**Elasticsearch**

**第二天**

学习目标：

1、能够完成创建索引的操作（第一章，1.2）

2、能够完成删除索引的操作（第一章，1.2）

3、能够完成创建映射的操作（第一章，1.3）

4、能够完成文档的增删改查（第一章，1.4、1.5）

5、能够完成文档的分页操作（第一章，1.6）

6、能够完成文档的高亮查询操作（第一章，1.7）

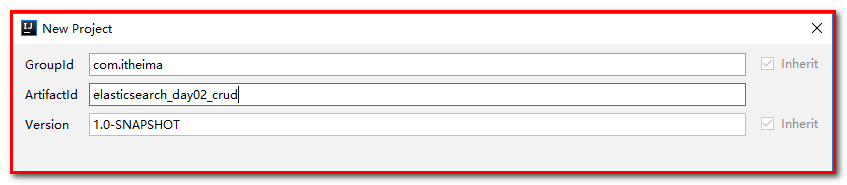
7、能够搭建Spring Data ElasticSearch的环境（第二章，2.2）

8、能够完成Spring Data ElasticSearch的基本增删改查操作（第二章，2.3.1）

9、能够掌握基本条件查询的方法命名规则（第二章，2.3.2）

# **第一章：ElasticSearch编程操作**

## **创建工程，导入坐标**



pom.xml坐标

*<?***xml version="1.0" encoding="UTF-8"***?>*<**project xmlns="http://maven.apache.org/POM/4.0.0"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"**>  
 <**modelVersion**>4.0.0</**modelVersion**>  
  
 <**groupId**>com.itheima</**groupId**>  
 <**artifactId**>elasticsearch\_day02\_crud</**artifactId**>  
 <**version**>1.0-SNAPSHOT</**version**>  
  
 <**dependencies**>  
 <**dependency**>  
 <**groupId**>org.elasticsearch</**groupId**>  
 <**artifactId**>elasticsearch</**artifactId**>  
 <**version**>5.6.8</**version**>  
 </**dependency**>  
 <**dependency**>  
 <**groupId**>org.elasticsearch.client</**groupId**>  
 <**artifactId**>transport</**artifactId**>  
 <**version**>5.6.8</**version**>  
 </**dependency**>  
 <**dependency**>  
 <**groupId**>org.apache.logging.log4j</**groupId**>  
 <**artifactId**>log4j-to-slf4j</**artifactId**>  
 <**version**>2.9.1</**version**>  
 </**dependency**>  
 <**dependency**>  
 <**groupId**>org.slf4j</**groupId**>  
 <**artifactId**>slf4j-api</**artifactId**>  
 <**version**>1.7.24</**version**>  
 </**dependency**>  
 <**dependency**>  
 <**groupId**>org.slf4j</**groupId**>  
 <**artifactId**>slf4j-simple</**artifactId**>  
 <**version**>1.7.21</**version**>  
 </**dependency**>  
 <**dependency**>  
 <**groupId**>log4j</**groupId**>  
 <**artifactId**>log4j</**artifactId**>  
 <**version**>1.2.12</**version**>  
 </**dependency**>  
 <**dependency**>  
 <**groupId**>junit</**groupId**>  
 <**artifactId**>junit</**artifactId**>  
 <**version**>4.10</**version**>  
 </**dependency**>  
 </**dependencies**>  
</**project**>

## **创建/删除索引index**

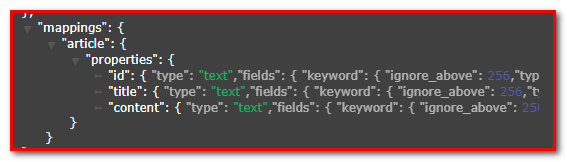
**public class** TestElasticSearchCRUD {  
  
 @Test  
 *//创建索引* **public void** test1() **throws** Exception{  
 *// 创建Client连接对象  
 //Settings settings = Settings.builder().put("cluster.name", "my-elasticsearch").build();* Settings settings = Settings.***EMPTY***;  
 TransportClient client = **new** PreBuiltTransportClient(settings)  
 .addTransportAddress(**new** InetSocketTransportAddress(InetAddress.*getByName*(**"127.0.0.1"**), 9300));  
 *//创建名称为blog2的索引* client.admin().indices().prepareCreate(**"blog2"**).get();  
 *//释放资源* client.close();  
 }  
}

@Test  
*//删除索引***public void** test2() **throws** Exception {  
 *//创建客户端访问对象* TransportClient client = **new** PreBuiltTransportClient(Settings.***EMPTY***).addTransportAddress(  
 **new** InetSocketTransportAddress(InetAddress.*getByName*(**"127.0.0.1"**), 9300));  
 *//删除索引* client.admin().indices().prepareDelete(**"blog2"**).get();  
  
 *//关闭资源* client.close();  
}

## **创建映射mapping**

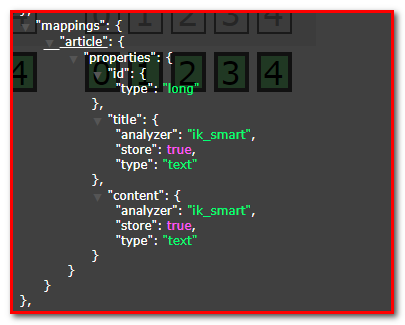
这里的映射表示创建表结构，如果不创建映射，Elasticsearch会默认根据创建的文档中的数据用来创建映射。

查看之前的blog的索引



@Test  
*//创建映射***public void** test3() **throws** Exception{  
 *// 创建Client连接对象* TransportClient client = **new** PreBuiltTransportClient(Settings.***EMPTY***)  
 .addTransportAddress(**new** InetSocketTransportAddress(InetAddress.*getByName*(**"127.0.0.1"**), 9300));  
  
 *//创建名称为blog2的索引* client.admin().indices().prepareCreate(**"blog2"**).get();  
 *// 添加映射  
 /\*\*  
  \* 格式：  
  \* "mappings" : {  
      "article" : {  
           "dynamic" : "false",  
           "properties" : {  
              "id" : { "type" : "string" },  
              "content" : { "type" : "string" },  
              "author" : { "type" : "string" }  
           }  
       }  
    }  
  \*/* XContentBuilder builder = XContentFactory.*jsonBuilder*()  
 .startObject()  
 .startObject(**"article"**)  
 .startObject(**"properties"**)  
 .startObject(**"id"**)  
 .field(**"type"**,**"long"**)  
 .endObject()  
 .startObject(**"title"**)  
 .field(**"type"**,**"string"**)  
 .field(**"store"**,**"true"**)  
 .field(**"analyzer"**,**"ik\_smart"**)  
 .endObject()  
 .startObject(**"content"**)  
 .field(**"type"**,**"string"**)  
 .field(**"store"**,**"true"**)  
 .field(**"analyzer"**,**"ik\_smart"**)  
 .endObject()  
 .endObject()  
 .endObject()  
 .endObject();  
 *// 创建映射* PutMappingRequest mapping = Requests.*putMappingRequest*(**"blog2"**).type(**"article"**).source(builder);  
 client.admin().indices().putMapping(mapping).get();  
 *//释放资源* client.close();  
}

