

ElasticSearch第二天

学习目标:

1. 能够使用Java客户端完成创建索引的操作
2. 能够使用Java客户端完成文档的增删改的操作
3. 能够使用Java客户端完成文档的查询操作
4. 能够完成文档的分页操作
5. 能够完成文档的高亮查询操作
6. 能够搭建Spring Data ElasticSearch的环境
7. 能够完成Spring Data ElasticSearch的文档基本增删改查操作
8. 能够掌握基本条件查询的方法命名规则

回顾ElasticSearch的主要概念

- 索引(indexes)：是文档存储的地方
- 类型(type)：索引的分类或分区
- 文档(document)：一个可以被索引的基本信息单元
- 字段(field)：文档中对不同类型进行分类存储的依靠字段
- 映射(mapping)：处理数据的方式和规则，对字段的限制，也就是字段的配置

类似概念

- 索引 → 数据库
- 类型 → 表
- 文档 → 行
- 字段 → 列
- 映射 → 字段配置ddl

回顾PostMan操作ElasticSearch的主要内容

1. 创建索引
2. 删除索引
3. 创建映射
4. 创建文档
5. 修改文档
6. 删除文档
7. 查询文档
 - 通过ID查询
 - query_string查询(带分词器)
 - term查询(关键词)

第一章 ElasticSearch Java客户端

1.1 客户端开发环境搭建

- 创建Maven工程
- 导入坐标

```
<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>
</dependencies>
```

1.2 创建索引index

目标：使用ES的编程工具包，编写Java代码，代码运行结果会在ES服务中创建索引

分析：Java客户端的操作，模仿我们通过postman发送请求调用RESTful接口调用的方式，本质还是请求获取响应。只不过使用的不是http协议而是tcp。

实现步骤：创建索引的接口调用：<http://localhost:9200/blog1>

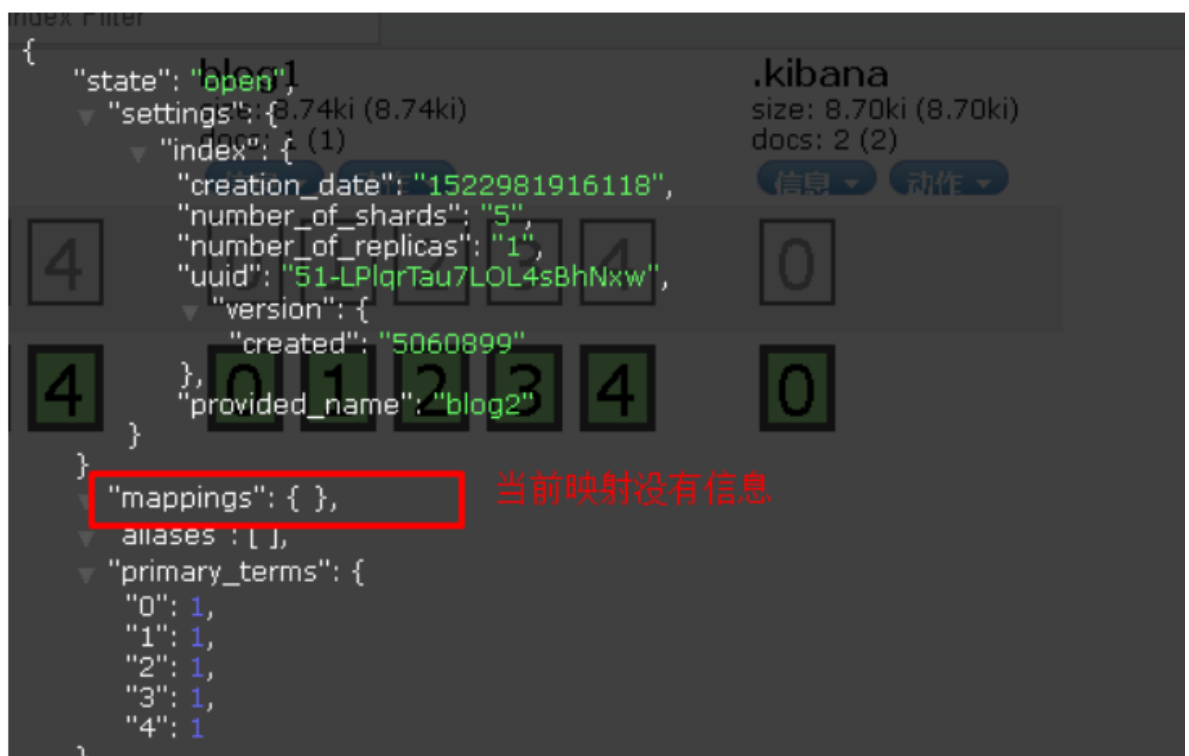
1. 客户端：Java中没有PostMan客户端，我们需要一个能够发送请求的客户端，Java中发送请求采用不是Http，而是TCP协议
2. 接口调用地址，tcp采用的是9300端口，地址还是127.0.0.1
3. 创建索引的请求对象：这个请求对象来自于索引管理权限客户端。
4. 执行请求发送
5. 请求完成返回响应对象
6. 关闭客户端

```
/**
 * 目标:创建索引
 * 步骤:
 * 1.连接ElasticSearch前置准备:确定连接地址,确定连接的集群名称。
 * 2.创建ElasticSearch连接客户端,客户端需要使用到配置信息。
 * 3.客户端创建索引的请求对象。
 * 4.请求对象执行发送请求操作,请求完成会获取响应对象。
 * 5.关闭客户端。
 */
@Test
public void createIndex() throws UnknownHostException {
    /** 1.连接ElasticSearch前置准备:
    // 确定连接地址,
    InetSocketAddress address = new
    InetSocketAddress(InetAddress.getByName("127.0.0.1"), 9300);
    // 确定连接的集群名称
    Settings settings = Settings.builder().put("cluster.name",
    "elasticsearch").build();
    /** 2.创建ElasticSearch连接客户端,客户端需要使用到配置信息
    PreBuiltTransportClient client = new PreBuiltTransportClient(settings);
    client.addTransportAddress(address);
```

```

/** 3.客户端创建索引的请求对象
CreateIndexRequestBuilder indexRequestBuilder =
client.admin().indices().prepareCreate("blog3");
/** 4.请求对象执行发送请求操作,请求完成会获取响应对象
CreateIndexResponse indexResponse = indexRequestBuilder.get();
System.out.println(indexResponse.index());
/** 5.关闭客户端
client.close();
}

