ElasticSearch第二天

学习目标：

1. 能够使用java客户端完成创建、删除索引的操作
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

第一章 ElasticSearch编程操作

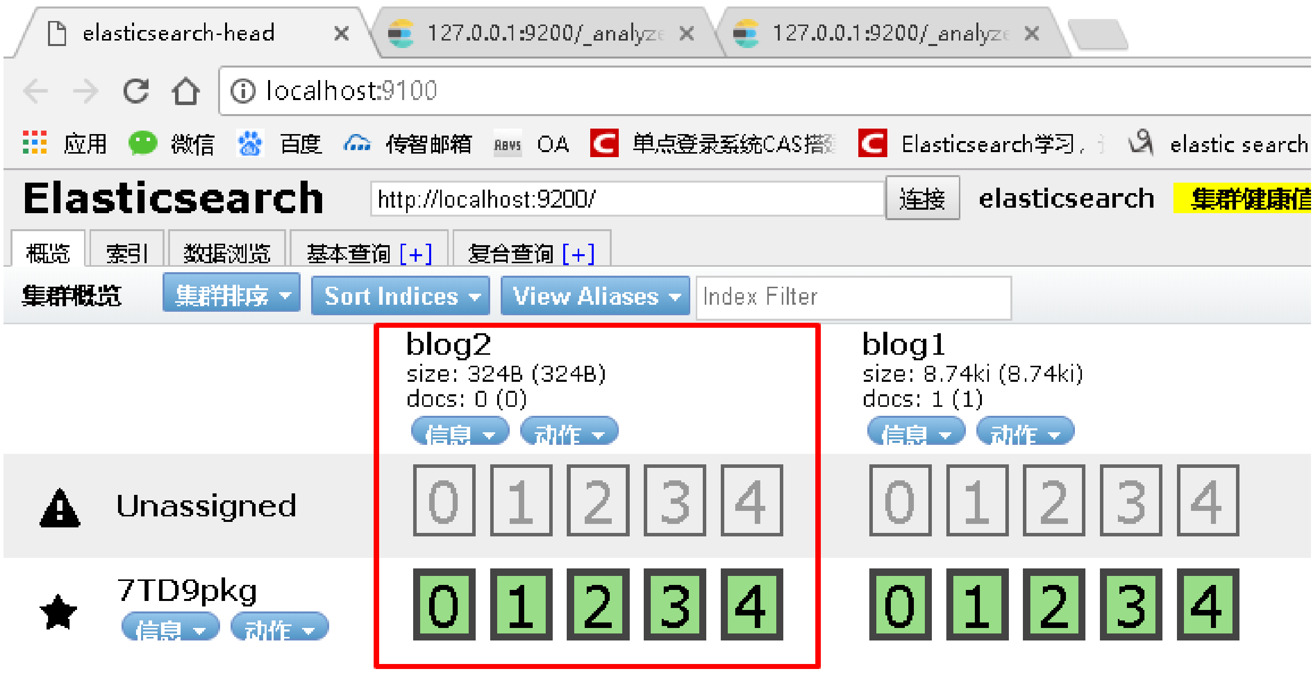
1.1 创建工程，导入坐标

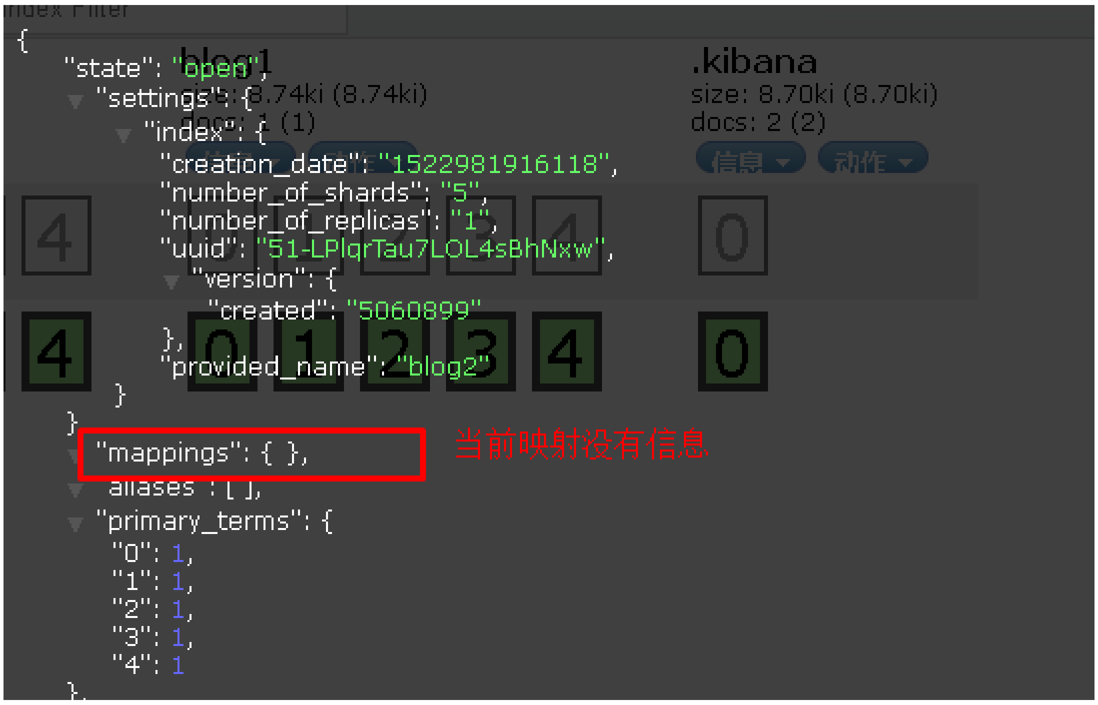
pom.xml坐标

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

1.2 创建索引index

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



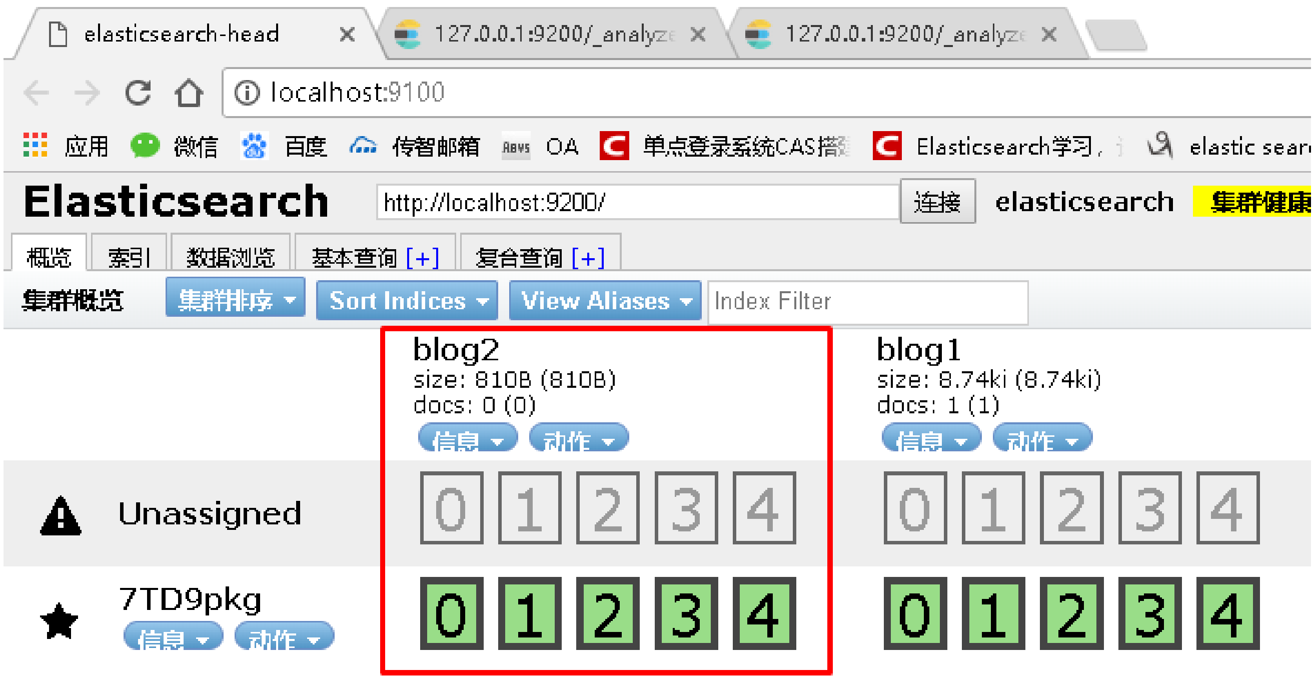


```
{
    "state": "open",
    "settings": {
        "index": {
            "creation_date": "1522981916118",
            "number_of_shards": "5",
            "number_of_replicas": "1",
            "uuid": "51-LPlqrTau7LOL4sBhNxw",
            "version": {
                "created": "5060899"
            },
            "provided_name": "blog2"
        }
    },
    "mappings": { },
    aliases : [ ],
    "primary_terms": {
        "0": 1,
        "1": 1,
        "2": 1,
        "3": 1,
        "4": 1
    },
```

1.3 创建映射mapping

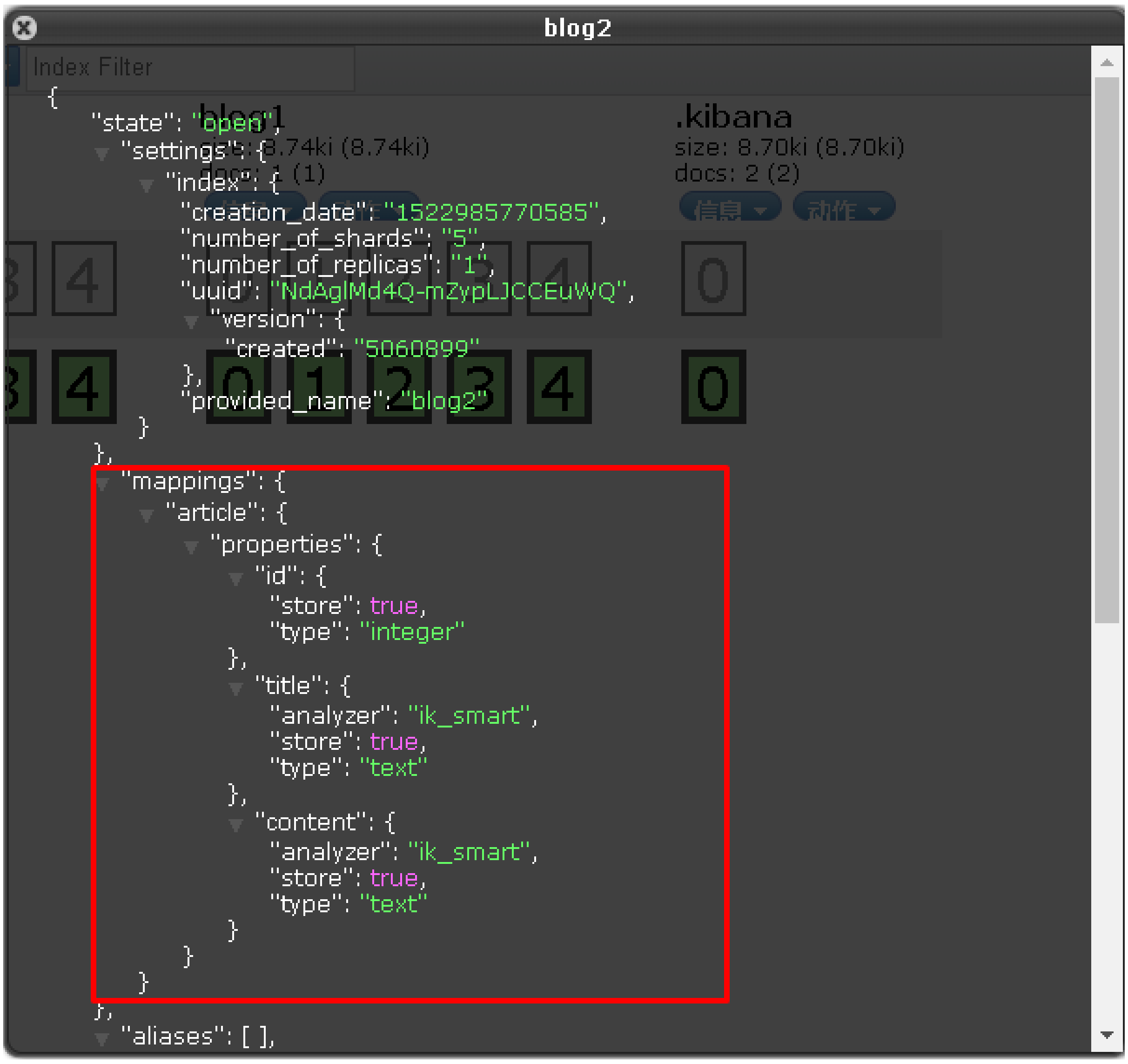
```
@Test
//创建映射
public void test3() throws Exception{
    // 创建Client连接对象
    Settings settings = Settings.builder().put("cluster.name", "my-elasticsearch").build();
    TransportClient client = new PreBuiltTransportClient(settings)
        .addTransportAddress(new InetSocketTransportAddress(InetAddress.getByName("127.0.0.1"),
9300));

    // 添加映射
    /**
         * 格式：
         * "mappings" : {
             "article" : {
                "dynamic" : "false",
                 "properties" : {
                    "id" : { "type" : "string" },
                     "content" : { "type" : "string" },
                    "author" : { "type" : "string" }
                 }
             }
         }
         */
    XContentBuilder builder = XContentFactory.jsonBuilder()
        .startObject()
        .startObject("article")
        .startObject("properties")

        .startObject("id")
```

```
                .field("type", "integer").field("store", "yes")
                .endObject()
                .startObject("title")
                .field("type", "string").field("store", "yes").field("analyzer", "ik_smart")
                .endObject()
                .startObject("content")
                .field("type", "string").field("store", "yes").field("analyzer", "ik_smart")
                .endObject()
                .endObject()
                .endObject()
                .endObject();
        // 创建映射
        PutMappingRequest mapping = Requests.putMappingRequest("blog2")
                .type("article").source(builder);
        client.admin().indices().putMapping(mapping).get();
        //释放资源
        client.close();
    }
```