查看最新的blog2的索引

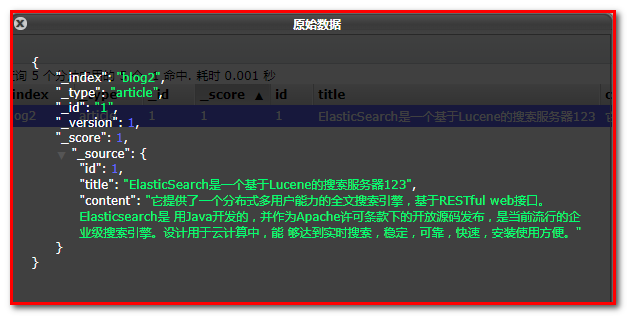


## **建立文档document**

### **回顾：建立文档（通过XContentBuilder）**

@Test  
*//创建文档***public void** test4() **throws** Exception {  
 *//创建客户端访问对象* TransportClient client = **new** PreBuiltTransportClient(Settings.***EMPTY***).addTransportAddress(  
 **new** InetSocketTransportAddress(InetAddress.*getByName*(**"127.0.0.1"**), 9300));  
  
  
 *//创建文档* XContentBuilder builder = XContentFactory.*jsonBuilder*()  
 .startObject()  
 .field(**"id"**,1)  
 .field(**"title"**,**"ElasticSearch是一个基于Lucene的搜索服务器123"**)  
 .field(**"content"**,**"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。Elasticsearch是\n"** +  
 **"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。设计用于云计算中，能\n"** +  
 **"够达到实时搜索，稳定，可靠，快速，安装使用方便。"**)  
 .endObject();  
  
 *//添加文档到指定索引库  
 /\*\*  
 \* 参数一blog1：表示索引对象  
 \* 参数二article：类型  
 \* 参数三1：建立id  
 \*/* client.prepareIndex(**"blog2"**,**"article"**,**"1"**).setSource(builder).get();  
  
 *//关闭资源* client.close();  
}

查看：



### **建立文档（使用Jackson转换实体）**

1. 创建Article实体

创建包com.itheima.es.domain，创建类Article.java

**public class** Article {  
 **private** Integer **id**;  
 **private** String **title**;  
 **private** String **content**;

*// 提供set/get方法  
 // 提供toString方法*

}

1. 添加jackson坐标

Jackson的包，可以将Java对象转换成Json字符串；也可以将Json的字符串，转化成Java对象。

<**dependency**>  
 <**groupId**>com.fasterxml.jackson.core</**groupId**>  
 <**artifactId**>jackson-core</**artifactId**>  
 <**version**>2.8.1</**version**>  
</**dependency**>  
<**dependency**>  
 <**groupId**>com.fasterxml.jackson.core</**groupId**>  
 <**artifactId**>jackson-databind</**artifactId**>  
 <**version**>2.8.1</**version**>  
</**dependency**>  
<**dependency**>  
 <**groupId**>com.fasterxml.jackson.core</**groupId**>  
 <**artifactId**>jackson-annotations</**artifactId**>  
 <**version**>2.8.1</**version**>  
</**dependency**>

1. 代码实现

创建文档

@Test  
*//创建文档(通过实体转json)***public void** test5() **throws** Exception{  
 *// 创建Client连接对象* TransportClient client = **new** PreBuiltTransportClient(Settings.***EMPTY***)  
 .addTransportAddress(**new** InetSocketTransportAddress(InetAddress.*getByName*(**"127.0.0.1"**), 9300));  
  
 *// 描述json 数据  
 //{id:xxx, title:xxx, content:xxx}* Article article = **new** Article();  
 article.setId(4);  
 article.setTitle(**"搜索工作其实很快乐"**);  
 article.setContent(**"我们希望我们的搜索解决方案要快，我们希望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的索引数据"** +  
 **"，我们希望我们的搜索服务器始终可用，我们希望能够一台开始并扩展到数百，我们要实时搜索，"** +  
 **"我们要简单的多租户，我们希望建立一个云的解决方案。Elasticsearch旨在解决所有这些问题和更多的问题。"**);  
  
 *//json格式对象转换器* ObjectMapper objectMapper = **new** ObjectMapper();  
  
 client.prepareIndex(**"blog2"**,**"article"**,article.getId().toString())  
 *//.setSource(objectMapper.writeValueAsString(article)).get()* .setSource(objectMapper.writeValueAsString(article).getBytes(), XContentType.***JSON***).get();  
 *//关闭资源* client.close();  
  
}

### 修改文档

方案一：使用prepareIndex、prepareUpdate修改文档

代码实现：

@Test  
*//使用prepareUpdate、prepareIndex修改文档***public void** test6() **throws** Exception{  
 *// 创建Client连接对象* TransportClient client = **new** PreBuiltTransportClient(Settings.***EMPTY***)  
 .addTransportAddress(**new** InetSocketTransportAddress(InetAddress.*getByName*(**"127.0.0.1"**), 9300));  
  
 *// 描述json 数据  
 //{id:xxx, title:xxx, content:xxx}* Article article = **new** Article();  
 article.setId(2);  
 article.setTitle(**"update修改：搜索工作我很喜欢"**);  
 article.setContent(**"update修改：我们希望我们的搜索解决方案要快，我们希望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的索引数据"** +  
 **"，我们希望我们的搜索服务器始终可用，我们希望能够一台开始并扩展到数百，我们要实时搜索，"** +  
 **"我们要简单的多租户，我们希望建立一个云的解决方案。Elasticsearch旨在解决所有这些问题和更多的问题。"**);  
  
 *//json格式对象转换器* ObjectMapper objectMapper = **new** ObjectMapper();  
  
 client.prepareUpdate(**"blog2"**,**"article"**,article.getId().toString()).setDoc(objectMapper.writeValueAsString(article)).get();  
  
 *//关闭资源* client.close();  
}

方案二：直接使用update()修改文档

代码实现：

@Test  
*//使用update修改文档***public void** test6() **throws** Exception{  
 *// 创建Client连接对象* TransportClient client = **new** PreBuiltTransportClient(Settings.***EMPTY***)  
 .addTransportAddress(**new** InetSocketTransportAddress(InetAddress.*getByName*(**"127.0.0.1"**), 9300));  
  
 *// 描述json 数据  
 //{id:xxx, title:xxx, content:xxx}* Article article = **new** Article();  
 article.setId(2);  
 article.setTitle(**"update123修改：搜索工作我很喜欢"**);  
 article.setContent(**"update123修改：我们希望我们的搜索解决方案要快，我们希望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的索引数据"** +  
 **"，我们希望我们的搜索服务器始终可用，我们希望能够一台开始并扩展到数百，我们要实时搜索，"** +  
 **"我们要简单的多租户，我们希望建立一个云的解决方案。Elasticsearch旨在解决所有这些问题和更多的问题。"**);  
  
 *//json格式对象转换器* ObjectMapper objectMapper = **new** ObjectMapper();  
  
 client.update(**new** UpdateRequest(**"blog2"**,**"article"**,article.getId().toString()).doc(objectMapper.writeValueAsString(article))).get();  
  
