Java第四阶段\_day01\_ElasticSearch案例

## **5.2.创建Demo工程**

我们新建一个demo，学习Elasticsearch

pom依赖：

<?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.bigdata.demo</groupId>  
    <artifactId>es-demo</artifactId>  
    <version>1.0-SNAPSHOT</version>  
​  
    <name>elasticsearch</name>  
    <description>Demo project for Spring Boot</description>  
​  
    <parent>  
        <groupId>org.springframework.boot</groupId>  
        <artifactId>spring-boot-starter-parent</artifactId>  
        <version>2.0.4.RELEASE</version>  
        <relativePath/> <!-- lookup parent from repository -->  
    </parent>  
​  
    <properties>  
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  
        <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>  
        <java.version>1.8</java.version>  
    </properties>  
​  
    <dependencies>  
       <dependency>  
            <groupId>org.springframework.boot</groupId>  
            <artifactId>spring-boot-starter-web</artifactId>  
        </dependency>  
        <dependency>  
            <groupId>org.springframework.boot</groupId>  
            <artifactId>spring-boot-starter-data-elasticsearch</artifactId>  
        </dependency>  
        <dependency>  
            <groupId>org.springframework.boot</groupId>  
            <artifactId>spring-boot-starter-test</artifactId>  
            <scope>test</scope>  
        </dependency>  
        <dependency>  
            <groupId>org.projectlombok</groupId>  
            <artifactId>lombok</artifactId>  
        </dependency>  
​  
    </dependencies>  
​  
    <build>  
        <plugins>  
            <plugin>  
                <groupId>org.springframework.boot</groupId>  
                <artifactId>spring-boot-maven-plugin</artifactId>  
            </plugin>  
        </plugins>  
    </build>  
</project>

启动类

package com.bigdata;  
​  
@SpringBootApplication  
public class EsApplication {  
​  
   public static void main(String[] args){  
       SpringApplication.run(EsApplication.class);  
    }  
}

application.yml文件配置：

spring:  
 data:  
   elasticsearch:  
     cluster-name: elasticsearch  
     cluster-nodes: 192.168.228.100:9300

## **5.3.索引操作**

### **5.3.1.创建索引和映射**

实体类

首先我们准备好实体类：

package com.bigdata.es.pojo;  
​  
public class Item {  
    private Long id;  
    private String title; //标题  
    private String category;// 分类  
    private String brand; // 品牌  
    private Double price; // 价格  
    private String images; // 图片地址  
}

映射

Spring Data通过注解来声明字段的映射属性，有下面的三个注解：

@Document 作用在类，标记实体类为文档对象，一般有两个属性

- indexName：对应索引库名称

- type：对应在索引库中的类型

- shards：分片数量，默认5

- replicas：副本数量，默认1

@Id作用在成员变量，标记一个字段作为id主键

@Field作用在成员变量，标记为文档的字段，并指定字段映射属性：

- type：字段类型，是是枚举：FieldType

- index：是否索引，布尔类型，默认是true

- store：是否存储，布尔类型，默认是false

- analyzer：分词器名称

示例：

package com.bigdata.es.pojo;  
​  
@AllArgsConstructor  
@NoArgsConstructor  
@Data  
@Document(indexName = "big",type = "item", shards = 1, replicas = 0)  
public class Item {  
​  
    //@Id  
    @Field(type = FieldType.Long)  
    private Long id;  
      
    @Field(type = FieldType.Text, analyzer = "ik\_smart")  
    private String title; //标题  
      
    @Field(type = FieldType.Keyword)  
    private String category;// 分类  
      
    @Field(type = FieldType.Keyword)  
    private String brand; // 品牌  
      
    @Field(type = FieldType.Double)  
    private Double price; // 价格  
      
    @Field(type = FieldType.Keyword,index = false)  
    private String images; // 图片地址  
}