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

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

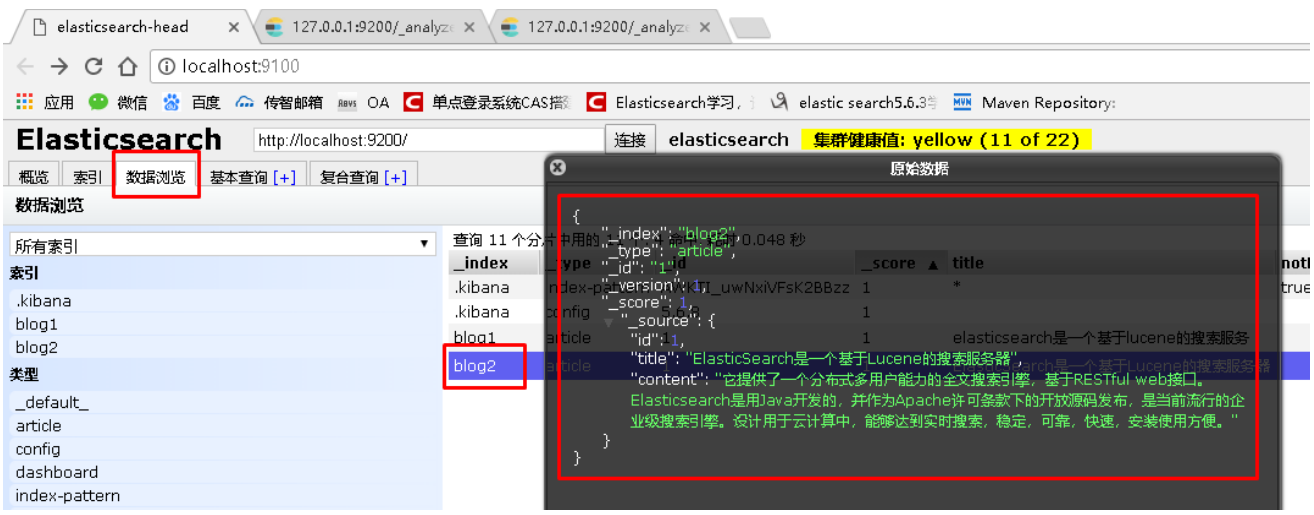


```
{
    "state": "open",
    "settings": {
        "index": {
            "creation_date": "1522985770585",
            "number_of_shards": "5",
            "number_of_replicas": "1",
            "uuid": "NdAgIMd4Q-mZypLJCCEuWQ",
            "version": {
                "created": "5060899"
            },
            "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 建立文档（通过XContentBuilder）

```
@Test
//创建文档(通过XContentBuilder)
public void test4() throws Exception{
    // 创建Client连接对象
    Settings settings = Settings.builder().put("cluster.name", "my-elasticsearch").build();
    TransportClient client = new PreBuiltTransportClient(settings)
        .addTransportAddress(new InetSocketTransportAddress(InetAddress.getByName("127.0.0.1"),
9300));

    //创建文档信息
    XContentBuilder builder = XContentFactory.jsonBuilder()
        .startObject()
        .field("id", 1)

        .field("title", "ElasticSearch是一个基于Lucene的搜索服务器")
        .field("content",
                "它提供了一个分布式多用户能力的全文搜索引擎，基于RESTful web接口。Elasticsearch是用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();

    // 建立文档对象
    /**
         * 参数一blog1：表示索引对象
         * 参数二article：类型
         * 参数三1：建立id
         */
    client.prepareIndex("blog2", "article", "1").setSource(builder).get();

    //释放资源
    client.close();
}
```



```
{
  "_index": "blog2",
  "_type": "article",
  "_id": "1",
  "_version": 1,
  "_score": 1,
  "_source": {
    "id": 1,
    "title": "ElasticSearch是一个基于Lucene的搜索服务器",
    "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。"
  }
}
```

1.4.2 建立文档（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...
}
```

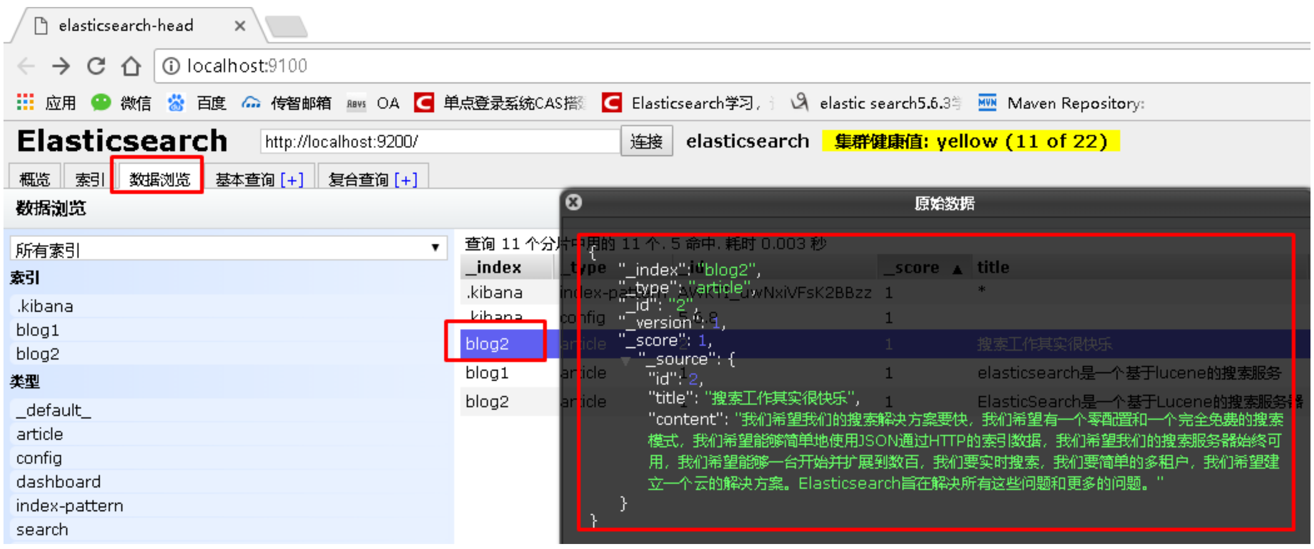
2）添加jackson坐标

```
<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

```
@Test
//创建文档(通过实体转json)
public void test5() throws Exception{
    // 创建Client连接对象
    Settings settings = Settings.builder().put("cluster.name", "my-elasticsearch").build();
    TransportClient client = new PreBuiltTransportClient(settings)
        .addTransportAddress(new InetSocketTransportAddress(InetAddress.getByName("127.0.0.1"),
9300));

    // 描述json 数据
    //{id:xxx, title:xxx, content:xxx}
    Article article = new Article();
    article.setId(2);
    article.setTitle("搜索工作其实很快乐");
    article.setContent("我们希望我们的搜索解决方案要快，我们希望有一个零配置和一个完全免费的搜索模式，
我们希望能够简单地使用JSON通过HTTP的索引数据，我们希望我们的搜索服务器始终可用，我们希望能够一台开始并扩
展到数百，我们要实时搜索，我们要简单的多租户，我们希望建立一个云的解决方案。Elasticsearch旨在解决所有这
些问题和更多的问题。");

    ObjectMapper objectMapper = new ObjectMapper();

    // 建立文档
    client.prepareIndex("blog2", "article", article.getId().toString())
        //.setSource(objectMapper.writeValueAsString(article)).get();
        .setSource(objectMapper.writeValueAsString(article).getBytes(),
XContentType.JSON).get();