 *//关闭资源* client.close();  
}

### 删除文档

方案一：通过prepareDelete 删除文档

@Test  
*//通过prepareDelete 删除文档***public void** test7() **throws** Exception{  
 *// 创建Client连接对象* TransportClient client = **new** PreBuiltTransportClient(Settings.***EMPTY***)  
 .addTransportAddress(**new** InetSocketTransportAddress(InetAddress.*getByName*(**"127.0.0.1"**), 9300));  
  
 *//删除文档* client.prepareDelete(**"blog2"**,**"article"**,**"2"**).get(); *//关闭资源* client.close();  
}

方案二：直接使用delete() 删除文档

@Test  
*//通过delete 删除文档***public void** test7() **throws** Exception{  
 *// 创建Client连接对象* TransportClient client = **new** PreBuiltTransportClient(Settings.***EMPTY***)  
 .addTransportAddress(**new** InetSocketTransportAddress(InetAddress.*getByName*(**"127.0.0.1"**), 9300));  
  
 *//删除文档* client.delete(**new** DeleteRequest(**"blog2"**,**"article"**,**"3"**)).get();  
 *//关闭资源* client.close();  
}

## **查询文档操作**

### **字符串查询**

@Test  
*//字符串查询***public void** test8() **throws** Exception {  
 *//创建客户端访问对象* TransportClient client = **new** PreBuiltTransportClient(Settings.***EMPTY***).addTransportAddress(  
 **new** InetSocketTransportAddress(  
 InetAddress.*getByName*(**"127.0.0.1"**), 9300));  
 *//设置查询条件（QueryBuilders.matchAllQuery()：查询所有）* SearchResponse searchResponse = client.prepareSearch(**"blog2"**).setTypes(**"article"**)  
 .setQuery(QueryBuilders.*queryStringQuery*(**"搜索"**)).get();  
 *//处理结果* SearchHits hits = searchResponse.getHits(); *//获得命中目标，即查询到了多少个对象* System.***out***.println(**"共查询"**+hits.getTotalHits()+**"条"**);  
 Iterator<SearchHit> ite = hits.iterator();  
 **while**(ite.hasNext()){  
 SearchHit searchHit = ite.next();  
 System.***out***.println(searchHit.getSourceAsString());  
 System.***out***.println(searchHit.getSource().get(**"title"**));  
 }  
 *//关闭资源* client.close();  
}

​

### **词条查询**

@Test  
*//词条查询***public void** test9() **throws** Exception {  
 *//创建客户端访问对象* TransportClient client = **new** PreBuiltTransportClient(Settings.***EMPTY***).addTransportAddress(  
 **new** InetSocketTransportAddress(  
 InetAddress.*getByName*(**"127.0.0.1"**), 9300));  
 *//设置查询条件* SearchResponse searchResponse = client.prepareSearch(**"blog2"**).setTypes(**"article"**)  
 .setQuery(QueryBuilders.*termQuery*(**"title"**,**"搜索"**)).get();  
 *//处理结果* SearchHits hits = searchResponse.getHits(); *//获得命中目标，即查询到了多少个对象* System.***out***.println(**"共查询"**+hits.getTotalHits()+**"条"**);  
 Iterator<SearchHit> ite = hits.iterator();  
 **while**(ite.hasNext()){  
 SearchHit searchHit = ite.next();  
 System.***out***.println(searchHit.getSourceAsString());  
 System.***out***.println(searchHit.getSource().get(**"title"**));  
 }  
 *//关闭资源* client.close();  
}

### **查询文档（使用Jackson转换实体）**

\* 查询所有

\* 词条查询

\* 字符串查询（指定字段）

\* 通配符查询

\* 相似度查询

\* 范围查询

@Test  
*// 各种查询***public void** test10() **throws** Exception{  
 *//创建客户端访问对象* TransportClient client = **new** PreBuiltTransportClient(Settings.***EMPTY***).addTransportAddress(  
 **new** InetSocketTransportAddress(  
 InetAddress.*getByName*(**"127.0.0.1"**), 9300));  
 *// 定义Jackson* ObjectMapper objectMapper = **new** ObjectMapper();  
 SearchResponse searchResponse = client.prepareSearch(**"blog2"**).setTypes(**"article"**)  
 *// .setQuery(new MatchAllQueryBuilder())// 查询所有（词条查询、通配符查询、模糊查询）* .setQuery(QueryBuilders.*queryStringQuery*(**"搜索"**).field(**"title"**).field(**"content"**)) *// 条件查询，默认是搜索所有字段；如果是某个字段  
 // .setQuery(QueryBuilders.wildcardQuery("title", "搜?")) // 通配符查询，\*表示任何字符，?表示任意单个字符  
 //.setQuery(QueryBuilders.fuzzyQuery("title", "lucenx")) // 相似度查询  
 //.setQuery(QueryBuilders.termQuery("title", "搜索")) // 词条查询*

*//.setQuery(QueryBuilders.rangeQuery("id").from(1).to(1))// 范围查询(id需要建立索引，否则：Cannot search on field [id] since it is not indexed. )*

.get();  
 *//处理结果* SearchHits hits = searchResponse.getHits(); *//获得命中目标，即查询到了多少个对象* System.***out***.println(**"共查询"**+hits.getTotalHits()+**"条"**);  
 Iterator<SearchHit> ite = hits.iterator();  
 **while**(ite.hasNext()){  
 SearchHit searchHit = ite.next();  
 System.***out***.println(**"得分:"**+searchHit.getScore());  
 *// json的字符串* System.***out***.println(searchHit.getSourceAsString());  
 *// 将json的字符串转换成Article对象* Article article = objectMapper.readValue(searchHit.getSourceAsString(), Article.**class**);  
 System.***out***.println(**"id:"**+article.getId());  
 System.***out***.println(**"title:"**+article.getTitle());  
 System.***out***.println(**"content:"**+article.getContent());  
 System.***out***.println(**"----------------------------------------------"**);  
 }  
 client.close();  
}

## **查询文档分页操作**

### **批量插入数据（100条）**

@Test  
*//批量插入文档***public void** test11() **throws** Exception{  
 *// 创建Client连接对象* TransportClient client = **new** PreBuiltTransportClient(Settings.***EMPTY***)  
 .addTransportAddress(**new** InetSocketTransportAddress(InetAddress.*getByName*(**"127.0.0.1"**), 9300));  
  
 *//json格式对象转换器* ObjectMapper objectMapper = **new** ObjectMapper();  
  
 **for** (**int** i = 1; i <= 100; i++) {  
  
 *// 描述json 数据  
 //{id:xxx, title:xxx, content:xxx}* Article article = **new** Article();  
 article.setId(i);  
 article.setTitle(i+**"搜索工作其实很快乐"**);  
 article.setContent(i+**"我们希望我们的搜索解决方案要快，我们希望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的索引数据"** +  
 **"，我们希望我们的搜索服务器始终可用，我们希望能够一台开始并扩展到数百，我们要实时搜索，"** +  
 **"我们要简单的多租户，我们希望建立一个云的解决方案。Elasticsearch旨在解决所有这些问题和更多的问题。"**);  
  
 client.prepareIndex(**"blog2"**,**"article"**,article.getId().toString()).setSource(objectMapper.writeValueAsString(article)).get();  
 }  
  
 *//关闭资源* client.close();  
  
