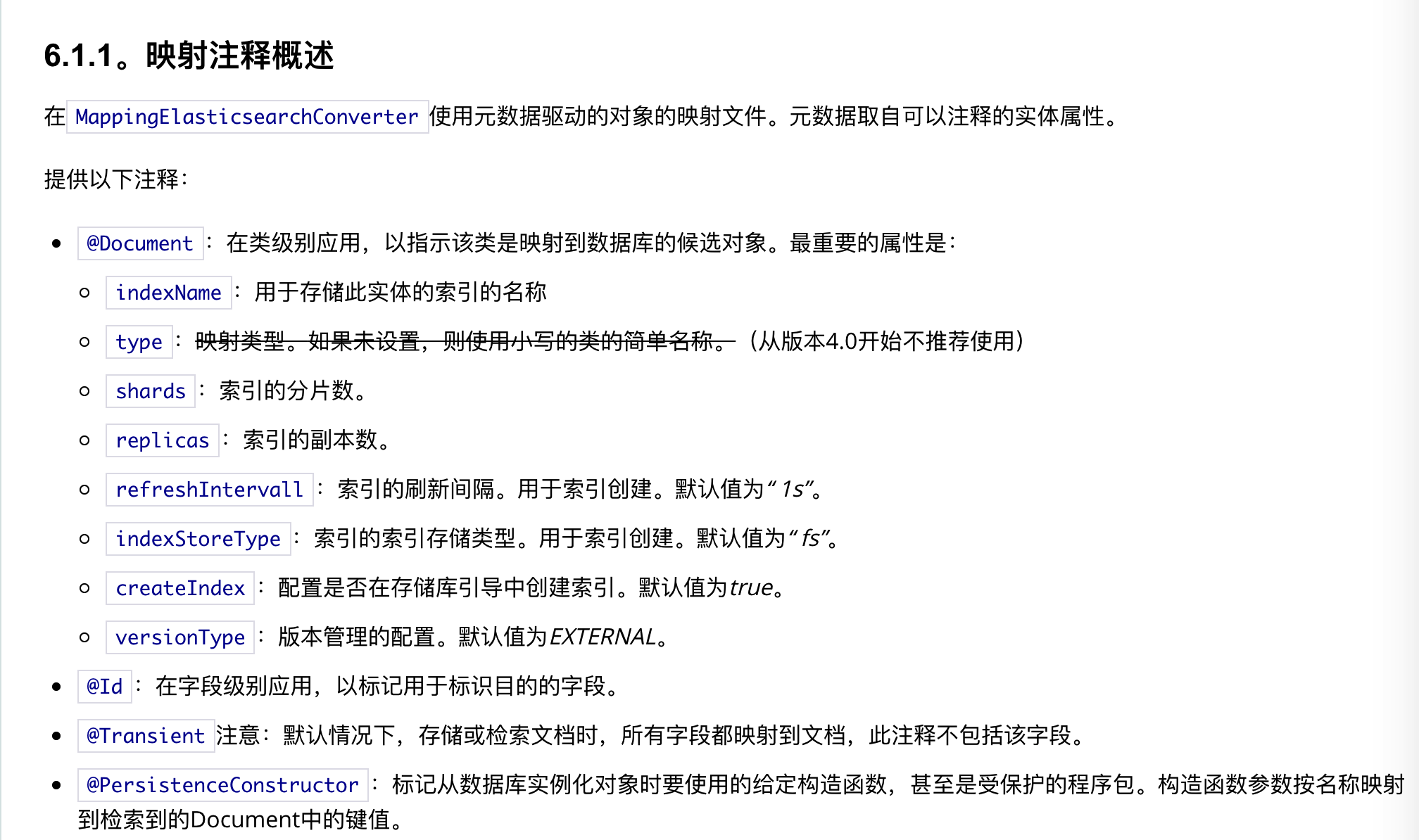
## 创建ElasticsearchTemplate实例

|  |
| --- |
| **package** com.zengziqiang.config;  **import** lombok.extern.log4j.Log4j2; **import** org.elasticsearch.client.Client; **import** org.elasticsearch.common.settings.Settings; **import** org.elasticsearch.common.transport.TransportAddress; **import** org.elasticsearch.transport.client.PreBuiltTransportClient; **import** org.springframework.beans.factory.annotation.Value; **import** org.springframework.context.annotation.Bean; **import** org.springframework.context.annotation.Configuration; **import** org.springframework.data.elasticsearch.core.ElasticsearchOperations; **import** org.springframework.data.elasticsearch.core.ElasticsearchTemplate; **import** org.springframework.data.elasticsearch.repository.config.EnableElasticsearchRepositories;  **import** java.net.InetAddress;  */\*\*  \** ***@author*** *iszengziqiang@163.com  \** ***@Version:*** *1.0  \** ***@desc*** *//****todo*** *\*/* @Log4j2 @Configuration @EnableElasticsearchRepositories(basePackages = **"com.zengziqiang.service"**) **public class** ElasticsearchConfig {  @Value(**"${elasticsearch.host}"**)  **private** String **esHost**;   @Value(**"${elasticsearch.port}"**)  **private int esPort**;   @Value(**"${elasticsearch.clustername}"**)  **private** String **esClusterName**;   @Value(**"${elasticsearch.search.pool.size}"**)  **private** Integer **threadPoolSearchSize**;    @Bean  **public** Client client() **throws** Exception {  Settings esSettings = Settings.*builder*()  .put(**"cluster.name"**, **esClusterName**)  *//增加嗅探机制，找到ES集群,非集群置为false* .put(**"client.transport.sniff"**, **true**)  *//增加线程池个数* .put(**"thread\_pool.search.size"**, **threadPoolSearchSize**)  .build();  **return new** PreBuiltTransportClient(esSettings)  .addTransportAddress(**new** TransportAddress(InetAddress.*getByName*(**esHost**), **esPort**));  }   @Bean  **public** ElasticsearchOperations elasticsearchTemplateCustom() **throws** Exception {  ElasticsearchTemplate elasticsearchTemplate;  **try** {  elasticsearchTemplate = **new** ElasticsearchTemplate(client());  System.***out***.println(**"初始化ElasticsearchTemplate成功"**);  **return** elasticsearchTemplate;  } **catch** (Exception e) {  ***log***.error(**"初始化ElasticsearchTemplate失败"**);  **return new** ElasticsearchTemplate(client());  }  } } |

