elasticSearch搜索引擎

【内容回顾】：

1 redis-

2 Spring data redis的使用---redisTemplate操作的(模板操作的代码需要记忆)

3 MQ-

应用解耦

流量削峰

消息通信

日志处理

activeMQ（apache），rabbitMQ，rocketMQ（阿里巴巴），kafka

消息模型：P2P ，pub/sub

mq实现方式：1 不与Spring整合 2 与Spring整合(重点)

mq中的队列itheima18，有多个消费者，能不能指定某个消费者去消费某个指定的任务

【今天内容安排】：

1、ElasticSearch 搜索服务器使用 （安装、配置、集成IK分词器、创建维护索引结构，实现对数据进行全文检索 ）

\* 创建索引

\* 创建文档

\* 查询文档（简单查询、复杂查询、分页查询、条件查询、多条件查询..）

\* 分页

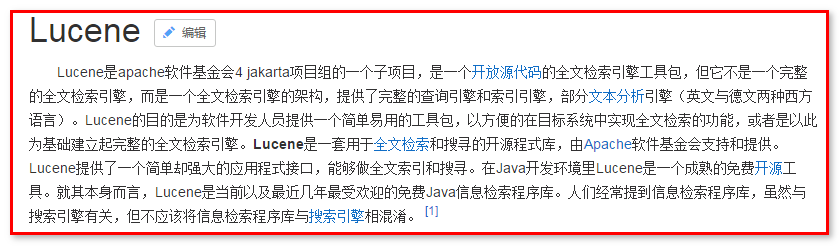
\* 高亮

2、spring Data Elasticsearch，用来简化Elasticsearch的开发

# ElasticSearch

## 什么是elasticsearch

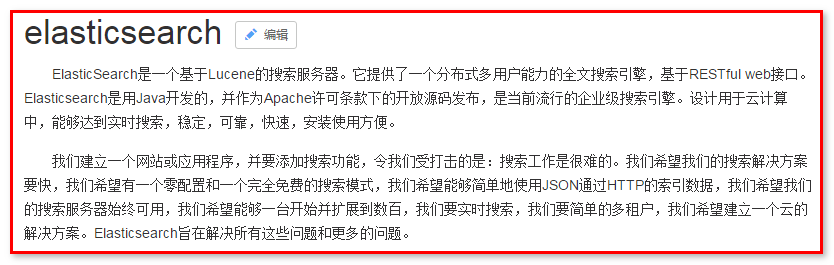
早期：



Lucene就是一套 全文检索 编程API ， 基于Lucene对数据建立索引，进行查询

很多框架对lucene进行了封装

什么是ElasticSearch ?

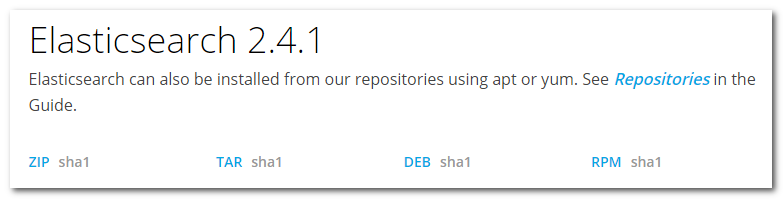


现在企业开发中，更常用是的solr搜索服务器和ElasticSearch搜索服务器

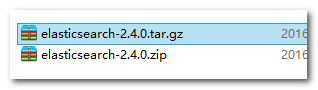
## ElasticSearch 安装配置使用入门

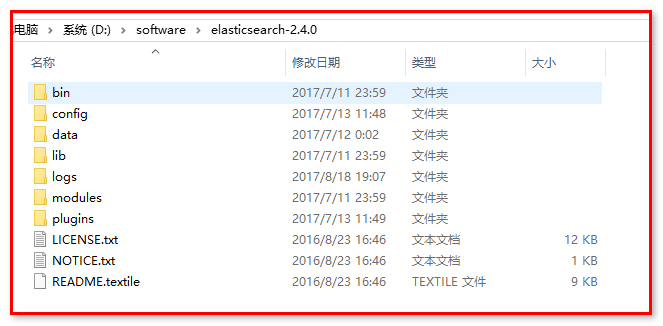


官网： <https://www.elastic.co/products/elasticsearch>



Window系统下载zip版本，linux系统下载tar版本





bin 存放elasticSearch 运行命令

config 存放配置文件

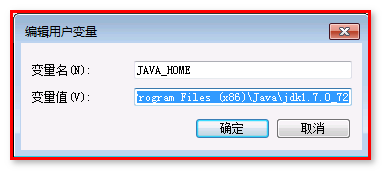
lib 存放elasticSearch运行依赖jar包

modules 存放elasticSearch 模块

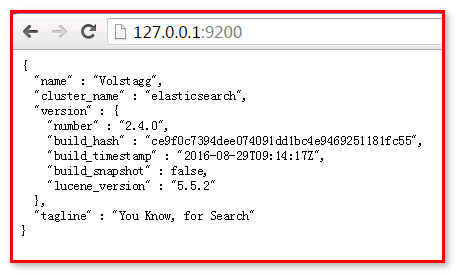
plugins 存放插件

运行elasticSearch/bin/elasticsearch.bat 文件

配置JAVA\_HOME环境变量



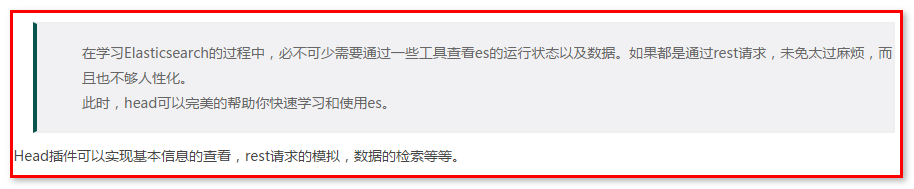
访问 <http://127.0.0.1:9200>



安装elasticSearch成功

## ElasticSearch 插件安装 es head

es head： 就是图形化界面工具的插件



### elasticsearch-head

elasticsearch-head是一个elasticsearch的集群管理工具，它是完全由**[HTML5](http://lib.csdn.net/base/html5" \o "HTML5知识库" \t "http://blog.csdn.net/xifeijian/article/details/_blank)**编写的独立网页程序，你可以通过插件把它集成到es。或直接下载源码，在本地打开index.html运行它。该工具的**[Git](http://lib.csdn.net/base/git" \o "Git知识库" \t "http://blog.csdn.net/xifeijian/article/details/_blank)**地址是： https://github.com/zt1115798334/elasticsearch-head-master

插件安装方法：

1.elasticsearch/bin/plugin -install Aconex/elasticsearch-head

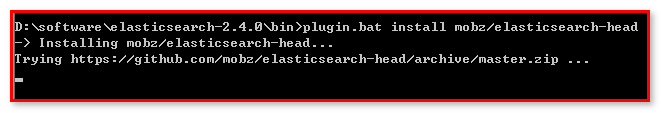
2.运行es

3.打开http://localhost:9200/\_plugin/head/

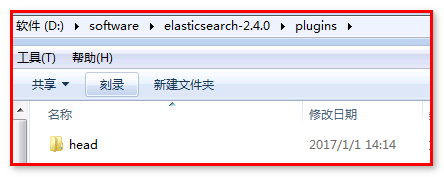
不想通过插件集成到es的话就可以直接在git上下载源码到本地运行。

在地址栏输入es服务器的ip地址和端口，点connect就可以连接到集群。下面是连接后的视图。这是主界面，在这里可以看到es集群的基本信息（如：节点情况，索引情况）