  
}



### 【组合查询】（复杂查询）

布尔查询

must(QueryBuilders) : AND，求交集

mustNot(QueryBuilders): NOT

should(QueryBuilders):OR ，求并集

在定义json：放置到Elasticsearch的HEAD插件中（DSL表达式）

{

"query" : {

"bool" : {

"must" : {

"term" : {

"title" : "elasticsearch"

}

},

"must" : {

"range" : {

"id" : {

"from" : 1,

"to" : 5

}

}

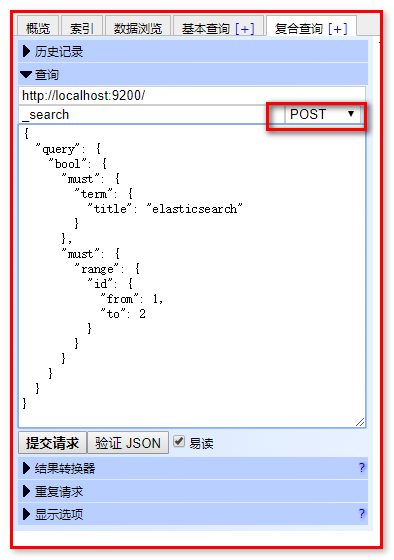
}

}

}

}

如果使用head插件查看索引库的信息，进行查询：



使用java代码：

@Test  
*//组合查询***public void** test12() **throws** Exception{  
 *// 创建Client连接对象* TransportClient client = **new** PreBuiltTransportClient(Settings.***EMPTY***)  
 .addTransportAddress(**new** InetSocketTransportAddress(InetAddress.*getByName*(**"127.0.0.1"**), 9300));  
  
 *//执行查询* SearchRequestBuilder requestBuilder = client.prepareSearch(**"blog2"**).setTypes(**"article"**)  
 .setQuery(QueryBuilders.*boolQuery*().must(QueryBuilders.*termQuery*(**"title"**, **"搜索"**))*// 词条查询  
 //.must(QueryBuilders.rangeQuery("id").from(1).to(5)) // 范围查询* .must(QueryBuilders.*wildcardQuery*(**"content"**, **"elastics\*ch"**)) *// 模糊查询* );*// 大家回去改成should试试看。默认查询10条  
  
 //获得结果* SearchResponse searchResponse = requestBuilder.get();  
 *//返回命中文档对象的数据* SearchHits hits = searchResponse.getHits();  
  
 System.***out***.println(**"共查询到"**+hits.getTotalHits()+**"条"**);  
 Iterator<SearchHit> ite = hits.iterator();  
 **while**(ite.hasNext()){  
 SearchHit searchHit = ite.next();  
 System.***out***.println(searchHit.getSourceAsString());  
 System.***out***.println(**"---------------------------"**);  
 }  
 *//关闭资源* client.close();  
}

这里发现：

.must(QueryBuilders.*wildcardQuery*("content", "elastics\*ch")) // 模糊查询

能搜索到到结果。

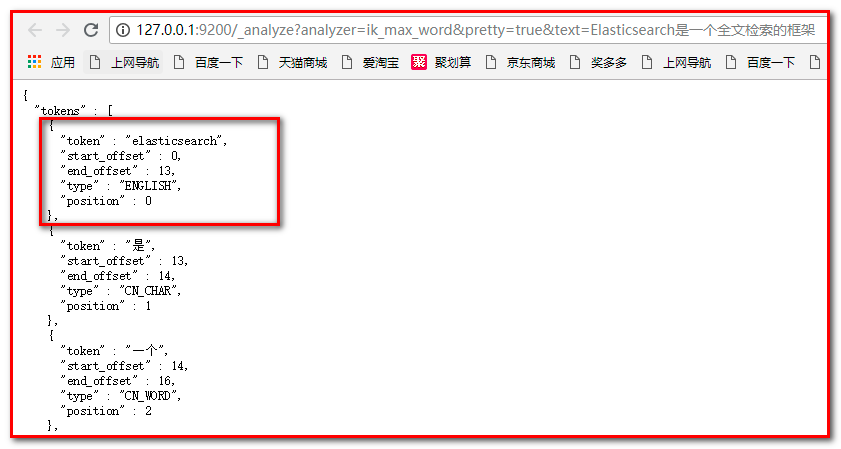
而使用

.must(QueryBuilders.*wildcardQuery*("content", "Elastics\*ch")) // 模糊查询

不能搜索到结果。为什么呢？

输入：

[http://127.0.0.1:9200/\_analyze?analyzer=ik\_max\_word&pretty=true&text=Elasticsearch是一个全文检索的框架](http://localhost:9200/_analyze?analyzer=ik&pretty=true&text=Elasticsearch是一个全文检索的框架)



因为IK分词器，在建立索引的时候将英文都变成了小写，这样方便我们在搜索的时候可以实现“不区分大小写”的搜索，只要我们在搜索的条件中添加.toLowerCase()的方法即可。

例如：

.must(QueryBuilders.*wildcardQuery*("content", "ELASTics\*ch".toLowerCase()))

将输入的值都先变成小写，再来搜索结果

### **分页查询和排序**

@Test  
*//分页查询和排序***public void** test13() **throws** Exception{  
 *// 创建Client连接对象* TransportClient client = **new** PreBuiltTransportClient(Settings.***EMPTY***)  
 .addTransportAddress(**new** InetSocketTransportAddress(InetAddress.*getByName*(**"127.0.0.1"**), 9300));  
  
 *//执行查询* SearchRequestBuilder requestBuilder = client.prepareSearch(**"blog2"**).setTypes(**"article"**)  
 .setQuery(QueryBuilders.*matchAllQuery*());*//默认查询10条  
 // 分页  
 //setFrom()：从第几条开始检索，默认是0。  
 //setSize():每页最多显示的记录数。* requestBuilder.setFrom(0).setSize(20);  
 *// 排序* requestBuilder.addSort(**"id"**, SortOrder.***ASC***);  
 *//获得结果* SearchResponse searchResponse = requestBuilder.get();  
 *//返回命中文档对象的数据* SearchHits hits = searchResponse.getHits();  
  
 System.***out***.println(**"共查询到"**+hits.getTotalHits()+**"条"**);  
  
 Iterator<SearchHit> ite = hits.iterator();  
  
 **while**(ite.hasNext()){  
 SearchHit searchHit = ite.next();  
 System.***out***.println(searchHit.getSourceAsString());  
 System.***out***.println(**"---------------------------"**);  
 }  
 *//关闭资源* client.close();  
}