    //释放资源
    client.close();
}
```


elasticsearch-head
localhost:9100
应用 微信 百度 传智邮箱 OA 单点登录系统CAS搭 Elasticsearch学习, elastic search5.6.3 Maven Repository:
Elasticsearch
http://localhost:9200/
连接
elasticsearch
集群健康值: yellow (11 of 22)
概览 索引 数据浏览 基本查询 [+] 复合查询 [+]
数据浏览
所有索引
索引
.kibana
blog1
blog2
类型
default
article
config
dashboard
index-pattern
search
查询 11 个分片中用的 11 个. 5 命中. 耗时 0.003 秒
_index _type _score title
原始数据

```
{
  "_index": "blog2",
  "_type": "article",
  "_id": "2",
  "_version": 1,
  "_score": 1,
  "_source": {
    "id": 2,
    "title": "搜索工作其实很快乐",
    "content": "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。"
  }
}
```

1.5 查询文档操作

1.5.1关键词查询

```
@Test
public void testTermQuery() throws Exception{
    //1、创建es客户端连接对象
    Settings settings = Settings.builder().put("cluster.name", "my-elasticsearch").build();
    TransportClient client = new PreBuiltTransportClient(settings)
        .addTransportAddress(new InetSocketTransportAddress(InetAddress.getByName("127.0.0.1"),
9300));

    //2、设置搜索条件
    SearchResponse searchResponse = client.prepareSearch("blog2")
        .setTypes("article")
        .setQuery(QueryBuilders.termQuery("content", "搜索")).get();

    //3、遍历搜索结果数据
    SearchHits hits = searchResponse.getHits(); // 获取命中次数，查询结果有多少对象
    System.out.println("查询结果有：" + hits.getTotalHits() + "条");
    Iterator<SearchHit> iterator = hits.iterator();
    while (iterator.hasNext()) {
        SearchHit searchHit = iterator.next(); // 每个查询对象
        System.out.println(searchHit.getSourceAsString()); // 获取字符串格式打印
        System.out.println("title:" + searchHit.getSource().get("title"));
    }

    //4、释放资源
    client.close();

}
```

2.5.2 字符串查询

```
@Test
public void testStringQuery() throws Exception{
    //1、创建es客户端连接对象
    Settings settings = Settings.builder().put("cluster.name", "my-elasticsearch").build();
    TransportClient client = new PreBuiltTransportClient(settings)
        .addTransportAddress(new InetSocketTransportAddress(InetAddress.getByName("127.0.0.1"),
9300));

    //2、设置搜索条件
    SearchResponse searchResponse = client.prepareSearch("blog2")
        .setTypes("article")
        .setQuery(QueryBuilders.queryStringQuery("搜索")).get();

    //3、遍历搜索结果数据
    SearchHits hits = searchResponse.getHits(); // 获取命中次数，查询结果有多少对象
    System.out.println("查询结果有：" + hits.getTotalHits() + "条");
    Iterator<SearchHit> iterator = hits.iterator();
    while (iterator.hasNext()) {
        SearchHit searchHit = iterator.next(); // 每个查询对象
        System.out.println(searchHit.getSourceAsString()); // 获取字符串格式打印
        System.out.println("title:" + searchHit.getSource().get("title"));
    }

    //4、释放资源
    client.close();

}
```

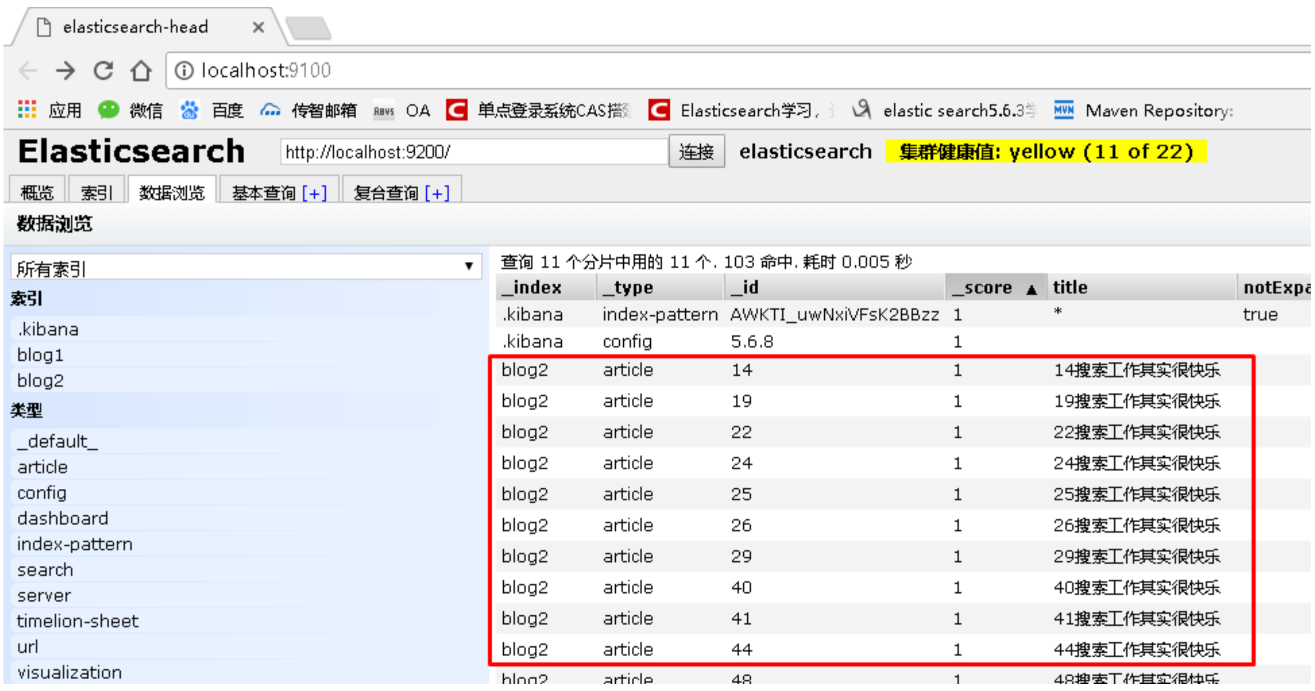
2.5.2 使用文档ID查询文档

```
@Test
    public void testIdQuery() throws Exception {
        //client对象为TransportClient对象
        SearchResponse response = client.prepareSearch("blog1")
                .setTypes("article")
                //设置要查询的id
                .setQuery(QueryBuilders.idsQuery().addIds("test002"))
                //执行查询
                .get();
        //取查询结果
        SearchHits searchHits = response.getHits();
        //取查询结果总记录数
        System.out.println(searchHits.getTotalHits());
        Iterator<SearchHit> hitIterator = searchHits.iterator();
        while(hitIterator.hasNext()) {
            SearchHit searchHit = hitIterator.next();
            //打印整行数据
            System.out.println(searchHit.getSourceAsString());
        }
    }
```

2.6 查询文档分页操作

2.6.1 批量插入数据

```
 @Test
//批量插入100条数据
public void test9() throws Exception{
        // 创建Client连接对象
        Settings settings = Settings.builder().put("cluster.name", "my-elasticsearch").build();
        TransportClient client = new PreBuiltTransportClient(settings)
                .addTransportAddress(new
InetSocketTransportAddress(InetAddress.getByName("127.0.0.1"), 9300));

        ObjectMapper objectMapper = new ObjectMapper();

        for (int i = 1; i <= 100; i++) {
            // 描述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。");

