**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

## 准备

我们可以使用昨天创建的工程elasticsearch\_demo，在test包下创建TestElasticSearchCRUD.java，可以复制昨天的TestElasticSearch.java改一下。

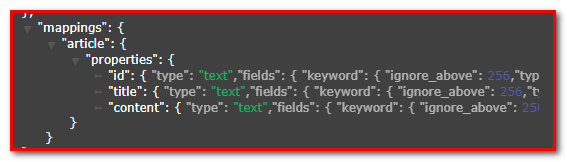
## 创建/删除索引index

**public class** TestElasticsearchCRUD {  
  
 *//创建索引、删除索引* @Test  
 **public void** testCreateIndex() **throws** Exception{  
 *//1.创建客户端访问对象-TransportClient* TransportClient client = **new** PreBuiltTransportClient(Settings.***EMPTY***);  
 *//2.创建Elasticsearch连接对象-new InetSocketTransportAddress(InetAddress.getByName(地址), 端口号)* InetSocketTransportAddress address = **new** InetSocketTransportAddress(InetAddress.*getByName*(**"127.0.0.1"**), 9300);  
 *//3.指定Elasticsearch连接地址与端口-client.addTransportAddress(address)* client.addTransportAddress(address);  
  
 *//创建索引  
 //client.admin().indices().prepareCreate("blog3").get();  
  
 //删除索引* client.admin().indices().prepareDelete(**"blog1"**, **"blog3"**).get();  
  
 *//8.关闭客户端-client.close()* client.close();  
 }  
}

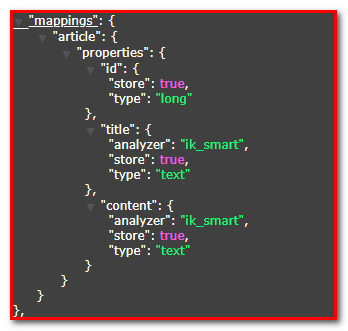
## 创建映射mapping

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

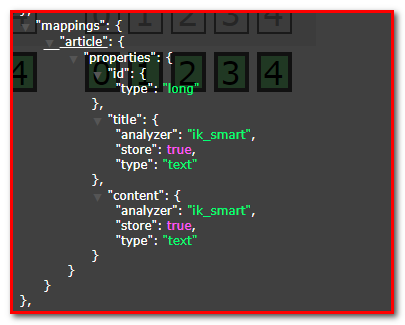


要想指定对每个字段是否使用分词器，需要手动创建映射。



查看最新的blog2的索引

*//创建映射*@Test  
**public void** testPutMappings() **throws** Exception{  
 *//1.创建客户端访问对象-TransportClient* TransportClient client = **new** PreBuiltTransportClient(Settings.***EMPTY***);  
 *//2.创建Elasticsearch连接对象-new InetSocketTransportAddress(InetAddress.getByName(地址), 端口号)* InetSocketTransportAddress address = **new** InetSocketTransportAddress(InetAddress.*getByName*(**"127.0.0.1"**), 9300);  
 *//3.指定Elasticsearch连接地址与端口-client.addTransportAddress(address)* client.addTransportAddress(address);  
  
 *//创建索引* client.admin().indices().prepareCreate(**"blog"**).get();  
  
 XContentBuilder builder = XContentFactory.*jsonBuilder*()  
 .startObject()  
 .startObject(**"article"**)  
 .startObject(**"properties"**)  
 .startObject(**"id"**)  
 .field(**"type"**,**"long"**) *//指定数据类型* .field(**"store"**,**false**) *//是否存储(默认就是false),使用默认的\_score来存储* .field(**"index"**,**true**) *//是否创建索引，默认true* .endObject()  
 .startObject(**"title"**)  
 .field(**"type"**,**"text"**) *//指定数据类型* .field(**"store"**,**false**) *//是否存储(默认就是false),使用默认的\_score来存储* .field(**"index"**,**true**) *//是否创建索引，默认true* .field(**"analyzer"**,**"ik\_smart"**) *//使用ik最小切分方式作为分词器* .endObject()  
 .startObject(**"content"**)  
 .field(**"type"**,**"text"**) *//指定数据类型* .field(**"store"**,**false**) *//是否存储(默认就是false),使用默认的\_score来存储* .field(**"index"**,**true**) *//是否创建索引，默认true* .field(**"analyzer"**,**"ik\_smart"**) *//使用ik最小切分方式作为分词器* .endObject()  
 .endObject()  
 .endObject()  
 .endObject();  
 PutMappingRequest request = Requests.*putMappingRequest*(**"blog"**).type(**"article"**).source(builder);  
 client.admin().indices().putMapping(request);  
  
 *//8.关闭客户端-client.close()* client.close();  
}

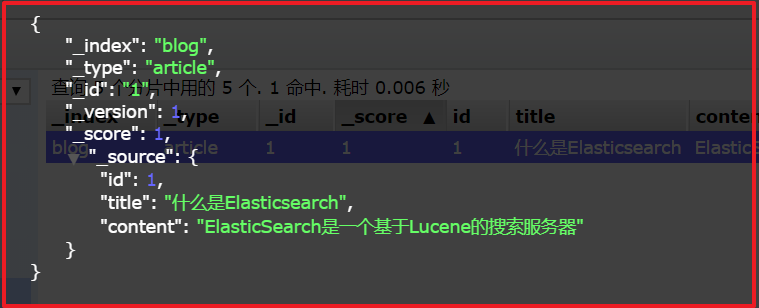


## 建立文档document

### 回顾：建立文档（通过XContentBuilder）

*//创建文档-XContentBuilder*@Test  
**public void** testCreateDocument() **throws** Exception{  
 */\*\*1.创建客户端访问对象-TransportClient\*/* TransportClient client = **new** PreBuiltTransportClient(Settings.***EMPTY***);  
 *//2.创建Elasticsearch连接对象-new InetSocketTransportAddress(InetAddress.getByName(地址), 端口号)* InetSocketTransportAddress address = **new** InetSocketTransportAddress(InetAddress.*getByName*(**"127.0.0.1"**), 9300);  
 *//3.指定Elasticsearch连接地址与端口-client.addTransportAddress(address)* client.addTransportAddress(address);  
  
 *//方式2：XContentBuilder(推荐)  
 //4.准备索引-client.prepareIndex(索引名,类型名,文档id)* IndexRequestBuilder indexRequestBuilder = client.prepareIndex(**"blog"**, **"article"**, **"1"**);  
 XContentBuilder builder = XContentFactory.*jsonBuilder*()  
 .startObject() *//{* .field(**"id"**,1)  
 .field(**"title"**,**"什么是Elasticsearch"**)  
 .field(**"content"**,**"ElasticSearch是一个基于Lucene的搜索服务器"**)  
 .endObject(); *//}* indexRequestBuilder.setSource(builder);  
  
 *//6.执行保存索引与文档-indexRequestBuilder.get()* IndexResponse response = indexRequestBuilder.get();  
 *//7.输出执行结果-response.sout()* System.***out***.println(response);  
 *//8.关闭客户端-client.close()* client.close();  
}

查看：



### 建立文档（使用Jackson转换实体）

#### 创建Article实体

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

**public class** Article {  
 **private** Long **id**;  
 **private** String **title**;  
 **private** String **content**;  
 *// 提供set/get方法  
 // 提供toString方法*}

#### 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**>

#### Jackson对象与字符串转换

@Test  
**public void** testJacksonMapper() **throws** Exception {  
 Article article = **new** Article();  
 article.setId(2L);  
 article.setTitle(**"2什么是Elasticsearch"**);  
 article.setContent(**"2ElasticSearch是一个基于Lucene的搜索服务器"**);  
  
 ObjectMapper mapper = **new** ObjectMapper();  
 *//把对象转json串* String json = mapper.writeValueAsString(article);  
 System.***out***.println(json);  
  