```



1.3 配置索引映射mapping

分析：创建索引和配置映射本质都是一次请求，相比于索引创建映射的创建需要请求体内容。

1. 客户端对象：添加请求地址

2. 映射配置请求对象，映射的配置不能通过普通客户端权限操作，需要索引管理权限客户端
3. 映射请求对象设置请求体，请求体是JSON数据类型，需要JSON构建器来构建
4. 映射请求对象添加索引信息，添加类型信息
5. 映射管理权限客户端发送映射请求获取响应
6. 关闭客户端

```
/**
 * 目标:创建索引的type的映射
 * 1.获取客户端
 * 2.创建mapping请求对象:需要设置index,type,请求体
 * 3.创建JSON请求体:参考第一天的设置Mapping的JSON数据
 * 4.请求对象执行发送请求操作,请求完成会获取响应对象
 * 5.关闭客户端
 */
@Test
public void createMapping() throws IOException, ExecutionException,
InterruptedException {
    //配置集群名称,注意采用的事情TCP接口调用
    Settings settings = Settings.builder().put("cluster.name",
"elasticsearch").build();
    //配置请求地址和端口
    InetSocketAddress address = new
InetSocketAddress(InetAddress.getByName("127.0.0.1"), 9300);
    PreBuiltTransportClient client = new PreBuiltTransportClient(settings);
    client.addTransportAddress(address);
    /** 1.获取客户端
    /** 2.创建mapping请求对象:需要设置index,type,请求体
    PutMappingRequestBuilder request =
client.admin().indices().preparePutMapping("blog3");
    request.setType("article");//设置type
    /** 3.创建JSON请求体:参考第一天的设置Mapping的JSON数据
    XContentBuilder jsonBuilder = XContentFactory.jsonBuilder();
    jsonBuilder.startObject()
        .startObject("article")
        .startObject("properties")

        .startObject("id").field("type","long").field("store","yes").field("index","not_
analyzed")
            .endObject()

        .startObject("title").field("type","string").field("store","yes").field("index",
"analyzed").field("analyzer","ik_smart")
            .endObject()

        .startObject("content").field("type","string").field("store","yes").field("index
","analyzed").field("analyzer","ik_smart")
            .endObject()
            .endObject()
            .endObject()
            .endObject();
    request.setSource(jsonBuilder);//设置请求体
    /** 4.请求对象执行发送请求操作,请求完成会获取响应对象
    PutMappingResponse response = request.get();
    System.out.println(response.toString());
    /** 5.关闭客户端
    client.close();
}
```

The screenshot displays the Elasticsearch Kibana interface. At the top, the 'Elasticsearch' header shows the URL 'http://localhost:9200/' and a '连接' (Connect) button. Below the header, there are tabs for '集群概览' (Cluster Overview), '索引' (Indices), '数据浏览' (Data Explorer), '基本查询' (Basic Search), and '复合查询' (Advanced Search). The '集群概览' tab is active, showing a table of indices. The table has columns for 'Index Filter', 'Index Name', 'Size', 'Docs', 'Shards', and 'Replicas'. Two indices are listed: 'blog1' and 'blog2'. 'blog1' has a size of 8.74ki, 1 doc, and 5 shards. 'blog2' has a size of 810B, 0 docs, and 5 shards. A red box highlights the 'blog2' index row, showing its details. Below the table, there is a section for 'Unassigned' and '7TD9pkg' (a package). The 'blog2' index details are shown in a modal window. The modal has a title 'blog2' and a 'Index Filter' dropdown. The main content is a JSON representation of the index settings. A red box highlights the 'mappings' section, which defines the 'article' type with fields 'id' (integer), 'title' (text), and 'content' (text).

blog2
size: 810B (810B)
docs: 0 (0)
信息 动作

blog1
size: 8.74ki (8.74ki)
docs: 1 (1)
信息 动作

Unassigned

7TD9pkg
信息 动作

blog2

```
{
  "state": "open",
  "settings": {
    "index": {
      "creation_date": "1522985770585",
      "number_of_shards": "5",
      "number_of_replicas": "1",
      "uuid": "NdAglMd4Q-mZypLJCCEuWQ",
      "version": {
        "created": "5060899"
      },
      "provided_name": "blog2"
    }
  },
  "mappings": {
    "article": {
      "properties": {
        "id": {
          "store": true,
          "type": "integer"
        },
        "title": {
          "analyzer": "ik_smart",
          "store": true,
          "type": "text"
        },
        "content": {
          "analyzer": "ik_smart",
          "store": true,
          "type": "text"
        }
      }
    }
  },
  "aliases": [ ]
}
```

