# ElasticSearch

**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;

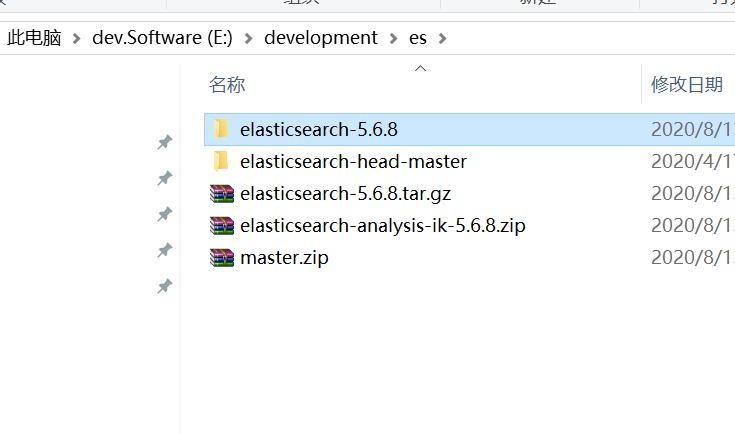
●Solr 支持更多格式的数据,而Elasticsearch仅支持json文件格式;

●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？

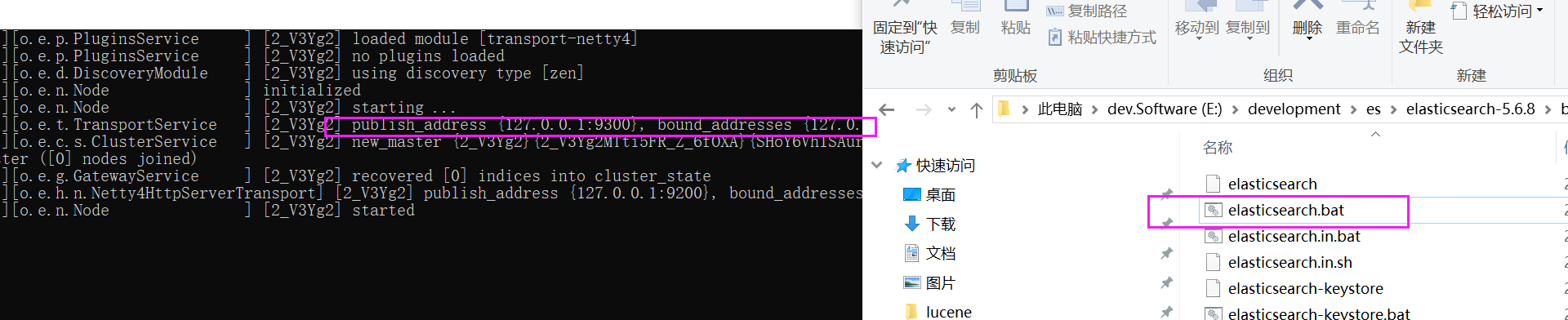


## 2.2安装

解压后就是安装了

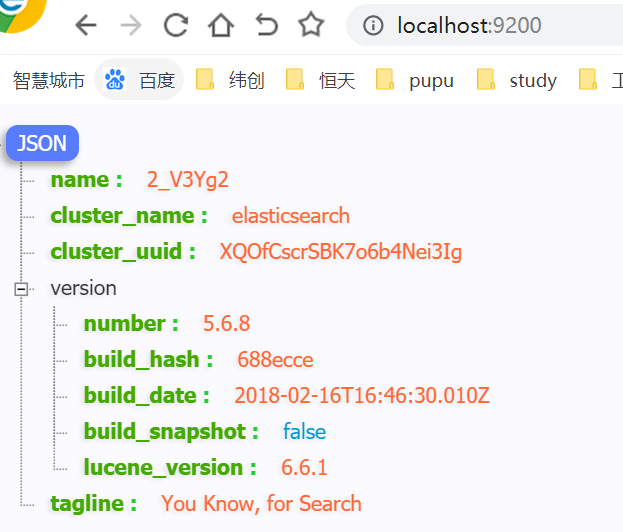
## 2.3 启动

需要jdk1.8以上



9300：Tcp

9200:Restful



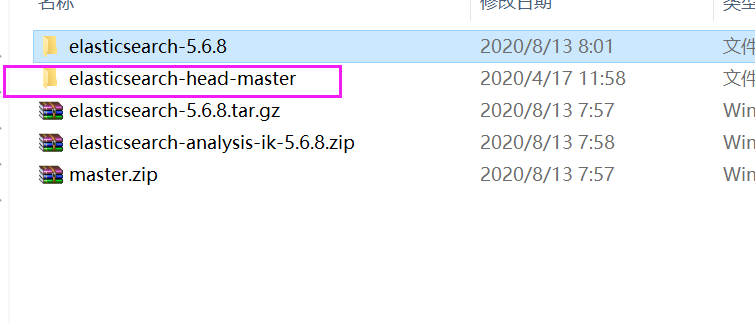
## 2.4 安装ES的图形化界面插件

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