创建索引

ElasticsearchTemplate中提供了创建索引的API：

- 可以根据类的信息自动生成，也可以手动指定indexName和Settings

映射

映射相关的API：

- 一样，可以根据类的字节码信息（注解配置）来生成映射，或者手动编写映射

我们这里采用类的字节码信息创建索引并映射：

package com.bigdata.es.demo;  
​  
@RunWith(SpringRunner.class)  
@SpringBootTest(classes = EsApplication.class)  
public class EsTest {  
​  
    @Autowired  
    public ElasticsearchTemplate template;  
​  
    @Test  
    public void  testCreate(){  
        // 创建索引，会根据Item类的@Document注解信息来创建  
        template.createIndex(Item.class);  
        // 配置映射，会根据Item类中的id、Field等字段来自动完成映射  
        template.putMapping(Item.class);  
        
   }  
}

查看索引信息

GET /big

结果：

{  
  "big": {  
    "aliases": {},  
    "mappings": {  
      "item": {  
        "properties": {  
          "brand": {  
            "type": "keyword"  
         },  
          "category": {  
            "type": "keyword"  
         },  
          "id": {  
            "type": "keyword"  
         },  
          "images": {  
            "type": "keyword",  
            "index": false  
         },  
          "price": {  
            "type": "double"  
         },  
          "title": {  
            "type": "text",  
            "analyzer": "ik\_smart"  
         }  
       }  
     }  
   },  
    "settings": {  
      "index": {  
        "refresh\_interval": "1s",  
        "number\_of\_shards": "1",  
        "provided\_name": "big",  
        "creation\_date": "1596942603048",  
        "store": {  
          "type": "fs"  
       },  
        "number\_of\_replicas": "0",  
        "uuid": "iNYUycw3Rkq9g1L5rDj4-A",  
        "version": {  
          "created": "6020499"  
       }  
     }  
   }  
 }  
}

### **5.3.2.删除索引**

示例：

   @Test  
    public void deleteIndex() {  
        template.deleteIndex(Item.class);  
   }

结果：

## **5.4.新增文档数据**

### **5.4.1.Repository接口**

Spring Data 的强大之处，就在于你不用写任何DAO处理，自动根据方法名或类的信息进行CRUD操作。只要你定义一个接口，然后继承Repository提供的一些子接口，就能具备各种基本的CRUD功能。

package com.bigdata.es.repository;  
​  
public interface ItemRepository extends ElasticsearchRepository<Item,Long>{  
​  
       
}

执行testCreate方法再次创建索引库，接下来测试新增数据：

### **5.4.2.新增一个对象**

package com.bigdata.es.demo;  
​  
​  
@RunWith(SpringRunner.class)  
@SpringBootTest(classes = EsApplication.class)  
public class EsTest {  
​  
   ...  
​  
    @Autowired  
    public ItemRepository itemRepository;  
​  
      
    @Test  
    public void index() {  
        Item item = new Item(1L, "小米手机7", " 手机",  
                "小米", 3499.00, "http://image.leyou.com/13123.jpg");  
        itemRepository.save(item);  
   }

去页面查询看看：

GET /big/\_search  
{  
    "query":{  
        "match\_all": {}  
   }  
}

{  
  "took": 1,  
  "timed\_out": false,  
  "\_shards": {  
    "total": 1,  
    "successful": 1,  
    "skipped": 0,  
    "failed": 0  
 },  
  "hits": {  
    "total": 1,  
    "max\_score": 1,  
    "hits": [  
     {  
        "\_index": "big",  
        "\_type": "item",  
        "\_id": "1",  
        "\_score": 1,  
        "\_source": {  
          "id": 1,  
          "title": "小米手机7",  
          "category": " 手机",  
          "brand": "小米",  
          "price": 3499,  
          "images": "http://image.leyou.com/13123.jpg"  
       }  
     }  
   ]  
 }  
}

### **5.4.3.批量新增**

代码：

package com.bigdata.es.demo;  
​  
...  
​  
@Test  
public void indexList() {  
        List<Item> list = new ArrayList<>();  
        list.add(new Item(1L, "小米手机7", "手机", "小米", 3299.00, "http://image.leyou.com/13123.jpg"));  
        list.add(new Item(2L, "坚果手机R1", "手机", "锤子", 3699.00, "http://image.leyou.com/13123.jpg"));  
        list.add(new Item(3L, "华为META10", "手机", "华为", 4499.00, "http://image.leyou.com/13123.jpg"));  
        list.add(new Item(4L, "小米Mix2S", "手机", "小米", 4299.00, "http://image.leyou.com/13123.jpg"));  
        list.add(new Item(5L, "荣耀V10", "手机", "华为", 2799.00, "http://image.leyou.com/13123.jpg"));  
        // 接收对象集合，实现批量新增  
        itemRepository.saveAll(list);  
}