创建实体类，及索引和type

|  |
| --- |
| **package** com.zengziqiang.dto;  **import** lombok.Data; **import** lombok.ToString; **import** org.springframework.data.elasticsearch.annotations.DateFormat; **import** org.springframework.data.elasticsearch.annotations.Document; **import** org.springframework.data.elasticsearch.annotations.Field; **import** org.springframework.data.elasticsearch.annotations.FieldType;  */\*\*  \** ***@author*** *iszengziqiang@163.com  \** ***@date*** *2020/7/25 16:50  \** ***@Version:*** *1.0  \** ***@desc*** *//****todo*** *\*/* @Data @ToString @Document(indexName = **"blog"**, type = **"article"**, shards = 3) **public class** Article {   */\*\*  \* 主键ID  \*/* @Field(type = FieldType.***Keyword***)  **private** String **id**;   */\*\*  \* 文章标题  \*/* @Field(type = FieldType.***Text***, analyzer = **"ik\_max\_word"**, searchAnalyzer = **"ik\_max\_word"**)  **private** String **title**;   */\*\*  \* 文章内容  \*/* @Field(type = FieldType.***Text***, analyzer = **"ik\_max\_word"**, searchAnalyzer = **"ik\_max\_word"**)  **private** String **content**;   */\*\*  \* 创建时间  \*/* @Field(type = FieldType.***Date***, pattern = **"yyyy-MM-dd HH:mm:ss"**, format = DateFormat.***custom***)  **private** String **createTime**;   @Field(type = FieldType.***Integer***)  **private** Integer **order**; } |

## 注解映射介绍





## 地理位置坐标



## 集合使用



## map使用



## 详细参考官方文档

[https://docs.spring.io/spring-data/elasticsearch/docs/4.0.2.RELEASE/reference/html/#repositories.query-methods](https://docs.spring.io/spring-data/elasticsearch/docs/4.0.2.RELEASE/reference/html/" \l "repositories.query-methods)

## service定义

用于对实体类的curd

|  |
| --- |
| **package** com.zengziqiang.service;  **import** com.zengziqiang.dto.Article; **import** org.springframework.data.elasticsearch.repository.ElasticsearchRepository; **import** org.springframework.stereotype.Repository;  **import** java.util.List;  */\*\*  \** ***@author*** *iszengziqiang@163.com  \** ***@date*** *2020/7/25 16:50  \** ***@Version:*** *1.0  \** ***@desc*** *//****todo*** *\*/* @Repository **public interface** ArticleRepository **extends** ElasticsearchRepository<Article, String> {  List<Article> findByOrderIn(  List<Integer> ids  ); } |