## **查询结果高亮操作**

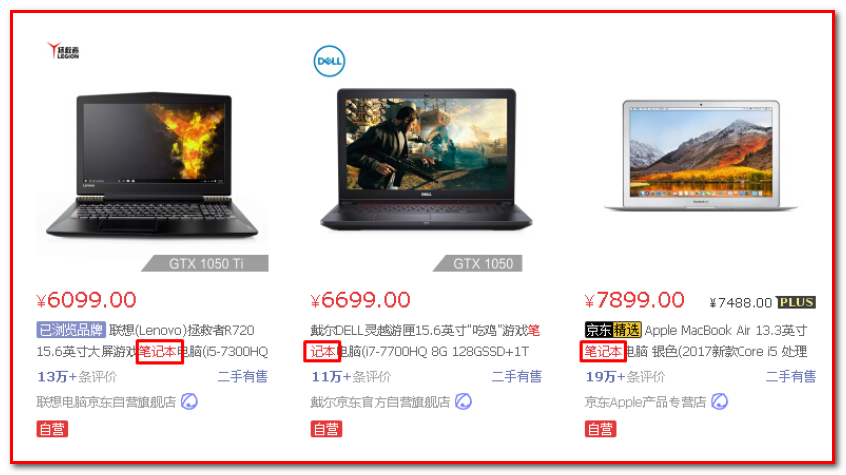
### **什么是高亮显示**

在进行关键字搜索时，搜索出的内容中的关键字会显示不同的颜色，称之为高亮

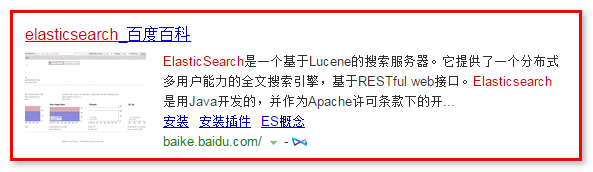
百度搜索关键字"传智播客"



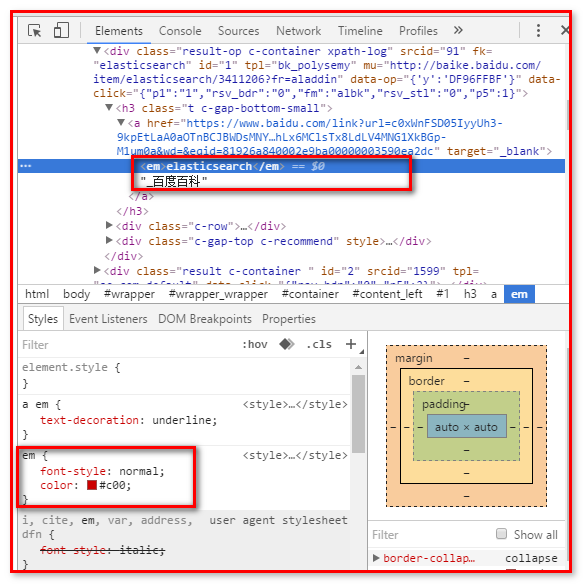
京东商城搜索"笔记本"



在百度搜索"elasticsearch"。



查看页面源码分析



### **高亮显示的html分析**

通过开发者工具查看高亮数据的html代码实现：

ElasticSearch可以对查询出的内容中关键字部分进行标签和样式的设置，但是你需要告诉ElasticSearch使用什么标签对高亮关键字进行包裹呢？

使用<em>高亮内容</em>

### **高亮显示代码实现**

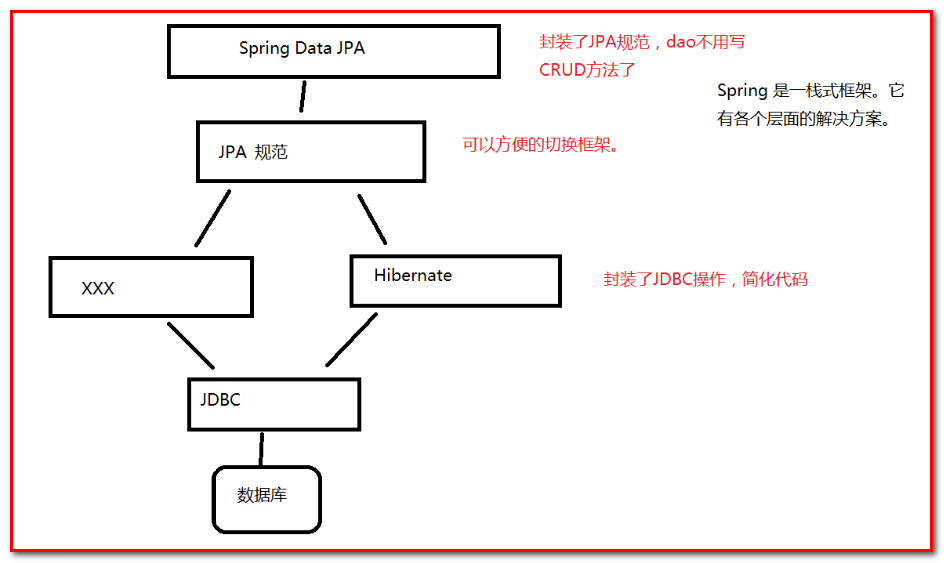
@Test  
*//高亮显示代码实现***public void** test14() **throws** Exception{  
 *// 创建Client连接对象* TransportClient client = **new** PreBuiltTransportClient(Settings.***EMPTY***)  
 .addTransportAddress(**new** InetSocketTransportAddress(InetAddress.*getByName*(**"127.0.0.1"**), 9300));  
  
 *//执行查询* SearchRequestBuilder requestBuilder = client.prepareSearch(**"blog2"**).  
 setTypes(**"article"**).setQuery(QueryBuilders.*termQuery*(**"title"**,**"搜索"**));*//默认查询10条  
  
 //第一步：设置高亮* HighlightBuilder highlightBuilder = **new** HighlightBuilder();  
 highlightBuilder.preTags(**"<font color='red'>"**);  
 highlightBuilder.postTags(**"</font>"**);  
 highlightBuilder.field(**"title"**);  
  
 requestBuilder.highlighter(highlightBuilder);  
  
 SearchResponse searchResponse = requestBuilder.get();  
 SearchHits hits = searchResponse.getHits();  
 *// Iterator<SearchHit> ite = hits.iterator();* **for** (SearchHit hit : hits) {  
 System.***out***.println(**"String方式打印文档搜索内容:"**);  
 System.***out***.println(hit.getSourceAsString());  
 System.***out***.println(**"Map方式打印高亮内容"**);  
 System.***out***.println(hit.getHighlightFields());  
 System.***out***.println(**"遍历高亮集合，打印高亮片段:"**);

*//第二步：获取高亮*  
 Text[] titles = hit.getHighlightFields().get(**"title"**).fragments(); *// 获取title字段的高亮文本* **for** (Text title : titles) {  
 System.***out***.println(title);  
 }  
 System.***out***.println(**"--------------------------"**);  
 }  
 *//关闭资源* client.close();  
}

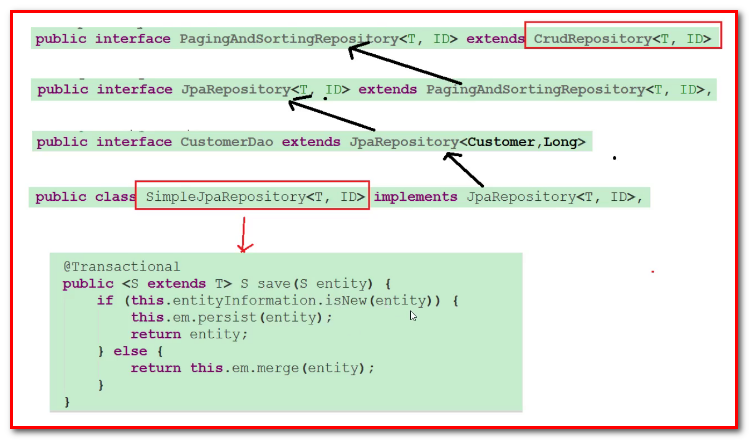
# **第二章：Spring Data ElasticSearch 使用（独立使用）**