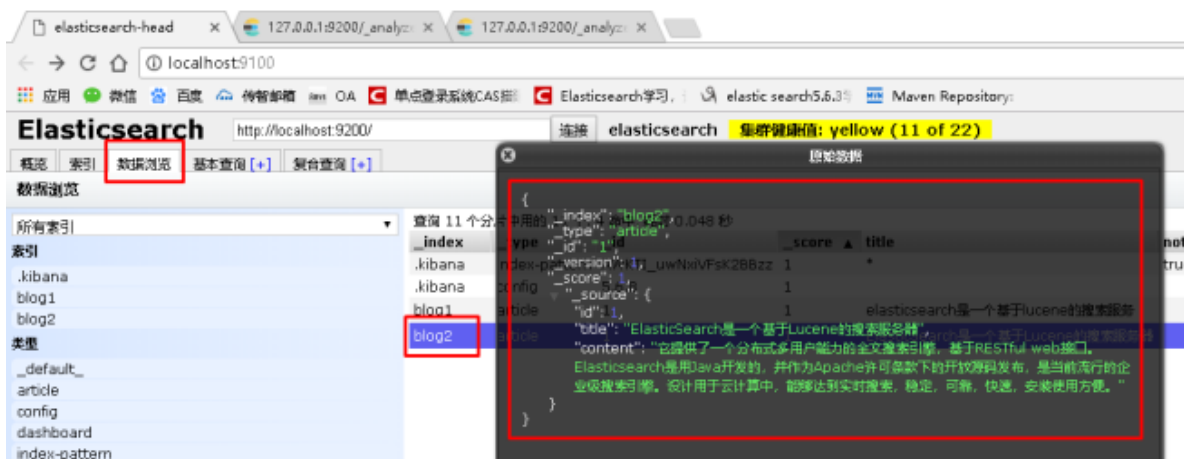
1.4 建立文档document

1.4.1 创建文档（通过XContentBuilder）

```

/**
 * 目标:创建文档
 * 1.获取客户端
 * 2.创建索引设置文档请求对象:需要设置index,type,id,请求体
 * 3.创建JSON请求体:参考第一天的设置创建文档的JSON数据
 * 4.请求对象执行发送请求操作,请求完成会获取响应对象
 * 5.关闭客户端
 */
@Test
public void createDocument() throws IOException {
    /** 1.获取客户端:设置集群名称,设置请求地址和端口TCP的
    Settings settings = Settings.builder().put("cluster.name",
    "elasticsearch").build();
    InetSocketAddress address = new
    InetSocketAddress(InetAddress.getByName("127.0.0.1"), 9300);
    PreBuiltTransportClient client = new PreBuiltTransportClient(settings);
    client.addTransportAddress(address);
    /** 3.创建JSON请求体:参考第一天的设置创建文档的JSON数据
    XContentBuilder jsonBuilder = XContentFactory.jsonBuilder();
    jsonBuilder.startObject()
        .field("id","1")
        .field("title","ElasticSearch是一个基于Lucene的搜索服务器")
        .field("content","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,能够达到实时搜索,稳定,可靠,快速,安装使用方便。")
        .endObject();
    /** 2.创建索引设置文档请求对象:需要设置index,type,id,请求体
    IndexRequestBuilder requestBuilder = client.prepareIndex("blog3", "article",
    "1");
    requestBuilder.setSource(jsonBuilder);
    /** 4.请求对象执行发送请求操作,请求完成会获取响应对象
    IndexResponse indexResponse = requestBuilder.get();
    System.out.println(indexResponse.toString());
    /** 5.关闭客户端
    client.close();
}

```



1.4.2 创建文档（使用Jackson转换实体）

1) 创建Article实体

```
public class Article {
    private Integer id;
    private String title;
    private String content;
    getter/setter...
}
```

