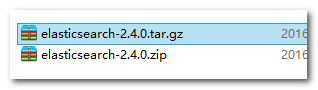
Spring Data Elasticsearch学习笔记

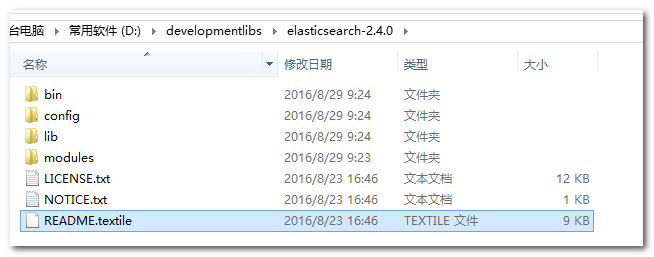
# 运行Elasticsearch

## ElasticSearch 安装配置使用入门

Window系统下载zip版本，linux系统下载tar版本



下载后的页面



bin 存放elasticSearch 运行命令

config 存放配置文件

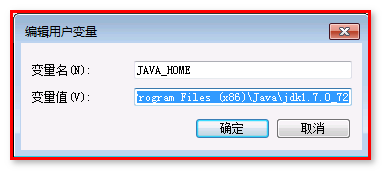
lib 存放elasticSearch运行依赖jar包

modules 存放elasticSearch 模块

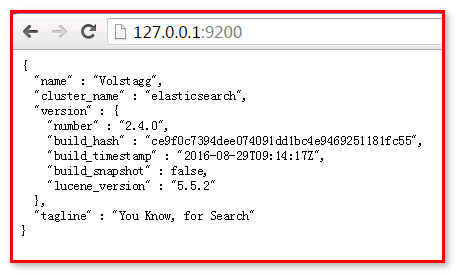
plugins 存放插件

运行elasticSearch/bin/elasticsearch.bat 文件

配置JAVA\_HOME环境变量



访问 <http://127.0.0.1:9200>



安装elasticSearch成功

## ElasticSearch 插件安装 es head

下载解压

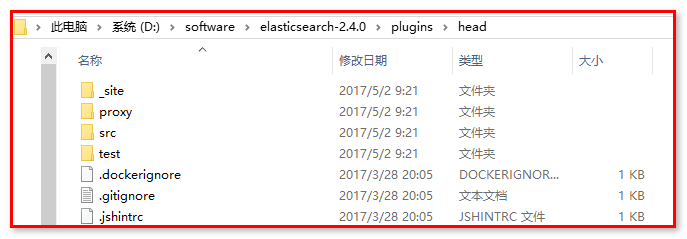
1、插件下载地址

[https://github.com/mobz/elasticsearch-head](https://github.com/mobz/elasticsearch-head" \t "http://www.open-open.com/lib/view/_blank)

2、安装方法：

将下载下的zip文件，解压缩到plugins/head目录下

目录路径如下：elasticsearch-2.4.0/plugins/head/

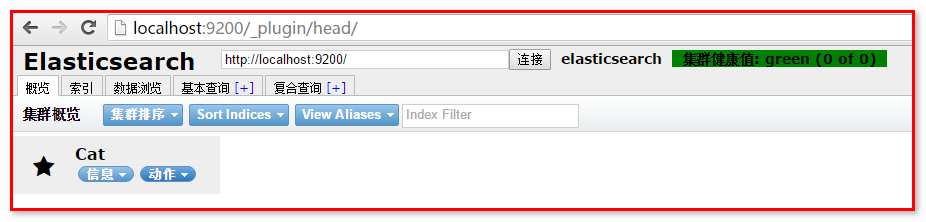


3、启动es  bin/elasticsearch

4、访问集群

   浏览器地址栏输入http://localhost:9200/\_plugin/head/

1. 显示结果如下图



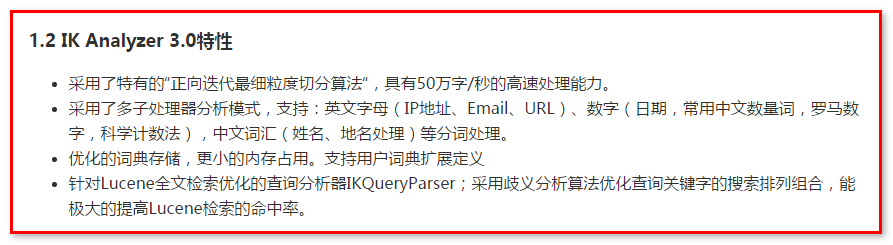
## ElasticSearch 安装IK分词器

ElasticSearch 默认采用分词器， 单个字分词 ，效果很差

搜索【IK Analyzer 3.0】

http://www.oschina.net/news/2660



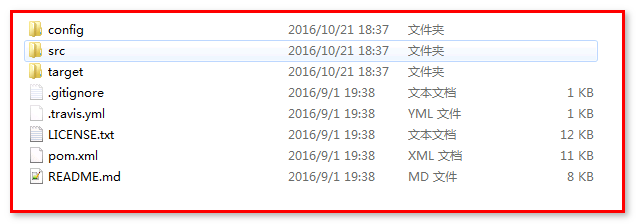


下载 <https://github.com/medcl/elasticsearch-analysis-ik/tree/2.x>

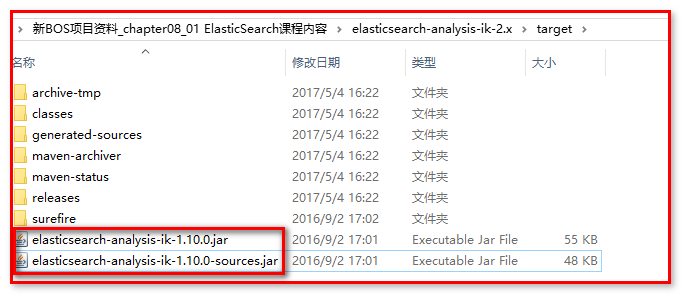
1. 下载开源项目



解压打开



执行mvn package后，打开target，生成需要的jar包



1. 进入target/release目录

将下列文件 ，拷贝到 %es%/plugins/analysis-ik



1. 进入target/release/config 目录

将所有配置文件，复制 %es%/config 下



1. 配置elasticsearch.yml

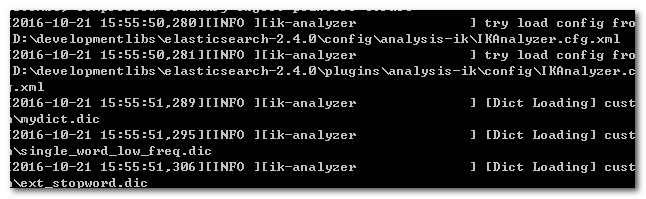
在%es%/config下

index.analysis.analyzer.ik.type: "ik"