再次去页面查询：

{  
  "took": 2,  
  "timed\_out": false,  
  "\_shards": {  
    "total": 1,  
    "successful": 1,  
    "skipped": 0,  
    "failed": 0  
 },  
  "hits": {  
    "total": 5,  
    "max\_score": 1,  
    "hits": [  
     {  
        "\_index": "big",  
        "\_type": "item",  
        "\_id": "1",  
        "\_score": 1,  
        "\_source": {  
          "id": 1,  
          "title": "小米手机7",  
          "category": " 手机",  
          "brand": "小米",  
          "price": 3499,  
          "images": "http://image.leyou.com/13123.jpg"  
       }  
     },  
     {  
        "\_index": "big",  
        "\_type": "item",  
        "\_id": "2",  
        "\_score": 1,  
        "\_source": {  
          "id": 2,  
          "title": "坚果手机R1",  
          "category": "手机",  
          "brand": "锤子",  
          "price": 3699,  
          "images": "http://image.leyou.com/13123.jpg"  
       }  
     },  
     {  
        "\_index": "big",  
        "\_type": "item",  
        "\_id": "3",  
        "\_score": 1,  
        "\_source": {  
          "id": 3,  
          "title": "华为META10",  
          "category": "手机",  
          "brand": "华为",  
          "price": 4499,  
          "images": "http://image.leyou.com/13123.jpg"  
       }  
     },  
     {  
        "\_index": "big",  
        "\_type": "item",  
        "\_id": "4",  
        "\_score": 1,  
        "\_source": {  
          "id": 4,  
          "title": "小米Mix2S",  
          "category": "手机",  
          "brand": "小米",  
          "price": 4299,  
          "images": "http://image.leyou.com/13123.jpg"  
       }  
     },  
     {  
        "\_index": "big",  
        "\_type": "item",  
        "\_id": "5",  
        "\_score": 1,  
        "\_source": {  
          "id": 5,  
          "title": "荣耀V10",  
          "category": "手机",  
          "brand": "华为",  
          "price": 2799,  
          "images": "http://image.leyou.com/13123.jpg"  
       }  
     }  
   ]  
 }  
}

### **5.4.4.修改**

修改和新增是同一个接口，区分的依据就是id，这一点跟我们在页面发起PUT请求是类似的。

### **5.4.5.删除**

package com.bigdata.es.demo;  
​  
@Test  
public  void delete(){  
       itemRepository.deleteById(2L);  
}

## **5.5.查询**

### **5.5.1.基本查询**

package com.bigdata.es.demo;  
​  
 @Test  
 public  void  testFind(){  
        Iterable<Item> all=itemRepository.findAll();  
        for (Item item:all){  
            System.out.println("item="+item);  
       }  
 }  
​  
 @Test  
 public void query(){  
        // 查询全部，并按照价格降序排序  
        Iterable<Item> items =  this.itemRepository.findAll(Sort.by("price").descending());  
        for (Item item : items) {  
            System.out.println("item = " + item);  
       }  
 }

### **5.5.2.自定义方法**

package com.bigdata.es.repository;  
​  
public interface ItemRepository extends ElasticsearchRepository<Item,Long>{  
​  
     /\*\*  
      \* 根据价格区间查询  
      \* @param begin  
      \* @param end  
      \* @return  
      \*/  
     List<Item> findByPriceBetween(Double begin, Double end);  
​  
}

不需要写实现类，然后我们直接去运行：

package com.bigdata.es.demo;  
​  
@Test  
public  void testFindBy(){  
       List<Item> items=this.itemRepository.findByPriceBetween(2000d,4000d);  
        for (Item item : items) {  
            System.out.println("item = " + item);  
       }  
}