%elasticsearch%/bin/plugin.bat install mobz/elasticsearch-head



执行较慢，下载插件后，查看elasticsearch的plugin文件夹：



访问 <http://localhost:9200/_plugin/head/>



方案二：

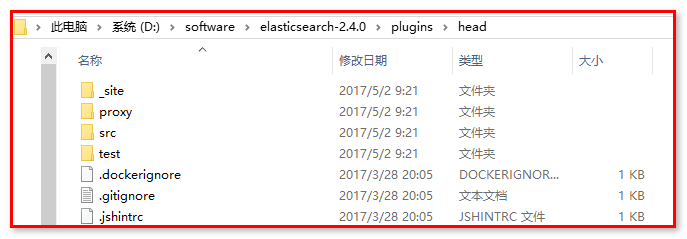
1、插件下载地址

<https://github.com/zt1115798334/elasticsearch-head-master>

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-2.2.0/plugins/head/

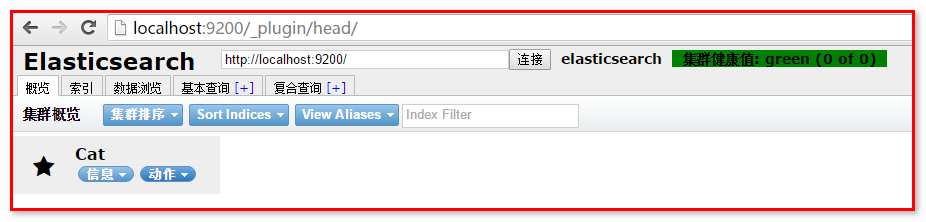


3、启动es  bin/elasticsearch

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



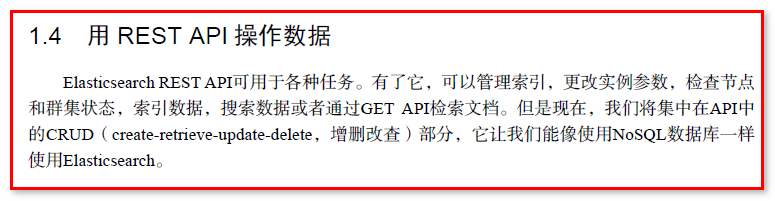
## ElasticSearch 基本操作入门

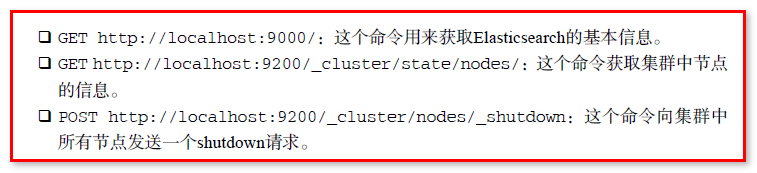
参考书籍 ： 《Elasticsearch服务器开发（第2版）.pdf 》



全文检索： 针对文本中每个词，创建词条建立索引，进行搜索。

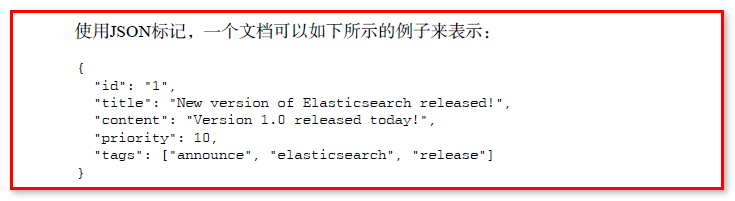
ElasticSearch 操作服务器上的数据，通过 Rest API 操作数据



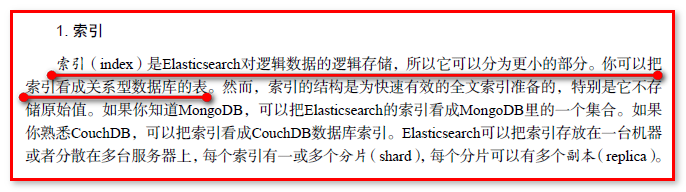


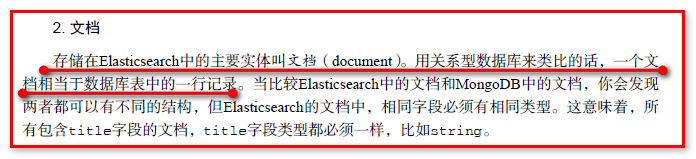


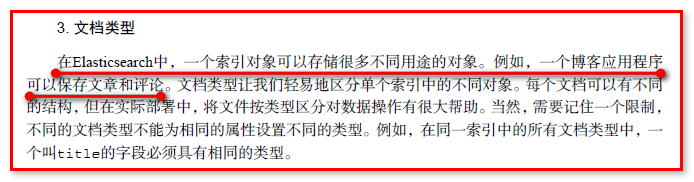
如果操作Elasticsearch上数据，访问提供Rest API的URL地址，传递json数据给服务器

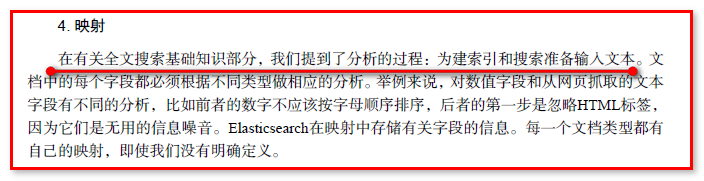


### ElasticSearch 基础 数据架构的主要概念

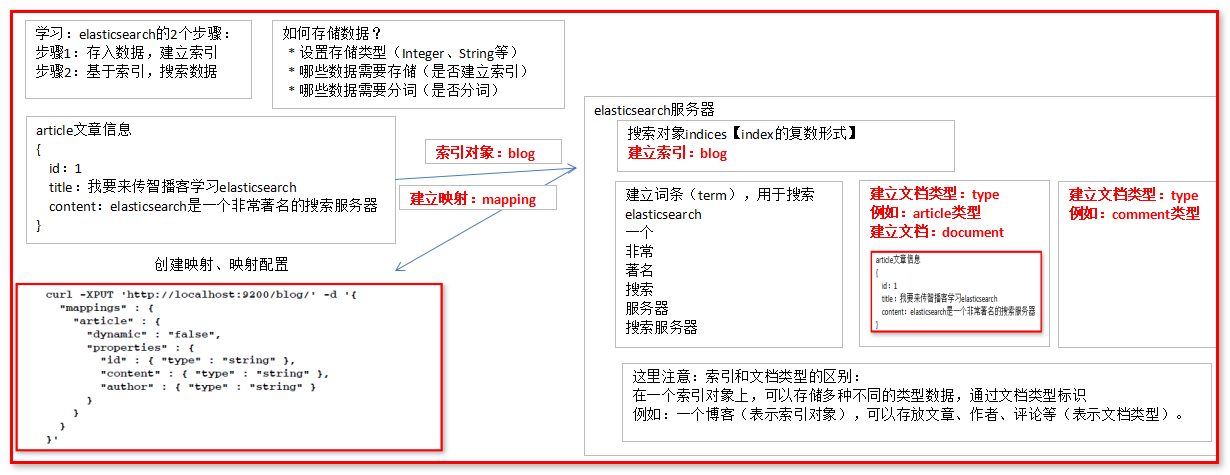
(索引库)







图解

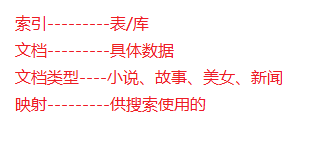


索引对象（blog）： 存储数据的表(库)结构 ，任何搜索数据，存放在索引对象上 。

映射（mapping）： 数据如何存放到索引对象上，需要有一个映射配置， 包括：数据类型、是否存储、是否分词 … 等。

文档类型（type）： 一个索引对象 存放多种类型数据， 数据用文档类型进行标识

文档（document）： 一条数据记录， 存在索引对象上



【后续编程】：

第一步：建立索引对象

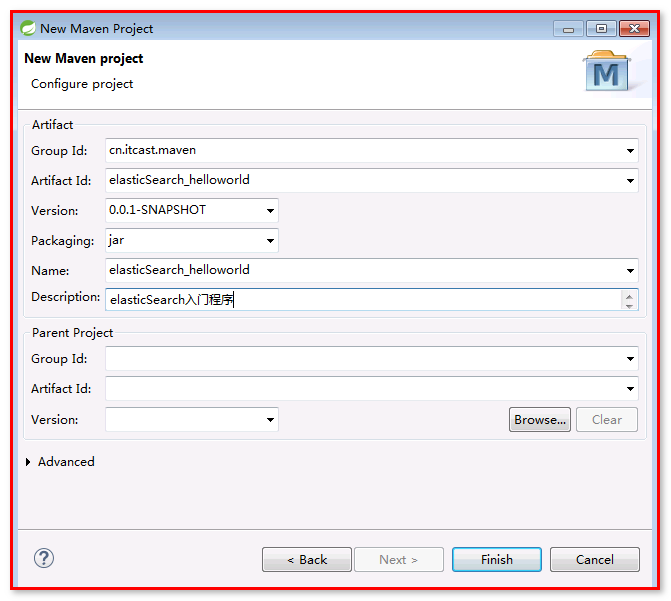
第二步：建立映射

第三步：存储数据【文档】

第四步：指定文档类型进行搜索数据【文档】

### 新建文档（自动创建索引和映射 ）

1. 需要新建 maven项目



1. 基于maven的pom 导入坐标依赖

<dependencies>

<dependency>

<groupId>org.elasticsearch</groupId>

<artifactId>elasticsearch</artifactId>

<version>2.4.0</version>

</dependency>

<dependency>

<groupId>junit</groupId>

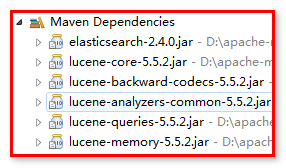
<artifactId>junit</artifactId>

<version>4.12</version>

</dependency>

</dependencies>

ElasticSearch2.4.0 依赖 lucene5.5.2 版本



当直接在ElasticSearch 建立文档对象时，如果索引不存在的，默认会自动创建，映射采用默认方式

* ElasticSearch 服务默认端口 9300
* Web 管理平台端口 9200

1. 建立文档， 自动创建索引

// ElasticSearch 测试程序

**public** **class** ElasticSearchTest {

@Test

// 直接在ElasticSearch中建立文档，自动创建索引，自动创建映射

**public** **void** demo1() **throws** IOException {

// 创建连接搜索服务器对象

Client client = TransportClient

.*builder*()