2) 添加jackson坐标

```
<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>
```

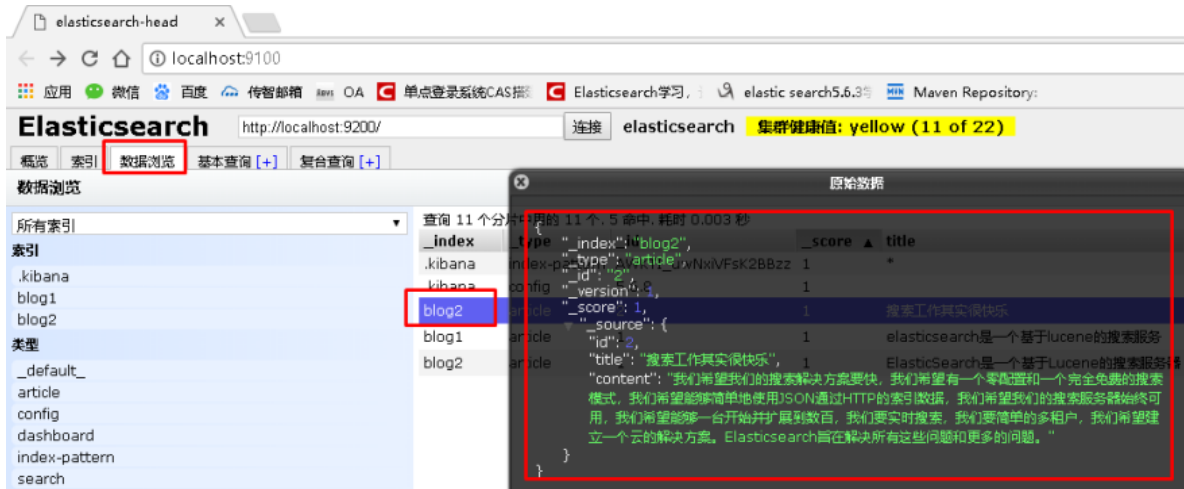
3) 代码实现

```
/**
 * 目标:创建文档(第二种方式对象转换且JSON字符串)
 * 1.获取客户端
 * 2.创建索引设置文档请求对象:需要设置index,type,id,请求体
 * 3.创建文章对象,对象转换JSON的字符串;请求对象设置字符串
 * 4.请求对象执行发送请求操作,请求完成会获取响应对象
 * 5.关闭客户端
 *
 * 其实还有:第三种方式JSONString,第四种方式map
 */
@Test
public void createDocumentTwo() throws UnknownHostException,
JsonProcessingException {
    /** 1.获取客户端
    PreBuiltTransportClient client = TransportClientUtil.getClient();
    /** 2.创建索引设置文档请求对象:需要设置index,type,id,请求体
    IndexRequestBuilder indexRequestBuilder = client.prepareIndex("blog3",
"article", "1");
    /** 3.创建文章对象,对象转换JSON的字符串;请求对象设置字符串
    Article article = new Article();
    article.setId(1);
    article.setTitle("ElasticSearch是一个基于Lucene的搜索服务器");
    article.setContent("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,能够达到实时搜索,稳定,可靠,快速,安装使用方便。");
    ObjectMapper objectMapper = new ObjectMapper();
    String jsonArticle = objectMapper.writeValueAsString(article);
    indexRequestBuilder.setSource(jsonArticle);
    /** 4.请求对象执行发送请求操作,请求完成会获取响应对象
```

```

IndexResponse indexResponse = indexRequestBuilder.get();
/** 5.关闭客户端
client.close();
}

```



1.5 查询文档操作

1.5.1 关键词查询

```

/**
 * 目标:term关键词的查询
 * 1.获取客户端
 * 2.创建搜索请求对象:需要设置index,type,查询对象
 * 3.创建Term查询对象,设置查询字段,和关键词
 * 4.请求对象执行发送请求操作,请求完成会获取响应对象
 * 5.响应对象中获取命中数据,循环遍历输出
 * 6.关闭客户端
 */
@Test
public void testTermQuery() throws UnknownHostException {
    /** 1.获取客户端
    PreBuiltTransportClient client = TransportClientUtil.getClient();
    /** 2.创建搜索请求对象:需要设置index,type,查询对象
    SearchRequestBuilder searchRequestBuilder = client.prepareSearch("blog3");
    /** 3.创建Term查询对象,设置查询字段,和关键词
    searchRequestBuilder.setQuery(QueryBuilders.termQuery("title","搜索"));
    /** 4.请求对象执行发送请求操作,请求完成会获取响应对象
    SearchResponse searchResponse = searchRequestBuilder.get();
    /** 5.响应对象中获取命中数据,循环遍历输出
    SearchHits hits = searchResponse.getHits();
    for (SearchHit hit : hits) {
        System.out.println(hit.getSourceAsString());
        System.out.println(hit.getSource().get("id"));
        System.out.println(hit.getSource().get("title"));
        System.out.println(hit.getSource().get("content"));
    }
    /** 6.关闭客户端
    client.close();
}

```


1.5.2 字符串查询

```
/**
 * 目标:查询,带分词器的字符串查询
 * 1.获取客户端
 * 2.创建搜索请求对象:需要设置index,type,查询对象
 * 3.创建Query_string查询对象,传入查询字符串
 * 4.请求对象执行发送请求操作,请求完成会获取响应对象
 * 5.响应对象中获取命中数据,循环遍历输出
 * 6.关闭客户端
 */
@Test
public void testStringQuery() throws UnknownHostException {
    //1.获取客户端
    PreBuiltTransportClient client = TransportClientUtil.getClient();
    //2.创建搜索请求对象:需要设置index,type,查询对象
    SearchRequestBuilder searchRequestBuilder = client.prepareSearch("blog2");
    searchRequestBuilder.setTypes("article");
    //3.创建Query_string查询对象,传入查询字符串
    searchRequestBuilder.setQuery(QueryBuilders.queryStringQuery("搜索"));
    //4.请求对象执行发送请求操作,请求完成会获取响应对象
    SearchResponse searchResponse = searchRequestBuilder.get();//发送请求获取响应
    //5.响应对象中获取命中数据,循环遍历输出
    SearchHits hits = searchResponse.getHits();
    System.out.println("搜索结果有:[" + hits.getTotalHits() + "]条");
    Iterator<SearchHit> iterator = hits.iterator();
    while (iterator.hasNext()){
        SearchHit next = iterator.next();
        System.out.println("ID:"+next.getSource().get("id"));
        System.out.println("title:"+next.getSource().get("title"));
        System.out.println("content:"+next.getSource().get("content"));
    }
    //6.关闭客户端
    client.close();
}
```

