# ElasticSearch

# 第一天

**学习目标**

* 什么是？
* 安装？
* 相关概念？
* 入门程序？
* postman发送Restful操作ES
* 理解分词的作用，并集成IK？
* 完成ES集群？

# 1是什么？

## 1.1 是什么？

laticsearch,简称为es，es是一个开源的高扩 展的分布式全文检索引擎,它可以近乎实时的存储、检索数据;本身扩展性很好。可以扩展到上百台服务器,处理PB级别的数据。es也使用Java开发并使用Lucene作为其核心来实现所有索引和搜索的功能,但是它的目的是通过简单的RESTful API来隐藏Lucene的复杂性,从而让全文搜索变得

## 1.2使用案例

●2013年初 , GitHub抛弃了Solr ,采取ElasticSearch来做PB级的搜索。"GitHub使用ElasticSearch搜索20TB的数据,包括13亿文件和1 300亿行代码”

●维基百科:启动以elasticsearch为基础的核心搜索架构

●SoundCloud : "SoundCloud使用ElasticSearch为1 .8亿用户提供即时而精准的音乐搜索服务”

●百度:百度目前广泛使用ElasticSearch作为文本数据分析,采集百度所有服务器上的各类指标数据及用户自定义数据，通过对各种数据进行多维分析展示,辅助定位分析实例异常或业务层面异常。目前覆盖百度内部20多个业务线(包括casio.云分析、网盟、预测、文库、直达号.钱包、风控等) , 单集群最大100台机器，200个ES节点，每天导入30TB+数据

●新浪使用ES分析处理32亿条实时日志

●阿里使用ES构建挖财自己的日志采集和分析体系

## 1.3 ES对比Solr？

●Solr 利用Zookeeper进行分布式管理。而Elasticsearch自身带有分布式协调管理功能;

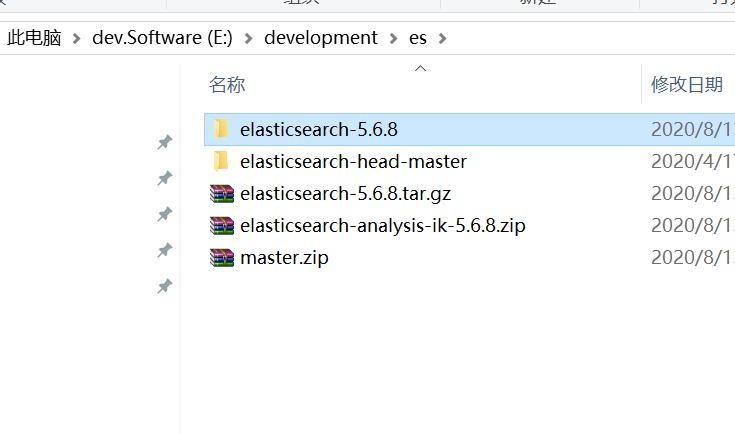
●Solr 支持更多格式的数据,而Elasticsearch仅支持json文件格式;

●Solr 官方提供的功能更多,而Elasticsearch本身更注重于核心功能,高级功能多有第三方插件提供;

●Solr 在传统的搜索应用中表现好于Elasticsearch ,但在处理**实时搜索**应用时效率明显低于Elasticsearch

# 2 安装与启动

## 2.1下载？

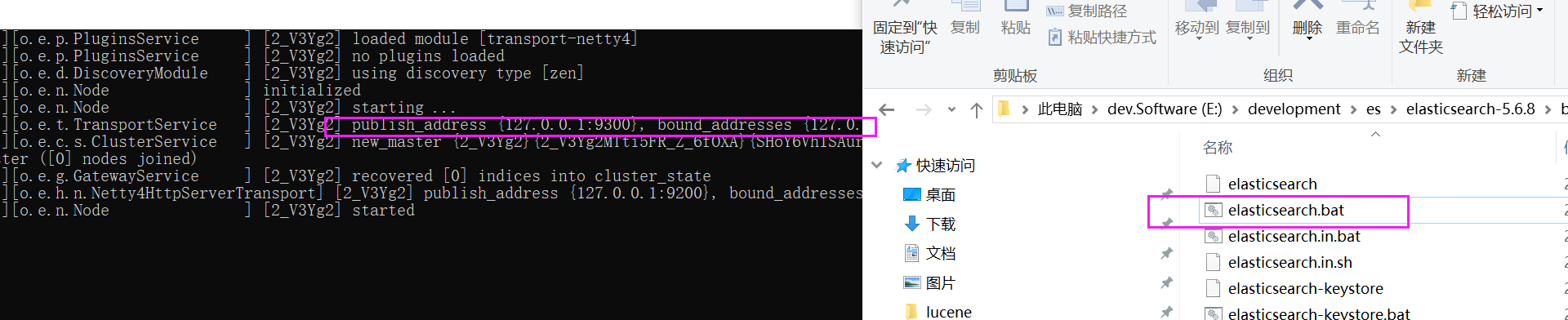


## 2.2安装

解压后就是安装了

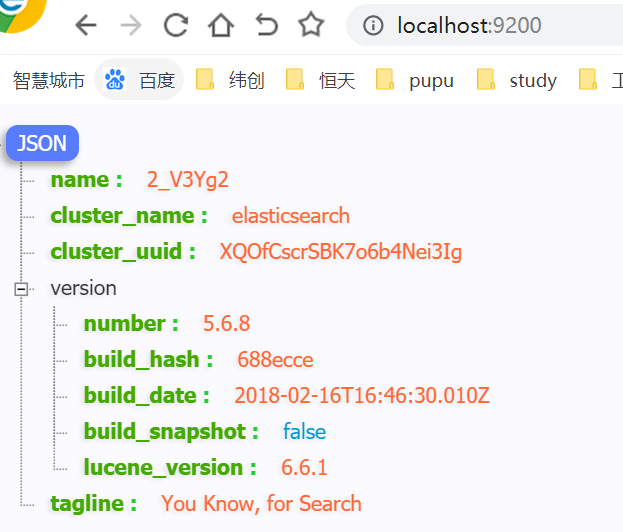
## 2.3 启动

需要jdk1.8以上



9300：Tcp

9200:Restful



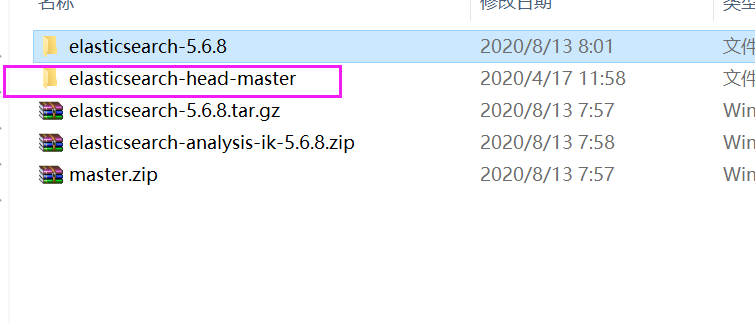
## 2.4 安装ES的图形化界面插件

ElasticSe arch不同于Solr自带图形化界面,我们可以通过安装ElasticSearch的head插件,完成图形化界面的效果，完成索引数据的查看。安装插件的方式有两种,在线安装和本地安装。本文档采用本地安装方式进行he ad插件的安装。elasticsearch-5-\*以 上版本安装head需要安装node和grunt

1 )下载head插件: bttps://github.com/mobz/elasticsearch-head

在资料中已经提供了elasticsearch-head-master插件压缩包:

是JS开发的

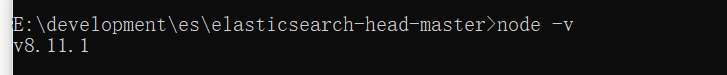


2 )将elasticsearch-head-master压缩包解压到任意目录,但是要和elasticsearch的安装目录区别开

3 )下载nodejs : https://nodeis. org/en/download.