            // 建立文档
            client.prepareIndex("blog2", "article", article.getId().toString())
                    //.setSource(objectMapper.writeValueAsString(article)).get();

.setSource(objectMapper.writeValueAsString(article).getBytes(),XContentType.JSON).get();
        }

        //释放资源
        client.close();
}
```



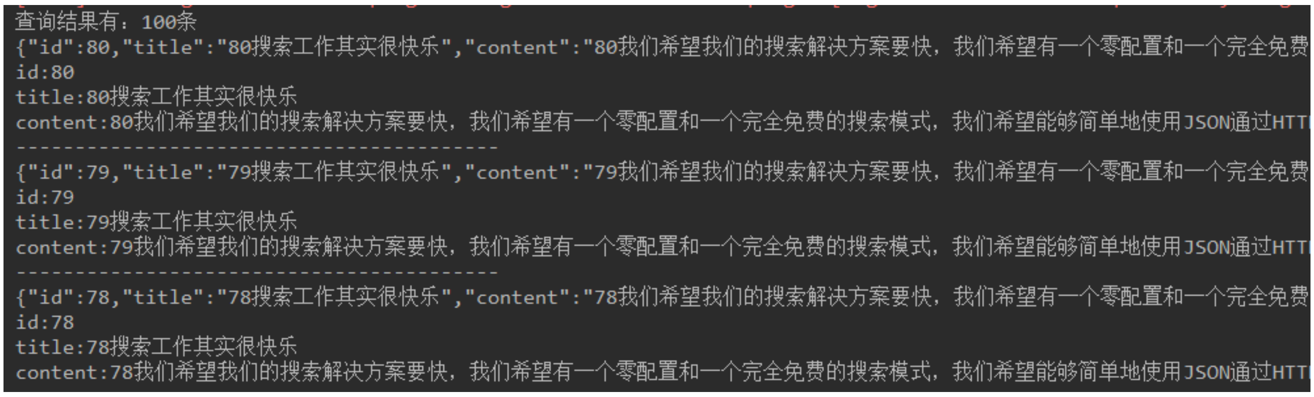
查询 11 个分片中用的 11 个. 103 命中. 耗时 0.005 秒

_index	_type	_id	_score ▲	title	notExpa
.kibana	index-pattern	AWKTI_uwNxiVFsK2BBzz	1	*	true
.kibana	config	5.6.8	1		
blog2	article	14	1	14搜索工作其实很快乐	
blog2	article	19	1	19搜索工作其实很快乐	
blog2	article	22	1	22搜索工作其实很快乐	
blog2	article	24	1	24搜索工作其实很快乐	
blog2	article	25	1	25搜索工作其实很快乐	
blog2	article	26	1	26搜索工作其实很快乐	
blog2	article	29	1	29搜索工作其实很快乐	
blog2	article	40	1	40搜索工作其实很快乐	
blog2	article	41	1	41搜索工作其实很快乐	
blog2	article	44	1	44搜索工作其实很快乐	
blog2	article	48	1	48搜索工作其实很快乐	

2.6.2 分页查询

```
@Test
//分页查询
public void test10() throws Exception{
    // 创建Client连接对象
    Settings settings = Settings.builder().put("cluster.name", "my-elasticsearch").build();
    TransportClient client = new PreBuiltTransportClient(settings)
        .addTransportAddress(new InetSocketTransportAddress(InetAddress.getByName("127.0.0.1"),
9300));

    // 搜索数据
    SearchRequestBuilder searchRequestBuilder =
client.prepareSearch("blog2").setTypes("article")
        .setQuery(QueryBuilders.matchAllQuery());//默认每页10条记录

    // 查询第2页数据，每页20条
    //setFrom()：从第几条开始检索，默认是0。
    //setSize():每页最多显示的记录数。
    searchRequestBuilder.setFrom(0).setSize(5);
    SearchResponse searchResponse = searchRequestBuilder.get();

    SearchHits hits = searchResponse.getHits(); // 获取命中次数，查询结果有多少对象
    System.out.println("查询结果有：" + hits.getTotalHits() + "条");
    Iterator<SearchHit> iterator = hits.iterator();
    while (iterator.hasNext()) {
        SearchHit searchHit = iterator.next(); // 每个查询对象
        System.out.println(searchHit.getSourceAsString()); // 获取字符串格式打印
        System.out.println("id:" + searchHit.getSource().get("id"));
        System.out.println("title:" + searchHit.getSource().get("title"));
        System.out.println("content:" + searchHit.getSource().get("content"));
        System.out.println("-----------------------------------------");