1.5.3 使用文档ID查询文档

```
/**
 * 目标:查询,使用ID查询
 * 1.获取客户端
 * 2.创建搜索请求对象:需要设置index,type,查询对象
 * 3.创建idsQuery查询对象,填入id
 * 4.请求对象执行发送请求操作,请求完成会获取响应对象
 * 5.响应对象中获取命中数据,循环遍历输出
 * 6.关闭客户端
 */
@Test
public void findById() throws UnknownHostException {
    //1.获取客户端
    PreBuiltTransportClient client = TransportClientUtil.getClient();
    //2.创建搜索请求对象:需要设置index,type,查询对象
    SearchRequestBuilder searchRequestBuilder = client.prepareSearch("blog1");
    searchRequestBuilder.setTypes("article");
    //3.创建idsQuery查询对象,填入id
```

```

searchRequestBuilder.setQuery(QueryBuilders.idsQuery().addIds("1"));
//4. 请求对象执行发送请求操作,请求完成会获取响应对象
SearchResponse searchResponse = searchRequestBuilder.get();
//5. 响应对象中获取命中数据,循环遍历输出
SearchHits hits = searchResponse.getHits();
System.out.println("搜索结果有:[" + hits.getTotalHits() + "]条");
Iterator<SearchHit> iterator = hits.iterator();
while (iterator.hasNext()){
    SearchHit next = iterator.next();
    System.out.println("ID:"+next.getSource().get("id"));
    System.out.println("title:"+next.getSource().get("title"));
    System.out.println("content:"+next.getSource().get("content"));
}
//6. 关闭客户端
client.close();
}

```

1.6 查询文档分页操作

1.6.1 批量插入数据

```

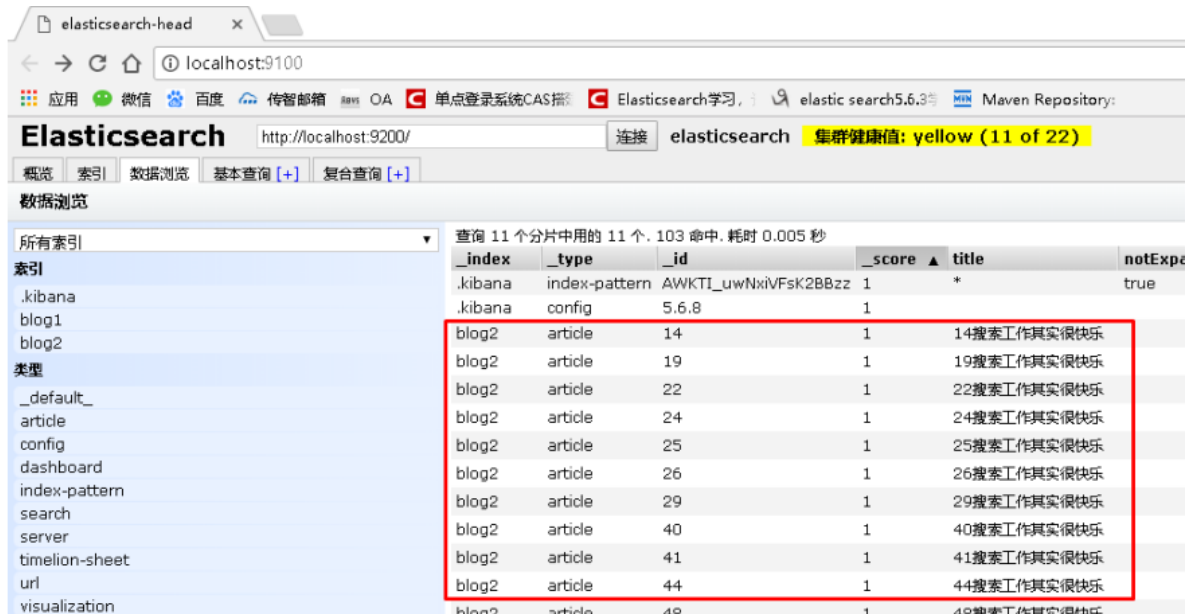
/**
 * 分页查询的准备:批量插入100条数据
 * 1. 获取客户端
 * 循环100次2,3,4的操作:
 * 2. 创建索引新增文档请求对象:需要设置index,type,id,请求体
 * 3. 创建JSON请求体:参考第一天的设置创建文档的JSON数据
 * 4. 请求对象执行发送请求操作,请求完成会获取响应对象
 *
 * 5. 关闭客户端
 */
@Test
public void testSaveList() throws UnknownHostException, JsonProcessingException
{
    //1. 获取客户端
    PreBuiltTransportClient client = TransportClientUtil.getClient();
    ObjectMapper objectMapper = new ObjectMapper();
    //循环100次2,3,4的操作:
    for (int i = 0; i < 100; i++) {
        //3. 创建JSON请求体:参考第一天的设置创建文档的JSON数据
        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。");
        //对象转换为JSON数据
        String jsonArticle = objectMapper.writeValueAsString(article);
        //2. 创建索引新增文档请求对象:需要设置index,type,id,请求体
        IndexRequestBuilder indexRequestBuilder = client.prepareIndex("blog2",
        "article", "" + i);
        indexRequestBuilder.setSource(jsonArticle, XContentType.JSON);//设置请求体
        //4. 请求对象执行发送请求操作,请求完成会获取响应对象
        IndexResponse indexResponse = indexRequestBuilder.get();
    }
}

```

```

        System.out.println(jsonArticle);
        System.out.println(indexResponse.toString());
    }
    //5.关闭客户端
    client.close();
}