在资料中已经提供了nodejs安装程序:

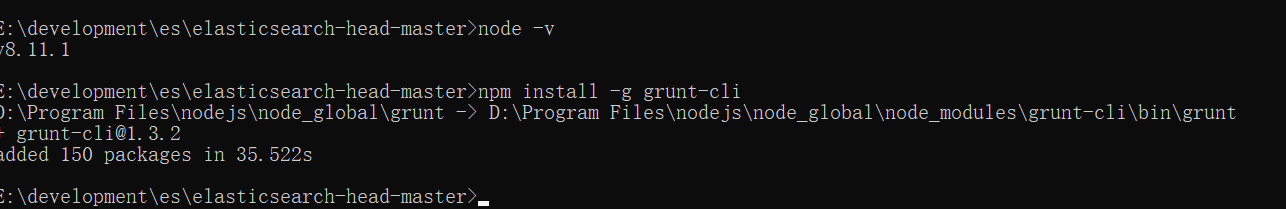
安装完毕,可以通过cmd控制台输入: node -V查看版本号



5 )将grunt安装为全局命令, Grunt是基于Node.js的项目构建工具

在cmd控制台中输入如下执行命令:

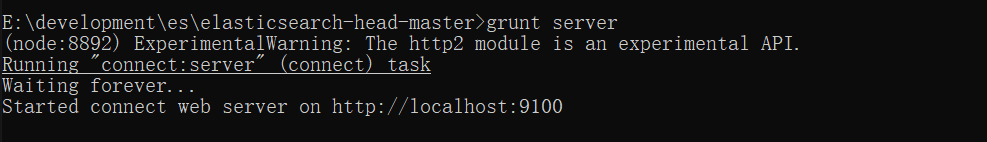
npm install -g grunt-cli



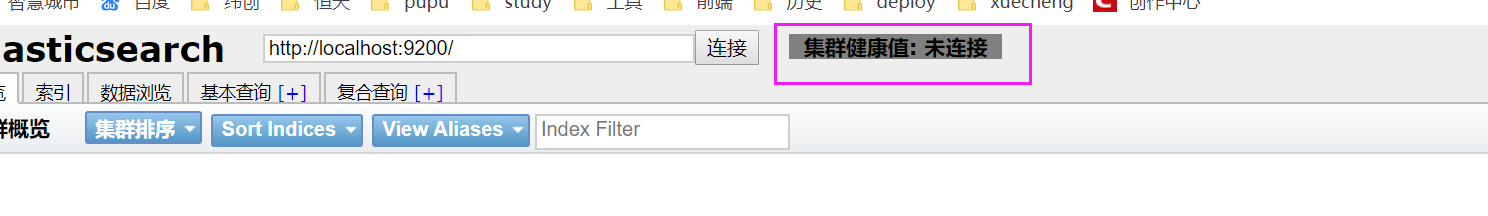
6 )进入elasticsearch-head-master目录启动head ,在命令提示符下输入命令:

>npm install

>grunt server



7 )打开浏览器,输入htp://localhost:9100 ,看到如下页面:

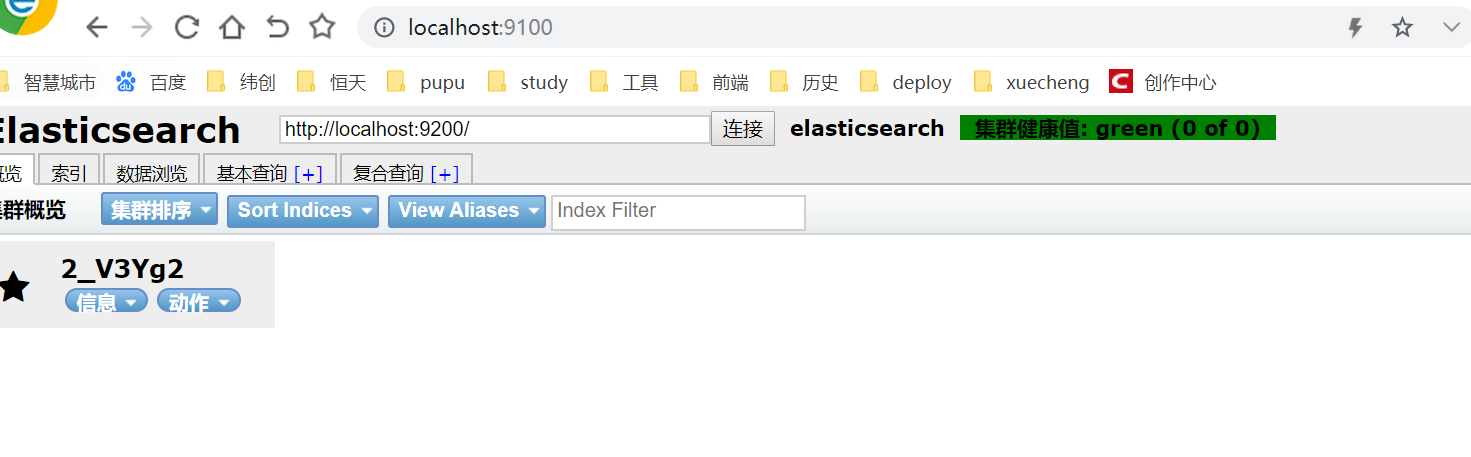


如果不能成功连接到es服务,需要修改ElasticSearch的config目录下的配置文件: config/elasticsearch.yml ,增加以下两句命令:

http.cors.enabled: true

http.cors.allow-origin: "\*"

然后重新启动ElasticSearch服务。



# 3 相关术语

## 3.1 概述

Elasticsearch是面向文档(document oriented)的,这意味着它可以存储整个对象或文档(document)。然而它不仅仅是存储,还会索引(index)每个文档的内容使之可以被搜索。

在Elasticsearch中 ,你可以对文档(而非成行成列的数据)进行索引、搜索.排序、过滤。Elasticsearch比传统关系型数据库如下:

Relational DB -> Databases -> Tables \*> Rows -> Columns

Elasticsearch -> Indices -> Types -> Documents -> Fields

## 3.2 核心概念

### 3.2.1 索引index

一个索引就是一个拥有几分相似特征的文档的集合。比如说,你可以有一个客户数据的索引，另一个产品目录的索引，还有一个订单数据的索引。一个索引由一个名字来标识(必须全部是小写字母的) , 并且当我们要对对应于这个索引中的文档进行索引、搜索、更新和删除的时候,都要使用到这个名字。在一个集群中 ,可以定义任意多的索引。

### 3.2.2 类型type

在一个索引中,你可以定义一种或多种类型。一个类型是你的索引的一个逻辑 上的分类/分区,其语义完全由你来定。通常,会为具有一组共同字段的文档定义一个类型。比如说,我们假设你运营一个博客 平台并且将你所有的数据存储到一个索引中。在这个索引中，你可以为用户数据定义一个类型,为博客数据定义另一个类型，当然，也可以为评论数据定义另一个类型。

### 3.2.3 字段Field

相当于是数据表的字段,对文档数据根据不同属性进行的分类标识

### 3.2.4 映射mapping

mapping是处理数据的方式和规则方面做一些限制 ,如某个字段的数据类型、默认值、分析器、是否被索引等等，这些都是映射里面可以设置的,其它就是处理es里面数据的一些使用规则设置也叫做映射,按着最优规则处理数据对性能提高很大，因此才需要建立映射,并且需要思考如何建立映射才能对性能更好。

### 3.2.5 文档document

一个文档是一个可被索引的基础信息单元。比如,你可以拥有某一个客户的文档 ,某一个产品的一个文档,当然，也可以拥有某个订单的一个文档。 文档以JSON ( Javascript Object Nottion )格式来表示,而JSON是一个到处存在的互联网数据交互格式。

在一个index/type里面 ,你可以存储任意多的文档。注意,尽管一个文档,物理上存在于一个索引之中,文档必须被索引/赋予一个索引的type.

### 3.2.6 接近实时NRT

Elasticsearch是一个接近实时的搜索平台。这意味着,从索引一个文档直到这个文档能够被搜索到有一个轻微的延迟(通常是1秒以内)

### 3.2.7 集群cluster

一个集群就是由一个或多个节点组织在一起,它们共同持有整个的数据,并- -起提供索引和搜索功能。一个集群由一个唯一的名字标识,这个名字默认就是"elasticsearch"。这个名字是重要的,因为-个节点只能通过指定某个集群的名字,来加入这个集群

### 3.2.8 节点node

一个节点是集群中的一个服务器,作为集群的一部分,它存储数据,参与集群的索引和搜索功能。和集群类似,一个节点也是由一个名字来标识的,默认情况下,这个名字是一个随机的漫威漫画角色的名字,这个名字会在启动的时候赋予节点。这个名字对于管理工作来说挺重要的，因为在这个管理过程中,你会去确定网络中的哪些服务器对应于Elasticsearch集群中的哪些节点。

一个节点可以通过配置集群名称的方式来加入-个指定的集群。默认情况下,每个节点都会被安排加入到一个叫做"elasticsearch'的集群中,这意味着,如果你在你的网络中启动了若干个节点,并假定它们能够相互发现彼此，它们将会自动地形成并加入到一个叫做"elasticsearch'的集群中。

在一个集群里，只要你想,可以拥有任意多个节点。而且,如果当前你的网络中没有运行任何Elasticsearch节点，这时启动一个节点,会默认创建并加入-个叫做"elasticsearch"的集群。