.build()

.addTransportAddress(

**new** InetSocketTransportAddress(InetAddress

.*getByName*("127.0.0.1"), 9300));//服务器对应9300

// 描述json 数据

/\*

\* {id:xxx, title:xxx, content:xxx}

\*/

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

\*/

IndexResponse indexResponse = client.prepareIndex("blog1", "article", "1").setSource(builder).get();

// 结果获取

String index = indexResponse.getIndex();

String type = indexResponse.getType();

String id = indexResponse.getId();

**long** version = indexResponse.getVersion();

**boolean** created = indexResponse.isCreated();

System.***out***.println(index + " : " + type + ": " + id + ": " + version + ": " + created);

// 关闭连接

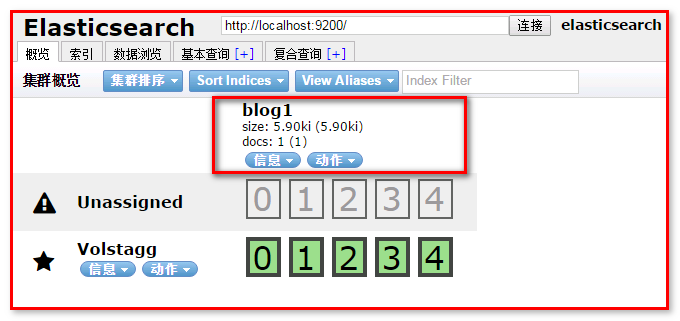
client.close();

}

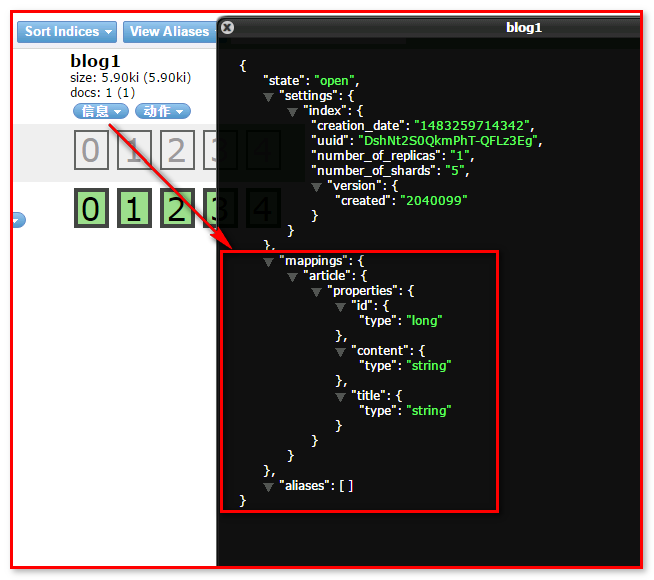
}

自动创建索引

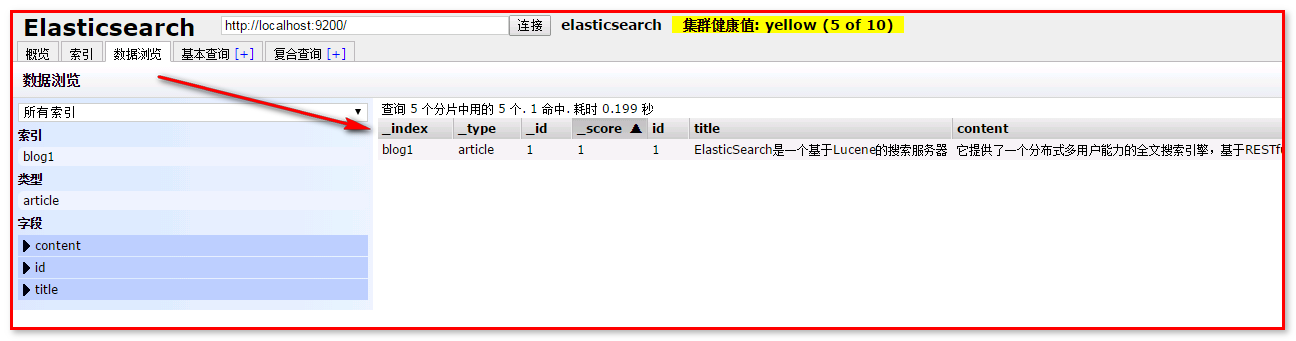
名称为blog1



自动创建索引映射



文档数据 （type 文档类型 ）



### 搜索文档数据

#### 【查询数据】

获取指定文档信息，通过id查询

/\*\*获取数据\*/

@Test

**public** **void** getIndexNoMapping() **throws** Exception {

// 创建Client连接对象

Client client = TransportClient.*builder*().build().addTransportAddress(**new** InetSocketTransportAddress(InetAddress.*getByName*("127.0.0.1"),9300));

GetResponse actionGet = client.prepareGet("blog", "article", "1").execute().actionGet();

System.***out***.println(actionGet.getSourceAsString());

client.close();

}

查询数据：复杂查询，主要依赖QueryBuilder对象 ，可以通过QueryBuilders获取各种查询 ：（基于lucene）

queryStringQuery：按照输入的文本进行分词查找，对所有内容进行分词

boolQuery() 布尔查询，可以用来组合多个查询条件

fuzzyQuery() 相似度查询

matchAllQuery() 查询所有数据

regexpQuery() 正则表达式查询

termQuery() 词条查询

wildcardQuery() 模糊查询

代码：

@Test

// 搜索在elasticSearch中创建文档对象

**public** **void** demo2() **throws** IOException {

// 创建连接搜索服务器对象

Client client = TransportClient

.*builder*()

.build()

.addTransportAddress(

**new** InetSocketTransportAddress(InetAddress

.*getByName*("127.0.0.1"), 9300));

// 搜索数据

// get() === execute().actionGet()

SearchResponse searchResponse = client.prepareSearch("blog1")

.setTypes("article")

// 按照输入的文本进行搜索（对所有字段进行分词查询 ）

// .setQuery(QueryBuilders.queryStringQuery("搜问"))

// 查询所有

//.setQuery(QueryBuilders.matchAllQuery())

// 词条查询

// .setQuery(QueryBuilders.termQuery("content", "搜索")) // content字段，搜索“搜索”的词

// 模糊查询

// .setQuery(QueryBuilders.wildcardQuery("content", "\*搜?")) // \*表示多个字符,?表示一个字符

// 相似度查询

.setQuery(QueryBuilders.*fuzzyQuery*("content", "Elasticsearcc")) //

.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("title:" + searchHit.getSource().get("title"));

}

// 关闭连接

client.close();

}

## 各种查询对象Query的使用

ElasticSearch 支持所有Lucene查询，并对其进行简化封装

matchAllQuery 查询所有

queryStringQuery 指定条件的搜索

TermQuery 词条查询

WildcardQuery 模糊查询

FuzzyQuery 相似度查询

BooleanQuery 布尔查询

1、ElasticSearch提供QueryBuileders.queryStringQuery(搜索内容) 查询方法，对所有字段进行分词查询

SearchResponse searchResponse = client.prepareSearch("blog1")

.setTypes("article")

.setQuery(QueryBuilders.*queryStringQuery*("全文")).get();

2、 只想查询content里包含全文 ，使用QueryBuilders.wildcardQuery模糊查询 \*任意字符串 ?任意单个字符

SearchResponse searchResponse = client.prepareSearch("blog1")

.setTypes("article")

.setQuery(QueryBuilders.*wildcardQuery*("content", "\*全文\*")).get();