```



1.6.2 分页查询

```

/**
 * 目标:查询,分页查询
 * 1.获取客户端
 * 2.创建搜索请求对象:需要设置index,type,查询对象
 * 3.创建查询所有记录对象,并设置分页信息:form表示起始页,size表示每页多少条
 * 4.请求对象执行发送请求操作,请求完成会获取响应对象
 * 5.响应对象中获取命中数据,循环遍历输出
 * 6.关闭客户端
 */
@Test
public void findByPageable() throws UnknownHostException {
    //1.获取客户端
    PreBuiltTransportClient client = TransportClientUtil.getClient();
    //2.创建搜索请求对象:需要设置index,type,查询对象
    SearchRequestBuilder searchRequestBuilder = client.prepareSearch("blog2");
    searchRequestBuilder.setTypes("article");
    //3.创建查询所有记录对象,并设置分页信息:form表示起始页,size表示每页多少条
    searchRequestBuilder.setQuery(QueryBuilders.matchAllQuery()); //默认每页10条
    searchRequestBuilder.setFrom(0); //form表示起始页
    searchRequestBuilder.setSize(20); //size表示每页多少条
    //4.请求对象执行发送请求操作,请求完成会获取响应对象
    SearchResponse searchResponse = searchRequestBuilder.get();
    //5.响应对象中获取命中数据,循环遍历输出
    SearchHits hits = searchResponse.getHits();
    System.out.println("搜索结果有:[" + hits.getTotalHits() + "]条");
    System.out.println("当前页有:[" + hits.getHits().length + "]条");
    Iterator<SearchHit> iterator = hits.iterator();
    while (iterator.hasNext()){
        SearchHit next = iterator.next();
    }
}

```

```

        System.out.println("ID:"+next.getSource().get("id"));
        System.out.println("title:"+next.getSource().get("title"));
        System.out.println("content:"+next.getSource().get("content"));
    }
    //6.关闭客户端
    client.close();
}

```

```

查询结果有: 100条
{"id":80,"title":"80搜索工作其实很快乐","content":"80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
id:80
title:80搜索工作其实很快乐
content:80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HTT
-----
{"id":79,"title":"79搜索工作其实很快乐","content":"79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
id:79
title:79搜索工作其实很快乐
content:79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HTT
-----
{"id":78,"title":"78搜索工作其实很快乐","content":"78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
id:78
title:78搜索工作其实很快乐
content:78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HTT

```

1.7 查询结果高亮操作


1.7.1 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"



GTX 1050 Ti

¥6099.00


已浏览品牌 联想(Lenovo)拯救者R720
15.6英寸大屏游戏笔记本电脑(i5-7300HQ
13万+条评价 二手有售
联想电脑京东自营旗舰店
自营



GTX 1050

¥6699.00

戴尔DELL灵越游匣15.6英寸"吃鸡"游戏笔记本电脑(i7-7700HQ 8G 128GSSD+1T
11万+条评价 二手有售
戴尔京东自营官方旗舰店
自营



¥7899.00 ¥7488.00 PLUS

京东精选 Apple MacBook Air 13.3英寸笔记本电脑 银色(2017新款Core i5 处理
19万+条评价 二手有售
京东Apple产品专营店
自营

1.7.2 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：

¥4099.00

华硕 (ASUS) 笔记本电脑
超薄13.3英寸轻薄手提
已有3086人评价



¥2799.00



GTX 1050 Ti

¥6099.00

已浏览品牌 联想(Lenovo)拯救者R720 15.6英寸大屏游戏笔记本电脑(i5-7300HQ
13万+条评价 二手有售
联想电脑京东自营旗舰店
自营



GTX 1050

¥6699.00

戴尔DELL灵越游匣15.6英寸"吃鸡"游戏笔记本电脑(i7-7700HQ 8G 128GSSD+1T
11万+条评价 二手有售
戴尔京东自营官方旗舰店
自营

font.skcolor_ljg | 36 x 14

是添加的样式的文字

```

<div class="p-price">...</div>
<div class="p-name p-name-type-2">
  <a target="_blank" title="全新升级，吃鸡利器！拯救者“黑金”耀眼归来！“双硬盘”硬实力，玩儿视觉来“4G”！ 详
  5512841.html" onclick="searchlog(1,5512841,0,1,',', 'flagsC1k=1077949064')">
    <em>
      <span class="p-tag">已浏览品牌</span>
      联想(Lenovo)拯救者R720 15.6英寸大屏游戏
      <font class="skcolor_ljg">笔记本</font> == $0
      电脑(i5-7300HQ 8G 1T+128G SSD 61X105011 4G IPS 黑金)"
    </em>
    <i class="promo-words" id="J_AD_5512841">全新升级，吃鸡利器！拯救者“黑金”耀眼归来！“双硬盘”硬实力，
  </i>
  </a>

```

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

1.7.3 高亮显示代码实现

```

/**
 * 目标:搜索结果高亮
 * 1. 获取客户端
 * 2. 创建搜索请求对象:需要设置index,type,查询对象
 * 3. 创建Term查询对象,设置查询字段,和关键词
 * 3.1 查询对象设置数据高亮配置:配置高亮标签font,配置高亮字段title
 * 4. 请求对象执行发送请求操作,请求完成会获取响应对象
 * 5. 响应对象中获取命中数据,循环遍历输出