    }
```

```
        //释放资源
        client.close();
    }
```



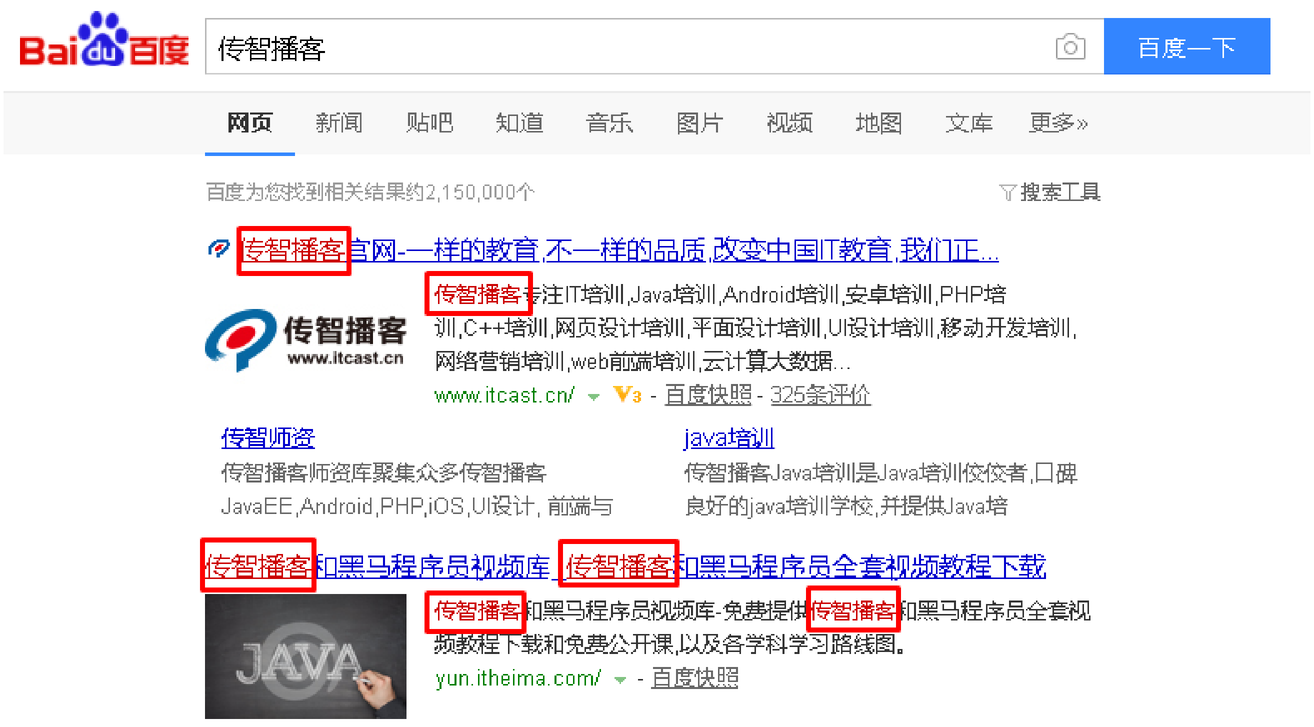
```
查询结果有：100条
{"id":80,"title":"80搜索工作其实很快乐","content":"80我们希望我们的搜索解决方案要快，我们希望有一个零配置和一个完全免费
id:80
title:80搜索工作其实很快乐
content:80我们希望我们的搜索解决方案要快，我们希望有一个零配置和一个完全免费的搜索模式，我们希望能够简单地使用JSON通过HTT
----------------------------------------
{"id":79,"title":"79搜索工作其实很快乐","content":"79我们希望我们的搜索解决方案要快，我们希望有一个零配置和一个完全免费
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
```

2.7 查询结果高亮操作

2.7.1 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"



百度为您找到相关结果约2,150,000个

传智播客官网-一样的教育,不一样的品质,改变中国IT教育,我们正...

传智播客专注IT培训,Java培训,Android培训,安卓培训,PHP培训,C++培训,网页设计培训,平面设计培训,UI设计培训,移动开发培训,网络营销培训,web前端培训,云计算大数据...

www.itcast.cn/ - V3 - 百度快照 - 325条评价

传智师资
传智播客师资库聚集众多传智播客JavaEE,Android,PHP,iOS,UI设计, 前端与

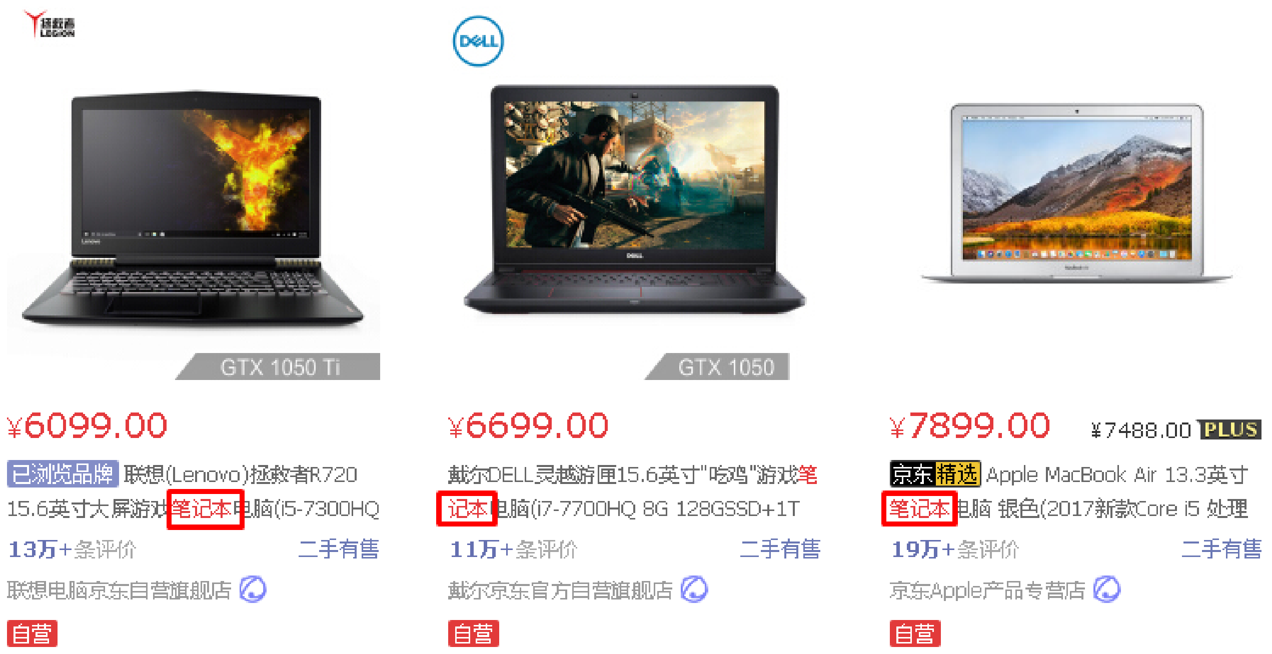
java培训
传智播客Java培训是Java培训佼佼者,口碑良好的java培训学校,并提供Java培

传智播客和黑马程序员视频库_传智播客和黑马程序员全套视频教程下载

传智播客和黑马程序员视频库-免费提供传智播客和黑马程序员全套视频教程下载和免费公开课,以及各学科学习路线图。

yun.itheima.com/ - 百度快照

京东商城搜索"笔记本"

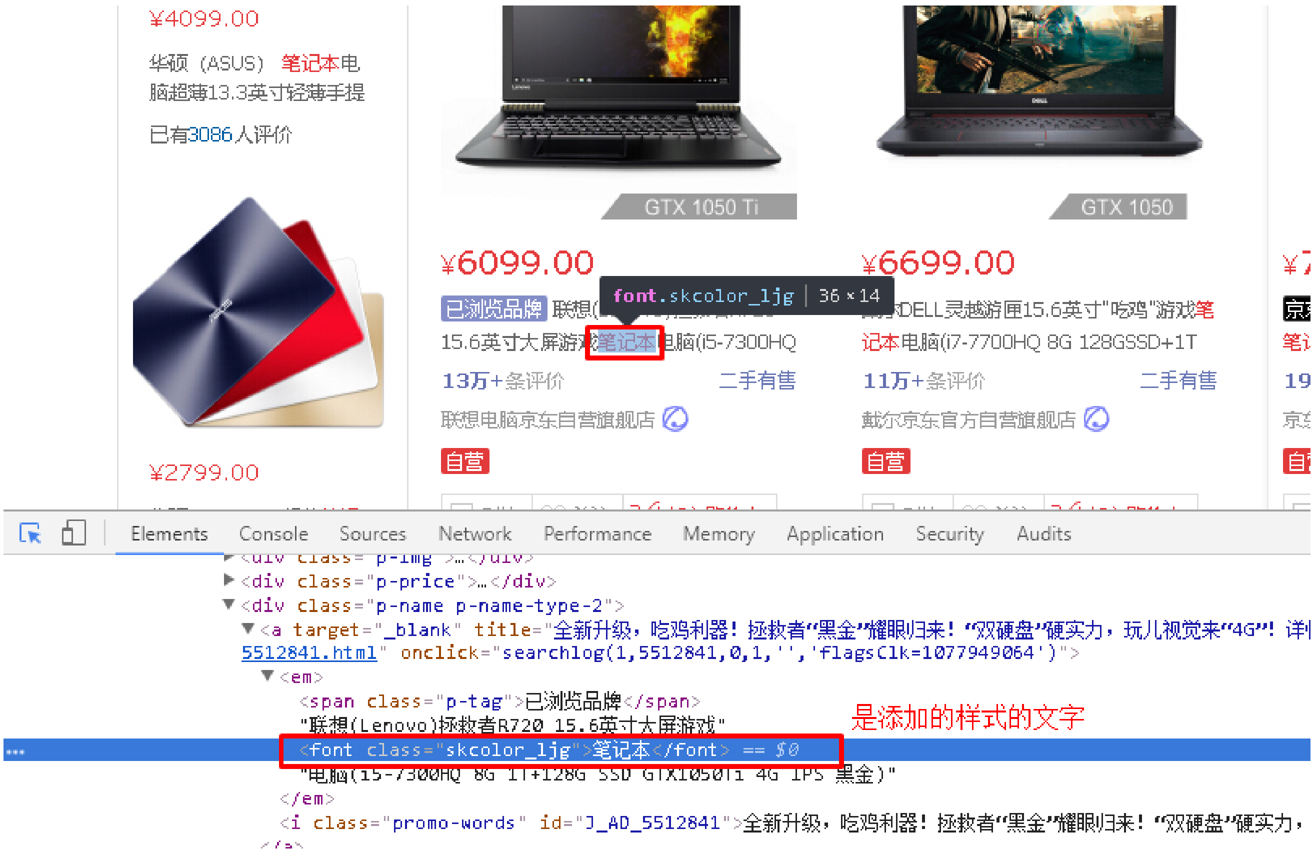

GTX 1050 Ti
¥6099.00
已浏览品牌 联想(Lenovo)拯救者R720 15.6英寸大屏游戏笔记本电脑(i5-7300HQ
13万+条评价 二手有售
联想电脑京东自营旗舰店
自营

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
戴尔京东官方自营旗舰店
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

2.7.2 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：



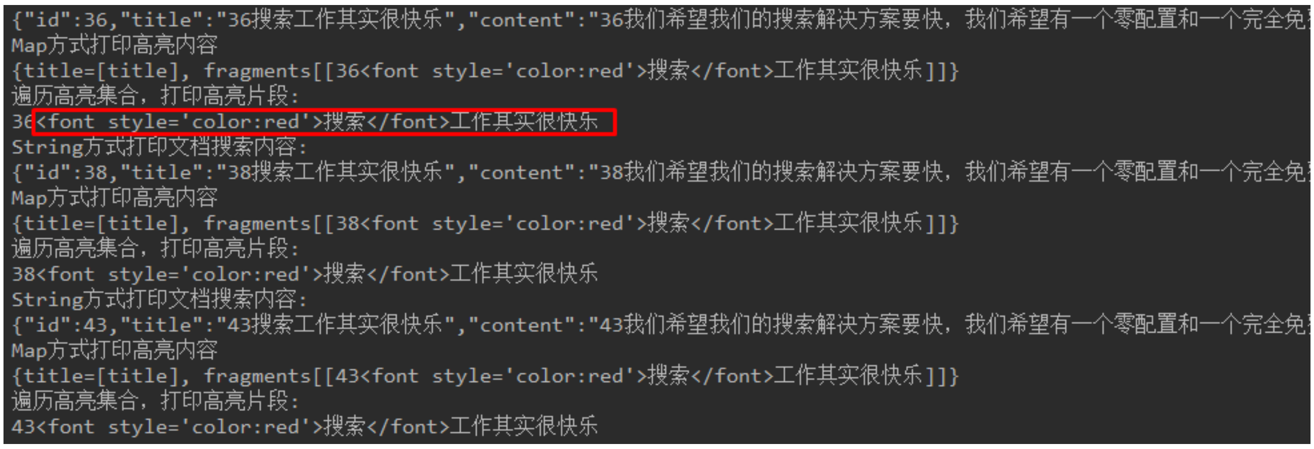
¥4099.00
华硕（ASUS） 笔记本电脑超薄13.3英寸轻薄手提
已有3086人评价
¥2799.00
GTX 1050 Ti
¥6099.00
已浏览品牌 联想(Lenovo)拯救者R720 15.6英寸大屏游戏笔记本电脑(i5-7300HQ
font.skcolor_ljg | 36 × 14
13万+条评价 二手有售
联想电脑京东自营旗舰店
自营
GTX 1050
¥6699.00
戴尔DELL灵越游匣15.6英寸"吃鸡"游戏笔记本电脑(i7-7700HQ 8G 128GSSD+1T
11万+条评价 二手有售
戴尔京东官方自营旗舰店
自营
Elements Console Sources Network Performance Memory Application Security Audits
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,'','flagsClk=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 GTX1050Ti 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，
```