## 接口测试

|  |
| --- |
| **package** com.zengziqiang.controller;  **import** com.alibaba.fastjson.JSONObject; **import** com.zengziqiang.dto.Article; **import** com.zengziqiang.service.ArticleRepository; **import** org.elasticsearch.index.query.MatchPhraseQueryBuilder; **import** org.elasticsearch.index.query.MatchQueryBuilder; **import** org.elasticsearch.index.query.QueryBuilders; **import** org.springframework.beans.factory.annotation.Autowired; **import** org.springframework.data.domain.PageRequest; **import** org.springframework.data.domain.Pageable; **import** org.springframework.data.domain.Sort; **import** org.springframework.data.elasticsearch.core.ElasticsearchTemplate; **import** org.springframework.data.elasticsearch.core.query.NativeSearchQueryBuilder; **import** org.springframework.data.elasticsearch.core.query.SearchQuery; **import** org.springframework.data.geo.Point; **import** org.springframework.web.bind.annotation.GetMapping; **import** org.springframework.web.bind.annotation.PathVariable; **import** org.springframework.web.bind.annotation.RestController;  **import** java.text.SimpleDateFormat; **import** java.util.\*;  */\*\*  \** ***@author*** *iszengziqiang@163.com  \** ***@date*** *2020/8/4 17:05  \** ***@Version:*** *1.0  \** ***@desc*** *//****todo*** *\*/* @RestController **public class** EsController {   @Autowired  **private** ArticleRepository **articleRepository**;   @Autowired  **private** ElasticsearchTemplate **elasticsearchTemplate**;   */\*\*  \* 新增数据  \*/* @GetMapping(value = **"/add"**)  **public void** saveArticle() {  **int** i = **new** Random().nextInt(1000);  String title = **"谷歌是如何做Code Review的"** + **new** Random().nextInt(1000);  String content = **"Code Review的主要目的是始终保证随着时间的推移，谷歌代码越来越健康，所有Code Review的工具和流程也是针对于此设计的。"** + **new** Random().nextInt(1000);  Article article = *createArticle*(title, content);  **articleRepository**.save(article);  *// 44db8516-1565-4fea-b19b-cc550b47f85d* System.***out***.println(article.getId());   title = **"iOS 13大更新曝光：苹果手机或要调整位置权限"** + **new** Random().nextInt(1000);  content = **"据外媒报道称，苹果正在对iOS 13系统进行调整，主要是修复之前出现的Bug，并且还打算iOS 13的位置权限设置进行调整，因为这个细节，他们正在接受反垄断调查。"** + **new** Random().nextInt(1000);  Article article2 = *createArticle*(title, content);  **articleRepository**.save(article2);  *// 73a50b15-e1c9-4afd-b675-fba2ffdc6e3e* System.***out***.println(article2.getId());   title = **"日媒：中国手机为何在东南亚受欢迎？"** + **new** Random().nextInt(1000);  content = **"人类可能地球上是最不珍惜粮食的物种之一，根据全球农业与食品营养问题委员会的统计数据，全球每年食物浪费总量达到 13 亿吨，其中超过一半的水果和蔬菜被浪费。"** + **new** Random().nextInt(1000);  Article article3 = *createArticle*(title, content);  **articleRepository**.save(article3);  *// feec732b-bf04-4c23-97fa-3d28ffd23a83* System.***out***.println(article3.getId());  }    **private static** Article createArticle(String title, String content) {  *//UUID模拟ID* UUID uuid = UUID.*randomUUID*();  String id = uuid.toString();  *//创建Article* Article article = **new** Article();  article.setId(id);  article.setTitle(title);  article.setContent(content);  article.setCreateTime(**new** SimpleDateFormat(**"yyyy-MM-dd HH:mm:ss"**).format(**new** Date()));  article.setOrder(**new** Random().nextInt(1000));  **return** article;  }   @GetMapping(value = **"/id/{id}"**)  **public** String findById(  @PathVariable(name = **"id"**) String id  ) {  System.***out***.println(**"接收的参数："** + id);  **long** start = System.*currentTimeMillis*();  Optional<Article> byId = **articleRepository**.findById(id);  **long** end = System.*currentTimeMillis*();  System.***out***.println(end - start);  **if** (byId.isPresent()) {  *//有值* Article article = byId.get();  **return** JSONObject.*toJSONString*(article);  } **else** {  *//无值* **return "数据为空"**;  }  }   */\*\*  \* 根据关键字在文章title中进行搜索  \* 分词  \*/* @GetMapping(value = **"/code/{code}"**)  **public void** findArticleByTitle(  @PathVariable(name = **"code"**) String code  ) {  String titleKeyWord = code;*//谷歌中国  //matchQuery 会对关键字分词后进行搜索:谷歌中国---> 谷歌 中国* MatchQueryBuilder matchQueryBuilder = QueryBuilders.