### 3.2.9 分片和复制shard & replicas

一个索引可以存储超出单个结点硬件限制的大量数据。比如，一个具有10亿文档的索引占据1TB的磁盘空间,而任一节点都没有这样大的磁盘空间 ;或者单个节点处理搜索请求,响应太慢。为了解决这个问题, Elasticsearch提供了将索引划分成多份的能力,这些份就叫做分片。当你创建一个索引的时候 ,你可以指定你想要的分片的数量。每个分片本身也是一个功能完善并且独立的“索引" ,这个“索引可以被放置到集群中的任何节点上。分片很重要,主要有两方面的原因: 1 )允许你水平分割/扩展你的内容容量。2 )允许你在分片(潜在地,位于多个节点上)之上进行分布式的、并行的操作,进而提高性能/吞吐量。

至于一个分片怎样分布,它的文档怎样聚合回搜索请求,是完全由Elasticsearch管理的,对于作为用户的你来说，这些都是透明的。

在一个网络/云的环境里,失败随时都可能发生,在某个分片/节点不知怎么的就处于离线状态,或者由于任何原因消失了,这种情况下,有一个故障转移机制是非常有用并且是强烈推荐的。为此目的, Elasticsearch允许你创建分片的一份或多份拷贝,这些拷贝叫做复制分片,或者直接叫复制。

复制之所以重要,有两个主要原因:在分片/节点失败的情况下,提供了高可用性。因为这个原因,注意到复制分片从不与原/主要( original/primary )分片置于同一~节点上是非常重要的。扩展你的搜索量/吞吐量,因为搜索可以在所有的复制上并行运行。总之,每个索引可以被分成多个分片。-个索引也可以被复制0次(意思是没有复制)或多次。一旦复制了,每个索引就有了主分片(作为复制源的原来的分片)和复制分片(主分片的拷贝)之别。分片和复制的数量可以在索引创建的时候指定。在索引创建之后,你可以在任何时候动态地改变复制的数量,但是你事后不能改变分片的数量。

默认情况下, Elasticsearch中的每个索引被分片5个主分片和1个复制,这意味着,如果你的集群中至少有两个节点，你的索引将会有5个主分片和另外5个复制分片( 1个完全拷贝) , 这样的话每个索引总共就有10个分片。