放置到最后

名称命名为ik。后续用到分词器的时候，使用ik这个名词。

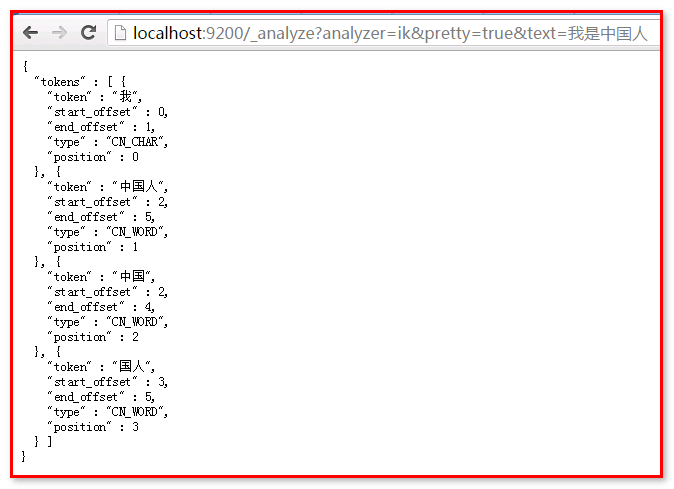
1. 重启es



发现ik分词器被加载

1. 访问

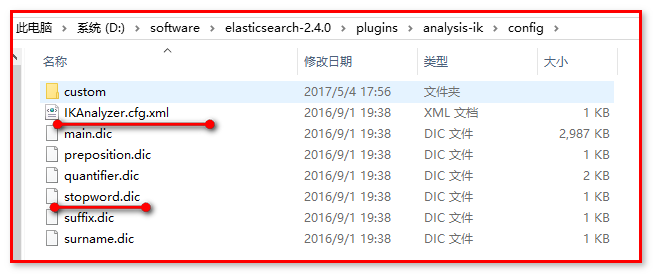
<http://localhost:9200/_analyze?analyzer=ik&pretty=true&text=我是中国人>



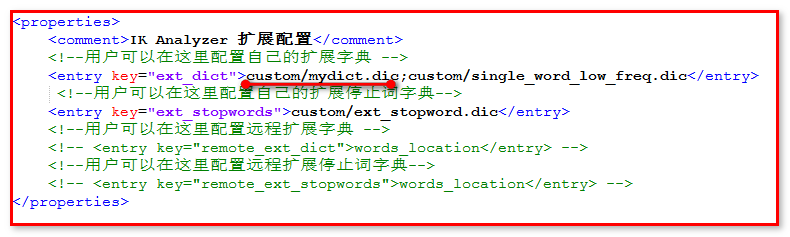
如果输入，其实“传智播客”使用IK中文分词器的时候，进行单字分词了。



如何自定义词库呢？



修改IKAnalyzer.cfg.xml



打开custom文件夹，mydict.dic，编辑文件



重新启动es。



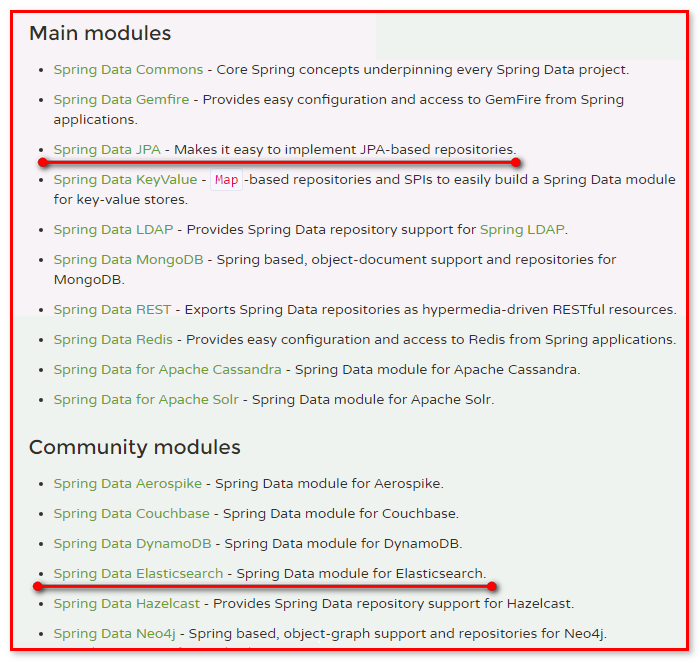
重新修改文档，再次搜索，可以查询到结果。

# Spring Data Elasticsearch

## 什么是Spring Data Elasticsearch

### Spring Data ElasticSearch简介

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



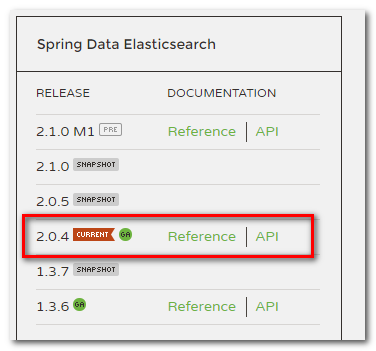
### 什么是spring data elasticSearch ?

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 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。

Spring Data Elasticsearch为文档的存储，查询，排序和统计提供了一个高度抽象的模板。在使用中，你会发现Spring Data Elasticsearch和Spring Data Solr/Mongodb有许多相似之处，都提供了Template和Repository简化开发。

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



下载官方规范文档

Maven坐标 ，本课程使用2.0.4的版本。使用spring的4的版集成

<dependencies>

<dependency>

<groupId>org.springframework.data</groupId>

<artifactId>spring-data-elasticsearch</artifactId>

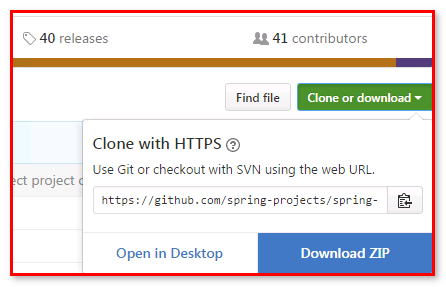
<version>2.0.4.RELEASE</version>

</dependency>

</dependencies>

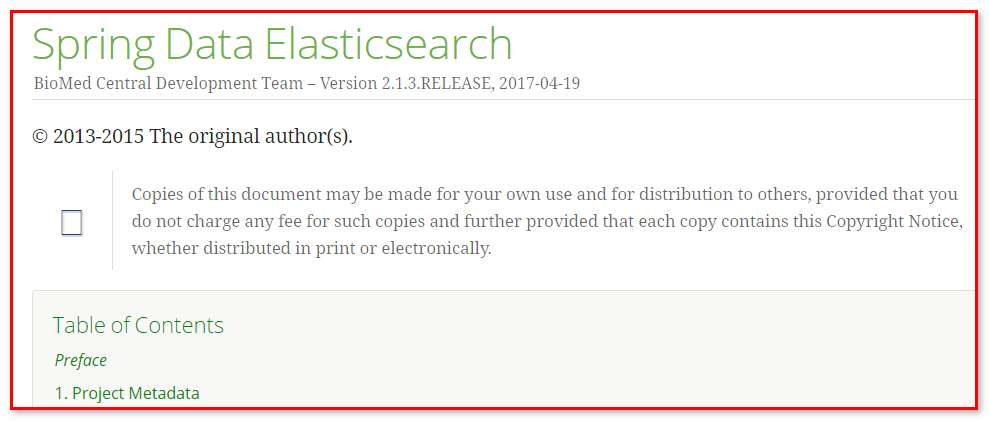
Jar包下载网址：

<https://github.com/spring-projects/spring-data-elasticsearch>



英文帮助文档地址:

<http://docs.spring.io/spring-data/elasticsearch/docs/current/reference/html/>



## Spring命名空间

Spring Data Elasticsearch模块包含一个自定义的命名空间，它允许我们定义repository bean和初始化一个ElasticsearchServer。

下面，我们像[创建Repository实例](https://es.yemengying.com/5/5.1/http:" \t "https://es.yemengying.com/5/5.1/_blank)中描述的那样使用repositories元素查找Spring Data repository。

<?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:elasticsearch="http://www.springframework.org/schema/data/elasticsearch"xsi:schemaLocation="http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans-3.1.xsd

http://www.springframework.org/schema/data/elasticsearch

http://www.springframework.org/schema/data/elasticsearch/spring-elasticsearch-1.0.xsd">

<elasticsearch:repositories base-package="com.itheima.repositories" />

</beans>

定义查找所有继承\*Repository的repository接口时所扫描包的路径，该路径下所有的包都会被扫描。允许使用通配符

使用Transport Client和Node Client元素注册一个Elasticsearch Server实例。

使用Transport Client

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

http://www.springframework.org/schema/beans/spring-beans-3.1.xsd

http://www.springframework.org/schema/data/elasticsearch

http://www.springframework.org/schema/data/elasticsearch/spring-elasticsearch-1.0.xsd">

<elasticsearch:transport-client id="client" cluster-nodes="localhost:9300,someip:9300" />

</beans>

使用Node Client

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

http://www.springframework.org/schema/beans/spring-beans-3.1.xsd

http://www.springframework.org/schema/data/elasticsearch

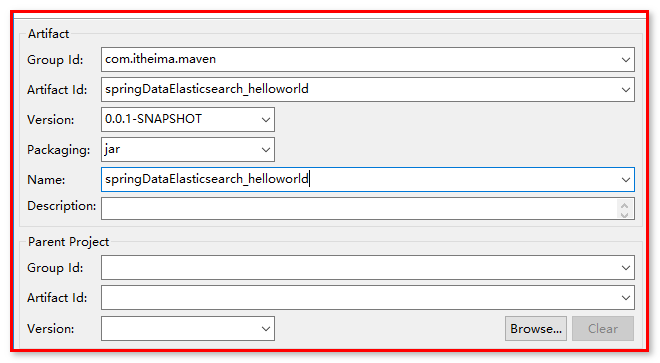
http://www.springframework.org/schema/data/elasticsearch/spring-elasticsearch-1.0.xsd">

<elasticsearch:node-client id="client" local="true"" />

</beans>

## 准备工作

1. 创建maven 工程



1. 基于maven导入坐标

Spring data elasticsearch 对 elasticsearch api 简化封装

导入elasticsearch的jar包和spring-data整合elsticsearch的jar包。

<dependency>

<groupId>org.elasticsearch</groupId>

<artifactId>elasticsearch</artifactId>

<version>2.4.0</version>

</dependency>

<dependency>

<groupId>org.springframework.data</groupId>

<artifactId>spring-data-elasticsearch</artifactId>

<version>2.0.4.RELEASE</version>

</dependency>

导入spring-test 和 junit 编写测试用例

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-test</artifactId>

<version>4.2.8.RELEASE</version>

</dependency>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.12</version>

</dependency>

导入Slf4j-log4j 日志包

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-log4j12</artifactId>

<version>1.7.12</version>

</dependency>

1. 在src/main/resources 下建立 applicationContext.xml 和 log4j.properties

学习参考：《spring-data-elasticsearch-reference.pdf》



引入 spring data elasticsearch 名称空间 ，并添加配置

<?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 -->

<elasticsearch:transport-client id="*client*" cluster-nodes=*"localhost:9300"* />

<!-- spring data elasticSearcheDao 必须继承 ElasticsearchTemplate -->

<bean id=*"elasticsearchTemplate"* class=*"org.springframework.data.elasticsearch.core.ElasticsearchTemplate"*>

<constructor-arg name=*"client"* ref=*"client"*></constructor-arg>

</bean>

</beans>

log4j.properties

### direct log messages to stdout ###

log4j.appender.stdout=org.apache.log4j.ConsoleAppender

log4j.appender.stdout.Target=System.out

log4j.appender.stdout.layout=org.apache.log4j.PatternLayout

log4j.appender.stdout.layout.ConversionPattern=%d{ABSOLUTE} %5p %c**{1}**:%L - %m%n

### direct messages to file mylog.log ###

log4j.appender.file=org.apache.log4j.FileAppender

log4j.appender.file.File=c:/mylog.log

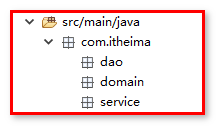
log4j.appender.file.layout=org.apache.log4j.PatternLayout

log4j.appender.file.layout.ConversionPattern=%d{ABSOLUTE} %5p %c**{1}**:%L - %m%n

### set log levels - for more verbose logging change 'info' to 'debug' ###

log4j.rootLogger=info, stdout

1. 创建domain、dao、service 包



1. 编写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 + "]";

}

}

1. 编写Service ，处理操作索引库业务逻辑

ArticleService接口：

**public** **interface** ArticleService {

}

ArticleServiceImpl实现类

@Service

**public** **class** ArticleServiceImpl **implements** ArticleService{

}

1. 索引和映射如何创建 --- 基于spring data elasticsearch注解

在使用spring data elasticsearch开发， 需要将索引和映射信息 配置实体类上面

@Document 文档对象 （索引信息、文档类型 ）

indexName：索引名称

type：索引类型

@Id 文档主键 唯一标识

@Field 每个文档的字段配置（类型、是否分词、是否存储、分词器 ）

index：是否建立索引，同时设置是否分词，如果不建立索引，无法搜索

stroe：是否存储，如果不存储，无法查询数据

searchAnalyzer：采用的分词器

@Document(indexName="blog3",type="article")