### **5.5.3.自定义查询**

先来看最基本的match query：

package com.bigdata.es.demo;  
​  
@Test  
public void search(){  
        // 构建查询条件  
        NativeSearchQueryBuilder queryBuilder = new NativeSearchQueryBuilder();  
        // 添加基本分词查询  
        queryBuilder.withQuery(QueryBuilders.matchQuery("title", "小米手机"));  
        // 搜索，获取结果  
        Page<Item> items = this.itemRepository.search(queryBuilder.build());  
        // 总条数  
        long total = items.getTotalElements();  
        System.out.println("total = " + total);  
        for (Item item : items) {  
            System.out.println(item);  
       }  
}

### **5.5.4.分页查询**

利用NativeSearchQueryBuilder可以方便的实现分页：

package com.bigdata.es.demo;  
​  
 @Test  
 public void searchByPage(){  
        // 构建查询条件  
        NativeSearchQueryBuilder queryBuilder = new NativeSearchQueryBuilder();  
        // 添加基本分词查询  
        queryBuilder.withQuery(QueryBuilders.termQuery("category", "手机"));  
        // 分页：从0开始  
        int page = 0;  
        int size = 2;  
        queryBuilder.withPageable(PageRequest.of(page,size));  
​  
        // 搜索，获取结果  
        Page<Item> items = this.itemRepository.search(queryBuilder.build());  
        // 总条数  
        System.out.println("总条数 = " + items.getTotalElements());  
        // 总页数  
        System.out.println("总页数 = " + items.getTotalPages());  
        // 当前页  
        System.out.println("当前页：" + items.getNumber());  
        // 每页大小  
        System.out.println("每页大小：" + items.getSize());  
​  
        for (Item item : items) {  
            System.out.println(item);  
       }  
 }

### **5.5.5.排序**

排序也通用通过NativeSearchQueryBuilder完成：

package com.bigdata.es.demo;  
​  
 @Test  
 public void searchAndSort(){  
        // 构建查询条件  
        NativeSearchQueryBuilder queryBuilder = new NativeSearchQueryBuilder();  
        // 添加基本分词查询  
        queryBuilder.withQuery(QueryBuilders.termQuery("category", "手机"));  
        // 排序  
        queryBuilder.withSort(SortBuilders.fieldSort("price").order(SortOrder.ASC));  
​  
        // 搜索，获取结果  
        Page<Item> items = this.itemRepository.search(queryBuilder.build());  
        // 总条数  
        long total = items.getTotalElements();  
        System.out.println("总条数 = " + total);  
​  
        for (Item item : items) {  
            System.out.println(item);  
       }  
 }

## **5.6.聚合**

### **5.6.1.聚合为桶**

package com.bigdata.es.demo;  
​  
 @Test  
 public void testAgg(){  
        NativeSearchQueryBuilder queryBuilder = new NativeSearchQueryBuilder();  
        // 0、不查询任何结果  
        queryBuilder.withSourceFilter(new FetchSourceFilter(new String[]{""}, null));  
        // 1、添加一个新的聚合，聚合类型为terms，聚合名称为brands，聚合字段为brand  
        queryBuilder.addAggregation(AggregationBuilders.terms("brands").field("brand"));  
        // 2、查询,需要把结果强转为AggregatedPage类型  
        AggregatedPage<Item> aggPage = (AggregatedPage<Item>) this.itemRepository.search(queryBuilder.build());  
        // 3、解析  
        // 3.1、从结果中取出名为brands的那个聚合，  
        // 因为是利用String类型字段来进行的term聚合，所以结果要强转为StringTerm类型  
        StringTerms agg = (StringTerms) aggPage.getAggregation("brands");  
        // 3.2、获取桶  
        List<StringTerms.Bucket> buckets = agg.getBuckets();  
        // 3.3、遍历  
        for (StringTerms.Bucket bucket : buckets) {  
            // 3.4、获取桶中的key，即品牌名称  
            System.out.println(bucket.getKeyAsString());  
            // 3.5、获取桶中的文档数量  
            System.out.println(bucket.getDocCount());  
       }  
​  
 }