```

```

* 6.关闭客户端
*/
@Test
public void testHighlight() throws UnknownHostException {
    //1.获取客户端
    PreBuiltTransportClient client = TransportClientUtil.getClient();
    //2.创建搜索请求对象:需要设置index,type,查询对象
    SearchRequestBuilder searchRequestBuilder = client.prepareSearch("blog2");
    searchRequestBuilder.setTypes("article");
    //3.创建Term查询对象,设置查询字段,和关键词
    searchRequestBuilder.setQuery(QueryBuilders.termQuery("title", "搜索")); //默认
    每页10条

    //3.1 查询对象设置数据高亮配置
    HighlightBuilder highlightBuilder = new HighlightBuilder();
    highlightBuilder.preTags("<font style='color:red'>");
    highlightBuilder.postTags("</font>");
    highlightBuilder.field("title");

    searchRequestBuilder.highlighter(highlightBuilder);

    //4.请求对象执行发送请求操作,请求完成会获取响应对象
    SearchResponse searchResponse = searchRequestBuilder.get();
    //5.响应对象中获取命中数据,循环遍历输出
    SearchHits hits = searchResponse.getHits();
    System.out.println("共搜索到:[" + hits.getTotalHits() + "]条结果");
    for (SearchHit hit : hits) {
        System.out.println("String方式打印高亮内容:");
        System.out.println(hit.getSourceAsString());
        System.out.println("Map方式打印高亮内容:");
        System.out.println(hit.getHighlightFields());
        Map<String, HighlightField> highlightFieldMap =
            hit.getHighlightFields();

        Text[] titles = hit.getHighlightFields().get("title").fragments();
        for (Text title : titles) {
            System.out.println(title);
        }
    }
    //6.关闭客户端
    client.close();
}

```

```

{"id":36,"title":"36搜索工作其实很快乐","content":"36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
Map方式打印高亮内容
{title=[title], fragments[[36<font style='color:red'>搜索</font>工作其实很快乐]]}
遍历高亮集合, 打印高亮片段:
36<font style='color:red'>搜索</font>工作其实很快乐
String方式打印文档搜索内容:
{"id":38,"title":"38搜索工作其实很快乐","content":"38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
Map方式打印高亮内容
{title=[title], fragments[[38<font style='color:red'>搜索</font>工作其实很快乐]]}
遍历高亮集合, 打印高亮片段:
38<font style='color:red'>搜索</font>工作其实很快乐
String方式打印文档搜索内容:
{"id":43,"title":"43搜索工作其实很快乐","content":"43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
Map方式打印高亮内容
{title=[title], fragments[[43<font style='color:red'>搜索</font>工作其实很快乐]]}
遍历高亮集合, 打印高亮片段:
43<font style='color:red'>搜索</font>工作其实很快乐

```


第二章 Spring Data Elasticsearch 使用

2.1 Spring Data Elasticsearch简介

2.1.1 什么是Spring Data

Spring Data是一个用于简化数据访问，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。Spring Data可以极大的简化数据操作的写法，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：

Main modules

- **Spring Data Commons** - Core Spring concepts underpinning every Spring Data project.
- **Spring Data Gemfire** - Provides easy configuration and access to GemFire from Spring applications.
- **Spring Data JPA** - Makes it easy to implement JPA-based repositories.
- **Spring Data JDBC** - JDBC-based repositories.
- **Spring Data KeyValue** - Map-based repositories and SPIs to easily build a Spring Data module for key-value stores.
- **Spring Data LDAP** - Provides Spring Data repository support for Spring LDAP.
- **Spring Data MongoDB** - Spring based, object-document support and repositories for MongoDB.
- **Spring Data REST** - Exports Spring Data repositories as hypermedia-driven RESTful resources.
- **Spring Data Redis** - Provides easy configuration and access to Redis from Spring applications.
- **Spring Data for Apache Cassandra** - Spring Data module for Apache Cassandra.
- **Spring Data for Apache Solr** - Spring Data module for Apache Solr.

Community modules

- **Spring Data Aerospike** - Spring Data module for Aerospike.
- **Spring Data ArangoDB** - Spring Data module for ArangoDB.
- **Spring Data Couchbase** - Spring Data module for Couchbase.
- **Spring Data Azure DocumentDB** - Spring Data module for Microsoft Azure DocumentDB.
- **Spring Data DynamoDB** - Spring Data module for DynamoDB.
- **Spring Data Elasticsearch** - Spring Data module for Elasticsearch.
- **Spring Data Hazelcast** - Provides Spring Data repository support for Hazelcast.
- **Spring Data Jest** - Spring Data for Elasticsearch based on the Jest REST client.
- **Spring Data Neo4j** - Spring based, object-graph support and repositories for Neo4j.
- **Spring Data Spanner** - Google Spanner support via Spring Cloud GCP.
- **Spring Data Vault** - Vault repositories built on top of Spring Data KeyValue.

2.1.2 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Elasticsearch交互文档和轻松地编写一个存储库数据访问层。

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

2.2 Spring Data Elasticsearch环境搭建

步骤:

1. 导入maven依赖的坐标
2. 编写配置文件applicationContext.xml: 引入ElasticSearch的命名空间
3. 编写实体类Article, Dao层, Service层的代码
4. 编写配置文件: applicationContext.xml: 配置持久层包扫描, 配置service包扫描, 配置ElasticSearch的集群的名称和地址
5. 编写测试类: 需引入配置文件

1) 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>itheima_elasticsearch_demo3</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>
    </dependencies>
</project>
```