 String str = **"{\"id\":3,\"title\":\"3什么是Elasticsearch\",\"content\":\"3ElasticSearch是一个基于Lucene的搜索服务器\"}"**;  
 *//把json字符串转对象* Article article1 = mapper.readValue(str, Article.**class**);  
 System.***out***.println(article1);  
}

#### 创建文档-代码实现

创建文档

*//创建文档-Article*@Test  
**public void** testCreateDocumentByArticle() **throws** Exception{  
 */\*\*1.创建客户端访问对象-TransportClient\*/* TransportClient client = **new** PreBuiltTransportClient(Settings.***EMPTY***);  
 *//2.创建Elasticsearch连接对象-new InetSocketTransportAddress(InetAddress.getByName(地址), 端口号)* InetSocketTransportAddress address = **new** InetSocketTransportAddress(InetAddress.*getByName*(**"127.0.0.1"**), 9300);  
 *//3.指定Elasticsearch连接地址与端口-client.addTransportAddress(address)* client.addTransportAddress(address);  
  
 *//方式2：XContentBuilder(推荐)  
 //4.准备索引-client.prepareIndex(索引名,类型名,文档id)* IndexRequestBuilder indexRequestBuilder = client.prepareIndex(**"blog"**, **"article"**, **"2"**);  
  
 Article article = **new** Article();  
 article.setId(2L);  
 article.setTitle(**"2什么是Elasticsearch"**);  
 article.setContent(**"2ElasticSearch是一个基于Lucene的搜索服务器"**);  
 *//使用对象json串保存* ObjectMapper mapper = **new** ObjectMapper();  
 indexRequestBuilder.setSource(mapper.writeValueAsString(article), XContentType.***JSON***);  
  
 *//6.执行保存索引与文档-indexRequestBuilder.get()* IndexResponse response = indexRequestBuilder.get();  
 *//7.输出执行结果-response.sout()* System.***out***.println(response);  
 *//8.关闭客户端-client.close()* client.close();  
}

### 修改文档

#### 方案一：使用prepareIndex，刚才上面的保存文档方式，同样可以更新(推荐)

代码略…

#### 方案二：使用prepareUpdate修改文档

代码实现：

*/\*\*  
 \* 修改文档  
 \* 1.4.3.1. 方案一：使用prepareIndex  
 \* 1.4.3.2. 方案二：使用prepareUpdate  
 \*/*@Test  
**public void** testUpdateDocument() **throws** Exception{  
 */\*\*1.创建客户端访问对象-TransportClient\*/* TransportClient client = **new** PreBuiltTransportClient(Settings.***EMPTY***);  
 *//2.创建Elasticsearch连接对象-new InetSocketTransportAddress(InetAddress.getByName(地址), 端口号)* InetSocketTransportAddress address = **new** InetSocketTransportAddress(InetAddress.*getByName*(**"127.0.0.1"**), 9300);  
 *//3.指定Elasticsearch连接地址与端口-client.addTransportAddress(address)* client.addTransportAddress(address);  
   
 *//方案一：使用prepareIndex(推荐)  
 /\*//4.准备索引-client.prepareIndex(索引名,类型名,文档id)  
 IndexRequestBuilder indexRequestBuilder = client.prepareIndex("blog", "article", "2");  
 Article article = new Article();  
 article.setId(2L);  
 article.setTitle("2-1什么是Elasticsearch");  
 article.setContent("2-1ElasticSearch是一个基于Lucene的搜索服务器");  
 //使用对象json串保存  
 ObjectMapper mapper = new ObjectMapper();  
 indexRequestBuilder.setSource(mapper.writeValueAsString(article), XContentType.JSON);  
 //6.执行保存索引与文档-indexRequestBuilder.get()  
 IndexResponse response = indexRequestBuilder.get();\*/  
   
 //方案二：使用prepareUpdate* Article article = **new** Article();  
 article.setId(2L);  
 article.setTitle(**"2【修改】什么是Elasticsearch"**);  
 article.setContent(**"2【修改】ElasticSearch是一个基于Lucene的搜索服务器"**);  
 ObjectMapper mapper = **new** ObjectMapper();  
 *//更新文档* UpdateResponse response = client.prepareUpdate(**"blog"**, **"article"**, **"2"**)  
 .setDoc(mapper.writeValueAsString(article), XContentType.***JSON***).get();  
  
 *//7.输出执行结果-response.sout()* System.***out***.println(response);  
 *//8.关闭客户端-client.close()* client.close();  
}

#### 方案三：直接使用update()修改文档

代码实现：

*/\*\*  
 \* 修改文档  
 \* 1.4.3.1. 方案一：使用prepareIndex  
 \* 1.4.3.2. 方案二：使用prepareUpdate  
 \* 1.4.3.3. 方案三：直接使用update()修改文档  
 \*/*@Test  
**public void** testUpdateDocument() **throws** Exception{  
 */\*\*1.创建客户端访问对象-TransportClient\*/* TransportClient client = **new** PreBuiltTransportClient(Settings.***EMPTY***);  
 *//2.创建Elasticsearch连接对象-new InetSocketTransportAddress(InetAddress.getByName(地址), 端口号)* InetSocketTransportAddress address = **new** InetSocketTransportAddress(InetAddress.*getByName*(**"127.0.0.1"**), 9300);  
 *//3.指定Elasticsearch连接地址与端口-client.addTransportAddress(address)* client.addTransportAddress(address);  
  
 *//方案一：使用prepareIndex(推荐)  
 /\*//4.准备索引-client.prepareIndex(索引名,类型名,文档id)  
 IndexRequestBuilder indexRequestBuilder = client.prepareIndex("blog", "article", "2");  
 Article article = new Article();  
 article.setId(2L);  
 article.setTitle("2-1什么是Elasticsearch");  
 article.setContent("2-1ElasticSearch是一个基于Lucene的搜索服务器");  
 //使用对象json串保存  
 ObjectMapper mapper = new ObjectMapper();  
 indexRequestBuilder.setSource(mapper.writeValueAsString(article), XContentType.JSON);  
 //6.执行保存索引与文档-indexRequestBuilder.get()  
 IndexResponse response = indexRequestBuilder.get();\*/  
  
 //方案二：使用prepareUpdate  
 /\*Article article = new Article();  
 article.setId(2L);  
 article.setTitle("2【修改】什么是Elasticsearch");  
 article.setContent("2【修改】ElasticSearch是一个基于Lucene的搜索服务器");  
 ObjectMapper mapper = new ObjectMapper();  
 //更新文档  
 UpdateResponse response = client.prepareUpdate("blog", "article", "2")  
 .setDoc(mapper.writeValueAsString(article), XContentType.JSON).get();\*/  
  
 //方案三：直接使用update()修改文档* Article article = **new** Article();  
 article.setId(2L);  
 article.setTitle(**"3【修改】什么是Elasticsearch"**);  
 article.setContent(**"3【修改】ElasticSearch是一个基于Lucene的搜索服务器"**);  
 ObjectMapper mapper = **new** ObjectMapper();  
 ActionFuture<UpdateResponse> response = client.update(**new** UpdateRequest(**"blog"**, **"article"**, **"2"**)  
 .doc(mapper.writeValueAsString(article), XContentType.***JSON***));  
  
 *//7.输出执行结果-response.sout()* System.***out***.println(response);  
 *//8.关闭客户端-client.close()* client.close();  
}

### 删除文档

方案一：通过prepareDelete 删除文档

*/\*\*  
 \* 删除文档  
 \*/*@Test  
**public void** testDeleteDocument() **throws** Exception{  
 */\*\*1.创建客户端访问对象-TransportClient\*/* TransportClient client = **new** PreBuiltTransportClient(Settings.***EMPTY***);  
 *//2.创建Elasticsearch连接对象-new InetSocketTransportAddress(InetAddress.getByName(地址), 端口号)* InetSocketTransportAddress address = **new** InetSocketTransportAddress(InetAddress.*getByName*(**"127.0.0.1"**), 9300);  
 *//3.指定Elasticsearch连接地址与端口-client.addTransportAddress(address)* client.addTransportAddress(address);  
  