发现查询不到！！！！ ，涉及词条查询理解， 说明没有词条包含“全文”

3、 查询content词条为“搜索” 内容，使用TermQuery

SearchResponse searchResponse = client.prepareSearch("blog1")

.setTypes("article")

.setQuery(QueryBuilders.*termQuery*("content", "全文")).get();

发现查询不到！！！，说明没有 “搜索”这个词条

词条： 就是将文本内容存入搜索服务器，搜索服务器进行分词之后的内容

例如1 ：“ElasticSearch是一个基于Lucene的搜索服务器”

分词（好的）： ElasticSearch、是、一个、基于、Lucene、搜索、服务、服务器

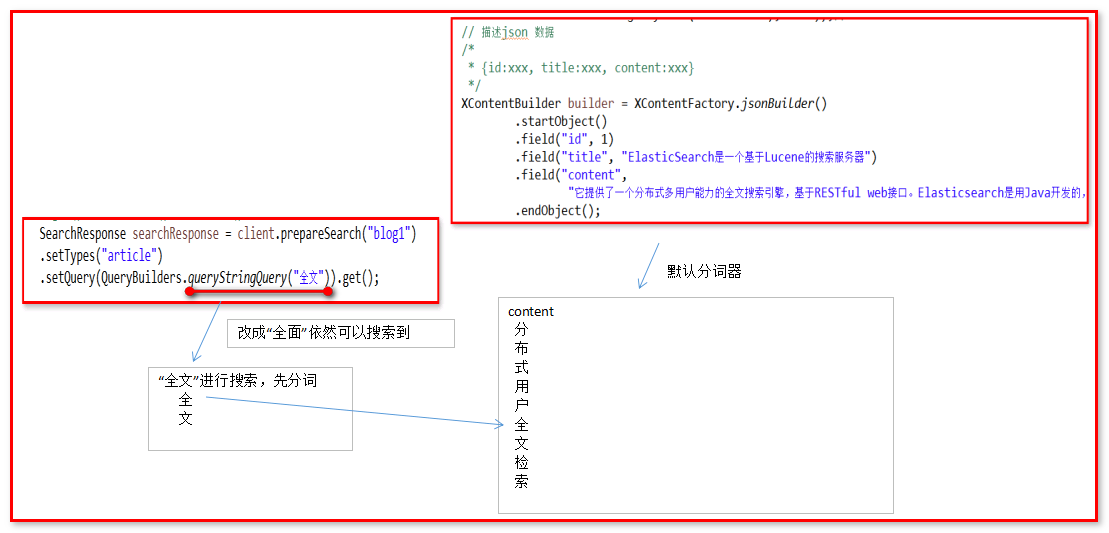
默认单字分词（差的）： ElasticSearch、 是、一、个、基、于、搜、索

例如2 ：消费者在四川星级酒店用面盆洗脚突然垮塌

分词(好的)：消费者 消费 四川 星级 酒店 星级酒店 面盆 突然 洗脚 垮塌

差的:消 费 者 四 川 星 级 酒 店 面 盆 洗 脚 突 然 垮 塌

搜索“全面” 也能够查询到，这是为什么呢？看图：



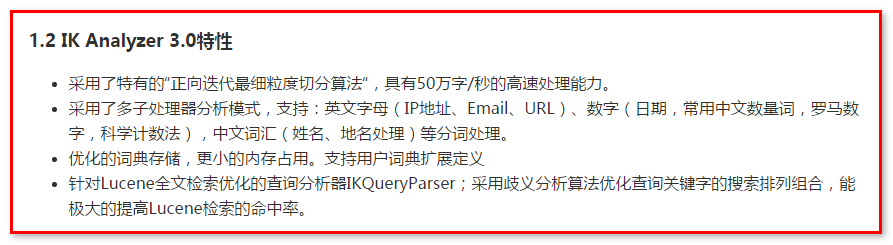
## IK分词器和ElasticSearch集成使用

ElasticSearch 默认采用分词器， 单个字分词 ，效果很差

搜索【IK Analyzer 3.0】

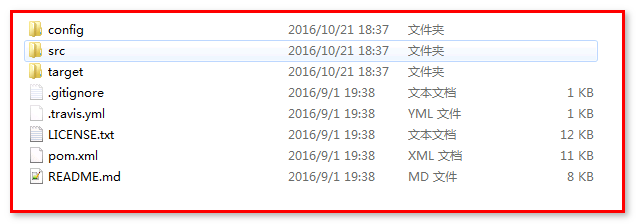
http://www.oschina.net/news/2660





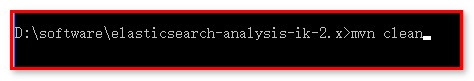
下载 <https://github.com/medcl/elasticsearch-analysis-ik/tree/2.x>

1. 下载开源项目

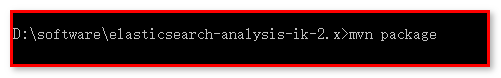


1. 打包ik分词器

mvn clean 清空

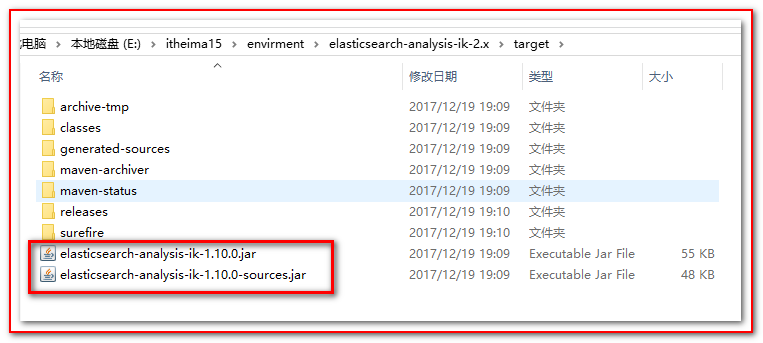


mvn package 打包



生成：

这里不用执行，需要联网下载，而且很慢，我们使用现成的即可。



1. 进入target/release目录

将下列文件 ，拷贝到 %es%/plugins/analysis-ik



这里注意：elasticsearch-analysis-ik-1.10.0的文件夹和elasticssearch-analysis-ik-1.10.0.zip，2个文件不要复制

1. 配置elasticsearch.yml

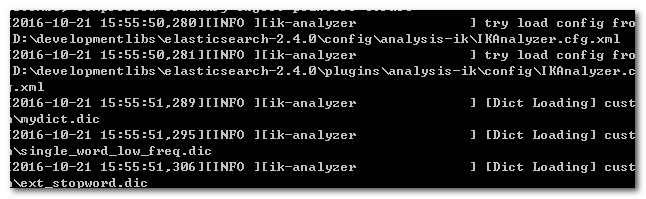
在%es%/config下

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

放置到最后

名称命名为ik。这里注意:后面需要有一个空格。

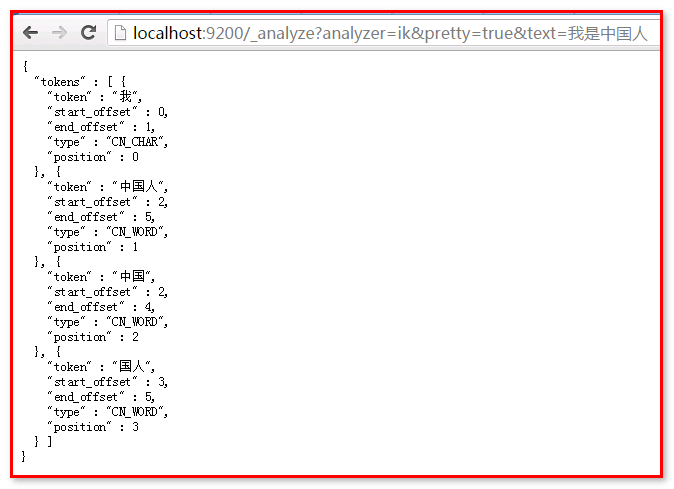
1. 重启es



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>



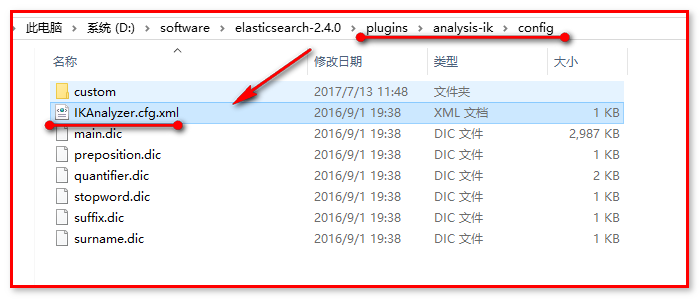
其中，“我”是一个词，“是”是停用词，“中国人”可以分为“中国人”、“中国”、“国人”。