```

</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>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>
<dependency>
    <groupId>org.springframework.data</groupId>
    <artifactId>spring-data-elasticsearch</artifactId>
    <version>3.0.5.RELEASE</version>
    <exclusions>
        <exclusion>
            <groupId>org.elasticsearch.plugin</groupId>
            <artifactId>transport-netty4-client</artifactId>
        </exclusion>
    </exclusions>
</dependency>
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-test</artifactId>
    <version>5.0.4.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

```
package com.itheima.domain;

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.dao;

import com.itheima.domain.Article;
import
org.springframework.data.elasticsearch.repository.ElasticsearchRepository;

@Repository
public interface ArticleRepository extends ElasticsearchRepository<Article,
Integer> {

}

```

5) 编写Service

```

package com.itheima.service;

import com.itheima.domain.Article;

public interface ArticleService {

    public void save(Article article);

}

```

```

package com.itheima.service.impl;

import com.itheima.dao.ArticleRepository;
import com.itheima.domain.Article;
import com.itheima.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
    ">

    <!-- 扫描Dao包，自动创建实例 -->
    <elasticsearch:repositories base-package="com.itheima.dao"/>

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

    <!-- 配置elasticsearch的连接 -->
    <!-- 配置elasticsearch的连接 -->
    <elasticsearch:transport-client id="client" cluster-nodes="localhost:9300"
cluster-name="elasticsearch"/>

    <!-- 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

```

package com.itheima.domain;

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;

//@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

    //@Field 每个文档的字段配置（类型、是否分词、是否存储、分词器 ）
    @Field(store=true, index = false,type = FieldType.Integer)
    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
+ "]\n";
    }
}

```

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

searchAnalyzer: 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.test;

import com.itheima.domain.Article;
import com.itheima.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 SpringDataESTest {

    @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Spring Data为
        Elasticsearch Elasticsearch项目提供集成搜索引擎");
        articleService.save(article);
    }
}

```

2.3 Spring Data ElasticSearch的常用操作

2.3.1 增删改查方法测试

```

package com.itheima.service;

import com.itheima.domain.Article;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.Pageable;

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);

}

```

```

package com.itheima.service.impl;

import com.itheima.dao.ArticleRepository;
import com.itheima.domain.Article;
import com.itheima.service.ArticleService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.Pageable;
import org.springframework.stereotype.Service;

@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);
    }

}

```

```

package com.itheima.test;

import com.itheima.domain.Article;
import com.itheima.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.domain.Page;
import org.springframework.data.domain.PageRequest;
import org.springframework.data.domain.Pageable;
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 SpringDataESTest {

    @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("ELasticSearch是一个基于Lucene的搜索服务器");
        article.setContent("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，能够达到实时搜索，稳定，可靠，快速，安装使用方便。");
        articleService.save(article);
    }

    /**测试更新*/
    @Test
    public void update(){
        Article article = new Article();
        article.setId(1001);
        article.setTitle("[更新]ELasticSearch是一个基于Lucene的搜索服务器");
        article.setContent("[更新]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，能够达到实时搜索，稳定，可靠，快速，安装使用方便。");
        articleService.save(article);
    }

    /**测试删除*/
    @Test
    public void delete(){
        Article article = new Article();
        article.setId(1001);
        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);
    }
}

/**分页查询*/
@Test
public void findAllPage(){
    Pageable pageable = PageRequest.of(1,10);
    Page<Article> page = articleService.findAll(pageable);
    for(Article article:page.getContent()){
        System.out.println(article);
    }
}
}

```

2.3.2 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

2.3.3 查询方法测试

1) dao层实现

```

// 这里应该写测试代码

```

```

package com.itheima.dao;

import com.itheima.domain.Article;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.Pageable;
import org.springframework.data.elasticsearch.repository.ElasticsearchRepository;
import java.util.List;

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);
}

```

2) service层实现

```

public interface ArticleService {
    //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);
}

```

```

package com.itheima.service.impl;

import com.itheima.dao.ArticleRepository;
import com.itheima.domain.Article;
import com.itheima.service.ArticleService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.Pageable;
import org.springframework.stereotype.Service;
import java.util.List;

@Service
public class ArticleServiceImpl implements ArticleService {

    @Autowired
    private ArticleRepository articleRepository;

    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

```
package com.itheima.test;

import com.itheima.domain.Article;
import com.itheima.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.domain.Page;
import org.springframework.data.domain.PageRequest;
import org.springframework.data.domain.Pageable;
import org.springframework.data.elasticsearch.core.ElasticsearchTemplate;
import org.springframework.test.context.ContextConfiguration;
import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;

import java.util.List;

@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration(locations="classpath:applicationContext.xml")
public class SpringDataESTest {

    @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);
        Page<Article> page = articleService.findByTitle(condition,pageable);
        for(Article article:page.getContent()){
            System.out.println(article);
        }
    }
}
```

2.3.4 使用Elasticsearch的原生查询对象进行查询。

```
/**
 * 原生查询
 * 1.创建原生查询对象构建器
 * 2.设置查询方式，设置分页
 * 3.构建查询对象
 * 4.使用ElasticSearch模板执行查询对象
 */
@Test
public void findByNativeQuery() {
    //创建一个SearchQuery对象
    SearchQuery searchQuery = new NativeSearchQueryBuilder()
        //设置查询条件，此处可以使用QueryBuilders创建多种查询
        .withQuery(QueryBuilders.queryStringQuery("备份节点上没有数
据").defaultField("title"))
        //还可以设置分页信息
        .withPageable(PageRequest.of(1, 5))
        //创建SearchQuery对象
        .build();

    //使用模板对象执行查询
    elasticsearchTemplate.queryForList(searchQuery, Article.class)
        .forEach(a-> System.out.println(a));
}
```