 *//方案一：通过prepareDelete 删除文档* DeleteResponse response = client.prepareDelete(**"blog"**, **"article"**, **"2"**).get();  
  
 *//7.输出执行结果-response.sout()* System.***out***.println(response);  
 *//8.关闭客户端-client.close()* client.close();  
}

方案二：直接使用delete() 删除文档

@Test  
**public void** testDeleteDocument() **throws** Exception{  
 */\*\*1.创建客户端访问对象-TransportClient\*/* TransportClient client = **new** PreBuiltTransportClient(Settings.***EMPTY***);  
 *//2.创建Elasticsearch连接对象-new InetSocketTransportAddress(InetAddress.getByName(地址), 端口号)* InetSocketTransportAddress address = **new** InetSocketTransportAddress(InetAddress.*getByName*(**"127.0.0.1"**), 9300);  
 *//3.指定Elasticsearch连接地址与端口-client.addTransportAddress(address)* client.addTransportAddress(address);  
  
 *//方案一：通过prepareDelete 删除文档  
 //DeleteResponse response = client.prepareDelete("blog", "article", "2").get();  
  
 //方案二：直接使用delete() 删除文档* DeleteResponse response = client.delete(**new** DeleteRequest(**"blog"**, **"article"**, **"1"**)).get();  
  
 *//7.输出执行结果-response.sout()* System.***out***.println(response);  
 *//8.关闭客户端-client.close()* client.close();  
}

## 查询文档操作

### 批量导入

*//批量导入文档*@Test  
**public void** testCreateDocumentByArticle100() **throws** Exception{  
 */\*\*1.创建客户端访问对象-TransportClient\*/* TransportClient client = **new** PreBuiltTransportClient(Settings.***EMPTY***);  
 *//2.创建Elasticsearch连接对象-new InetSocketTransportAddress(InetAddress.getByName(地址), 端口号)* InetSocketTransportAddress address = **new** InetSocketTransportAddress(InetAddress.*getByName*(**"127.0.0.1"**), 9300);  
 *//3.指定Elasticsearch连接地址与端口-client.addTransportAddress(address)* client.addTransportAddress(address);  
  
 *//批量执行器* BulkRequestBuilder bulkRequestBuilder = client.prepareBulk();  
  
 *//使用对象json串保存* ObjectMapper mapper = **new** ObjectMapper();  
 **for** (**long** i = 0; i < 100; i++) {  
 Article article = **new** Article();  
 article.setId(i);  
 article.setTitle(**"什么是Elasticsearch?"** + i);  
 article.setContent(**"ElasticSearch是一个基于Lucene的搜索服务器"** + i);  
 *//追加文档* bulkRequestBuilder.add(client.prepareIndex(**"blog"**, **"article"**, i+**""**)  
 .setSource(mapper.writeValueAsString(article), XContentType.***JSON***));  
 }  
 *//执行批量保存索引与文档* BulkResponse responses = bulkRequestBuilder.get();  
 *//7.输出执行结果-response.sout()* System.***out***.println(responses.status());  
 *//8.关闭客户端-client.close()* client.close();  
}

### 回顾-查询所有、字符串查询与词条查询

*//查询文档*@Test  
**public void** testQuery() **throws** Exception{  
 *//1.创建客户端访问对象-TransportClient* TransportClient client = **new** PreBuiltTransportClient(Settings.***EMPTY***);  
 *//2.创建Elasticsearch连接对象-new InetSocketTransportAddress(InetAddress.getByName(地址), 端口号)* InetSocketTransportAddress address = **new** InetSocketTransportAddress(InetAddress.*getByName*(**"127.0.0.1"**), 9300);  
 *//3.指定Elasticsearch连接地址与端口-client.addTransportAddress(address)* client.addTransportAddress(address);  
  
 SearchResponse response = client.prepareSearch(**"blog"**).setTypes(**"article"**)  
 *//.setQuery(QueryBuilders.matchAllQuery()).get(); //查询所有  
 /\*\*  
 \* 字符串查询  
 \* QueryBuilders.queryStringQuery("搜索")默认匹配所有域  
 \* 如果如果添加.field("content")：表示只在content字段进行搜索  
 \*/  
 //.setQuery(QueryBuilders.queryStringQuery("搜索").field("content")).get();* .setQuery(QueryBuilders.*termQuery*(**"content"**,**"搜索"**)).get(); *//词条查询  
 //7.输出执行结果-response.sout()* SearchHits hits = response.getHits();  
 System.***out***.println(**"总记录数："** + hits.getTotalHits());  
 **for** (SearchHit hit : hits.getHits()) {  
 System.***out***.println(**"文章json: "** + hit.getSourceAsString());  
 System.***out***.println(**"获取title:"** + hit.getSource().get(**"title"**));  
 System.***out***.println(**"-------------------------------------------------"**);  
 }  
 *//8.关闭客户端-client.close()* 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

\* 通配符查询

\* 相似度查询

\* 范围查询

*//查询文档*@Test  
**public void** testQuery() **throws** Exception{  
 *//1.创建客户端访问对象-TransportClient* TransportClient client = **new** PreBuiltTransportClient(Settings.***EMPTY***);  
 *//2.创建Elasticsearch连接对象-new InetSocketTransportAddress(InetAddress.getByName(地址), 端口号)* InetSocketTransportAddress address = **new** InetSocketTransportAddress(InetAddress.*getByName*(**"127.0.0.1"**), 9300);  
 *//3.指定Elasticsearch连接地址与端口-client.addTransportAddress(address)* client.addTransportAddress(address);  
  
 SearchResponse response = client.prepareSearch(**"blog"**).setTypes(**"article"**)  
 *//.setQuery(QueryBuilders.matchAllQuery()) //查询所有  
 /\*\*  
 \* 字符串查询  
 \* QueryBuilders.queryStringQuery("搜索")默认匹配所有域  
 \* 如果如果添加.field("content")：表示只在content字段进行搜索  
 \*/  
 //.setQuery(QueryBuilders.queryStringQuery("搜索").field("content"))  
  
 //.setQuery(QueryBuilders.termQuery("content","搜索")) //词条查询  
  
 //通配符查询：试试"\*8"与"?8"的区别  
 //.setQuery(QueryBuilders.wildcardQuery("content", "?8"))*

*//.setQuery(QueryBuilders.fuzzyQuery("content", "elasticseerch")) //相似度查询* .setQuery(QueryBuilders.*rangeQuery*(**"id"**).from(1).to(5)) *//范围查询(日期、数值...)* .get();  
 *//7.输出执行结果-response.sout()* SearchHits hits = response.getHits();  
 System.***out***.println(**"总记录数："** + hits.getTotalHits());  
 ObjectMapper mapper = **new** ObjectMapper();  
 **for** (SearchHit hit : hits.getHits()) {  
 Article article = mapper.readValue(hit.getSourceAsString(), Article.**class**);  
 System.***out***.println(article);  
 System.***out***.println(**"-------------------------------------------------"**);  
 }  
 *//8.关闭客户端-client.close()* client.close();  
}

## 查询文档分页操作

### 【组合查询】(复杂查询)-boolQuery

布尔查询

must(QueryBuilders) : AND，求交集

mustNot(QueryBuilders): NOT 不包含

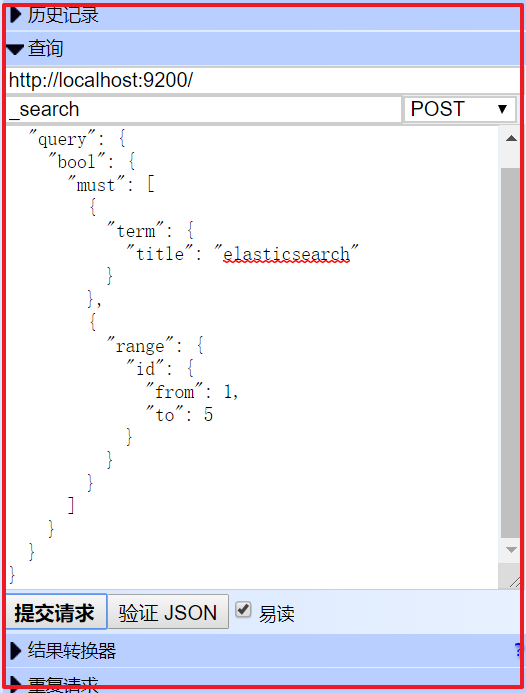
should(QueryBuilders):OR ，求并集

#### head查询

在定义json：放置到Elasticsearch的HEAD插件或者PostMan工具中（DSL表达式），使用restful风格编程，传递消息体

|  |
| --- |
| {  "query":{  "bool":{  "must":[  {  "term":{  "title":"elasticsearch"  }  },  {  "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：