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

2.7.3 高亮显示代码实现

```
@Test
//高亮查询
public void test11() throws Exception{
    // 创建Client连接对象
    Settings settings = Settings.builder().put("cluster.name", "my-elasticsearch").build();
    TransportClient client = new PreBuiltTransportClient(settings)
        .addTransportAddress(new InetSocketTransportAddress(InetAddress.getByName("127.0.0.1"),
9300));

    // 搜索数据
    SearchRequestBuilder searchRequestBuilder = client
        .prepareSearch("blog2").setTypes("article")
        .setQuery(QueryBuilders.termQuery("title", "搜索"));

    //设置高亮数据
    HighlightBuilder hiBuilder=new HighlightBuilder();
    hiBuilder.preTags("<font style='color:red'>");
    hiBuilder.postTags("</font>");
    hiBuilder.field("title");
    searchRequestBuilder.highlighter(hiBuilder);

    //获得查询结果数据
    SearchResponse searchResponse = searchRequestBuilder.get();

    //获取查询结果集
    SearchHits searchHits = searchResponse.getHits();
    System.out.println("共搜到:"+searchHits.getTotalHits()+"条结果!");
    //遍历结果
    for(SearchHit hit:searchHits){
        System.out.println("String方式打印文档搜索内容:");
        System.out.println(hit.getSourceAsString());
        System.out.println("Map方式打印高亮内容");
        System.out.println(hit.getHighlightFields());

        System.out.println("遍历高亮集合，打印高亮片段:");
        Text[] text = hit.getHighlightFields().get("title").getFragments();
        for (Text str : text) {
            System.out.println(str);
        }
    }

    //释放资源
    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
```