【回顾】

掌握Spring Data Elasticsearch之前，先掌握Spring Data JPA。



其中Spring Data JPA与JPA的关系



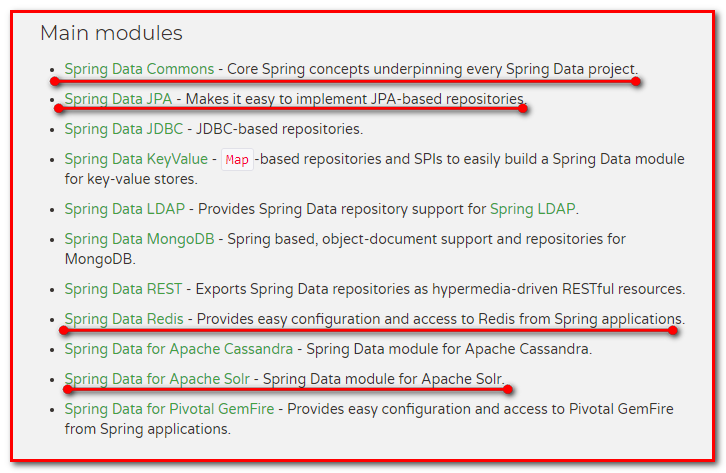
## **Spring Data ElasticSearch简介**

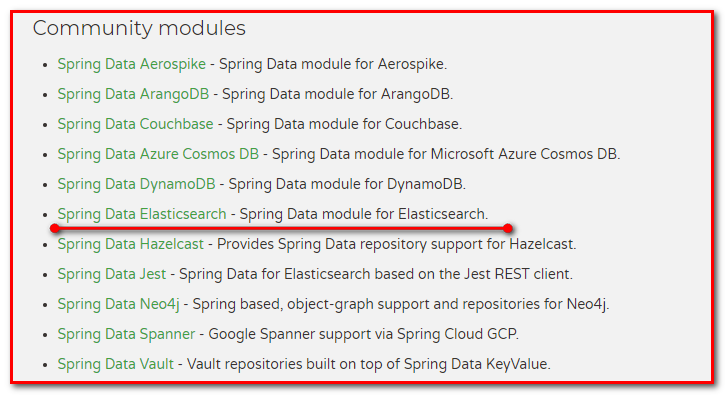
### **什么是Spring Data**

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/>

Spring Data常用的功能模块如下：





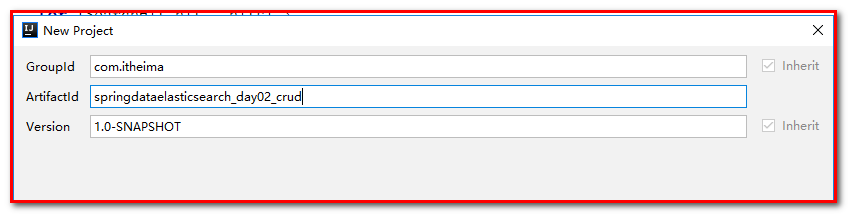
### **什么是Spring Data ElasticSearch**

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Elastichsearch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/>

## **Spring Data ElasticSearch入门**

1）创建工程



导入Spring Data ElasticSearch坐标

*<?***xml version="1.0" encoding="UTF-8"***?>*<**project xmlns="http://maven.apache.org/POM/4.0.0"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"**>  
 <**modelVersion**>4.0.0</**modelVersion**>  
  
 <**groupId**>com.itheima</**groupId**>  
 <**artifactId**>springdataelasticsearch\_day02\_crud</**artifactId**>  
 <**version**>1.0-SNAPSHOT</**version**>  
  
 <**dependencies**>  
 <**dependency**>  
 <**groupId**>org.elasticsearch</**groupId**>  
 <**artifactId**>elasticsearch</**artifactId**>  
 <**version**>5.6.8</**version**>  
 </**dependency**>  
 <**dependency**>  
 <**groupId**>org.elasticsearch.client</**groupId**>  
 <**artifactId**>transport</**artifactId**>  
 <**version**>5.6.8</**version**>  
 </**dependency**>  
 <**dependency**>  
 <**groupId**>org.apache.logging.log4j</**groupId**>  
 <**artifactId**>log4j-to-slf4j</**artifactId**>  
 <**version**>2.9.1</**version**>  
 </**dependency**>  
 <**dependency**>  
 <**groupId**>org.slf4j</**groupId**>  
 <**artifactId**>slf4j-api</**artifactId**>  
 <**version**>1.7.24</**version**>  
 </**dependency**>  
 <**dependency**>  
 <**groupId**>org.slf4j</**groupId**>  
 <**artifactId**>slf4j-simple</**artifactId**>  
 <**version**>1.7.21</**version**>  
 </**dependency**>  
 <**dependency**>  
 <**groupId**>log4j</**groupId**>  
 <**artifactId**>log4j</**artifactId**>  
 <**version**>1.2.12</**version**>  
 </**dependency**>  
 <**dependency**>  
 <**groupId**>junit</**groupId**>  
 <**artifactId**>junit</**artifactId**>  
 <**version**>4.12</**version**>  
 </**dependency**>  
  
 <**dependency**>  
 <**groupId**>org.springframework</**groupId**>  
 <**artifactId**>spring-test</**artifactId**>  
 <**version**>5.0.8.RELEASE</**version**>  
 </**dependency**>  
 <**dependency**>  
 <**groupId**>org.springframework.data</**groupId**>  
 <**artifactId**>spring-data-elasticsearch</**artifactId**>  
 <**version**>3.0.7.RELEASE</**version**>  
 </**dependency**>  
  
 </**dependencies**>  
</**project**>

2）创建applicationContext.xml配置文件，引入elasticsearch命名空间

*<?***xml version="1.0" encoding="UTF-8"***?>*<**beans xmlns="http://www.springframework.org/schema/beans"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xmlns:context="http://www.springframework.org/schema/context"  
 xmlns:elasticsearch="http://www.springframework.org/schema/data/elasticsearch"  
 xsi:schemaLocation="  
 http://www.springframework.org/schema/beans   
 http://www.springframework.org/schema/beans/spring-beans.xsd  
 http://www.springframework.org/schema/context   
 http://www.springframework.org/schema/context/spring-context.xsd  
 http://www.springframework.org/schema/data/elasticsearch  
 http://www.springframework.org/schema/data/elasticsearch/spring-elasticsearch-1.0.xsd  
 "**>  
  
</**beans**>

3）编写实体Article

**public class** Article {  
 **private** Integer **id**;  
 **private** String **title**;  
 **private** String **content**;  
   
 **public** Integer getId() {  
 **return id**;  
 }  
  
 **public void** setId(Integer id) {  
 **this**.**id** = id;  
 }  
  
 **public** String getTitle() {  
 **return title**;  
 }  
  
 **public void** setTitle(String title) {  
 **this**.**title** = title;  
 }  
  
 **public** String getContent() {  
 **return content**;  
 }  
  
 **public void** setContent(String content) {  
 **this**.**content** = content;  
 }  
  
 @Override  
 **public** String toString() {  
 **return "Article{"** +  
 **"id="** + **id** +  
 **", title='"** + **title** + **'\''** +  
 **", content='"** + **content** + **'\''** +  
 **'}'**;  
 }  
}