*/\*\*  
 \* 组合查询  
 \* must(QueryBuilders) : AND，求交集  
 \* mustNot(QueryBuilders): NOT 不包含  
 \* should(QueryBuilders):OR ，求并集  
 \*/*@Test  
**public void** testBoolQuery() **throws** Exception{  
 *//1.创建客户端访问对象-TransportClient* TransportClient client = **new** PreBuiltTransportClient(Settings.***EMPTY***);  
 *//2.创建Elasticsearch连接对象-new InetSocketTransportAddress(InetAddress.getByName(地址), 端口号)* InetSocketTransportAddress address = **new** InetSocketTransportAddress(InetAddress.*getByName*(**"127.0.0.1"**), 9300);  
 *//3.指定Elasticsearch连接地址与端口-client.addTransportAddress(address)* client.addTransportAddress(address);  
  
 SearchResponse response = client.prepareSearch(**"blog"**).setTypes(**"article"**)  
 .setQuery(QueryBuilders.*boolQuery*()  
 .must(QueryBuilders.*termQuery*(**"content"**,**"elasticsearch"**))  
 .must(QueryBuilders.*rangeQuery*(**"id"**).from(1).to(5))  
 ).get();  
 *//7.输出执行结果-response.sout()* SearchHits hits = response.getHits();  
 System.***out***.println(**"总记录数："** + hits.getTotalHits());  
 ObjectMapper mapper = **new** ObjectMapper();  
 **for** (SearchHit hit : hits.getHits()) {  
 Article article = mapper.readValue(hit.getSourceAsString(), Article.**class**);  
 System.***out***.println(article);  
 System.***out***.println(**"-------------------------------------------------"**);  
 }  
 *//8.关闭客户端-client.close()* client.close();  
}

这里发现：

.must(QueryBuilders.*termQuery*("content", "elasticsearch")) // 模糊查询

能搜索到到结果。

而使用

.must(QueryBuilders.*wildcardQuery*("content", "ElasticSearch")) // 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是一个基于Lucene的搜索服务器18](http://127.0.0.1:9200/_analyze?analyzer=ik_max_word&pretty=true&text=ElasticSearch是一个基于Lucene的搜索服务器18%20)



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.*termQuery*("content", "ELASTicsearch".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

*//分页与排序*@Test  
**public void** testQueryPage() **throws** Exception{  
 *//1.创建客户端访问对象-TransportClient* TransportClient client = **new** PreBuiltTransportClient(Settings.***EMPTY***);  
 *//2.创建Elasticsearch连接对象-new InetSocketTransportAddress(InetAddress.getByName(地址), 端口号)* InetSocketTransportAddress address = **new** InetSocketTransportAddress(InetAddress.*getByName*(**"127.0.0.1"**), 9300);  
 *//3.指定Elasticsearch连接地址与端口-client.addTransportAddress(address)* client.addTransportAddress(address);  
  
 SearchResponse response = client.prepareSearch(**"blog"**).setTypes(**"article"**)  
 .setQuery(QueryBuilders.*boolQuery*()  
 .must(QueryBuilders.*termQuery*(**"content"**,**"Elasticsearch"**.toLowerCase()))  
 *//.must(QueryBuilders.rangeQuery("id").from(1).to(5))* )  
 *//设置分页参数  
 //setFrom()：从第几条开始检索，默认是0。  
 //setSize():每页查询记录数* .setFrom(0).setSize(20)  
 *//设置排序-addSort(域名,排序方式asc|desc)* .addSort(**"id"**, SortOrder.***ASC***)  
 .get();  
 *//7.输出执行结果-response.sout()* SearchHits hits = response.getHits();  
 System.***out***.println(**"总记录数："** + hits.getTotalHits());  
 ObjectMapper mapper = **new** ObjectMapper();  
 **for** (SearchHit hit : hits.getHits()) {  
 Article article = mapper.readValue(hit.getSourceAsString(), Article.**class**);  
 System.***out***.println(article);  
 System.***out***.println(**"-------------------------------------------------"**);  
 }  
 *//8.关闭客户端-client.close()* client.close();  
}

## 查询结果高亮操作

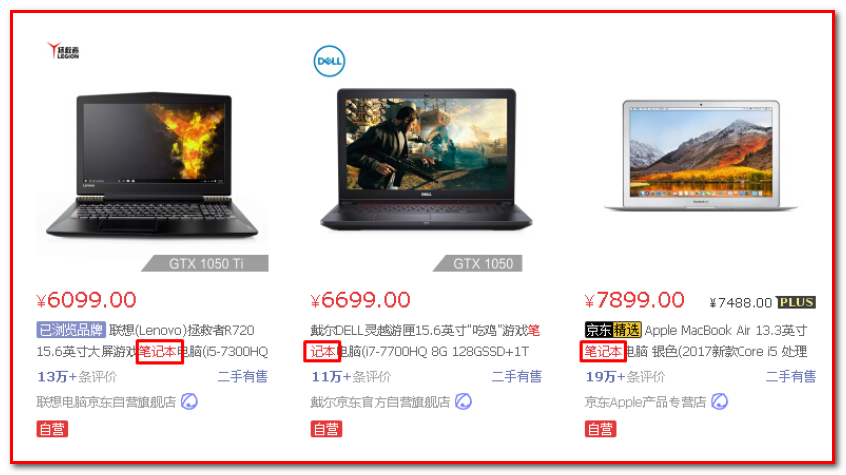
### 什么是高亮显示

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

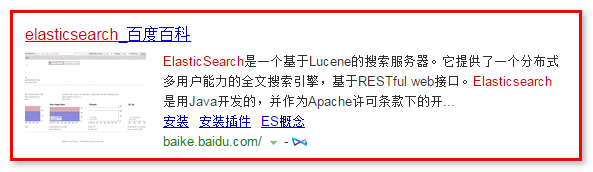
百度搜索关键字"传智播客"



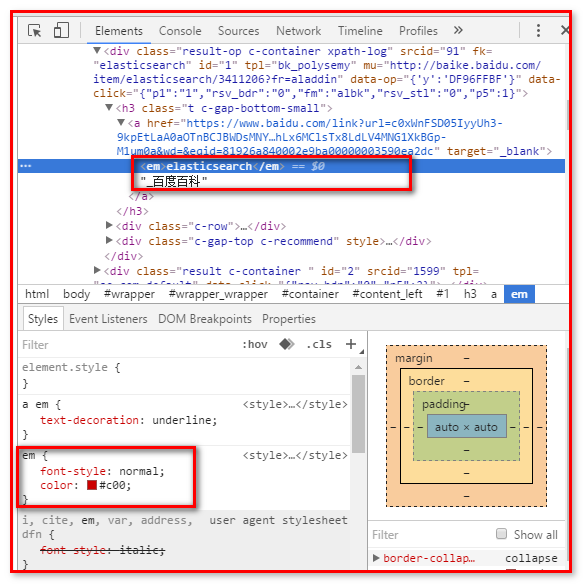
京东商城搜索"笔记本"