如果我们输出“传智播客”呢？



怎么解决呢？

我们找到plugins/analysis-ik/config的文件夹，打开IKAnalyzer.cfg.xml文件



打开

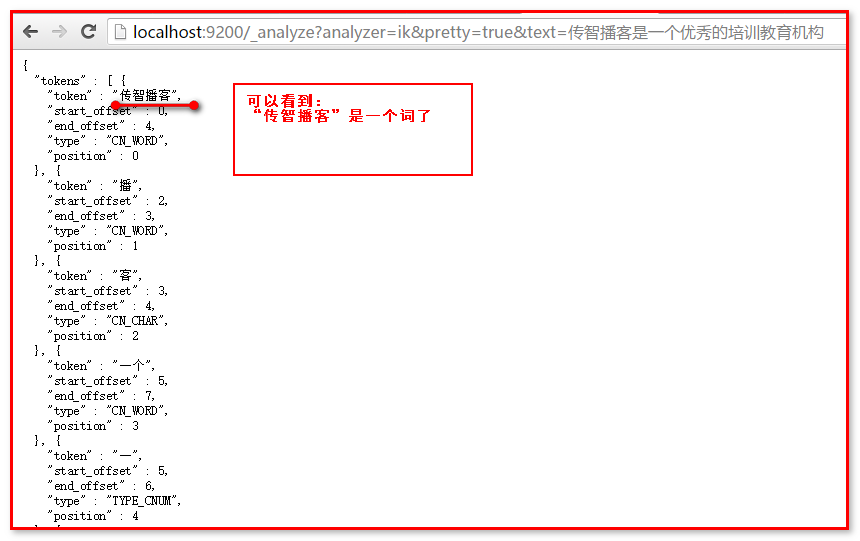


找到拓展词库的位置：custom/mydict.dic，打开文件

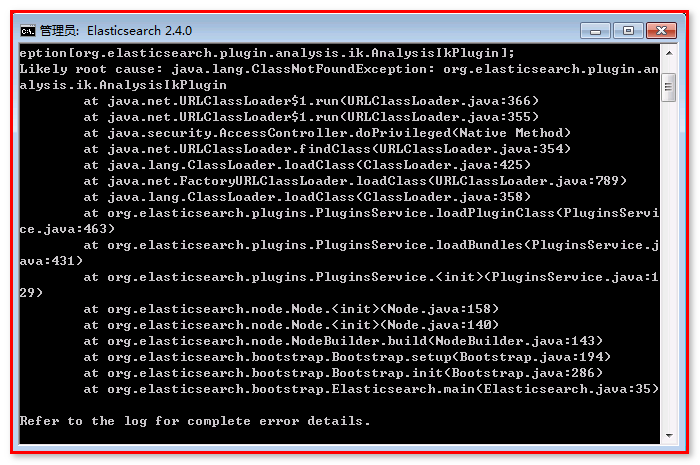


这里需要重启elasticsearch服务器。

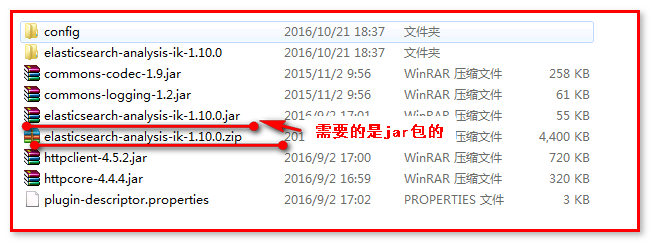
再次访问：



【出现异常】：



别导错包：



## ElasticSearch 常用编程操作 --手动创建索引、映射、使用IK分词器

在ElasticSearch没有索引情况下，插入文档，默认创建索引和索引映射 （但是无法使用ik分词器）

要想使用IK分词器，需要创建索引。

### 索引相关操作

#### 【创建索引】

/\*\*创建索引\*/

@Test

**public** **void** createIndex() **throws** Exception {

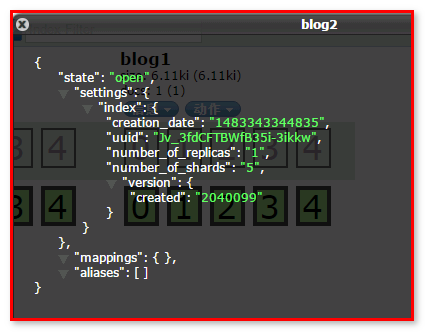
// 创建Client连接对象

TransportClient client = TransportClient.*builder*().build().addTransportAddress(**new** InetSocketTransportAddress(InetAddress.*getByName*("127.0.0.1"), 9300));

client.admin().indices().prepareCreate("blog2").get();

client.close();

}



默认创建好索引，mappings为空

#### 【删除索引】

/\*\*删除索引\*/

@Test

**public** **void** deleteIndex() **throws** Exception {

// 创建Client连接对象

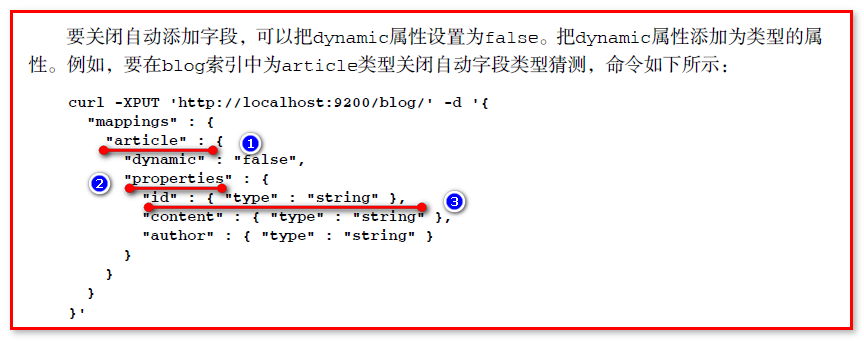
TransportClient client = TransportClient.*builder*().build().addTransportAddress(**new** InetSocketTransportAddress(InetAddress.*getByName*("127.0.0.1"), 9300));

client.admin().indices().prepareDelete("blog2").get();

client.close();

}

### 映射相关操作



#### 【创建映射】

PutMappingRequest mapping = Requests.*putMappingRequest*("blog2").type("article").source(builder);

client.admin().indices().putMapping(mapping).get();

【代码】

@Test

// 映射操作

**public** **void** demo5() **throws** IOException, InterruptedException,

ExecutionException {

// 创建连接搜索服务器对象

Client client = TransportClient

.*builder*()

.build()

.addTransportAddress(

**new** InetSocketTransportAddress(InetAddress

.*getByName*("127.0.0.1"), 9300));

//创建索引

client.admin().indices().prepareCreate("blog2").get();

// 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")

.endObject()

.startObject("content")

.field("type", "string").field("store", "yes").field("analyzer", "ik")

.endObject()

.endObject()

.endObject()

.endObject();

// 创建映射

PutMappingRequest mapping = Requests.*putMappingRequest*("blog2")

.type("article").source(builder);

client.admin().indices().putMapping(mapping).get();

// 关闭连接

client.close();

}

查看：http://localhost:9200/\_plugin/head/



### 文档相关操作 （CRUD）

#### 【建立文档】：XContentBuilder的形式

1. 回顾：直接在XContentBuilder中构建json数据，建立文档 。

Client client = TransportClient.*builder*().build().addTransportAddress(**new** InetSocketTransportAddress(InetAddress.*getByName*("127.0.0.1"),9300));

// 描述json 数据

/\*

\* {id:xxx, title:xxx, content:xxx}

\*/

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("blog1", "article", "1").setSource(builder).get();

client.close();

针对XContentFactory对象创建文档，有些复杂，是否可以使用对象的形式创建呢？

#### 【建立文档】：对一个已经存在对象，转换为json的形式

创建包，cn.itcast.elasticsearch.domain

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

**private** Integer id;

**private** String title;

**private** String content;

}

问题：如何将Article对象，转换为json数据 ---- Jackson 转换开发包

Jackson 是一个 Java 用来处理 JSON 格式数据的类库，性能非常好。

Jackson可以轻松的将Java对象转换成json对象和xml文档，同样也可以将json、xml转换成Java对象。Jackson库于2012.10.8号发布了最新的2.1版。  
 Jackson源码目前托管于GitHub，地址：https://github.com/FasterXML/