**public** **class** Article {

@Id

@Field(index=FieldIndex.***not\_analyzed***,store=**true**)

**private** Integer id;

@Field(index=FieldIndex.***analyzed***,analyzer="ik",store=**true**,searchAnalyzer="ik")

**private** String title;

@Field(index=FieldIndex.***analyzed***,analyzer="ik",store=**true**,searchAnalyzer="ik")

**private** String content;

@MultiField(

mainField = @Field(type=FieldType.***Nested***, index=FieldIndex.***analyzed***,store=**true**)

)

Author author = **new** Author();

}

文字对象中包括作者的对象。

Author.java

**public** **class** Author {

**private** String id;

**private** String name;

**public** String getId() {

**return** id;

}

**public** **void** setId(String id) {

**this**.id = id;

}

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

@Override

**public** String toString() {

**return** "Author [id=" + id + ", name=" + name + "]";

}

}

创建测试类：ArticleServiceTest.java，通过ElasticsearchTemplate 创建索引和添加映射

@RunWith(SpringJUnit4ClassRunner.**class**)

@ContextConfiguration(locations="classpath:applicationContext.xml")

**public** **class** ArticleServiceTest {

@Autowired

**private** ArticleService articleService;

}

## 调用ElasticsearchTemplate

第一步：在Service中注入ElasticsearchTemplate，通过ElasticsearchTemplate对索引库进行操作。

@Service

**public** **class** ArticleServiceImpl **implements** ArticleService{

@Autowired

**private** ElasticsearchTemplate elasticsearchTemplate;

}

第二步：创建测试类TestElasticsearchTemplate.java

@RunWith(SpringJUnit4ClassRunner.**class**)

@ContextConfiguration(locations="classpath:applicationContext.xml")

**public** **class** TestElasticsearchTemplate {

@Autowired

**private** ArticleService articleService;

}

### 【创建索引/删除索引】

TestElasticsearchTemplate.java

/\*\*创建索引和映射\*/

@Test

**public** **void** createIndex(){

articleService.createIndex(Article.**class**);

}

/\*\*删除索引\*/

@Test

**public** **void** deleteIndex(){

articleService.deleteIndex(Article.**class**);

}

ArticleServiceImpl.java

/\*\*创建索引\*/

**public** **void** createIndex(Class<Article> articleClass) {

**if** (!elasticsearchTemplate.indexExists(articleClass)) {

elasticsearchTemplate.createIndex(articleClass);

elasticsearchTemplate.putMapping(articleClass);

}

}

/\*\*删除索引\*/

**public** **void** deleteIndex(Class<Article> articleClass) {

**if** (!elasticsearchTemplate.indexExists(articleClass)) {

elasticsearchTemplate.deleteIndex(articleClass);

}

}

### 【创建文档/更新文档】

IndexQuery的使用

TestElasticsearchTemplate.java

/\*\*新增文档\*/

@Test

**public** **void** createDocument(){

Article article = **new** Article();

article.setId(2);

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。");

Author author = **new** Author();

author.setId("1");

author.setName("冰冰");

article.setAuthor(author);

articleService.createDocument(article);

}

/\*\*更新文档\*/

@Test

**public** **void** updateDocument(){

Article article = **new** Article();

article.setId(1);

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。");

Author author = **new** Author();

author.setId("1");

author.setName("若若");

article.setAuthor(author);

articleService.createDocument(article);

}

ArticleServiceImpl.java

/\*\*创建/更新文档数据\*/

**public** **void** createDocument(Article article) {

IndexQuery indexQuery = **new** IndexQueryBuilder()

.withId(article.getId().toString()).withObject(article).build();

elasticsearchTemplate.index(indexQuery);

}

### 【更新文档】

使用UpdateQuery

TestElasticsearchTemplate.java

/\*\*更新文档\*/

@Test

**public** **void** updateDocument\_1(){

Article article = **new** Article();

article.setId(1);

article.setTitle("标题，ElasticSearch是一个基于Lucene的搜索服务器");

article.setContent("内容，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。");

Author author = **new** Author();

author.setId("1");

author.setName("周芷若");

article.setAuthor(author);

articleService.updateDocument(article);

}

ArticleServiceImpl.java

/\*\*更新文档\*/

**public** **void** updateDocument(Article article) {

IndexRequest indexRequest = **new** IndexRequest();

// 只能设置一个资源，最后的一个设置是生效的

// indexRequest.source("id", article.getId());

// indexRequest.source("title",article.getTitle());

// indexRequest.source("content",article.getContent());

// 同时将资源放置到一行（推荐）

indexRequest.source("id",article.getId(),"title",article.getTitle(),"content",article.getContent());

UpdateQuery updateQuery = **new** UpdateQueryBuilder().withId(article.getId().toString())

.withDoUpsert(**true**).withClass(Article.**class**)

.withIndexRequest(indexRequest).build();

elasticsearchTemplate.update(updateQuery);

}

### 【删除文档】

TestElasticsearchTemplate.java

@Test

**public** **void** deleteDocument(){

Article article = **new** Article();

article.setId(1);

articleService.deleteDocument(article);

}

ArticleServiceImpl.java

/\*\*删除文档\*/

**public** **void** deleteDocument(Article article) {

elasticsearchTemplate.delete(Article.**class**,article.getId().toString());

}

### 【查询文档数量】

TestElasticsearchTemplate.java

/\*\*查询文档数量\*/

@Test

**public** **void** findDocumentCount(){

**long** count = articleService.findDocumentCount();

System.***out***.println(count);

}

ArticleServiceImpl.java

/\*\*查询数量\*/

**public** **long** findDocumentCount() {

CriteriaQuery criteriaQuery = **new** CriteriaQuery(**new** Criteria());

**long** count = elasticsearchTemplate.count(criteriaQuery, Article.**class**);

**return** count;

}

### 【使用主键ID查询文档】

TestElasticsearchTemplate.java

/\*\*ID查询唯一文档\*/

@Test

**public** **void** findDocumentById(){

Integer id = 2;

Article article = articleService.findDocumentById(id);

System.***out***.println(article);

}

ArticleServiceImpl.java

/\*\*使用ID主键查询\*/

**public** Article findDocumentById(Integer id) {

/\*\*1:GetQuery\*/

GetQuery query = **new** GetQuery();

query.setId(String.*valueOf*(id));

**return** elasticsearchTemplate.queryForObject(query, Article.**class**);

}

### 【使用多个主键ID查询文档】

TestElasticsearchTemplate.java

/\*\*使用多个ID查询\*/

@Test

**public** **void** findDocumentListByIds(){

Integer id = 1;

Integer id2 = 2;

List<Article> list = articleService.findDocumentListByIds(id,id2);

System.***out***.println(list);