在百度搜索"elasticsearch"。



查看页面源码分析

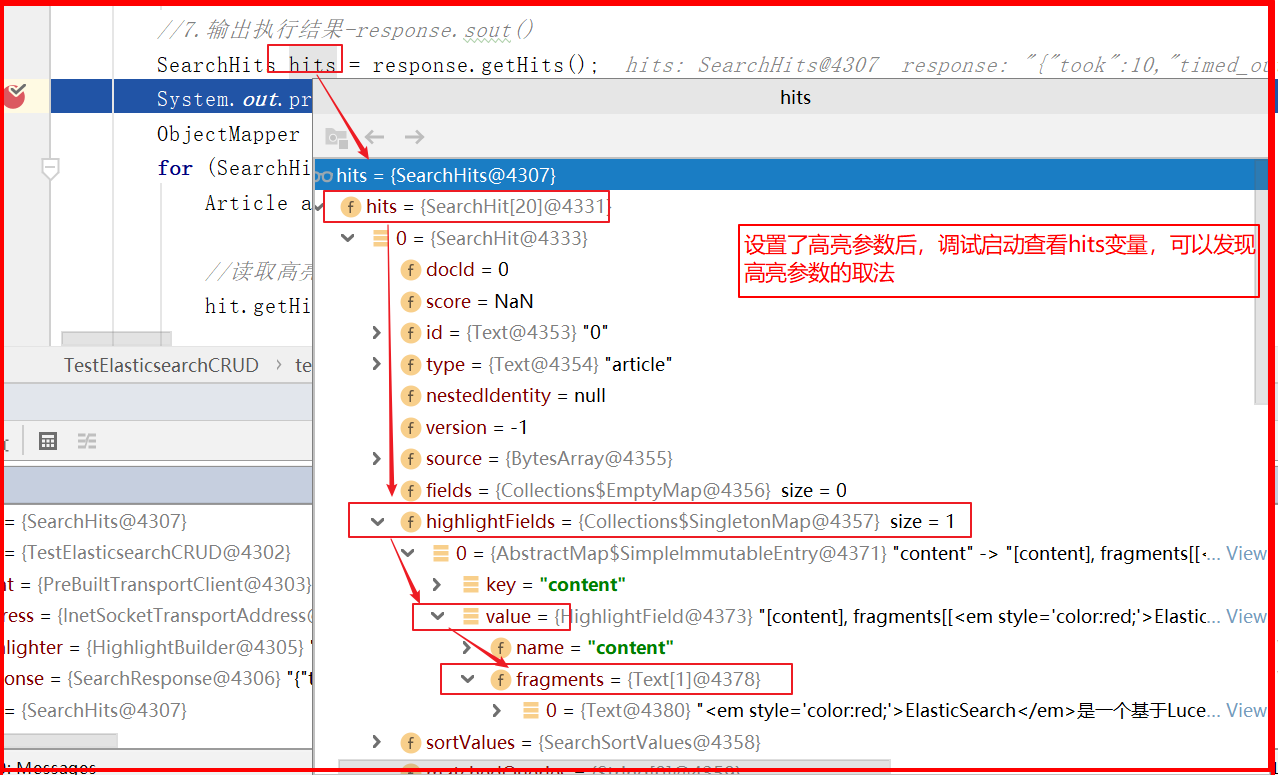


### 高亮显示的html分析

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？

经过测试：使用<em>高亮内容</em>



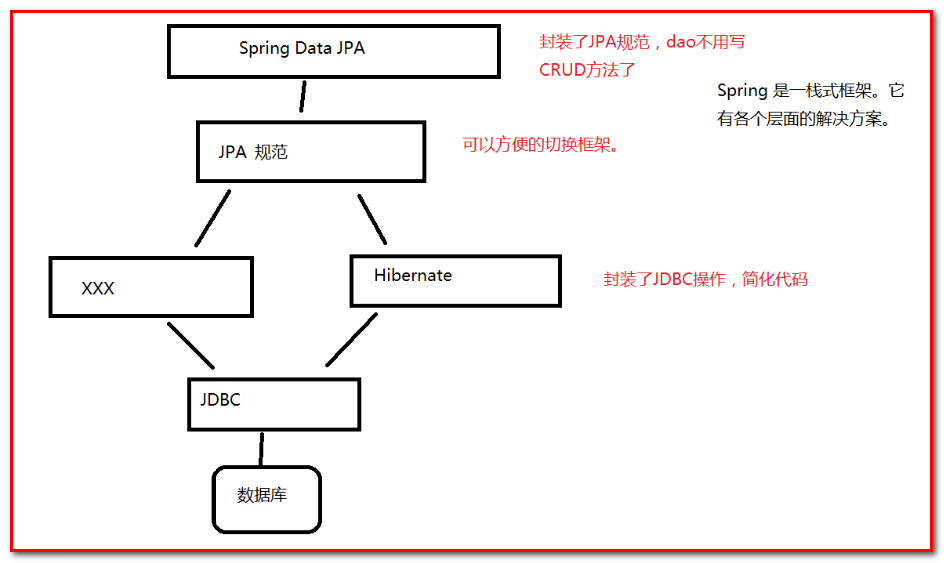
### 高亮显示代码实现

*//高亮查询*@Test  
**public void** testQueryHighlight() **throws** Exception{  
 *//1.创建客户端访问对象-TransportClient* TransportClient client = **new** PreBuiltTransportClient(Settings.***EMPTY***);  
 *//2.创建Elasticsearch连接对象-new InetSocketTransportAddress(InetAddress.getByName(地址), 端口号)* InetSocketTransportAddress address = **new** InetSocketTransportAddress(InetAddress.*getByName*(**"127.0.0.1"**), 9300);  
 *//3.指定Elasticsearch连接地址与端口-client.addTransportAddress(address)* client.addTransportAddress(address);  
  
 *//设置高亮参数* HighlightBuilder highlighter = **new** HighlightBuilder();  
 *//设置前缀* highlighter.preTags(**"<em style='color:red;'>"**);  
 *//设置后缀* highlighter.postTags(**"</em>"**);  
 *//设置高亮域* highlighter.field(**"content"**);  
  
 SearchResponse response = client.prepareSearch(**"blog"**).setTypes(**"article"**)  
 .setQuery(QueryBuilders.*boolQuery*()  
 .must(QueryBuilders.*termQuery*(**"content"**,**"Elasticsearch"**.toLowerCase()))  
 *//.must(QueryBuilders.rangeQuery("id").from(1).to(5))* )  
 *//设置分页参数  
 //setFrom()：从第几条开始检索，默认是0。  
 //setSize():每页查询记录数* .setFrom(0).setSize(20)  
 *//设置排序-addSort(域名,排序方式asc|desc)* .addSort(**"id"**, SortOrder.***ASC***)  
 *//高亮查询* .highlighter(highlighter)  
 .get();  
 *//7.输出执行结果-response.sout()* SearchHits hits = response.getHits();  
 System.***out***.println(**"总记录数："** + hits.getTotalHits());  
 ObjectMapper mapper = **new** ObjectMapper();  
 **for** (SearchHit hit : hits.getHits()) {  
 Article article = mapper.readValue(hit.getSourceAsString(), Article.**class**);  
  
 *//读取高亮数据* Text[] fragments = hit.getHighlightFields().get(**"content"**).fragments();  
 String content = **""**;  
 *//读取高亮碎片* **for** (Text text : fragments) {  
 content += text;  
 }  
 article.setContent(content);  
  
 System.***out***.println(article);  
 System.***out***.println(**"-------------------------------------------------"**);  
 }  
 *//8.关闭客户端-client.close()* client.close();  
}