# 4 客户端

## 4.1 安装postman

实现一个文件的搜索功能，通过关键字搜索文件，凡是文件名或文件内容包括关键字的文件都需要找出来。还可以根据中文词语进行查询，并且需要支持多个条件查询。

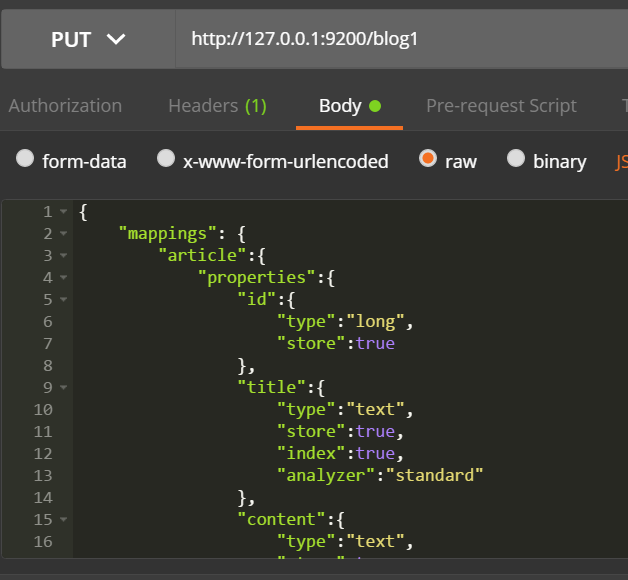
本案例中的原始内容就是磁盘上的文件，如下图：

## 4.2 下载postman

## 4.3 使用postman操作ES

### 4.2.1

### 4.2.2 创建index同时设置mapping



**{**

**"article":{**

**"properties":{**

**"id":{**

**"type":"long",**

**"store":true**

**},**

**"title":{**

**"type":"text",**

**"store":true,**

**"index":true,**

**"analyzer":"standar"**

**},**

**"content":{**

**"type":"text",**

**"store":true,**

**"index":true,**

**"analyzer":"standar"**

**}**

**}**

**}**

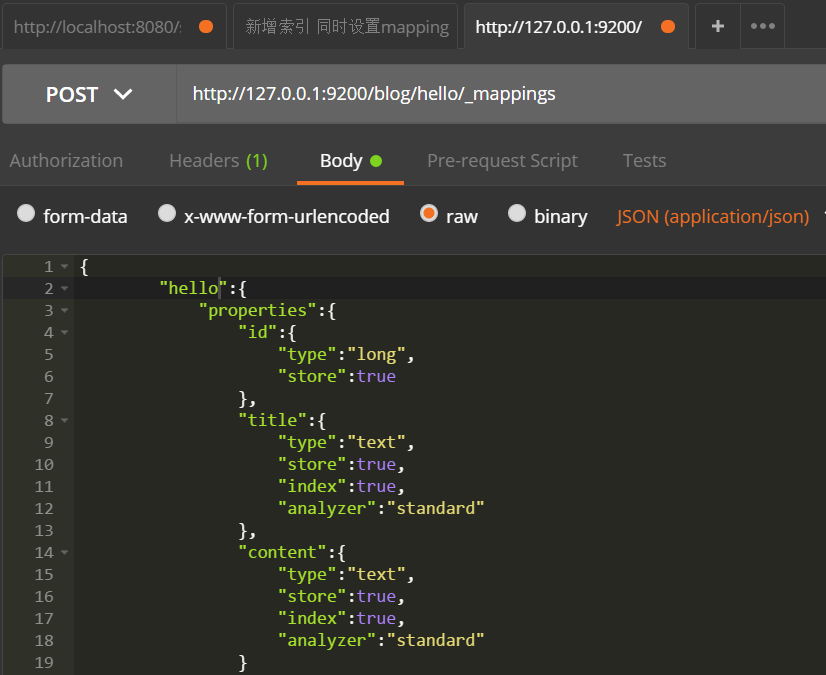
**}**

**article:**表示的是有多少个type

**properties:**表示的是有多少个document

再往里就是Field

### 4.2.3 创建索引后设置mapping



**{**

**"hello":{**

**"properties":{**

**"id":{**

**"type":"long",**

**"store":true**

**},**

**"title":{**

**"type":"text",**

**"store":true,**

**"index":true,**

**"analyzer":"standar"**

**},**

**"content":{**

**"type":"text",**

**"store":true,**

**"index":true,**

**"analyzer":"standar"**

**}**

**}**

**}**

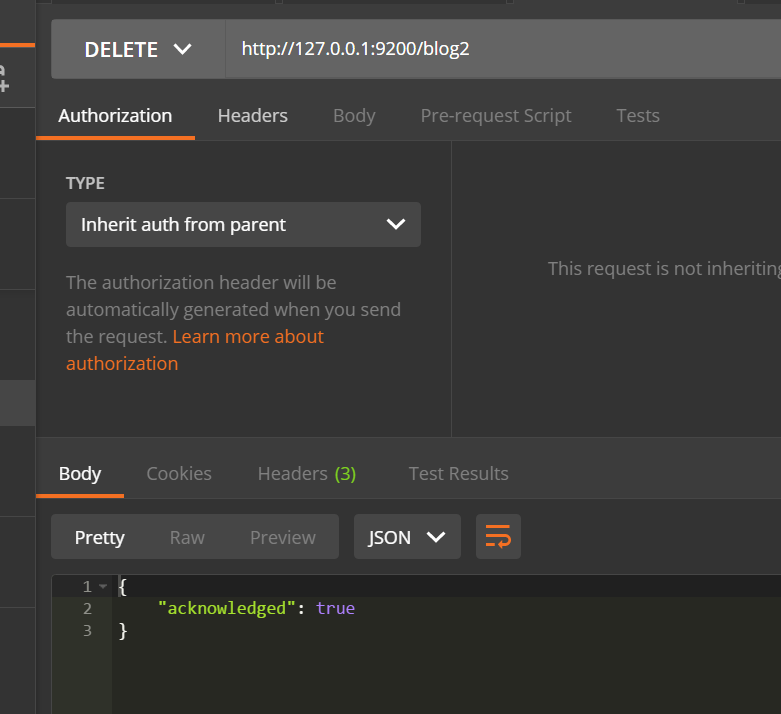
**}**

**article:**表示的是有多少个type

**properties:**表示的是有多少个document

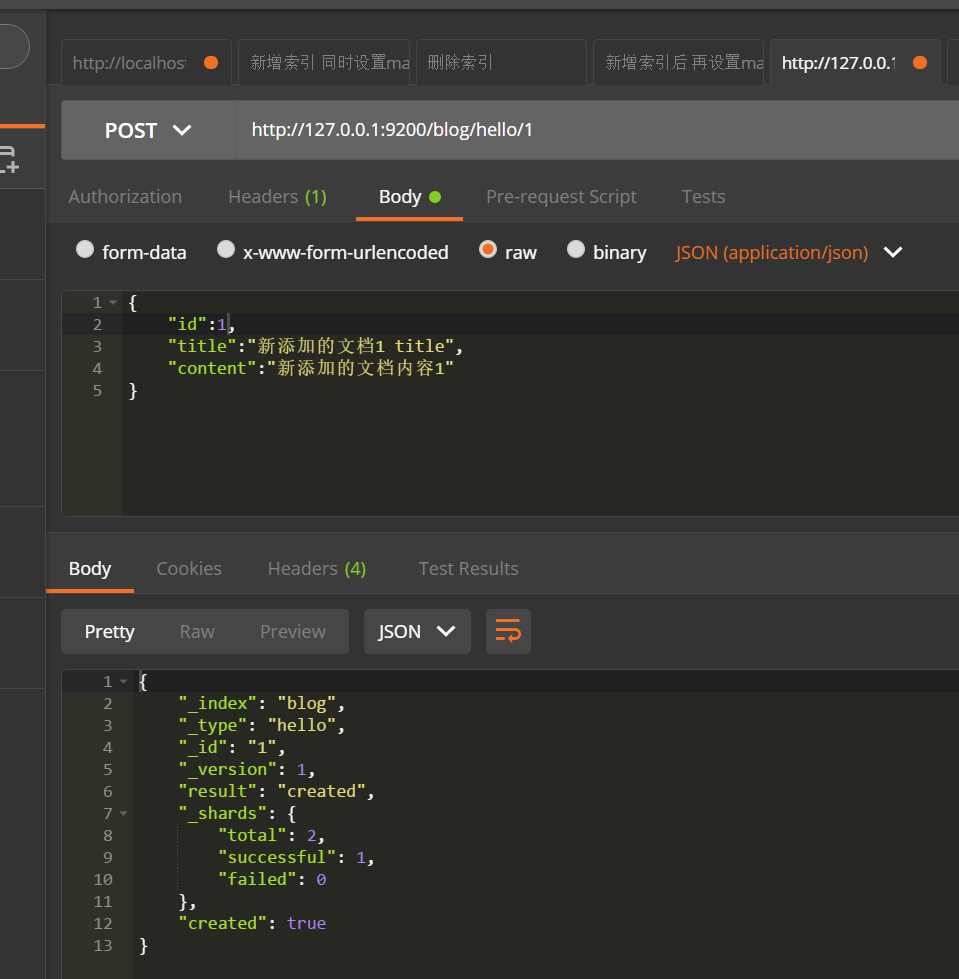
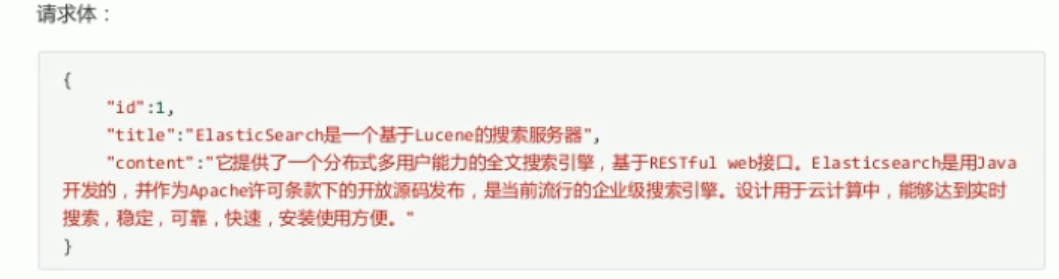
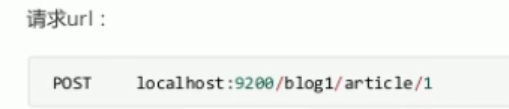
再往里就是Field