}

ArticleServiceImpl.java

/\*\*使用多个ID查询\*/

**public** List<Article> findDocumentListByIds(Integer id, Integer id2) {

SearchQuery query = **new** NativeSearchQueryBuilder().withIds(Arrays.*asList*(String.*valueOf*(id), String.*valueOf*(id2))).build();

LinkedList<Article> list = elasticsearchTemplate.multiGet(query, Article.**class**);

**return** list;

}

### 【查询所有文档】

TestElasticsearchTemplate.java

/\*\*查询所有文档\*/

@Test

**public** **void** findDocumentList(){

List<Article> list = articleService.findDocumentList();

System.***out***.println(list);

}

ArticleServiceImpl.java

/\*\*查询所有文档\*/

**public** List<Article> findDocumentList() {

QueryBuilder queryBuilder = **new** MatchAllQueryBuilder();

/\*\*1：使用SearchQuery\*/

// SearchQuery query = new NativeSearchQueryBuilder().withQuery(queryBuilder).build();

/\*\*2：使用StringQuery\*/

// StringQuery query = new StringQuery(queryBuilder.toString());

/\*\*3:使用CriteriaQuery\*/

Criteria criteria = **new** Criteria();

CriteriaQuery query = **new** CriteriaQuery(criteria);

**return** elasticsearchTemplate.queryForList(query, Article.**class**);

}

### 【词条查询】

Term

TestElasticsearchTemplate.java

/\*\*词条查询\*/

@Test

**public** **void** findDocumentListByTerm(){

String content = "搜索";

List<Article> list = articleService.findDocumentListByTerm(content);

System.***out***.println(list);

}

ArticleServiceImpl.java

/\*\*词条搜索\*/

**public** List<Article> findDocumentListByTerm(String content) {

QueryBuilder queryBuilder = **new** TermQueryBuilder("content", content);

SearchQuery query = **new** NativeSearchQueryBuilder()

.withQuery(queryBuilder)

// .withIndices(INDEX\_NAME)

// .withTypes(TYPE\_NAME)

// .withFields("message")

// .withPageable(new PageRequest(0, 10))

.build();

**return** elasticsearchTemplate.queryForList(query, Article.**class**);

}

### 【通配符查询】

TestElasticsearchTemplate.java

/\*\*通配符查询，\*匹配多个字符，?匹配一个字符\*/

@Test

**public** **void** findDocumentListByWildcard(){

String content = "\*搜?";

List<Article> list = articleService.findDocumentListByWildcard(content);

System.***out***.println(list);

}

ArticleServiceImpl.java

/\*\*通配符查询\*/

**public** List<Article> findDocumentListByWildcard(String content) {

QueryBuilder queryBuilder = **new** WildcardQueryBuilder("content",content);

SearchQuery query = **new** NativeSearchQueryBuilder()

.withQuery(queryBuilder)

// .withIndices(INDEX\_NAME)

// .withTypes(TYPE\_NAME)

// .withFields("message")

// .withPageable(new PageRequest(0, 10))

.build();

**return** elasticsearchTemplate.queryForList(query, Article.**class**);

}

### 【模糊查询】

即便一个单词书写有误，也能查询到对应的结果

TestElasticsearchTemplate.java

/\*\*即便一个单词书写有误，也能查询到对应的结果\*/

@Test

**public** **void** findDocumentListByFuzzy(){

// 即便一个单词书写有误，也能查询到对应的结果

String content = "lucena";

List<Article> list = articleService.findDocumentListByFuzzy(content);

System.***out***.println(list);

}

ArticleServiceImpl.java

/\*\*模糊查询\*/

**public** List<Article> findDocumentListByFuzzy(String content) {

QueryBuilder queryBuilder = **new** FuzzyQueryBuilder("content",content);

SearchQuery query = **new** NativeSearchQueryBuilder()

.withQuery(queryBuilder)// 指定使用的查询构造器

.build();

**return** elasticsearchTemplate.queryForList(query, Article.**class**);

}

### 【高级查询bool查询】

组织查询条件

TestElasticsearchTemplate.java

/\*\*高级查询，bool查询\*/

@Test

**public** **void** findDocumentListByBool(){

String title = "elasticsearch";

String content = "搜索";

List<Article> list = articleService.findDocumentListByBool(title,content);

System.***out***.println(list);

}

ArticleServiceImpl.java

/\*\*组合查询\*/

**public** List<Article> findDocumentListByBool(String title,String content) {

QueryBuilder queryBuilder = **new** BoolQueryBuilder().must(**new** TermQueryBuilder("title", title)).must(**new** TermQueryBuilder("content", content));

SearchQuery query = **new** NativeSearchQueryBuilder().withQuery(queryBuilder).build();

**return** elasticsearchTemplate.queryForList(query, Article.**class**);

}

### 【排序】

TestElasticsearchTemplate.java

/\*\*排序\*/

@Test

**public** **void** findDocumentListBySort(){

List<Article> list = articleService.findDocumentListBySort();

System.***out***.println(list);

}

ArticleServiceImpl.java

/\*\*排序\*/

**public** List<Article> findDocumentListBySort() {

QueryBuilder queryBuilder = **new** MatchAllQueryBuilder();

SearchQuery query = **new** NativeSearchQueryBuilder().withQuery(queryBuilder).withSort(**new** FieldSortBuilder("id").order(SortOrder.***DESC***)).build();

**return** elasticsearchTemplate.queryForList(query, Article.**class**);

}

### 【Criteria条件查询】

TestElasticsearchTemplate.java

/\*\*Criteria条件查询\*/

@Test

**public** **void** findDocumentListByCriteria(){

List<Article> list = articleService.findDocumentListByCriteria();

System.***out***.println(list);

}

ArticleServiceImpl.java

/\*\*Criteria条件查询\*/

**public** List<Article> findDocumentListByCriteria() {

// content字段包含“搜索”

Criteria criteria = **new** Criteria("content").contains("搜索").and("title").contains("Elasticsearch");

// Criteria criteria = new Criteria().where("id").is("1").and("title").is("Elasticsearch").and("content").is("搜索");

CriteriaQuery query = **new** CriteriaQuery(criteria);

// query.addIndices(INDEX\_NAME);

// query.addTypes(TYPE\_NAME);

// query.setPageable(new PageRequest(0, 10));

**return** elasticsearchTemplate.queryForList(query, Article.**class**);

}

### 【分页】

TestElasticsearchTemplate.java

/\*\*新增文档100条\*/

@Test

**public** **void** createDocument100(){

**for**(**int** i=1;i<=100;i++){

Article article = **new** Article();

article.setId(i);

article.setTitle(i+"elasticSearch 3.0版本发布..，");

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.createDocument(article);

}

}

/\*\*分页条件查询\*/

@Test

**public** **void** findDocumentListByPage(){

String content = "搜索";