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

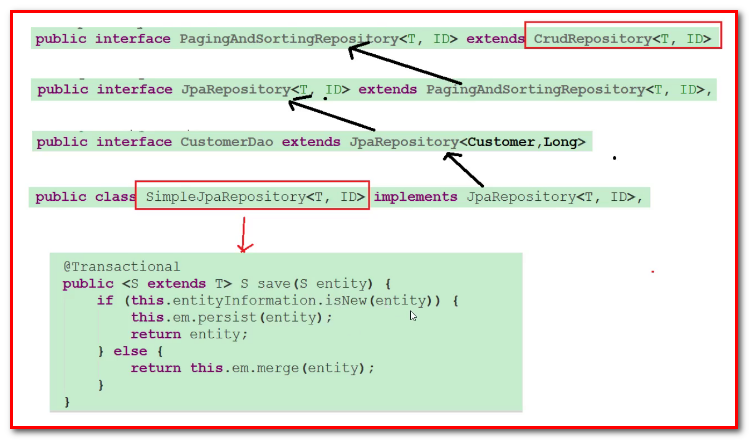
【回顾】

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



总结：JPA是一个规范，真正操作数据库的是Hibernate，而springdatajpa是对jpa的封装，将CRUD的方法封装到指定的方法中，操作的时候，只需要调用方法即可。

其中Spring Data JPA与JPA的关系原理



总结：

1：Jpa中所有的方法都封装到JpaRepository的接口中。

2：Springdatajpa的原理：通过jdk的动态代理机制，通过调用接口中的方法，从而产生代理对象，通过代理对象访问SimpleJPARepository类（这个类是真正实现了JpaRepository接口的类）

Spring Data Jpa的实现过程：

1：定义实体，实体类添加Jpa的注解

2：定义接口，接口要继承JpaRepository的接口

3：配置spring容器，applicationContext.xml

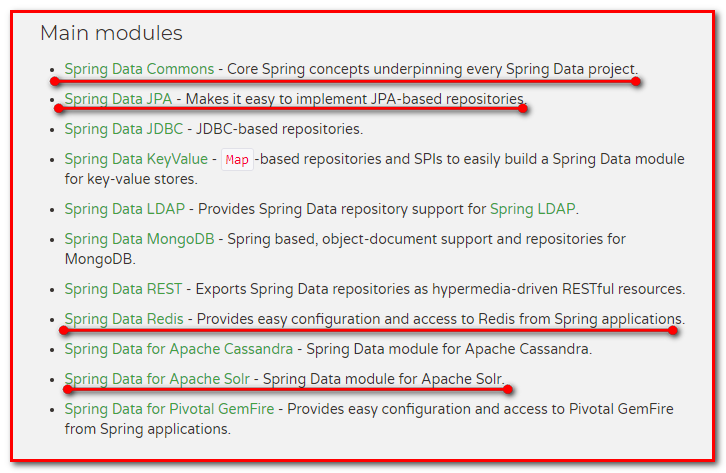
## Spring Data ElasticSearch简介

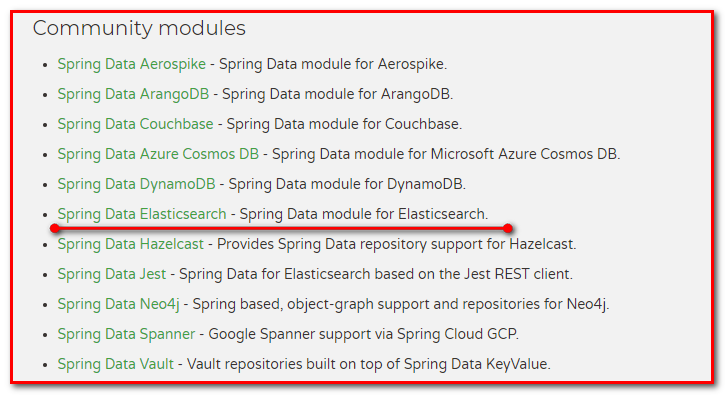
### 什么是Spring Data

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（Elasticsearch...）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索引库数据访问层。

官方网站：<http://projects.spring.io/spring-data-elasticsearch/>

## Spring Data ElasticSearch入门

### 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**>spring\_data\_elasticsearch\_demo</**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**>

#### 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**>

#### 编写实体Article

**public class** Article {  
 **private** Long **id**;  
 **private** String **title**;  
 **private** String **content**;  
  
 @Override  
 **public** String toString() {  
 **return "Article{"** +  
 **"id="** + **id** +  
 **", title='"** + **title** + **'\''** +  
 **", content='"** + **content** + **'\''** +  
 **'}'**;  
 }  
 **public** Long getId() {  
 **return id**;  
 }  
 **public void** setId(Long 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;  
 }  
}

#### 编写Dao

**package** com.itheima.spring.es.dao;  
  
**import** com.itheima.spring.es.pojo.Article;  
**import** org.springframework.data.elasticsearch.repository.ElasticsearchRepository;  
  
*/\*\*  
 \* es文章信息持久化接口  
 \** ***@author*** *Steven  
 \** ***@description*** *com.itheima.spring.es.dao  
 \*/***public interface** ArticleDao **extends** ElasticsearchRepository<Article,Long> {  
}

#### 编写Service

接口ArticleService.java

**package** com.itheima.spring.es.service;  
**import** com.itheima.spring.es.pojo.Article;  
*/\*\*  
 \* 文章信息业务逻辑接口  
 \** ***@author*** *Steven  
 \** ***@description*** *com.itheima.spring.es.service  
 \*/***public interface** ArticleService {  
 **public void** save(Article article);  
}

实现类ArticleServiceImpl.java

*/\*\*  
 \* 文章信息业务逻辑实现  
 \** ***@author*** *Steven  
 \** ***@description*** *com.itheima.spring.es.service.impl  
 \*/*@Service  
**public class** ArticleServiceImpl **implements** ArticleService {  
 @Autowired  
 **private** ArticleDao **articleDao**;  
   
 **public void** save(Article article) {  
 **articleDao**.save(article);  
 }  
}

### 配置spring\_data\_es

#### 配置applicationContext.xml

步骤：

<!-- 1：扫描Service包，创建Service的实体 -->

<!-- 2：扫描Dao包，自动创建实例，扫描所有继承ElasticsearchRepository接口的接口 -->

<!-- 3：配置Elasticsearch的连接对象Client -->

<!-- 4：Elasticsearch模版对象（底层使用模板操作，需要用spring创建，并注入client -->

*<?***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.service.impl"**/>  
 *<!-- 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.dao"**/>  
 *<!-- 3：配置Elasticsearch的连接对象Client -->* <**elasticsearch:transport-client id="client" cluster-nodes="127.0.0.1:9300" cluster-name="elasticsearch"**/>  
 *<!-- 4：Elasticsearch模版对象（底层使用模板操作，需要用spring创建，并注入client -->  
 <!--注意，此处的id只能是id="elasticsearchTemplate" 因为Dao在注册对象时用到了这个名字-->* <**bean id="elasticsearchTemplate" class="org.springframework.data.elasticsearch.core.ElasticsearchTemplate"**>  
 *<!--绑定真实连接，引用transport-client的id-->* <**constructor-arg ref="client"**/>  
 </**bean**>  
</**beans**>