4）编写Dao

**package** com.itheima.es.dao;  
  
**import** com.itheima.es.domain.Article;  
**import** org.springframework.data.elasticsearch.repository.ElasticsearchRepository;  
**import** org.springframework.stereotype.Repository;  
  
@Repository  
**public interface** ArticleRepository **extends** ElasticsearchRepository<Article, Integer> {  
   
}

5）编写Service

接口ArticleService.java

**package** com.itheima.es.service;  
  
**import** com.itheima.es.domain.Article;  
**public interface** ArticleService {   
 **public void** save(Article article);  
}

实现类ArticleServiceImpl.java

**package** com.itheima.es.service.impl;  
  
**import** com.itheima.es.dao.ArticleRepository;  
**import** com.itheima.es.domain.Article;  
**import** com.itheima.es.service.ArticleService;  
**import** org.springframework.beans.factory.annotation.Autowired;  
**import** org.springframework.stereotype.Service;  
  
@Service  
**public class** ArticleServiceImpl **implements** ArticleService {  
 @Autowired  
 **private** ArticleRepository **articleRepository**;

**public void** save(Article article) {   
 **articleRepository**.save(article);   
 }  
}

6） 配置applicationContext.xml

*<?***xml version="1.0" encoding="UTF-8"***?>*<**beans xmlns="http://www.springframework.org/schema/beans"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xmlns:context="http://www.springframework.org/schema/context"  
 xmlns:elasticsearch="http://www.springframework.org/schema/data/elasticsearch"  
 xsi:schemaLocation="  
 http://www.springframework.org/schema/beans  
 http://www.springframework.org/schema/beans/spring-beans.xsd  
 http://www.springframework.org/schema/context  
 http://www.springframework.org/schema/context/spring-context.xsd  
 http://www.springframework.org/schema/data/elasticsearch  
 http://www.springframework.org/schema/data/elasticsearch/spring-elasticsearch-1.0.xsd  
 "**>

*<!-- 1：扫描Service包，创建Service的实体 -->* <**context:component-scan base-package="com.itheima.es.service"**/>

*<!-- 2：扫描Dao包，自动创建实例，扫描所有继承ElasticsearchRepository接口的接口 -->* <**elasticsearch:repositories base-package="com.itheima.es.dao"**/>  
   
 *<!--3：配置elasticSearch的连接 -->* <**elasticsearch:transport-client id="client" cluster-nodes="localhost:9300" cluster-name="elasticsearch"**/>  
   
 *<!-- 4：ElasticSearch模版对象 -->* <**bean id="elasticsearchTemplate" class="org.springframework.data.elasticsearch.core.ElasticsearchTemplate"**>  
 <**constructor-arg name="client" ref="client"**></**constructor-arg**>  
 </**bean**>  
   
</**beans**>

7）配置实体

基于spring data elasticsearch注解配置索引、映射和实体的关系

*//@Document 文档对象 （索引信息、文档类型 ）*@Document(indexName=**"blog3"**,type=**"article"**)  
**public class** Article {  
 *//@Id 文档主键 唯一标识* @Id  
 **private** Integer **id**;  
 @Field(index=**true**,analyzer=**"ik\_smart"**,store=**true**,searchAnalyzer=**"ik\_smart"**,type = FieldType.***Text***)   
 **private** String **title**;  
 @Field(index=**true**,analyzer=**"ik\_smart"**,store=**true**,searchAnalyzer=**"ik\_smart"**,type = FieldType.***Text***)   
 **private** String **content**;  
  
 **public** Integer getId() {  
 **return id**;  
 }  
  
 **public void** setId(Integer id) {  
 **this**.**id** = id;  
 }  
  
 **public** String getTitle() {  
 **return title**;  
 }  
  
 **public void** setTitle(String title) {  
 **this**.**title** = title;  
 }  
  
 **public** String getContent() {  
 **return content**;  
 }  
  
 **public void** setContent(String content) {  
 **this**.**content** = content;  
 }  
  
 @Override  
 **public** String toString() {  
 **return "Article{"** +  
 **"id="** + **id** +  
 **", title='"** + **title** + **'\''** +  
 **", content='"** + **content** + **'\''** +  
 **'}'**;  
 }  
}

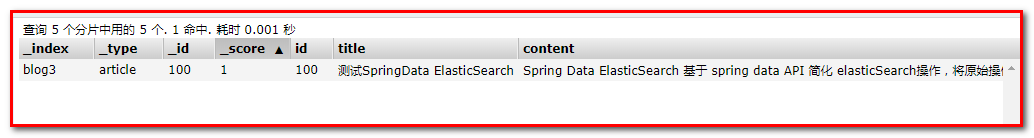
其中，注解解释如下：  
@Document(indexName="blob3",type="article")：  
   indexName：索引的名称（必填项）  
   type：索引的类型  
@Id：主键的唯一标识  
@Field(index=true,analyzer="ik\_smart",store=true,searchAnalyzer="ik\_smart",type = FieldType.Text)  
   index：是否设置分词  
   analyzer：存储时使用的分词器  
   searchAnalyze：搜索时使用的分词器  
   store：是否存储  
   type: 数据类型

8）创建测试类SpringDataESTest

**package** com.itheima.es.test;  
  
**import** com.itheima.es.domain.Article;  
**import** com.itheima.es.service.ArticleService;  
**import** org.elasticsearch.client.transport.TransportClient;  
**import** org.junit.Test;  
**import** org.junit.runner.RunWith;  
**import** org.springframework.beans.factory.annotation.Autowired;  
**import** org.springframework.data.elasticsearch.core.ElasticsearchTemplate;  
**import** org.springframework.test.context.ContextConfiguration;  
**import** org.springframework.test.context.junit4.SpringJUnit4ClassRunner;  
  
@RunWith(SpringJUnit4ClassRunner.**class**)  
@ContextConfiguration(locations=**"classpath:applicationContext.xml"**)  
**public class** SpringDataElasticSearchTest {  
 @Autowired  
 **private** ArticleService **articleService**;  
  
 @Autowired  
 **private** TransportClient **client**;  
  
 @Autowired  
 **private** ElasticsearchTemplate **elasticsearchTemplate**;  
  
 */\*\*创建索引和映射\*/* @Test  
 **public void** createIndex(){  
 **elasticsearchTemplate**.createIndex(Article.**class**);  
 **elasticsearchTemplate**.putMapping(Article.**class**);  
 }  
  
 */\*\*测试保存文档\*/* @Test  
 **public void** saveArticle(){  
 Article article = **new** Article();  
 article.setId(100);  
 article.setTitle(**"测试SpringData ElasticSearch"**);  
 article.setContent(**"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 \n"** +  
 **"   Spring Data为Elasticsearch Elasticsearch项目提供集成搜索引擎"**);  
 **articleService**.save(article);  
 }  
}

查看Elasticsearch





## **Spring Data ElasticSearch的常用操作**

### **增删改查方法测试**

在ArticleService.java中添加：

**public interface** ArticleService {  
 *// 保存/修改* **public void** save(Article article);  
 *//删除* **public void** delete(Article article);  
 *//查询全部* **public** Iterable<Article> findAll();  
 *//分页查询* **public** Page<Article> findAll(Pageable pageable);  
   