Page<Article> page = articleService.findDocumentListByPage(content);

System.***out***.println("总页数："+page.getTotalPages());

System.***out***.println("总记录数："+page.getTotalElements());

**for**(Article article:page.getContent()){

System.***out***.println(article);

}

}

ArticleServiceImpl.java

/\*\*分页查询\*/

**public** Page<Article> findDocumentListByPage(String content) {

/\*\*1：SearchQuery\*/

// SearchQuery query = new NativeSearchQueryBuilder().withQuery(new TermQueryBuilder("content", content))

// .withPageable(new PageRequest(0, 20)) // 从第1条，到第20条

// .build();

/\*\*2：CriteriaQuery\*/

Criteria criteria = **new** Criteria("content").is(content);

CriteriaQuery query = **new** CriteriaQuery(criteria);

query.setPageable(**new** PageRequest(0, 20));

Page<Article> page = elasticsearchTemplate.queryForPage(query, Article.**class**);

**return** page;

}

### 【文字分页高亮】

TestElasticsearchTemplate.java

/\*\*文字高亮\*/

@Test

**public** **void** findDocumentListByPageAndHighLight(){

String queryString = "搜索";

Page<Article> page = articleService.findDocumentListByPageAndHighLight(queryString);

System.***out***.println("总页数："+page.getTotalPages());

System.***out***.println("总记录数："+page.getTotalElements());

**for**(Article article:page.getContent()){

System.***out***.println(article);

}

}

ArticleServiceImpl.java

/\*\*文字高亮\*/

**public** Page<Article> findDocumentListByPageAndHighLight() {

QueryBuilder queryBuilder = **new** QueryStringQueryBuilder("搜索").field("content").field("title");

SearchQuery searchQuery = **new** NativeSearchQueryBuilder().withQuery(queryBuilder)

.withHighlightFields(**new** HighlightBuilder.Field("content").preTags("<em>").postTags("</em>").fragmentSize(10)

,**new** HighlightBuilder.Field("title").preTags("<em>").postTags("</em>").fragmentSize(10))

.build();

// Page<Article> article = elasticsearchTemplate.queryForPage(searchQuery, Article.class);

Long count = elasticsearchTemplate.count(searchQuery,Article.**class**);

// System.out.println("count:"+count);

// 只封装分页查询的结果，没有总的记录数

Page<Article> article = elasticsearchTemplate.queryForPage(searchQuery, Article.**class**,**new** SearchResultMapper() {

**public** <T> Page<T> mapResults(SearchResponse response, Class<T> clazz, Pageable pageable) {

List<Article> list = **new** ArrayList<Article>();

System.***out***.println(pageable.getPageNumber()+" "+pageable.getPageSize());

System.***out***.println(response.contextSize());

**for** (SearchHit searchHit : response.getHits()) {

**if** (response.getHits().getHits().length <= 0) {

**return** **null**;

}

Article article = **new** Article();

article.setId(Integer.*parseInt*(searchHit.getId()));

**if**(searchHit.getHighlightFields().get("title")==**null**){

article.setTitle((String)searchHit.getSource().get("title"));

}

**else**{

article.setTitle(searchHit.getHighlightFields().get("title").fragments()[0].toString());

}

// 如果字段没有高亮的值，使用searchHit.getSource().get("content")获取值

**if**(searchHit.getHighlightFields().get("content")==**null**){

article.setContent((String)searchHit.getSource().get("content"));

}

// 如果字段存在高亮的值，使用searchHit.getHighlightFields().get("content").fragments()[0].toString()获取值

**else**{

article.setContent(searchHit.getHighlightFields().get("content").fragments()[0].toString());

}

list.add(article);

}

Page<Article> page = **new** PageImpl<Article>(list);

**return** (Page<T>) page;

}

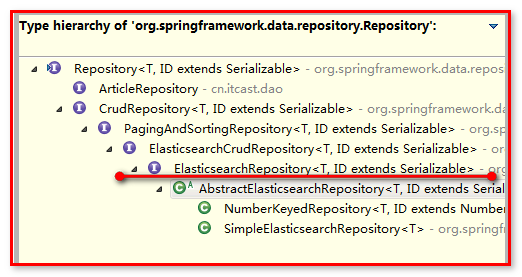
});

**return** article;

}

## Repository的 CRUD操作

1. 编写DAO



编写DAO 自动操作elasticsearch ，需要继承 ElasticsearchRepository 接口

**public** **interface** ArticleRepository **extends** ElasticsearchRepository<Article, Integer> {

}

ElasticsearchRepository 继承了CrudRepository，提供增删改查 save、delete、findAll 、findOne等基本功能

同时继承了PagingAndSortingRepository， 提供分页和排序功能

1. ：ArticleServiceImpl.java类

@Service

**public** **class** ArticleServiceImpl **implements** ArticleService{

@Autowired

**private** ArticleRepository articleRepository;

}

1. 创建测试类，TestRepository.java

@RunWith(SpringJUnit4ClassRunner.**class**)

@ContextConfiguration(locations="classpath:applicationContext.xml")

**public** **class** TestRepository {

@Autowired

**private** ArticleService articleService;

}

### 【创建索引】

TestRepository.java

/\*\*创建索引和映射\*/

@Test

**public** **void** createIndex(){

articleService.createIndex(Article.**class**);

}

/\*\*创建索引和映射\*/

@Test

**public** **void** deleteIndex(){

articleService.deleteIndex(Article.**class**);

}

ArticleServiceImpl.java

/\*\*创建索引\*/

**public** **void** createIndex(Class<Article> articleClass) {

**if** (!elasticsearchTemplate.indexExists(articleClass)) {

elasticsearchTemplate.createIndex(articleClass);

elasticsearchTemplate.putMapping(articleClass);

}

}

/\*\*删除索引\*/

**public** **void** deleteIndex(Class<Article> articleClass) {

**if** (!elasticsearchTemplate.indexExists(articleClass)) {

elasticsearchTemplate.deleteIndex(articleClass);

}

}

### 【创建文档】

TestRepository.java

/\*\*测试保存\*/

@Test

**public** **void** save(){

Article article = **new** Article();

Author author = **new** Author();

author.setId("1");

author.setName("萱萱");

article.setId(1);

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。");

article.setAuthor(author);

articleService.save(article);

}

ArticleServiceImpl.java

**public** **void** save(Article article) {

articleRepository.save(article);

}

### 【更新文档】

TestRepository.java

/\*\*测试更新\*/

@Test

**public** **void** update(){

Article article = **new** Article();

Author author = **new** Author();

author.setId("2");

author.setName("菲菲");

article.setId(1);

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。");

article.setAuthor(author);

articleService.save(article);

}

ArticleServiceImpl.java

**public** **void** save(Article article) {

articleRepository.save(article);