#### 配置Article实体

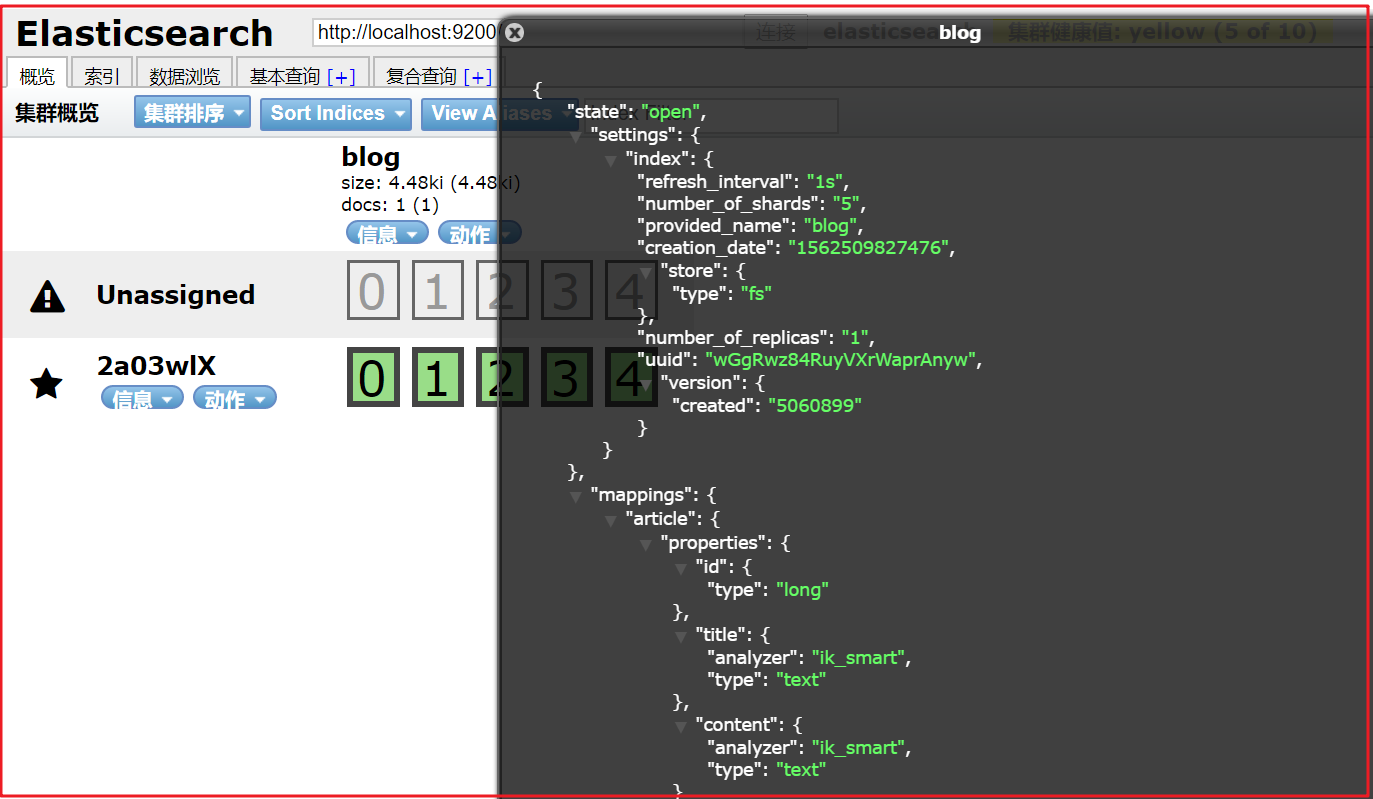
基于spring data elasticsearch注解配置索引、映射和实体的关系

**package** com.itheima.spring.es.pojo;  
**import** org.springframework.data.annotation.Id;  
**import** org.springframework.data.elasticsearch.annotations.Document;  
**import** org.springframework.data.elasticsearch.annotations.Field;  
**import** org.springframework.data.elasticsearch.annotations.FieldType;  
*/\*\*  
 \* 创建映射信息  
 \** ***@Document*** *文档对象(索引信息*、*文档类型)  
 \** ***@Id*** *映射文档id  
 \** ***@Field(*** *\* type = 数据类型,  
 \* analyzer = 索引时使用的分词器,  
 \* searchAnalyzer = (查询时使用的分词器，可以省略，默认与analyzer一样),  
 \* index = 是否索引，默认值true,  
 \* store = 是否存储，默认false* *,Elasticsearch默认使用\_source存放我们数据内容  
 \* )  
 \*/*@Document(indexName = **"blog"**,type = **"article"**)  
**public class** Article {  
 @Id  
 **private** Long **id**;  
 @Field(type = FieldType.***Text***,analyzer = **"ik\_smart"**,searchAnalyzer = **"ik\_smart"**,index = **true**,store = **false**)  
 **private** String **title**;  
 @Field(type = FieldType.***Text***,analyzer = **"ik\_smart"**,searchAnalyzer = **"ik\_smart"**,index = **true**,store = **false**)  
 **private** String **content**;  
  
 @Override  
 **public** String toString() {  
 **return "Article{"** +  
 **"id="** + **id** +  
 **", title='"** + **title** + **'\''** +  
 **", content='"** + **content** + **'\''** +  
 **'}'**;  
 }  
 **public** Long getId() {  
 **return id**;  
 }  
 **public void** setId(Long 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;  
 }  
}

#### 创建测试类SpringDataESTest

**package** com.itheima.es.test;  
  
**import** com.itheima.spring.es.dao.ArticleDao;  
**import** com.itheima.spring.es.pojo.Article;  
**import** com.itheima.spring.es.service.ArticleService;  
**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;  
  
*/\*\*  
 \** ***@author*** *Steven  
 \** ***@description*** *com.itheima.es.test  
 \*/*@RunWith(SpringJUnit4ClassRunner.**class**)  
@ContextConfiguration(locations = **"classpath:applicationContext.xml"**)  
**public class** SpringDataESTest {  
 @Autowired  
 **private** ElasticsearchTemplate **elasticsearchTemplate**;  
 @Autowired  
 **private** ArticleService **articleService**;  
  
 *//创建索引* @Test  
 **public void** testCreateIndex(){  
 *//创建索引* **elasticsearchTemplate**.createIndex(Article.**class**);  
 *//创建映射* **elasticsearchTemplate**.putMapping(Article.**class**);  
 }  
  
 *//创建文档* @Test  
 **public void** testCreateDouement(){  
 Article article = **new** Article();  
 article.setId(1L);  
 article.setTitle(**"什么是Elasticsearch"**);  
 article.setContent(**"ElasticSearch是一个基于Lucene的搜索服务器"**);  
 *//保存文档* **articleService**.save(article);  
 }  
}

查看Elasticsearch





## Spring Data ElasticSearch的常用操作

### 增删改查方法测试

#### 修改ArticleService.java

*/\*\*  
 \* 文章信息业务逻辑接口  
 \** ***@author*** *Steven  
 \** ***@description*** *com.itheima.spring.es.service  
 \*/***public interface** ArticleService {  
 *//新增与修改文档* **public void** save(Article article);  
 *//删除文档* **public void** delete(Article article);  
 *//批量导入数据* **public void** save100(List<Article> articleList);  
 *//查询全部* **public** Iterable<Article> findAll();  
 *//分页查询* **public** Page<Article> findPage(Pageable pageable);  
}

#### 修改ArticleSermpl.java

*/\*\*  
 \* 文章信息业务逻辑实现  
 \** ***@author*** *Steven  
 \** ***@description*** *com.itheima.spring.es.service.impl  
 \*/*@Service  
**public class** ArticleServiceImpl **implements** ArticleService {  
 @Autowired  
 **private** ArticleDao **articleDao**;  
  
 **public void** save(Article article) {  
 **articleDao**.save(article);  
 }  
  
 **public void** delete(Article article) {  
 **articleDao**.delete(article);  
 }  
  
 **public void** save100(List<Article> articleList) {  
 **articleDao**.saveAll(articleList);  
 }  
  
 **public** Iterable<Article> findAll() {  
 **return articleDao**.findAll();  
 }  
  
 **public** Page<Article> findPage(Pageable pageable) {  
 **return articleDao**.findAll(pageable);  
 }  
}

#### 测试类

@RunWith(SpringJUnit4ClassRunner.**class**)  
@ContextConfiguration(locations = **"classpath:applicationContext.xml"**)  
**public class** SpringDataESTest {  
 @Autowired  
 **private** ElasticsearchTemplate **elasticsearchTemplate**;  
 @Autowired  
 **private** ArticleService **articleService**;  
  
 *//创建索引* @Test  
 **public void** testCreateIndex(){  
 *//创建索引* **elasticsearchTemplate**.createIndex(Article.**class**);  
 *//创建映射* **elasticsearchTemplate**.putMapping(Article.**class**);  
 }  
  
 *//创建与修改文档* @Test  
 **public void** testCreateDouement(){  
 Article article = **new** Article();  
 article.setId(1L);  
 article.setTitle(**"什么是Elasticsearch"**);  
 article.setContent(**"ElasticSearch是一个基于Lucene的搜索服务器"**);  
 *//保存文档* **articleService**.save(article);  
 }  
  