*matchQuery*(**"title"**, titleKeyWord);  QueryBuilders.*commonTermsQuery*(**"title"**, code);  Iterable<Article> search = **articleRepository**.search(matchQueryBuilder);  Iterator<Article> iterator = search.iterator();  **while** (iterator.hasNext()) {  Article next = iterator.next();  */\*  \* Article(id=44db8516-1565-4fea-b19b-cc550b47f85d,  \* title=谷歌是如何做Code Review的,  \* content=Code Review的主要目的是始终保证随着时间的推移，谷歌代码越来越健康，所有Code Review的工具和流程也是针对于此设计的。,  \* createTime=2020-04-09 14:01:00)  \* Article(id=f8df737a-fc46-4471-8b80-e2756ca8c85c,  \* title=日媒：中国手机为何在东南亚受欢迎？,  \* content=人类可能地球上是最不珍惜粮食的物种之一，根据全球农业与食品营养问题委员会的统计数据，全球每年食物浪费总量达到 13 亿吨，其中超过一半的水果和蔬菜被浪费。,  \* createTime=2020-04-09 14:02:55)  \* ... ...  \*/* System.***out***.println(next);  }   }   @GetMapping(value = **"/update/{id}"**)  **public void** updateById(  @PathVariable(name = **"id"**) String id  ) {  Article article = **new** Article();  article.setId(id);  article.setCreateTime(**new** SimpleDateFormat(**"yyyy-MM-dd HH:mm:ss"**).format(**new** Date()));  Article save = **articleRepository**.save(article);  System.***out***.println(save);  }   */\*\*  \* 根据关键字在文章title中进行搜索  \* 全匹配  \*/* @GetMapping(value = **"/code2/{code}"**)  **void** findArticleByTitle2(  @PathVariable(name = **"code"**) String code  ) {  String titleKeyWord = code;  *//matchPhraseQueryBuilder 对关键字不进行分词，全匹配查询* MatchPhraseQueryBuilder matchPhraseQueryBuilder = QueryBuilders.*matchPhraseQuery*(**"title"**, titleKeyWord);  Iterable<Article> search = **articleRepository**.search(matchPhraseQueryBuilder);  Iterator<Article> iterator = search.iterator();   **while** (iterator.hasNext()) {  Article next = iterator.next();  */\*  \* Article(id=cf7037d7-1b0e-4568-be94-221371903651,  \* title=谷歌是如何做Code Review的,  \* content=Code Review的主要目的是始终保证随着时间的推移，谷歌代码越来越健康，所有Code Review的工具和流程也是针对于此设计的。,  \* createTime=2020-04-09 11:00:00)  \* Article(id=1fb7c348-3fa2-4c05-aaf3-c677338afd31,  \* title=谷歌是如何做Code Review的,  \* content=Code Review的主要目的是始终保证随着时间的推移，谷歌代码越来越健康，所有Code Review的工具和流程也是针对于此设计的。,  \* createTime=2020-04-09 14:01:00)  \* ... ...  \*/* System.***out***.println(next);  }  }   */\*\*  \* 根据关键字在文章title中进行搜索  \* 分页+排序  \* es应尽量避免深层分页  \*/* @GetMapping(value = **"/order/{code}"**)  **public void** findArticleByTitlePage(  @PathVariable(name = **"code"**) String code  ) {  Sort createTime = Sort.*by*(**"createTime"**).descending();  Pageable pageable = PageRequest.*of*(0, 10, createTime);   String titleKeyWord = code;  *//matchQuery 会对关键字分词后进行搜索:谷歌中国---> 谷歌 中国* MatchQueryBuilder matchQueryBuilder = QueryBuilders.*matchQuery*(**"title"**, titleKeyWord);  QueryBuilders.*commonTermsQuery*(**"title"**, code);  Iterable<Article> search = **articleRepository**.search(matchQueryBuilder, pageable);  Iterator<Article> iterator = search.iterator();  **while** (iterator.hasNext()) {  Article next = iterator.next();  */\*  \* Article(id=44db8516-1565-4fea-b19b-cc550b47f85d,  \* title=谷歌是如何做Code Review的,  \* content=Code Review的主要目的是始终保证随着时间的推移，谷歌代码越来越健康，所有Code Review的工具和流程也是针对于此设计的。,  \* createTime=2020-04-09 14:01:00)  \* ... ...  \*/* Point point = **new** Point(1, 2);  System.***out***.println(next);  }  }   @GetMapping(value = **"/in"**)  **public void** findByOrderIn(   ) {  List<Integer> ids = **new** ArrayList<>();  ids.add(88);  ids.add(500);  List<Article> byOrderIn = **articleRepository**.findByOrderIn(ids);  System.***out***.println(byOrderIn);  SearchQuery searchQuery = **new** NativeSearchQueryBuilder().withQuery(QueryBuilders.*queryStringQuery*(**"中国"**)).withPageable(PageRequest.*of*(0, 100)).build();   List<Article> articles = **elasticsearchTemplate**.queryForList(searchQuery, Article.**class**);  System.***out***.println(articles);  }  } |