}

### 【删除文档】

TestRepository.java

/\*\*测试删除\*/

@Test

**public** **void** delete(){

Article article = **new** Article();

article.setId(1);

articleService.delete(article);

}

ArticleServiceImpl.java

**public** **void** delete(Article article) {

articleRepository.delete(article);

}

### 【ID查询文档】

TestRepository.java

/\*\*使用ID查询\*/

@Test

**public** **void** findOne(){

Integer id = 1;

Article article = articleService.findOne(id);

System.***out***.println(article);

}

ArticleServiceImpl.java

**public** Article findOne(Integer id) {

**return** articleRepository.findOne(id);

}

### 【查询所有】

TestRepository.java

/\*\*查询所有\*/

@Test

**public** **void** findAll(){

Iterable<Article> list = articleService.findAll();

**for**(Article article:list){

System.***out***.println(article);

}

}

ArticleServiceImpl.java

**public** Iterable<Article> findAll() {

// 查询所有

Iterable<Article> iter = articleRepository.findAll();

**return** iter;

}

### 【批量插入100条】

TestRepository.java

/\*\*批量插入100条数据\*/

@Test

**public** **void** save100(){

**for**(**int** i=1;i<=100;i++){

Article article = **new** Article();

article.setId(i);

article.setTitle(i+"ElasticSearch是一个基于Lucene的搜索服务器");

article.setContent(i+"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);

}

}

### 【排序】

TestRepository.java

/\*\*排序查询\*/

@Test

**public** **void** findAllSort(){

Iterable<Article> list = articleService.findAllBySort();

**for**(Article article:list){

System.***out***.println(article);

}

}

ArticleServiceImpl.java

/\*\*排序\*/

**public** Iterable<Article> findAllBySort() {

// 排序

Iterable<Article> iter = articleRepository.findAll(**new** Sort(**new** Sort.Order(Sort.Direction.***ASC***, "id")));

**return** iter;

}

### 【分页】

TestRepository.java

/\*\*分页查询\*/

@Test

**public** **void** findAllPage(){

Pageable pageable = **new** PageRequest(0, 10);

Page<Article> page = articleService.findAll(pageable);

**for**(Article article:page.getContent()){

System.***out***.println(article);

}

}

ArticleServiceImpl.java

**public** Page<Article> findAll(Pageable pageable) {

**return** articleRepository.findAll(pageable);

}

### 【条件查询】

条件查询，在ArticleServiceImpl中定义

/\*\*条件查询\*/

**public** Page<Article> findSearchByContent(String content) {

QueryBuilder query = **new** TermQueryBuilder("content",content);

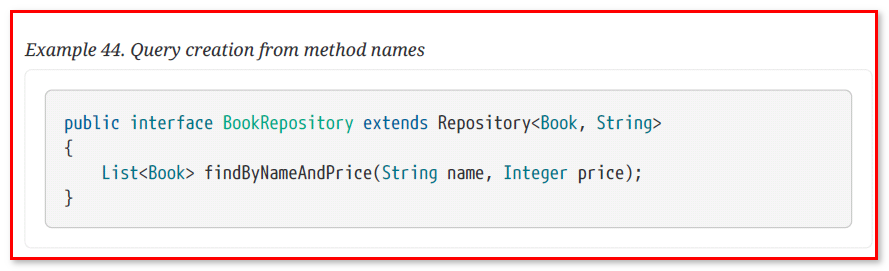
SearchQuery searchQuery = **new** NativeSearchQuery(query);

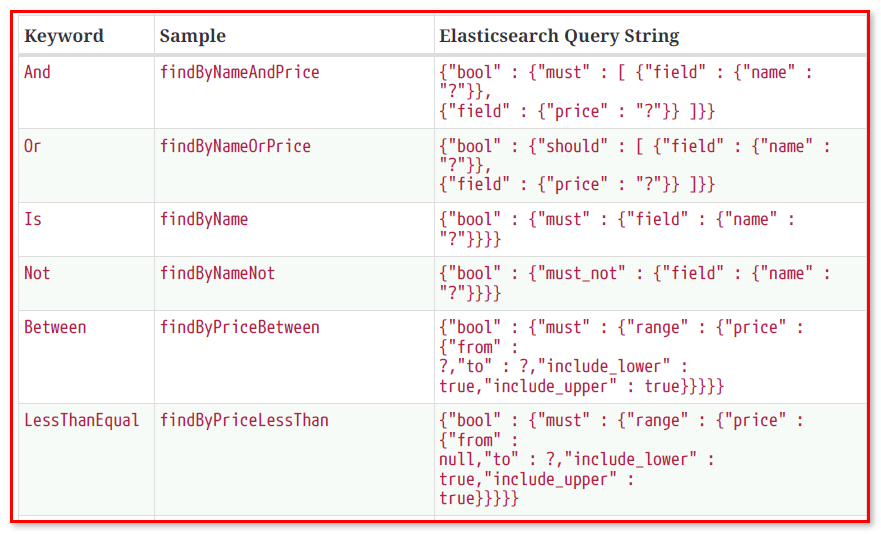
// 默认采用分页的效果，每页10条显示

**return** articleRepository.search(searchQuery);

}

自定义查询方法，例如：





（1）查询标题方法 （不分页）

Dao层：ArticleRepository.java

**public** **interface** ArticleRepository **extends** ElasticsearchRepository<Article, Integer> {

List<Article> findByTitle(String title);

}

Service层：ArticleServiceImpl.java

**public** List<Article> findByTitle(String title) {

**return** articleRepository.findByTitle(title);

}

测试：TestRepository.java

/\*\*条件查询\*/

@Test

**public** **void** findByTitle(){

String title = "搜索";

List<Article> list = articleService.findByTitle(title);

**for**(Article article:list){

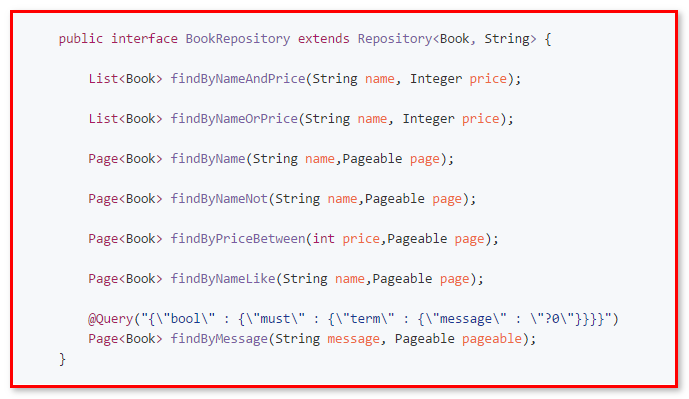
System.***out***.println(article);