### 4.2.4 删除index



### 4.2.5 创建文档document到index

1是文档的id

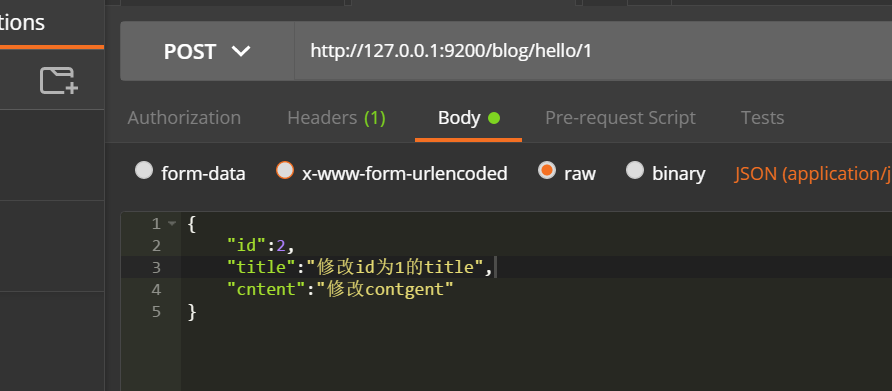


如果你不指定id，会自动生成随机的id

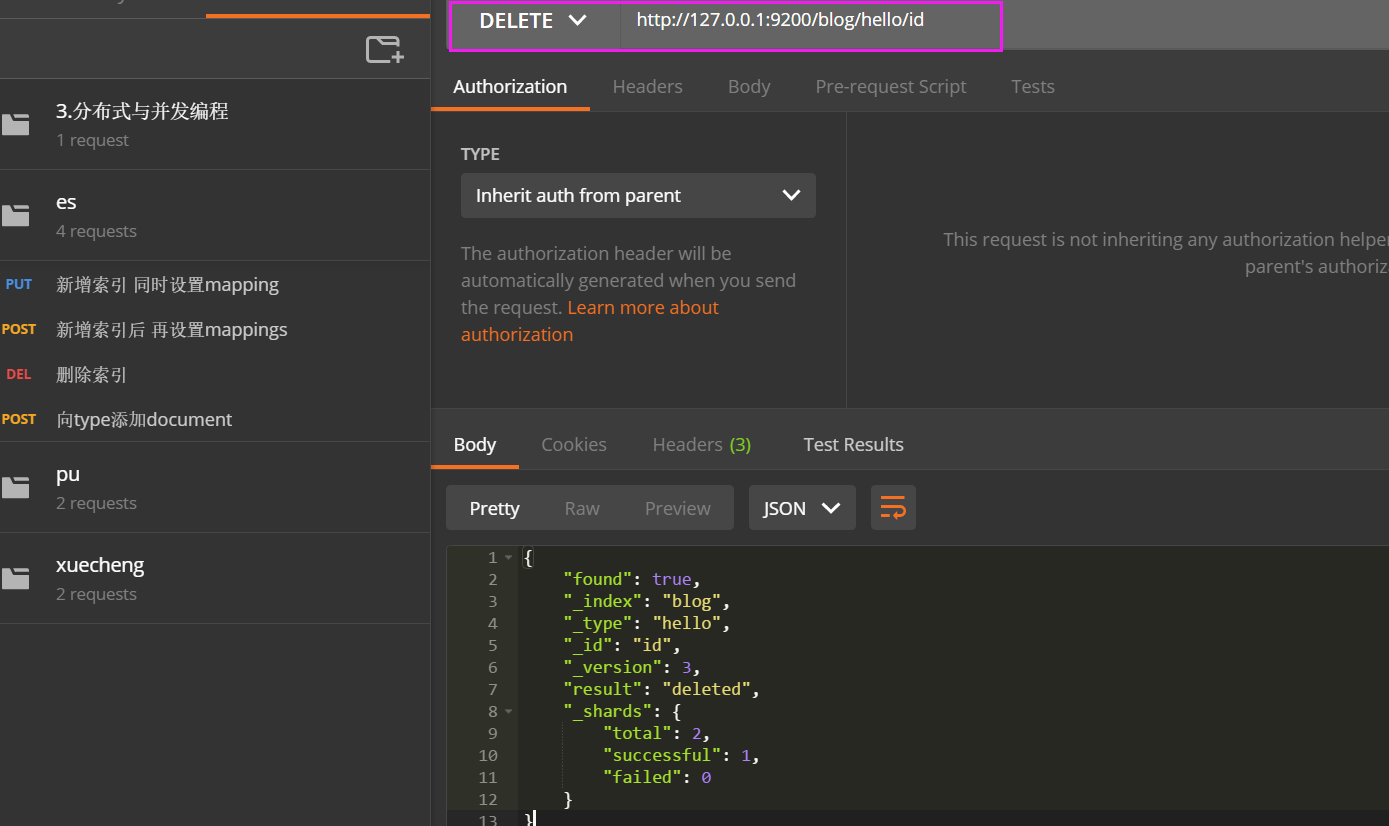
### 4.2.6 修改document

先删除后添加

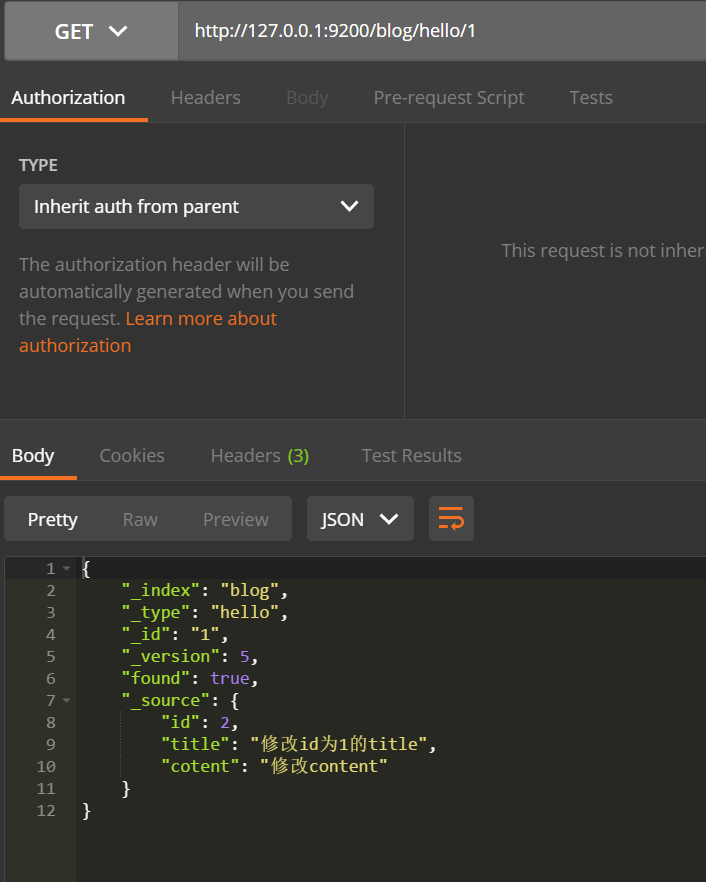
和添加是同一个操作



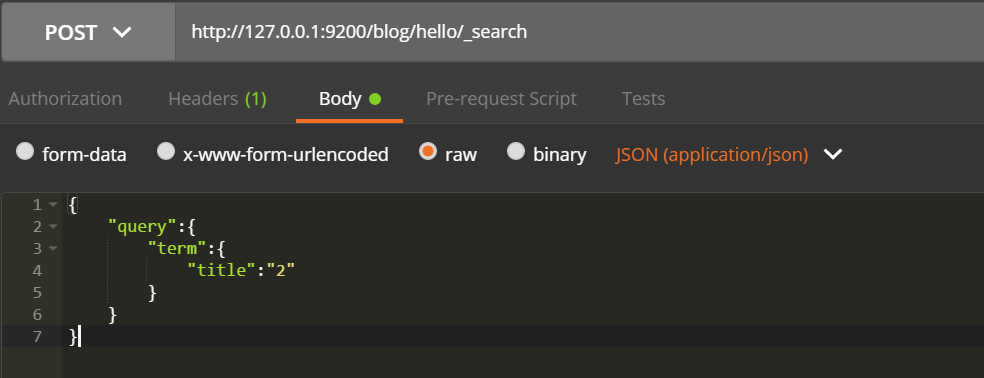
### 4.2.7 删除document



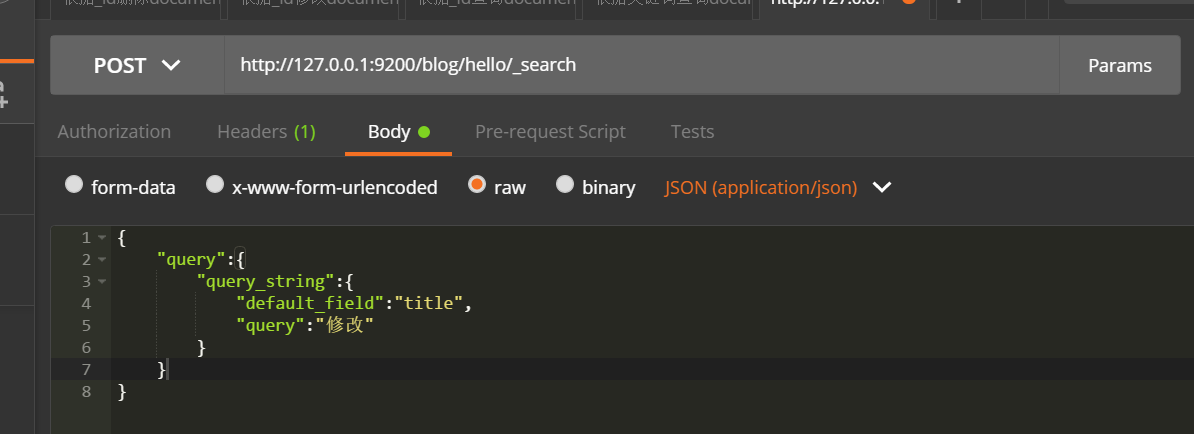
### 4.2.8 查询文档--根据id查询



### 4.2.9 查询文档-query 关键词



### 4.2.10 查询文档-querystring(就是Lucene中的queryparse)



### 4.2.11



must &&

should ||

must\_not !

# 5 IK分词器和ElasticSearch集成使用

## 5.1 上述查询存在的问题

在进行字符串查询时,我们发现去搜索”搜索服务器"和”钢索"都可以搜索到数据;

而在进行词条查询时,我们搜索搜索"却没有搜索到数据;

究其原因是ElasticSearch的标准分词器导致的,当我们创建索引时,字段使用的是标准分词器:

分析中文的话，会把每一个字分出来，没有组合的；只能一个字一个字的查

标准分词器分词效果测试:

http://127.0.0.1:9200/ analyze?analyzer =standard&pretty-true&text-我是程序员

## 5.2 IK分词器简介

## 5.3 ES集成IK

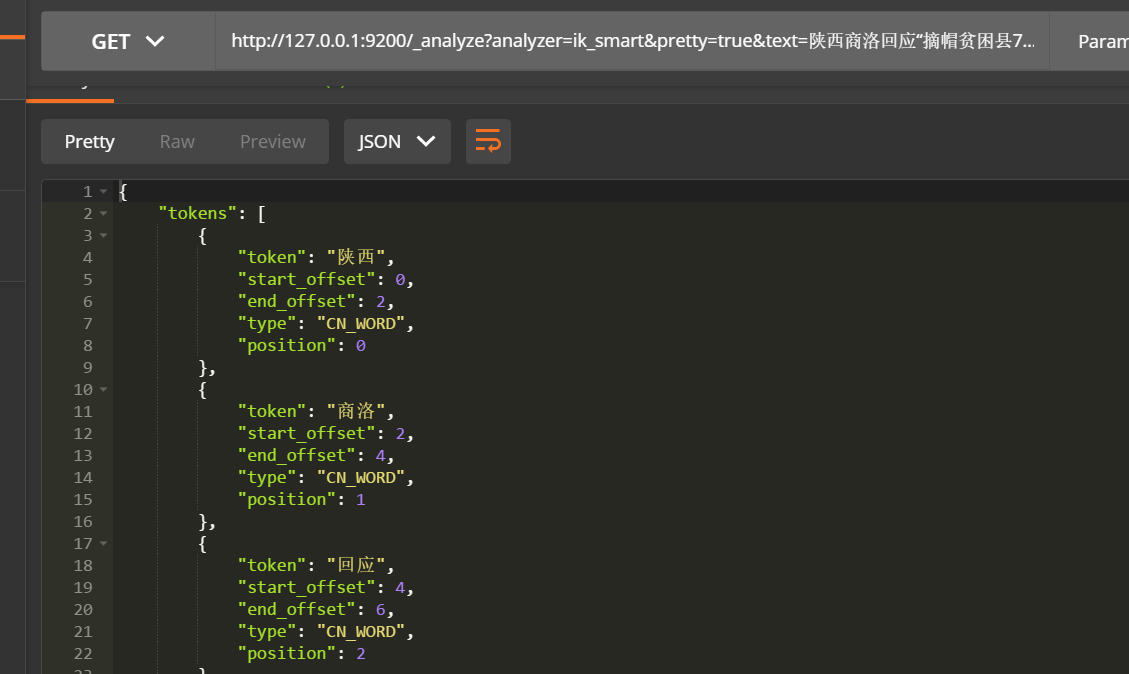
### 5.3.1 安装

解压放进去

### 5.3.2 测试

ik\_smart

ik\_max\_word



## 5.4 修改索引映射mapping

mappings建好后不能删

### 5.4.1 重建索引

自己做

### 5.4.2 再次测试querstring

自己做

### 5.4.3 再次测试term

自己做

# 6 ElasticSearch集群

## 6.1 相关术语

### 6.1.1 集群cluster

### 6.1.2 节点node

## 6.2 集群搭建

### 6.2.1准备3台es

赋值三个改下配置

### 6.2.2修改每台服务器的配置

#节点1的配置信息:

#集群名称，保证唯一

cluster.name: my-elasticsearch

#节点名称，必须不一样

node.name: node-1

#必须为本机的ip地址

network.host: 127.0.0.1

#http服务端口号，在同一机器下必须不一样

http.port: 9201

#集群间通信端口号，在同一机器下必须不一样

transport.tcp.port: 9301

#设置集群自动发现机器1p集合

discovery.zen.ping.unicast.hosts: ["127.0.0.1:9301","127.0.0.1:9302","127.0.0.1:9303"]

#节点2的配置信息:

#集群名称，保证唯一

cluster.name: my-elasticsearch

#节点名称，必须不一样

node.name: node-2

#必须为本机的ip地址

network.host: 127.0.0.1

#http服务端口号，在同一机器下必须不一样

http.port: 9202

#集群间通信端口号，在同一机器下必须不一样

transport.tcp.port: 9302

#设置集群自动发现机器1p集合

discovery.zen.ping.unicast.hosts: ["127.0.0.1:9301","127.0.0.1:9302","127.0.0.1:9303"]

#节点3的配置信息:

#集群名称，保证唯一

cluster.name: my-elasticsearch

#节点名称，必须不一样

node.name: node-3

#必须为本机的ip地址

network.host: 127.0.0.1

#http服务端口号，在同一机器下必须不一样

http.port: 9203

#集群间通信端口号，在同一机器下必须不一样

transport.tcp.port: 9303

#设置集群自动发现机器1p集合

discovery.zen.ping.unicast.hosts: ["127.0.0.1:9301","127.0.0.1:9302","127.0.0.1:9303"]

### 6.2.3启动每个节点服务器

自己做

### 6.2.4集群测试



## 6.3 索引库的修改

集群操作几乎和单机一样

# 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

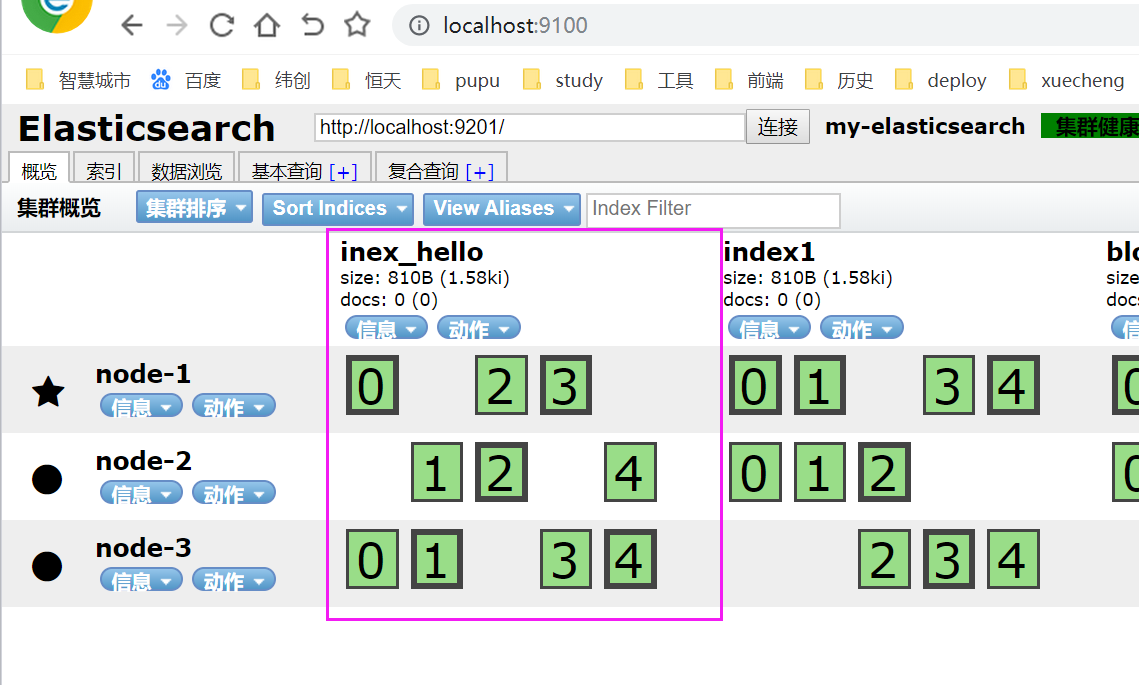
8.能够掌握基本条件查询的方法命名规则

# 1 es编程操作

## 1.1 创建工程，导入项目

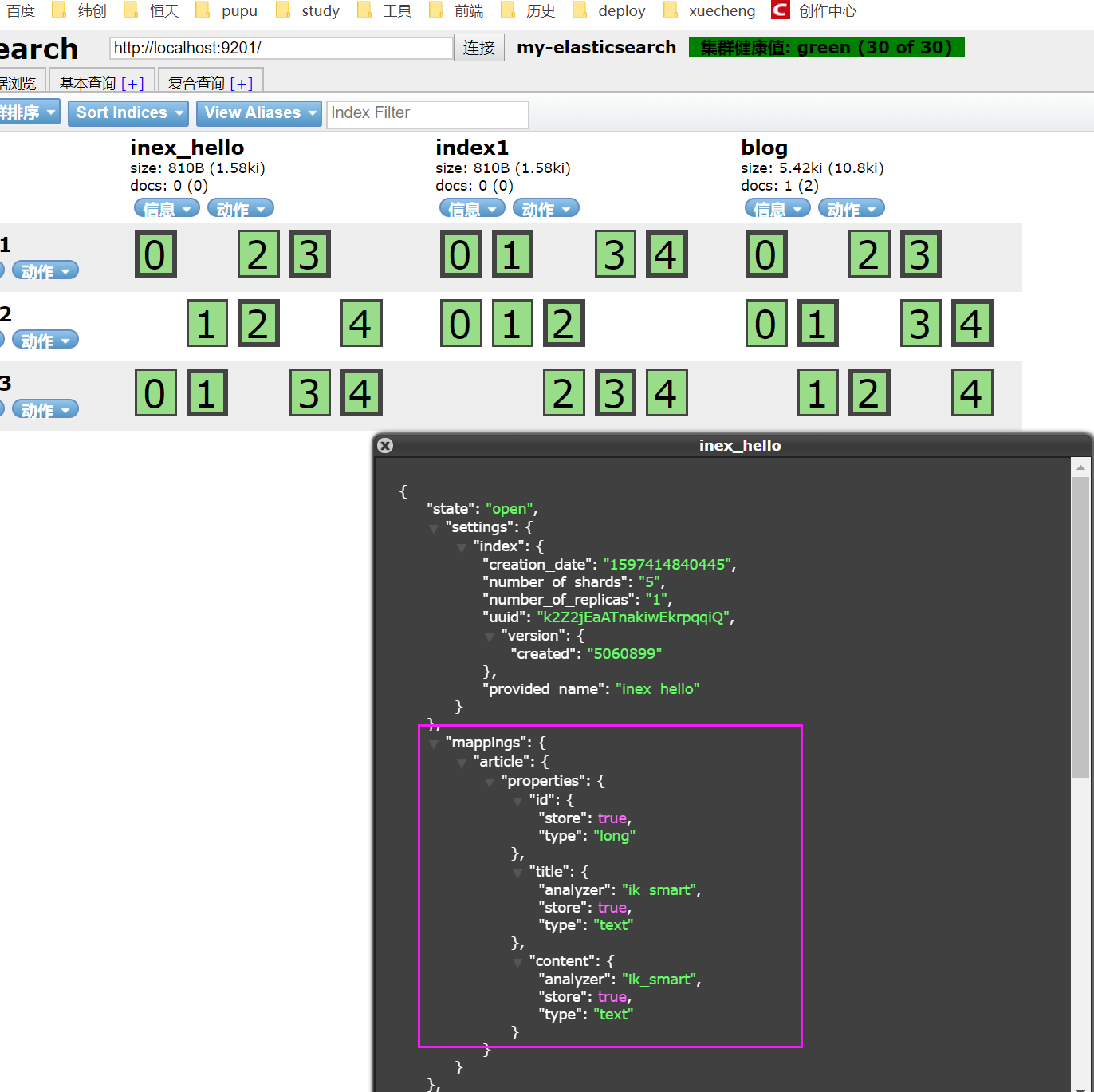
## 1.2 创建索引index

*/\*\*  
 \* 1.2创建索引库  
 \* 1、创建一个Settings对象，相当于是一配置信息。主要配置集群的名称。确定要连接那个集群  
 \* 2、创建一个客户端Client对象  
 \* 3、使用client对象创建一 个索引库  
 \* 4、关闭client对象  
 \*  
 \** ***@author*** *lipu  
 \** ***@since*** *2020/8/14 22:09  
 \*/* @Test  
 public void createIndex() throws Exception {  
 Settings settings = Settings.*builder*()  
 .put("cluster.name", "my-elasticsearch")  
 .build();  
  