第三章 Spring Data ElasticSearch 使用

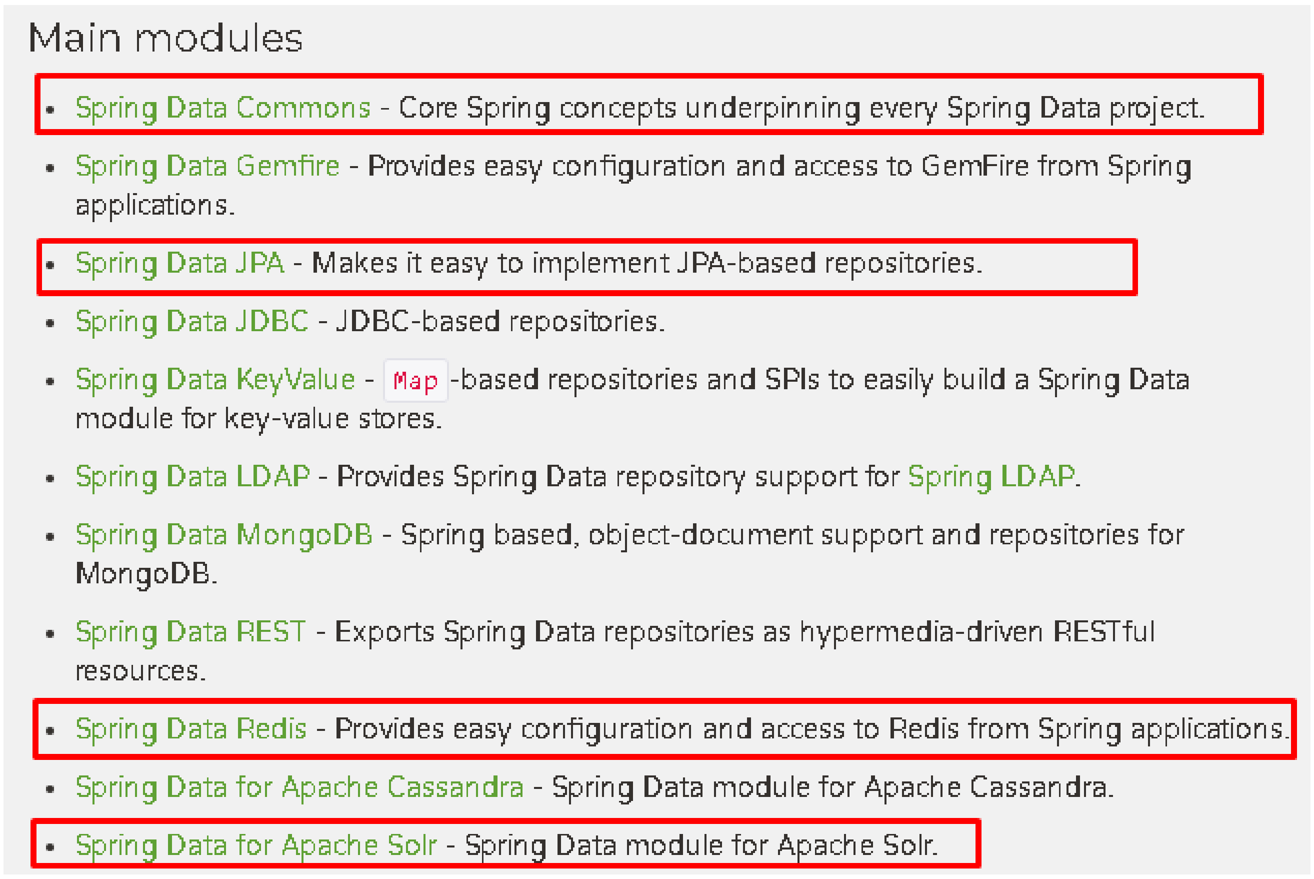
3.1 Spring Data ElasticSearch简介

3.1.1 什么是Spring Data

Spring Data是一个用于简化数据库访问，并支持云服务的开源框架。其主要目标是使得对数据的访问变得方便快捷，并支持map-reduce框架和云计算数据服务。 Spring Data可以极大的简化JPA的写法，可以在几乎不用写实现的情况下，实现对数据的访问和操作。除了CRUD外，还包括如分页、排序等一些常用的功能。

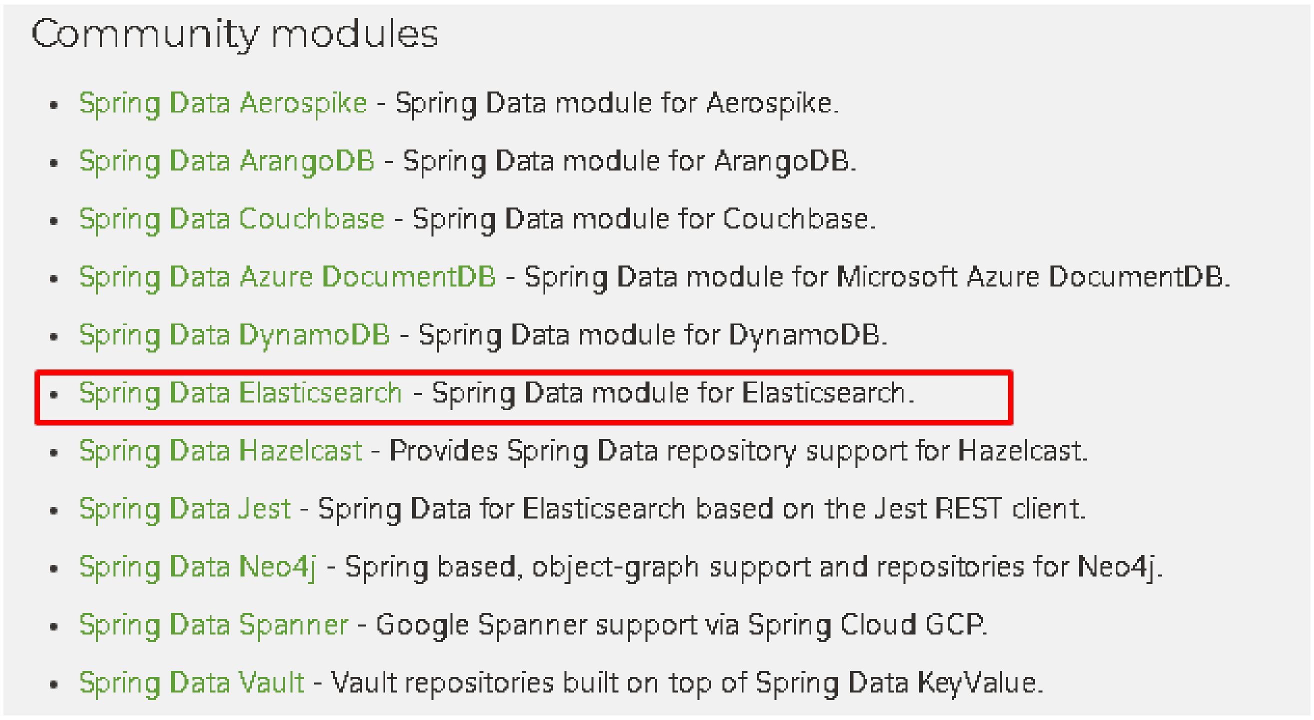
Spring Data的官网：http://projects.spring.io/spring-data/

Spring Data常用的功能模块如下：



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.

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

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

3.2 Spring Data ElasticSearch入门

1）导入Spring Data ElasticSearch坐标

```xml
<?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>
        <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

```java
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

```java
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="my-
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 + "]";
    }

}
```

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

8）创建测试类SpringDataESTest

```
package com.itheima.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 \n" +
                "    Spring Data为Elasticsearch Elasticsearch项目提供集成搜索引擎");
        articleService.save(article);
    }

}
```

3.3 Spring Data ElasticSearch的常用操作

3.3.1 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;
```

```java
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);
    }
}
```

```java
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("测试SpringData ElasticSearch");
        article.setContent("Spring Data ElasticSearch 基于 spring data API 简化 elasticSearch操
作，将原始操作elasticSearch的客户端API 进行封装 \n" +
                "    Spring Data为Elasticsearch Elasticsearch项目提供集成搜索引擎");
        articleService.save(article);
    }

    /**测试保存*/
    @Test
    public void save(){
        Article article = new Article();
        article.setId(1001);
        article.setTitle("elasticSearch 3.0版本发布");
        article.setContent("ElasticSearch是一个基于Lucene的搜索服务器。它提供了一个分布式多用户能力的
全文搜索引擎，基于RESTful web接口");
        articleService.save(article);
    }

    /**测试更新*/
    @Test
    public void update(){
        Article article = new Article();
        article.setId(1001);
        article.setTitle("elasticSearch 3.0版本发布...更新");
        article.setContent("ElasticSearch是一个基于Lucene的搜索服务器。它提供了一个分布式多用户能力的
全文搜索引擎，基于RESTful web接口");
        articleService.save(article);
    }

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

        article.setId(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);
        }
    }
}
```

3.3.2 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

3.3.3 查询方法测试

1）dao层实现

```
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;
```

```java
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);
        }
    }

}
```

3.3.4使用Elasticsearch的原生查询对象进行查询。

```
@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));
    }
```