}

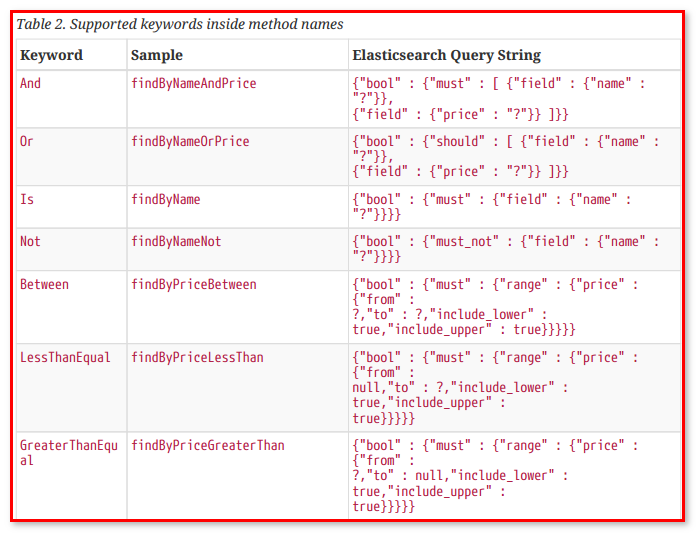
}

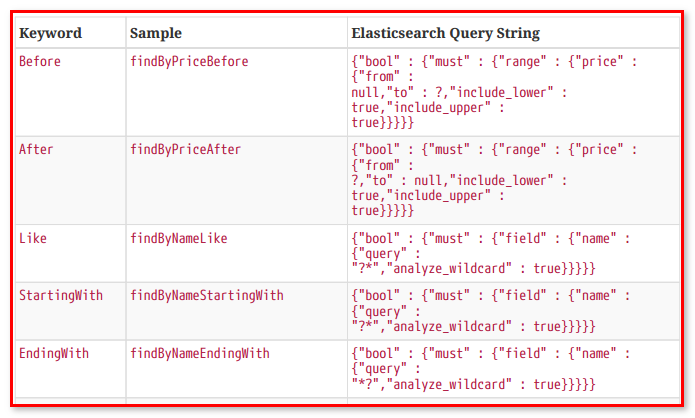
### 【多条件查询】

https://github.com/spring-projects/spring-data-elasticsearch



http://docs.spring.io/spring-data/elasticsearch/docs/current/reference/pdf/spring-data-elasticsearch-reference.pdf







（1）查询标题方法 （不分页）

Dao层：ArticleRepository.java

**public** **interface** ArticleRepository **extends** ElasticsearchRepository<Article, Integer> {

List<Article> findByTitleLikeAndContent(String title, String content);

}

Service层：ArticleServiceImpl.java

**public** List<Article> findByTitleLikeAndContent(String title, String content) {

**return** articleRepository.findByTitleLikeAndContent(title,content);

}

测试：TestRepository.java

/\*\*多条件查询\*/

@Test

**public** **void** findByTitleLikeAndContent(){

String title = "\*搜?";

String content = "全文";

List<Article> list = articleService.findByTitleLikeAndContent(title,content);

**for**(Article article:list){

System.***out***.println(article);

}

}

### 【分页、排序、条件查询】

分页条件查询，只需要在查询方法中，添加Pageable对象

排序条件查询，只需要在查询方法中，添加Sort对象

Dao层：ArticleRepository.java

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

Service层：ArticleServiceImpl.java

**public** Page<Article> findByTitleLikeAndContentPage(String title, String content, Pageable pageable) {

**return** articleRepository.findByTitleLikeAndContentOrderByIdDesc(title,content, pageable);

}

测试类：ArticleServiceTest.java

/\*\*条件查询、分页查询、排序\*/

@Test

**public** **void** findByTitleLikeAndContentPage(){

String title = "\*搜?";

String content = "全文";

Pageable pageable = **new** PageRequest(0, 10);

Page<Article> page = articleService.findByTitleLikeAndContentPage(title,content,pageable);

System.***out***.println("总记录数"+page.getTotalElements());

System.***out***.println("总页数"+page.getTotalPages());

**for**(Article article:page.getContent()){

System.***out***.println(article);

}

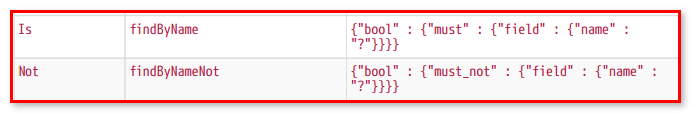
}

### 【声明查询Query】

通过解析方法名来构建查询有时可能满足不了开发者的需求，或造成方法名可读性差。这时可以使用@Query注解来声明一个查询(参考[使用@Query注解](https://es.yemengying.com/5/5.2/5.2.3.html))

#### 1：单字段搜索

使用is



ArticleRepository.java

@Query("{\"bool\" : {\"must\" : {\"term\" : {\"title\" : \"?0\"}}}}")

List<Article> queryByTitle(String title);

ArticleServiceImpl.java

**public** List<Article> queryByTitle(String title) {

**return** articleRepository.queryByTitle(title);

}

TestRepository.java

/\*\*声明查询Query\*/

@Test

**public** **void** queryByTitle(){

String title = "搜索";

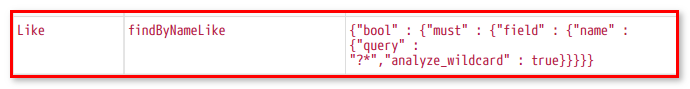
List<Article> list = articleService.queryByTitle(title);

System.***out***.println(list);

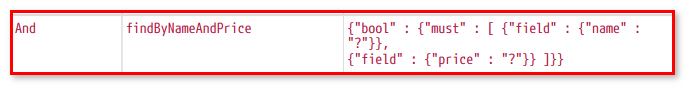
}

#### 2：多字段搜索

使用like



使用and



ArticleRepository.java

@Query("{\"bool\" : {\"must\" : [{\"wildcard\" : {\"title\" : \"?0\"}},{\"term\" : {\"content\" : \"?1\"}} ]}}")

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

ArticleServiceImpl.java

**public** Page<Article> queryByTitleLikeAndContentPage(String title, String content, Pageable pageable) {

**return** articleRepository.queryByTitleLikeAndContentPage(title,content, pageable);

}

TestRepository.java

/\*\*声明查询Query\*/

@Test

**public** **void** queryByTitleLikeAndContentPage(){

String title = "\*搜?";

String content = "全文";

Pageable pageable = **new** PageRequest(0, 10);

Page<Article> page = articleService.queryByTitleLikeAndContentPage(title,content,pageable);

System.***out***.println("总记录数"+page.getTotalElements());

System.***out***.println("总页数"+page.getTotalPages());

**for**(Article article:page.getContent()){

System.***out***.println(article);

}

}