 PreBuiltTransportClient client = new PreBuiltTransportClient(settings);  
 *//第一个参数根据ip获取名称;这里是停车票连接对应端口是9300* System.*out*.println(InetAddress.*getByName*("127.0.0.1"));  
 client.addTransportAddress(new InetSocketTransportAddress(InetAddress.*getByName*("127.0.0.1"),9301));  
 client.addTransportAddress(new InetSocketTransportAddress(InetAddress.*getByName*("127.0.0.1"),9302));  
 client.addTransportAddress(new InetSocketTransportAddress(InetAddress.*getByName*("127.0.0.1"),9303));  
  
 *//获取管理员，做索引库的操作，然后指定要创建的索引库的名称,设置后，调用get()执行* client.admin().indices().prepareCreate("inex\_hello").get();  
  
 client.close();  
  
 }  
}



## 1.3 创建mapping

*/\*\*  
 \* 1.3给index添加mappings:  
 \* 1)创建一个Settings对 象  
 \* 2)创建一个Client对象  
 \* 3)创建一个mapping信息，应该是一个json数据，可以是字符串，也可以是XContextBuilder对象  
 \* 4)使用client向es服务器发送mapping信息  
 \* 5)关闭client对象  
 \*  
 \** ***@author*** *lipu  
 \** ***@since*** *2020/8/14 22:26  
 \*/*@Test  
public void setMappings() throws Exception {  
 Settings settings = Settings.*builder*()  
 .put("cluster.name", "my-elasticsearch")  
 .build();  
  
 PreBuiltTransportClient client = new PreBuiltTransportClient(settings);  
 *//第一个参数根据ip获取名称;这里是停车票连接对应端口是9300* System.*out*.println(InetAddress.*getByName*("127.0.0.1"));  
 client.addTransportAddress(new InetSocketTransportAddress(InetAddress.*getByName*("127.0.0.1"),9301));  
 client.addTransportAddress(new InetSocketTransportAddress(InetAddress.*getByName*("127.0.0.1"),9302));  
 client.addTransportAddress(new InetSocketTransportAddress(InetAddress.*getByName*("127.0.0.1"),9303));  
  
 *//* XContentBuilder builder = XContentFactory.*jsonBuilder*()  
 .startObject()  
 .startObject("article")  
 .startObject("properties")  
 .startObject("id")  
 .field("type","long")  
 .field("store",true)  
 .endObject()  
 .startObject("title")  
 .field("type","text")  
 .field("store",true)  
 .field("analyzer","ik\_smart")  
 .endObject()  
 .startObject("content")  
 .field("type","text")  
 .field("store",true)  
 .field("analyzer","ik\_smart")  
 .endObject()  
 .endObject()  
 .endObject()  
 .endObject();  
  
 client.admin().indices().preparePutMapping("inex\_hello").setType("article").setSource(builder).get();  
 client.close();  
}

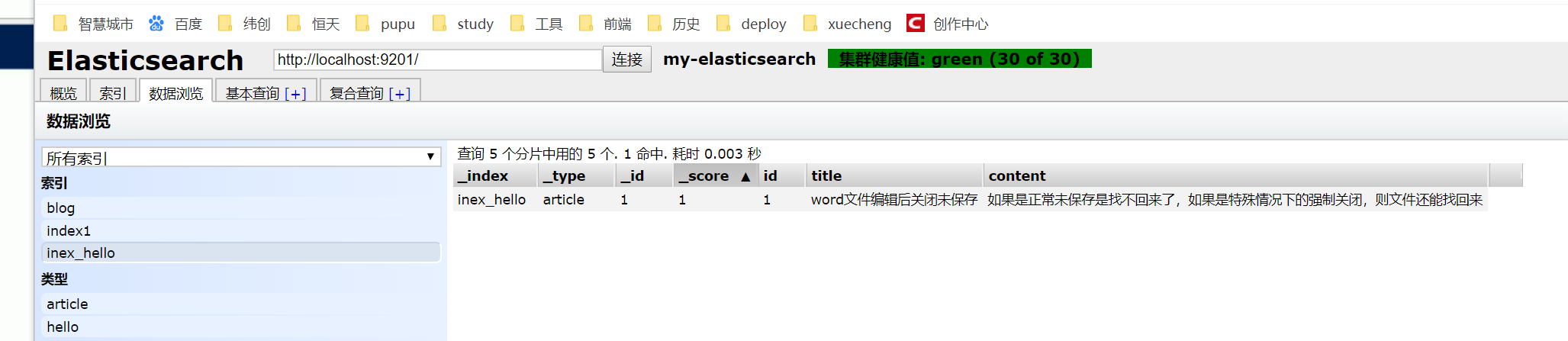


## 1.4 建立文档document

### 1.4.1 XcontextBuilder

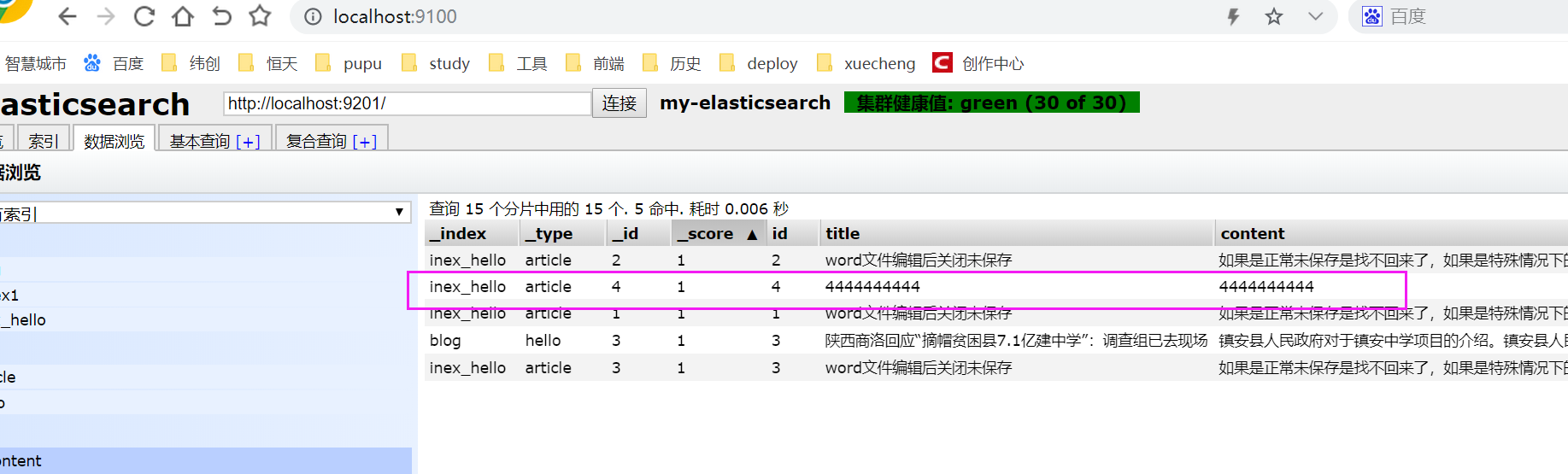
对象太麻烦，就是描述json

@Test  
public void testAddDocument() throws Exception {  
 XContentBuilder builder = XContentFactory.*jsonBuilder*()  
 .startObject()  
 .field("id",1)  
 .field("title","word文件编辑后关闭未保存")  
 .field("content","如果是正常未保存是找不回来了，如果是特殊情况下的强制关闭，则文件还能找回来")  
 .endObject();  
 *//获取管理员，做索引库的操作，然后指定要创建的索引库的名称,设置后，调用get()执行* client.prepareIndex("inex\_hello","article","1").setSource(builder).get();  
 client.close();  
}



### 1.4.2 Jackson

@Test  
public void testAddDocument2() throws Exception {  
 Article article = new Article();  
 article.setId(4L);  
 article.setTitle("4444444444");  
 article.setContent("4444444444");  
  
 ObjectMapper objectMapper = new ObjectMapper();  
 String jsonDocument = objectMapper.writeValueAsString(article);  
 System.*out*.println(jsonDocument);  
  
 *//获取管理员，做索引库的操作，然后指定要创建的索引库的名称,设置后，调用get()执行* client.prepareIndex("inex\_hello","article","4").setSource(jsonDocument, XContentType.*JSON*).get();  
 client.close();  
}