Jackson 2.x介绍

Jackson 2.x版提供了三个JAR包供下载：  
1. Core库：streaming parser/generator，即流式的解析器和生成器。  
下载：  
**http://repo1.maven.org/maven2/com/fasterxml/jackson/core/jackson-core/2.1.0/jackson-core-2.1.0.jar**  
2. Databind库：ObjectMapper, Json Tree Model，即对象映射器，JSON树模型。  
下载：  
**http://repo1.maven.org/maven2/com/fasterxml/jackson/core/jackson-databind/2.1.0/jackson-databind-2.1.0.jar**  
3. Annotations库：databinding annotations，即带注解的数据绑定包。  
下载：  
**http://repo1.maven.org/maven2/com/fasterxml/jackson/core/jackson-annotations/2.1.0/jackson-annotations-2.1.0.jar**  
  
从Jackson 2.0起，  
核心组件包括：jackson-annotations、jackson-core、jackson-databind。

引入jackson

1.x 版本

<dependency>  
          <groupId>org.codehaus.jackson</groupId>  
          <artifactId>jackson-core-asl</artifactId>  
          <version>1.9.13</version>  
      </dependency>  
      <dependency>  
          <groupId>org.codehaus.jackson</groupId>  
          <artifactId>jackson-mapper-asl</artifactId>  
          <version>1.9.13</version>  
      </dependency>

2.x 版本

<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>

* 【建立文档代码】：

@Test

// 文档相关操作

**public** **void** demo6() **throws** IOException, InterruptedException,

ExecutionException {

// 创建连接搜索服务器对象

Client client = TransportClient

.*builder*()

.build()

.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();

// 关闭连接

client.close();

}

测试完成搜索：

#### 【修改文档】

方式一： 使用prepareUpdate、prepareIndex都可以

client.prepareUpdate("blog2", "article", article.getId().toString())

.setDoc(objectMapper.writeValueAsString(article)).get();

方式二： 直接使用update

client.update(**new** UpdateRequest("blog2", "article", article.getId().toString())

.doc(objectMapper.writeValueAsString(article))).get();

#### 【删除文档】

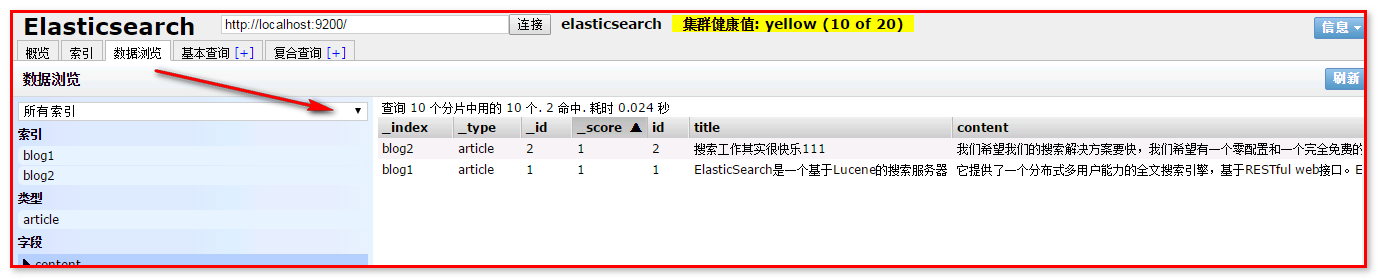
方式一： prepareDelete

client.prepareDelete("blog2", "article", article.getId().toString()).get();

方式二： 直接使用delete

client.delete(**new** DeleteRequest("blog2", "article", article.getId().toString())).get();

【http://localhost:9200/\_plugin/head：查询结果】：



### 查询文档分页操作

#### 【插入100条记录】

@Test

// 批量查询100条记录

**public** **void** demo7() **throws** IOException, InterruptedException,

ExecutionException {

// 创建连接搜索服务器对象

Client client = TransportClient

.*builder*()

.build()

.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();

}

// 关闭连接

client.close();

}

#### 【组合查询】（复杂查询）

must(QueryBuilders) : AND

mustNot(QueryBuilders): NOT

should(QueryBuilders):OR

在定义json：放置到Elasticsearch的插件中（DSL表达式）

{

"query" : {

"bool" : {

"must" : {

"term" : {

"title" : "elasticsearch"

}

},

"should" : {

"range" : {

"id" : {

"from" : 1,

"to" : 2

}

}

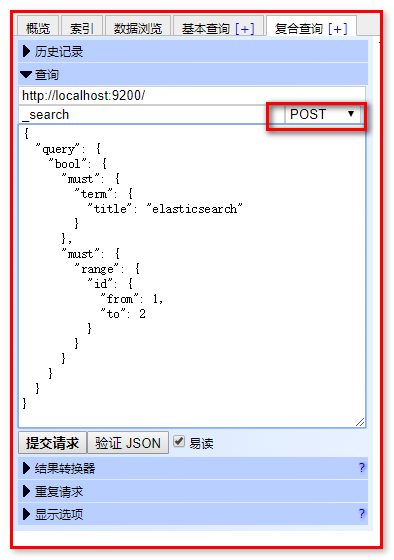
}

}

}

}

如果使用head插件查看索引库的信息，进行查询：



// 组合查询

@Test

**public** **void** combinationQuery() **throws** Exception{

TransportClient client = TransportClient.*builder*().build().addTransportAddress(**new** InetSocketTransportAddress(InetAddress.*getByName*("127.0.0.1"),9300));

SearchResponse searchResponse = client.prepareSearch("blog2").setTypes("article")

.setQuery(QueryBuilders.*boolQuery*().must(QueryBuilders.*termQuery*("title", "搜索"))// 词条查询

//.must(QueryBuilders.rangeQuery("id").from(1).to(5)) // 范围查询

.must(QueryBuilders.*wildcardQuery*("content", "elastics\*ch")) // 模糊查询

)// 大家回去改成should试试看

.get();

SearchHits hits = searchResponse.getHits();

System.***out***.println("总记录数："+hits.getTotalHits());

Iterator<SearchHit> iterator = hits.iterator();

**while**(iterator.hasNext()){

SearchHit searchHit = iterator.next();

System.***out***.println(searchHit.getSourceAsString());

Map<String, Object> source = searchHit.getSource();

System.***out***.println(source.get("id")+" "+source.get("title")+" "+source.get("content"));

}

client.close();

}

这里发现：

.must(QueryBuilders.*wildcardQuery*("content", "elastics\*ch")) // 模糊查询

能搜索到到结果。

而使用

.must(QueryBuilders.*wildcardQuery*("content", "Elastics\*ch")) // 模糊查询

不能搜索到结果。为什么呢？

输入：

<http://localhost:9200/_analyze?analyzer=ik&pretty=true&text=Elasticsearch是一个全文检索的框架>



因为IK分词器，在存储的时候将英文都变成了小写，这样方便我们在搜索的时候可以实现“不区分大小写”的搜索，只要我们在搜索的条件中添加.toLowerCase()的方法即可。

例如：

.must(QueryBuilders.*wildcardQuery*("content", "ELASTics\*ch".toLowerCase()))

将输入的值都先变成小写，再来搜索结果

#### 【分页查询】和【排序】

如果查询获取的数据在10条以上，默认按照查询结果每页以10条显示。

查询所有的方法

searchRequestBuilder 的 setFrom【从0开始】 和 setSize【查询多少条】方法实现

@Test

// 分页搜索

**public** **void** demo8() **throws** IOException {

// 创建连接搜索服务器对象

Client client = TransportClient

.*builder*()

.build()

.addTransportAddress(

**new** InetSocketTransportAddress(InetAddress

.*getByName*("127.0.0.1"), 9300));

// 搜索数据

// get() === execute().actionGet()

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(20).setSize(20);

// 排序

searchRequestBuilder.addSort("id",SortOrder.***DESC***);

SearchResponse searchResponse = searchRequestBuilder.get();

printSearchResponse(searchResponse);

// 关闭连接

client.close();

}

**private** **void** printSearchResponse(SearchResponse searchResponse) {

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"));