}

在ArticleSermpl.java中添加：

@Service  
**public class** ArticleServiceImpl **implements** ArticleService {  
 @Autowired  
 **private** ArticleRepository **articleRepository**;  
  
 **public void** save(Article article) {  
 **articleRepository**.save(article);  
 }  
  
 **public void** delete(Article article) {  
 **articleRepository**.delete(article);  
 }  
  
 **public** Iterable<Article> findAll() {  
 Iterable<Article> iter = **articleRepository**.findAll();  
 **return** iter;  
 }  
  
 **public** Page<Article> findAll(Pageable pageable) {  
 **return articleRepository**.findAll(pageable);  
 }  
}

测试类

@RunWith(SpringJUnit4ClassRunner.**class**)  
@ContextConfiguration(locations=**"classpath:applicationContext.xml"**)  
**public class** SpringDataElasticSearchTest {  
 @Autowired  
 **private** ArticleService **articleService**;  
  
 @Autowired  
 **private** TransportClient **client**;  
  
 @Autowired  
 **private** ElasticsearchTemplate **elasticsearchTemplate**;  
  
 */\*\*创建索引和映射\*/* @Test  
 **public void** createIndex(){  
 **elasticsearchTemplate**.createIndex(Article.**class**);  
 **elasticsearchTemplate**.putMapping(Article.**class**);  
 }  
  
 */\*\*测试保存文档\*/* @Test  
 **public void** saveArticle(){  
 Article article = **new** Article();  
 article.setId(100);  
 article.setTitle(**"测试SpringData ElasticSearch"**);  
 article.setContent(**"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 \n"** +  
 **"   Spring Data为Elasticsearch Elasticsearch项目提供集成搜索引擎"**);  
 **articleService**.save(article);  
 }  
  
 */\*\*测试更新\*/* @Test  
 **public void** update(){  
 Article article = **new** Article();  
 article.setId(100);  
 article.setTitle(**"elasticSearch 3.0版本发布...更新"**);  
 article.setContent(**"ElasticSearch是一个基于Lucene的搜索服务器。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"**);  
 **articleService**.save(article);  
 }  
  
 */\*\*测试删除\*/* @Test  
 **public void** delete(){  
 Article article = **new** Article();  
 article.setId(100);  
 **articleService**.delete(article);  
 }  
 */\*\*批量插入\*/* @Test  
 **public void** save100(){  
 **for**(**int** i=1;i<=100;i++){  
 Article article = **new** Article();  
 article.setId(i);  
 article.setTitle(i+**"elasticSearch 3.0版本发布..，更新"**);  
 article.setContent(i+**"ElasticSearch是一个基于Lucene的搜索服务器。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"**);  
 **articleService**.save(article);  
 }  
 }  
  
 */\*\*查询所有\*/* @Test  
 **public void** findAll(){  
 Iterable<Article> iterable = **articleService**.findAll();  
 **for**(Article article:iterable){  
 System.***out***.println(article);  
 }  
 }  
  
 */\*\*分页查询\*/* @Test  
 **public void** findAllPage(){  
 Pageable pageable = PageRequest.*of*(1,10);  
 Page<Article> page = **articleService**.findAll(pageable);

System.***out***.println(**"总记录数："**+page.getTotalElements());  
 **for**(Article article:page.getContent()){  
 System.***out***.println(article);  
 }  
 }  
}

### **常用查询命名规则**

| **关键字** | **命名规则** | **解释** | **示例** |
| --- | --- | --- | --- |
| and | findByField1AndField2 | 根据Field1和Field2获得数据 | findByTitleAndContent |
| or | findByField1OrField2 | 根据Field1或Field2获得数据 | findByTitleOrContent |
| is | findByField | 根据Field获得数据 | findByTitle |
| not | findByFieldNot | 根据Field获得补集数据 | findByTitleNot |
| between | findByFieldBetween | 获得指定范围的数据 | findByPriceBetween |
| lessThanEqual | findByFieldLessThan | 获得小于等于指定值的数据 | findByPriceLessThan |

### **查询方法测试**

1）dao层实现

@Repository  
**public interface** ArticleRepository **extends** ElasticsearchRepository<Article, Integer> {  
 *//根据标题查询* List<Article> findByTitle(String condition);  
  
 *//根据标题查询(含分页)* Page<Article> findByTitle(String condition, Pageable pageable);  
}

1. service层实现

接口ArticleService.java

**public interface** ArticleService {  
 *// 保存* **public void** save(Article article);  
 *//删除* **public void** delete(Article article);  
 *//查询全部* **public** Iterable<Article> findAll();  
 *//分页查询* **public** Page<Article> findAll(Pageable pageable);  
   
 *//根据标题查询* List<Article> findByTitle(String condition);  
 *//根据标题查询(含分页)* Page<Article> findByTitle(String condition, Pageable pageable);  
}

实现类ArticleServiceImpl.java

@Service  
**public class** ArticleServiceImpl **implements** ArticleService {  
 @Autowired  
 **private** ArticleRepository **articleRepository**;  
  
 **public void** save(Article article) {  
 **articleRepository**.save(article);  
 }  
  
 **public void** delete(Article article) {  
 **articleRepository**.delete(article);  
 }  
  
 **public** Iterable<Article> findAll() {  
 Iterable<Article> iter = **articleRepository**.findAll();  
 **return** iter;  
 }  
   
 **public** Page<Article> findAll(Pageable pageable) {  
 **return articleRepository**.findAll(pageable);  
 }  
   
 **public** List<Article> findByTitle(String condition) {  
 **return articleRepository**.findByTitle(condition);  
 }  
 **public** Page<Article> findByTitle(String condition, Pageable pageable) {  
 **return articleRepository**.findByTitle(condition,pageable);  
 }  
}

​

3）测试代码

@RunWith(SpringJUnit4ClassRunner.**class**)  
@ContextConfiguration(locations=**"classpath:applicationContext.xml"**)  
**public class** SpringDataElasticSearchTest {  
 @Autowired  
 **private** ArticleService **articleService**;  
  
 @Autowired  
 **private** TransportClient **client**;  
  
 @Autowired  
 **private** ElasticsearchTemplate **elasticsearchTemplate**;  
  
 */\*\*条件查询\*/* @Test  
 **public void** findByTitle(){  
 String condition = **"版本"**;  
 List<Article> articleList = **articleService**.findByTitle(condition);  
 **for**(Article article:articleList){  
 System.***out***.println(article);  
 }  
 }  
 */\*\*条件分页查询\*/* @Test  
 **public void** findByTitlePage(){  
 String condition = **"版本"**;  
 Pageable pageable = PageRequest.*of*(2,10,**new** Sort(Sort.Direction.***ASC***,**"id"**));  
 Page<Article> page = **articleService**.findByTitle(condition,pageable);  
 **for**(Article article:page.getContent()){  
 System.***out***.println(article);  
 }  
 }  
}

​

测试：and的用法，大家试试看。

在ArticleRepository中定义：

Page<Article> findByTitleAndContent(String title, String content, Pageable pageable);