## 1.5 查询文档操作

### 1.5.1 关键词term查询

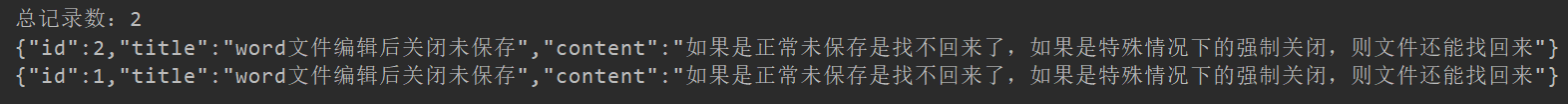
@Test  
 public void testSearchById() throws Exception {  
  
 *//获取管理员，做索引库的操作，然后指定要创建的索引库的名称,设置后，调用get()执行  
// IdsQueryBuilder query = QueryBuilders.idsQuery().addIds("1", "2");  
 //参数1：要搜索的字段field，参数2是要搜索的关键词* TermQueryBuilder builder = QueryBuilders.*termQuery*("title", "wprd");  
 SearchResponse response = client.prepareSearch("inex\_hello").setTypes("article").setQuery(builder).get();  
 SearchHits hits = response.getHits();  
 System.*out*.println("总记录数："+hits.totalHits);  
 Iterator<SearchHit> iterator = hits.iterator();  
 while (iterator.hasNext()){  
 SearchHit searchHit = iterator.next();  
 System.*out*.println(searchHit.getSourceAsString());  
 }  
 client.close();  
 }

### 1.5.2 字符串查询querystring

@Test  
 public void testSearchById() throws Exception {  
  
 *//获取管理员，做索引库的操作，然后指定要创建的索引库的名称,设置后，调用get()执行  
// IdsQueryBuilder builder = QueryBuilders.idsQuery().addIds("1", "2");  
 //参数1：要搜索的字段field，参数2是要搜索的关键词  
// TermQueryBuilder builder = QueryBuilders.termQuery("title", "wprd");* QueryStringQueryBuilder builder = QueryBuilders.*queryStringQuery*("则文件").defaultField("title");  
 SearchResponse response = client.prepareSearch("inex\_hello").setTypes("article").setQuery(builder).get();  
 SearchHits hits = response.getHits();  
 System.*out*.println("总记录数："+hits.totalHits);  
 Iterator<SearchHit> iterator = hits.iterator();  
 while (iterator.hasNext()){  
 SearchHit searchHit = iterator.next();  
 System.*out*.println(searchHit.getSourceAsString());  
 }  
 client.close();  
 }

### 1.5.3 使用文档id查询

@Test  
public void testSearchById() throws Exception {  
  
 *//获取管理员，做索引库的操作，然后指定要创建的索引库的名称,设置后，调用get()执行* IdsQueryBuilder query = QueryBuilders.*idsQuery*().addIds("1", "2");  
 SearchResponse response = client.prepareSearch("inex\_hello").setTypes("article").setQuery(query).get();  
 SearchHits hits = response.getHits();  
 System.*out*.println("总记录数："+hits.totalHits);  
 Iterator<SearchHit> iterator = hits.iterator();  
 while (iterator.hasNext()){  
 SearchHit searchHit = iterator.next();  
 System.*out*.println(searchHit.getSourceAsString());  
 }  
 client.close();  
}



## 1.6 查询文档分页操作

### 1.6.1 批量插入

循环

### 1.6.2 分页查询

默认是10条

@Test  
public void testQueryString() throws Exception {  
  
 QueryStringQueryBuilder builder = QueryBuilders.*queryStringQuery*("索引库名称")  
 .defaultField("title");  
 SearchResponse response = client  
 .prepareSearch("inex\_hello")  
 .setTypes("article")  
 .setQuery(builder)  
 *//从哪里开始分页* .setFrom(0)  
 *//每页显示的行数* .setSize(5)  
 .get();  
 SearchHits hits = response.getHits();  
 System.*out*.println("总记录数："+hits.totalHits);  
 Iterator<SearchHit> iterator = hits.iterator();  
 while (iterator.hasNext()){  
 SearchHit searchHit = iterator.next();  
 System.*out*.println(searchHit.getSourceAsString());  
 }  
 client.close();  
}

但是没有排序

## 1.7 查询结果高亮操作

@Test  
public void testSearchhighLight() throws Exception {  
  
 HighlightBuilder highlightBuilder = new HighlightBuilder();  
 *//高亮显示的字段* highlightBuilder.field("title");  
 highlightBuilder.preTags("<em>");  
 highlightBuilder.postTags("</em>");  
  
 QueryStringQueryBuilder builder = QueryBuilders.*queryStringQuery*("索引库名称")  
 .defaultField("title");  
 SearchResponse response = client  
 .prepareSearch("inex\_hello")  
 .setTypes("article")  
 .setQuery(builder)  
 *//从哪里开始分页* .setFrom(0)  
 *//每页显示的行数* .highlighter(highlightBuilder)  
 .setSize(5)  
 .get();  
 SearchHits hits = response.getHits();  
 System.*out*.println("总记录数："+hits.totalHits);  
 Iterator<SearchHit> iterator = hits.iterator();  
 while (iterator.hasNext()){  
 SearchHit searchHit = iterator.next();  
 System.*out*.println(searchHit.getSourceAsString());  
 System.*out*.println("\*\*\*\*\*\*\*\*\*\*\*高亮结果\*\*\*\*\*\*\*\*\*\*");  
 System.*out*.println(searchHit.getHighlightFields());  
 }  
 client.close();  
}

# 2 Spring Data ElasticSearch

## 2.1 概念

### 2.1.1 什么是pring 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IPA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: <https://spring.io/projects/spring-data>

Spring Data常用的功能模块如下:

### 2.1.2 什么是pring 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: <https://spring.io/projects/spring-data-elasticsearch>

## 2.2 入门

1）导入坐标

<?xml version="1.0" encoding="UTF-8"?>  
<project xmlns="http://maven.apache.org/POM/4.0.0"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">  
 <modelVersion>4.0.0</modelVersion>  
  
 <groupId>com.pupu</groupId>  
 <artifactId>springdata-elasticsearch</artifactId>  
 <version>1.0-SNAPSHOT</version>  
  
 <properties>  
 <maven.compiler.source>1.8</maven.compiler.source>  
 <maven.compiler.target>1.8</maven.compiler.target>  
 </properties>  
  
  
 <dependencies>  
  
 <!--elasticsearch-->  
 <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.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>  
  
 <!--jackson-->  
 <dependency>  
 <groupId>com.fasterxml.jackson.core</groupId>  
 <artifactId>jackson-databind</artifactId>  
 <version>2.9.6</version>  
 </dependency>  
 <dependency>  
 <groupId>com.fasterxml.jackson.core</groupId>  
 <artifactId>jackson-annotations</artifactId>  
 <version>2.9.0</version>  
 </dependency>  
  
  
 <!--log-->  
 <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>org.springframework</groupId>  
 <artifactId>spring-test</artifactId>  
 <version>5.0.4.RELEASE</version>  
 </dependency>  
  
 <!--junit-->  
 <dependency>  
 <groupId>junit</groupId>  
 <artifactId>junit</artifactId>  
 <version>4.12</version>  
 </dependency>  
 </dependencies>  
  
</project>

2）配置文件：

<?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">  
  
  
 <!--创建一个elasticsearch client 指定集群名称，节点信息-->  
 <elasticsearch:transport-client id="esClient" cluster-name="my-elasticsearch" cluster-nodes="127.0.0.1:9301,127.0.0.1:9302,127.0.0.1:9303"/>  
  
 <!-- 配置dao包扫描器,用于访问es -->  
 <elasticsearch:repositories base-package="com.pupu.es.repositories"/>  
  
 <!-- 简化es的增删改查等操作 -->  
 <bean id="elasticsearchTemplate" class="org.springframework.data.elasticsearch.core.ElasticsearchTemplate">  
 <!--需要配置es数据源 -->  
 <constructor-arg name="client" ref="esClient"/>  
 </bean>  
  
</beans>

## 2.3 概念

### 2.3.1 增删改查方法测试

### 2.3.2 常用查询命名规则

|  |  |  |  |
| --- | --- | --- | --- |
| **关键字** | **命名规则** | **解释** | **示例** |
| and | findByField1AndField2 |  |  |
| or | findByField1OrField2 |  |  |
| is | findByField |  | findByTirle |
| not | findByFieldNot |  |  |
| between | findByFieldBetween |  |  |
| lessThanEqual | findByFieldLessThan |  | findByPriceLessThan |

### 2.3.3 查询方法测试

分页默认从0页开始；

### 2.3.4 使用ElasticSearch的原则

### 2.3.5 高亮显示查询