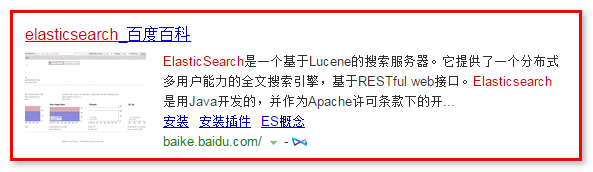
}

}

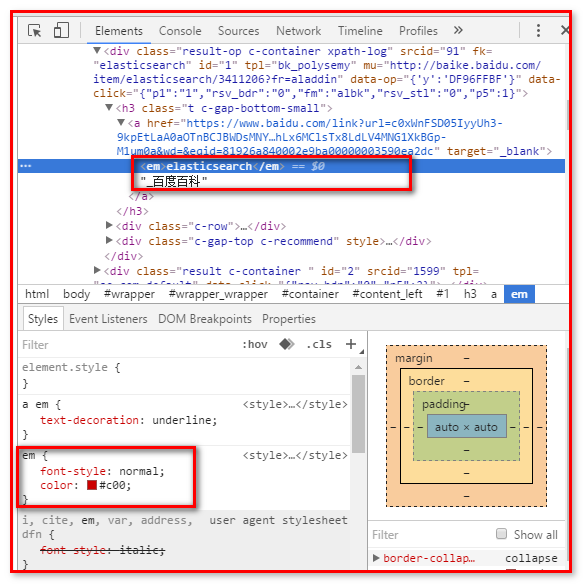
### 查询结果高亮显示

#### 【文字高亮】

在百度搜索elasticsearch，可以是文字高亮。



查看页面源码分析



【代码】

@Test

// 高亮查询结果 处理 搜索

**public** **void** demo9() **throws** IOException {

// 创建连接搜索服务器对象

Client client = TransportClient

.*builder*()

.build()

.addTransportAddress(

**new** InetSocketTransportAddress(InetAddress

.*getByName*("127.0.0.1"), 9300));

ObjectMapper objectMapper = **new** ObjectMapper();

// 搜索数据

SearchRequestBuilder searchRequestBuilder = client

.prepareSearch("blog2").setTypes("article")

//.setQuery(QueryBuilders.termQuery("content", "搜索"));

.setQuery(QueryBuilders.*multiMatchQuery*("搜索", "title","content"));

//.setQuery(QueryBuilders.queryStringQuery("搜索").field("title").field("content"))

1、 配置应用高亮

// 高亮定义

searchRequestBuilder.addHighlightedField("content").addHighlightedField("title"); // 设置高亮的字段; // 对content和title字段进行高亮

searchRequestBuilder.setHighlighterPreTags("<em>"); // 前置元素

searchRequestBuilder.setHighlighterPostTags("</em>");// 后置元素

// 设置摘要大小（高亮出现最多的区域）

searchRequestBuilder.setHighlighterFragmentSize(100);

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

2、 对结果的高亮片段做拼接处理，替换原有内容

Article article = objectMapper.readValue(searchHit.getSourceAsString(), Article.**class**);

// 获取高亮的结果，并把高亮的结果封装到Article中

Map<String, HighlightField> highlightFields = searchHit.getHighlightFields();

// 获取title中的文字高亮

HighlightField titleHighlightField = highlightFields.get("title");

// 如果存在高亮的结果，此时titleHighlightField对象不为null，组织高亮的结果，如果对象为null，说明没有高亮的效果

**if**(titleHighlightField!=**null**){

Text[] titleFragments = titleHighlightField.fragments();

String title = "";

**if**(titleFragments!=**null** && titleFragments.length>0){

**for**(Text text:titleFragments){

title += text;

}

article.setTitle(title);

}

}

// 获取content中的文字高亮

// 如果存在高亮的结果，此时contentHighlightField对象不为null，组织高亮的结果，如果对象为null，说明没有高亮的效果

HighlightField contentHighlightField = highlightFields.get("content");

**if**(contentHighlightField!=**null**){

Text[] contentFragments = contentHighlightField.fragments();

String content = "";

**if**(contentFragments!=**null** && contentFragments.length>0){

**for**(Text text:contentFragments){

content += text;

}

article.setContent(content);

}

}

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

}

// 关闭连接

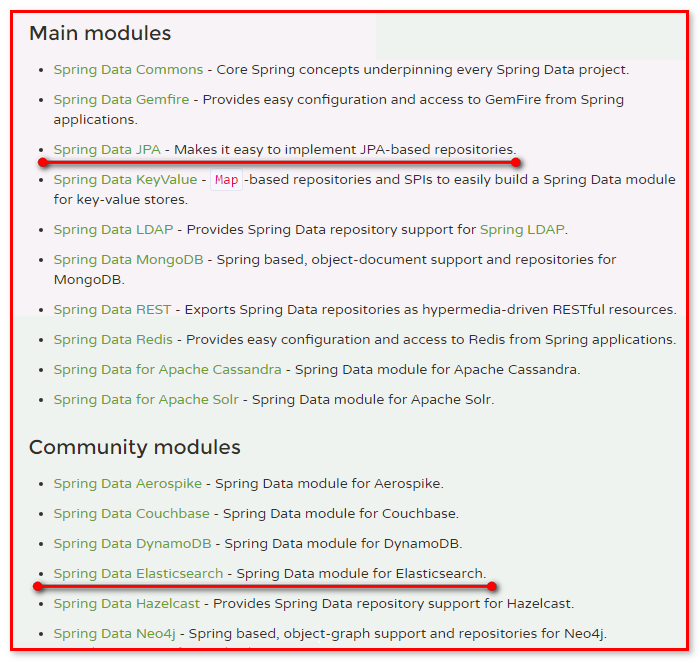
client.close();

}

## Spring Data ElasticSearch 使用 （重点）

### Spring Data ElasticSearch简介

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

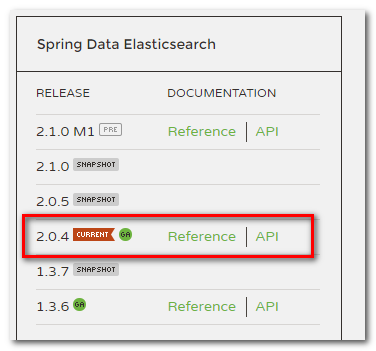


什么是spring data elasticSearch ?

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 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/>

下载官方规范文档

Maven坐标

<dependencies>

<dependency>

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

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

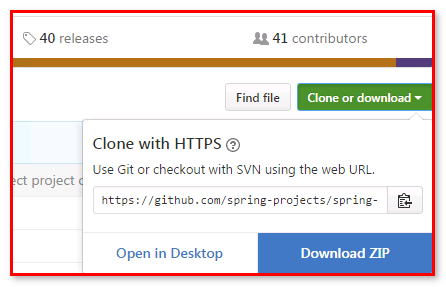
<version>2.4.0.RELEASE</version>

</dependency>

</dependencies>

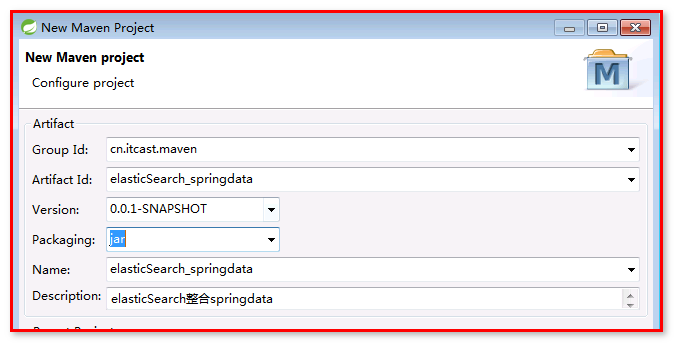
下载网址：

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



### Spring Data ElasticSearch 入门案例

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>

<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>

<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

<?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*

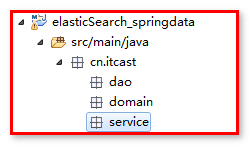
*http://www.springframework.org/schema/data/elasticsearch*

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

*"*>

</beans>

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

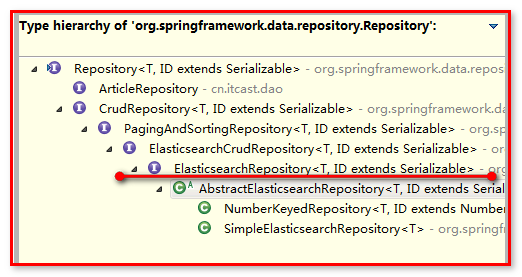
**public** String toString() {

**return** "Article [id=" + id + ", title=" + title + ", content=" + content + "]";

}

}

1. 编写DAO



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

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

}

1. 编写Service

ArticleService接口：

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

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

}

ArticleServiceImpl实现类

@Service

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

@Autowired

**private** ArticleRepository articleRepository;

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

articleRepository.save(article);