 *//删除文档* @Test  
 **public void** testDeleteDoc(){  
 Article article = **new** Article();  
 article.setId(1L);  
 **articleService**.delete(article);  
 }  
  
 *//批量导入数据* @Test  
 **public void** save100(){  
 List<Article> articleList = **new** ArrayList<Article>();  
 **for** (**long** i = 0; i < 100; i++) {  
 Article article = **new** Article();  
 article.setId(i);  
 article.setTitle(**"什么是Elasticsearch"** + i);  
 article.setContent(**"ElasticSearch是一个基于Lucene的搜索服务器"** + i);  
  
 articleList.add(article);  
 }  
 **articleService**.save100(articleList);  
 }  
  
 *//查询所有数据* @Test  
 **public void** testFindAll(){  
 Iterable<Article> all = **articleService**.findAll();  
 **for** (Article article : all) {  
 System.***out***.println(article);  
 }  
 }  
  
 *//分页查询数据* @Test  
 **public void** testFindPage(){  
 *//设置分页参数：of(当前页0开始,每页查询条数,排序方式)* Pageable pageable = PageRequest.*of*(0,20,**new** Sort(Sort.Direction.***ASC***,**"id"**));  
 *//分页查询* Page<Article> page = **articleService**.findPage(pageable);  
 System.***out***.println(**"总页数："** + page.getTotalPages());  
 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

#### dao层实现

*/\*\*  
 \* es文章信息持久化接口  
 \** ***@author*** *Steven  
 \** ***@description*** *com.itheima.spring.es.dao  
 \*/***public interface** ArticleDao **extends** ElasticsearchRepository<Article,Long> {  
 *//根据内容查询* List<Article> findByContent(String content);  
  
 */\*\*  
 \* 根据内容分页查询  
 \** ***@param content*** *内容  
 \** ***@param pageable*** *分页参数  
 \** ***@return*** *分页结果  
 \*/* Page<Article> findByContent(String content,Pageable pageable);  
}

#### service层实现

接口ArticleService.java

*/\*\*  
 \* 文章信息业务逻辑接口  
 \** ***@author*** *Steven  
 \** ***@description*** *com.itheima.spring.es.service  
 \*/***public interface** ArticleService {  
 *//新增与修改文档* **public void** save(Article article);  
 *//删除文档* **public void** delete(Article article);  
 *//批量导入数据* **public void** save100(List<Article> articleList);  
 *//查询全部* **public** Iterable<Article> findAll();  
 *//分页查询* **public** Page<Article> findPage(Pageable pageable);  
 *//根据内容查询* List<Article> findByContent(String content);  
 */\*\*  
 \* 根据内容分页查询  
 \** ***@param content*** *内容  
 \** ***@param pageable*** *分页参数  
 \** ***@return*** *分页结果  
 \*/* Page<Article> findByContent(String content,Pageable pageable);  
}

实现类ArticleServiceImpl.java

*/\*\*  
 \* 文章信息业务逻辑实现  
 \** ***@author*** *Steven  
 \** ***@description*** *com.itheima.spring.es.service.impl  
 \*/*@Service  
**public class** ArticleServiceImpl **implements** ArticleService {  
 @Autowired  
 **private** ArticleDao **articleDao**;  
  
 **public void** save(Article article) {  
 **articleDao**.save(article);  
 }  
  
 **public void** delete(Article article) {  
 **articleDao**.delete(article);  
 }  
  
 **public void** save100(List<Article> articleList) {  
 **articleDao**.saveAll(articleList);  
 }  
  
 **public** Iterable<Article> findAll() {  
 **return articleDao**.findAll();  
 }  
  
 **public** Page<Article> findPage(Pageable pageable) {  
 **return articleDao**.findAll(pageable);  
 }  
  
 **public** List<Article> findByContent(String content) {  
 **return articleDao**.findByContent(content);  
 }  
  
 **public** Page<Article> findByContent(String content, Pageable pageable) {  
 **return articleDao**.findByContent(content,pageable);  
 }  
}

​

#### 测试代码

*/\*\*  
 \** ***@author*** *Steven  
 \** ***@description*** *com.itheima.es.test  
 \*/*@RunWith(SpringJUnit4ClassRunner.**class**)  
@ContextConfiguration(locations = **"classpath:applicationContext.xml"**)  
**public class** SpringDataESTest {  
 @Autowired  
 **private** ElasticsearchTemplate **elasticsearchTemplate**;  
 @Autowired  
 **private** ArticleService **articleService**;  
  
 *//创建索引* @Test  
 **public void** testCreateIndex(){  
 *//创建索引* **elasticsearchTemplate**.createIndex(Article.**class**);  
 *//创建映射* **elasticsearchTemplate**.putMapping(Article.**class**);  
 }  
  
 *//创建与修改文档* @Test  
 **public void** testCreateDouement(){  
 Article article = **new** Article();  
 article.setId(1L);  
 article.setTitle(**"什么是Elasticsearch"**);  
 article.setContent(**"ElasticSearch是一个基于Lucene的搜索服务器"**);  
 *//保存文档* **articleService**.save(article);  
 }  
  
 *//删除文档* @Test  
 **public void** testDeleteDoc(){  
 Article article = **new** Article();  
 article.setId(1L);  
 **articleService**.delete(article);  
 }  
  
 *//批量导入数据* @Test  
 **public void** save100(){  
 List<Article> articleList = **new** ArrayList<Article>();  
 **for** (**long** i = 0; i < 100; i++) {  
 Article article = **new** Article();  
 article.setId(i);  
 article.setTitle(**"什么是Elasticsearch"** + i);  
 article.setContent(**"ElasticSearch是一个基于Lucene的搜索服务器"** + i);  
 articleList.add(article);  
 }  
 **articleService**.save100(articleList);  
 }  
  
 *//查询所有数据* @Test  
 **public void** testFindAll(){  
 Iterable<Article> all = **articleService**.findAll();  
 **for** (Article article : all) {  
 System.***out***.println(article);  
 }  
 }  
 *//分页查询数据* @Test  
 **public void** testFindPage(){  
 *//设置分页参数：of(当前页0开始,每页查询条数,排序方式)* Pageable pageable = PageRequest.*of*(0,20,**new** Sort(Sort.Direction.***ASC***,**"id"**));  
 *//分页查询* Page<Article> page = **articleService**.findPage(pageable);  
 System.***out***.println(**"总页数："** + page.getTotalPages());  
 System.***out***.println(**"总记录数："** + page.getTotalElements());  
 **for** (Article article : page.getContent()) {  
 System.***out***.println(article);  
 }  
 }  
  
 @Test  
 **public void** testFindByContent(){  
 *//条件可以分词，支持通配符：\*搜索\*、搜索服务器* List<Article> articleList = **articleService**.findByContent(**"\*搜索\*"**);  
 **for** (Article article : articleList) {  
 System.***out***.println(article);  
 }  
 }  
  
 @Test  
 **public void** testFindByContentPage(){  
 *//条件可以分词，支持通配符：\*搜索\*、搜索服务器* Pageable pageable = PageRequest.*of*(0, 20);  
 Page<Article> page = **articleService**.findByContent(**"搜索服务器"**, pageable);  
 System.***out***.println(**"总页数："** + page.getTotalPages());  
 System.***out***.println(**"总记录数："** + page.getTotalElements());  
 **for** (Article article : page.getContent()) {  
 System.***out***.println(article);  
 }  
 }  
}

​

#### 测试：and的用法（布尔查询 must）

在ArticleDao中定义：

*/\*\*  
 \* 根据内容与标题分页查询  
 \** ***@param content*** *内容  
 \** ***@param title*** *标题  
 \** ***@param pageable*** *分页参数  
 \** ***@return*** *分页结果  
 \*/*Page<Article> findByContentAndTitle(String content,String title,Pageable pageable);

其它代码略…