}

}

1. 配置applicationContext-es.xml

<!-- 扫描Dao包，自动创建实例 -->

<elasticsearch:repositories base-package=*"cn.itcast.dao"*/>

<!-- 扫描Service包，创建Service的实体 -->

<context:component-scan base-package=*"cn.itcast.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>

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 ）

@Id 文档主键 唯一标识 （Elasticsearch来说，定义String类型）

@Field 每个文档的字段配置（类型、是否分词、是否存储、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;

}

其中：

1：@Document(indexName="blob3",type="article")：

indexName：索引的名称（必填项）

type：索引的类型

2：@Id：主键的唯一标识

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

index：是否设置分词器

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：字段类型

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;

@Autowired

**private** Client client;

@Autowired

**private** ElasticsearchTemplate elasticsearchTemplate;

/\*\*创建索引和映射\*/

@Test

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

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

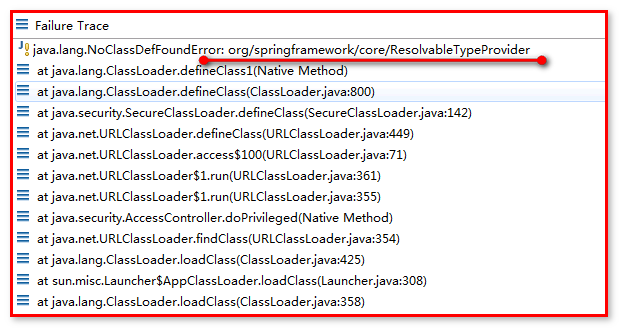
elasticsearchTemplate.putMapping(Article.**class**);

}

}

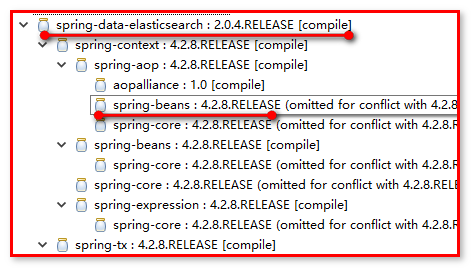
常见错误（一）： 导包冲突

java.lang.NoClassDefFoundError: org/springframework/core/ResolvableTypeProvider



解决方案： spring 导包版本要一致

Spring-data-elasticSearch默认导入的版本是4.2.8，所以测试包需要的版本要大于等于4.2.8



修改：

<dependency>

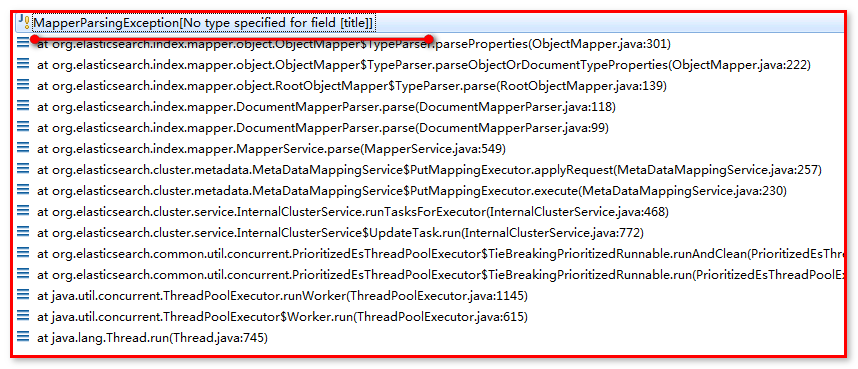
<groupId>org.springframework</groupId>

<artifactId>spring-test</artifactId>

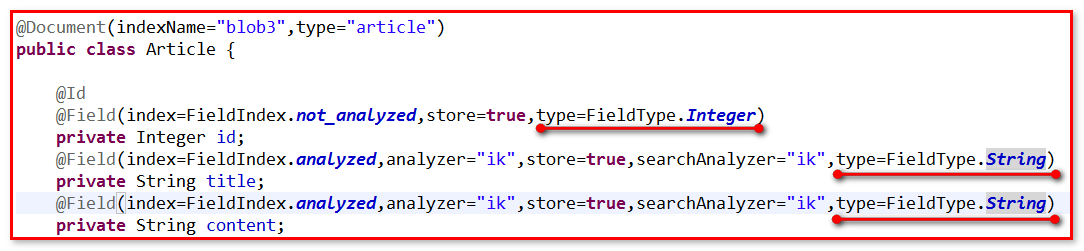
<version>4.2.8.RELEASE</version>

</dependency>

常见错误（二）：



解决： 配置字段类型



1. Spring data ElasticSearch CRUD操作

CurdRepository 提供增删改查 save、delete、findAll 、findOne

PagingAndSortingRepository 提供分页和排序

1. ：ArticleServiceImpl.java类

@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** Article findOne(Integer id) {

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

}

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

// 查询所有

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

// 排序

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

**return** iter;

}

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

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

}

}

1. 测试，ArticleServiceTest.java

@RunWith(SpringJUnit4ClassRunner.**class**)

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

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

@Autowired

**private** ArticleService articleService;

@Autowired

**private** Client 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\*/

@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);

}

/\*\*使用ID查询\*/

@Test

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

Integer id = 1001;

Article article = articleService.findOne(id);

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

}

}

/\*\*排序查询\*/

@Test

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

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

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

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

}

}

/\*\*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()){

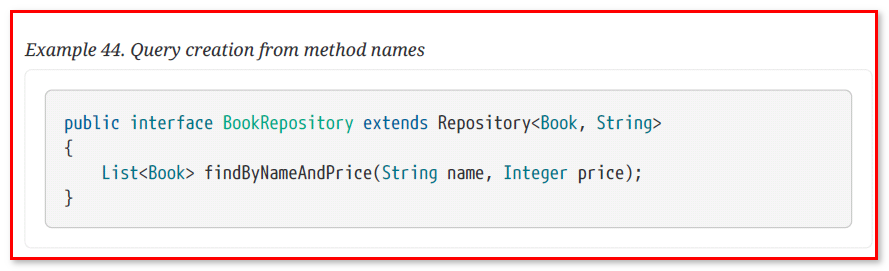
System.***out***.println(article);

}

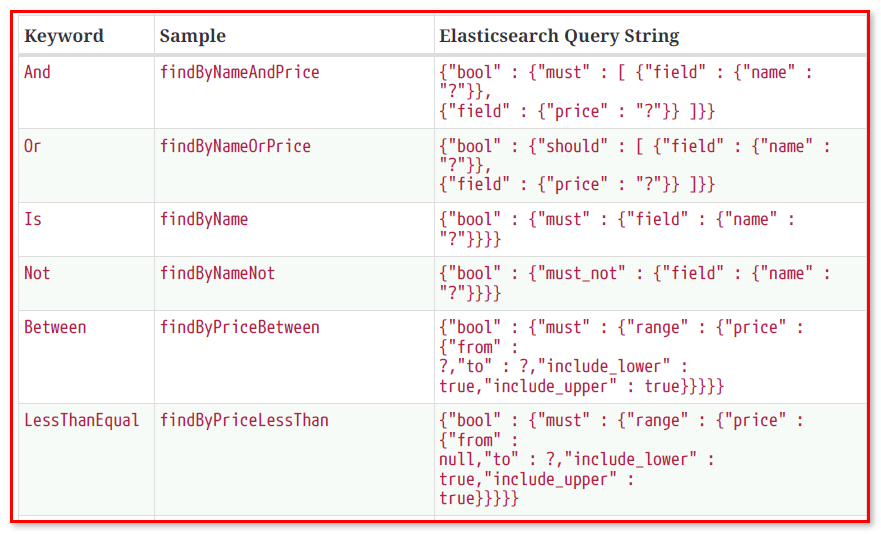
}

}

### 条件查询 （分页）



*Table 2. Supported keywords inside method names*



（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 condition);

}

Service层：

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

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

}

测试：ArticleServiceImpl.java

/\*\*条件查询\*/

@Test

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

String condition = "版本";

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

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

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

}

}

（2）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

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

List<Article> findByTitle(String condition);

Page<Article> findByTitle(String condition, Pageable pageable);

;

}

Service层：ArticleServiceImpl.java

**public** Page<Article> findByTitle(String condition, Pageable pageable) {

**return** articleRepository.findByTitle(condition,pageable);

}

测试类：ArticleServiceTest.java

/\*\*条件分页查询\*/

@Test

**public** **void** findByTitlePage(){

String condition = "版本";

Pageable pageable = **new** PageRequest(0, 10,Direction.***DESC***,"id");

Page<Article> page = articleService.findByTitle(condition,pageable);

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

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

}

}