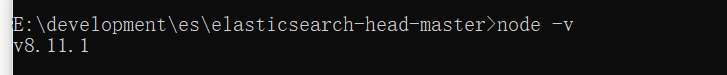


2 )将elasticsearch-head-master压缩包解压到任意目录,但是要和elasticsearch的安装目录区别开

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

在资料中已经提供了nodejs安装程序:

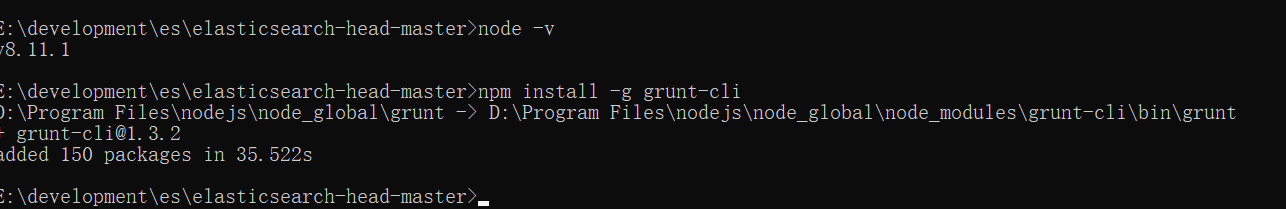
安装完毕,可以通过cmd控制台输入: node -V查看版本号



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:

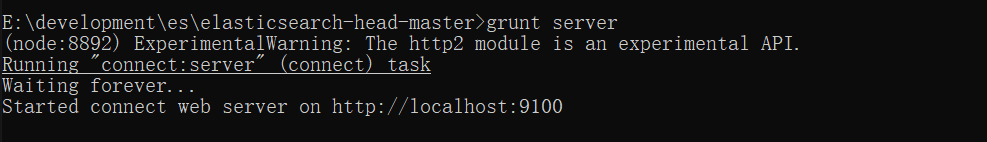
npm install -g grunt-cli



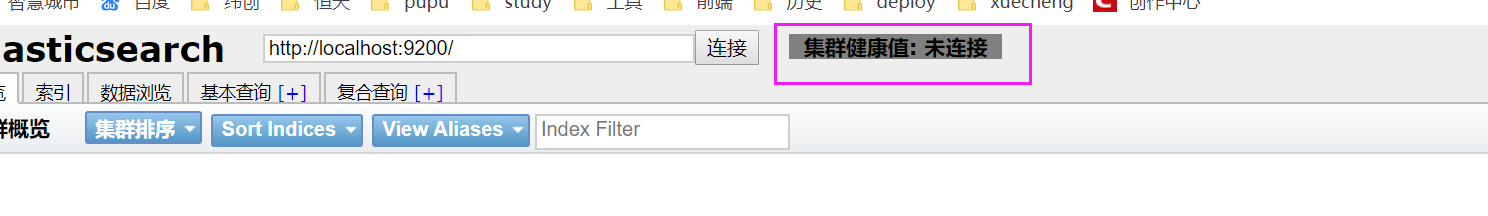
6 )进入elasticsearch-head-master目录启动head ,在命令提示符下输入命令:

>npm install

>grunt server



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

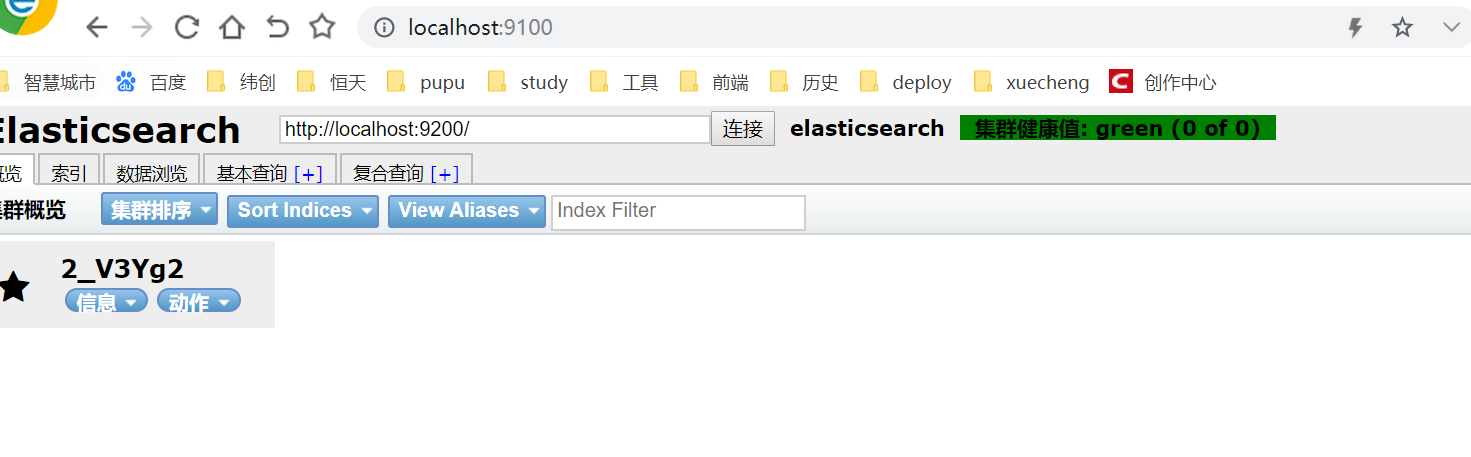


如果不能成功连接到es服务,需要修改ElasticSearch的config目录下的配置文件: config/elasticsearch.yml ,增加以下两句命令:

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。

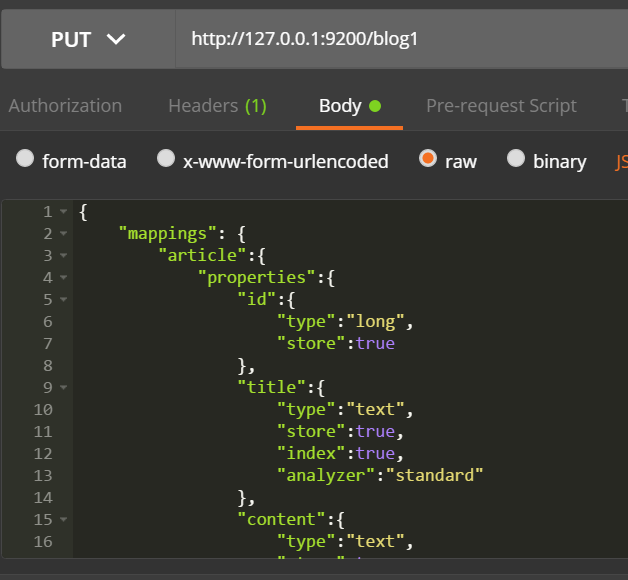
本案例中的原始内容就是磁盘上的文件，如下图：

## 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"**

**}**

**}**

**}**

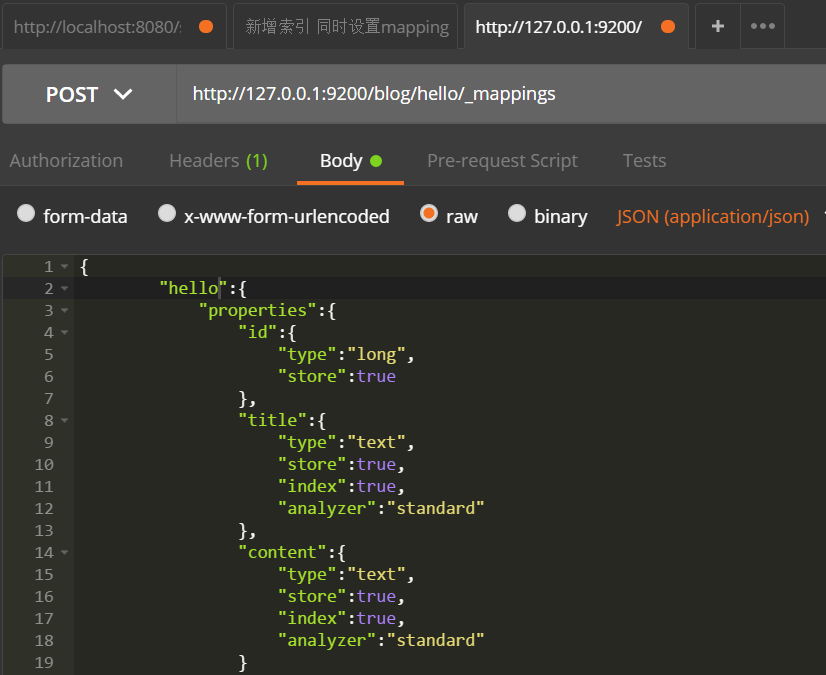
**}**

**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"**

**}**

**}**

**}**

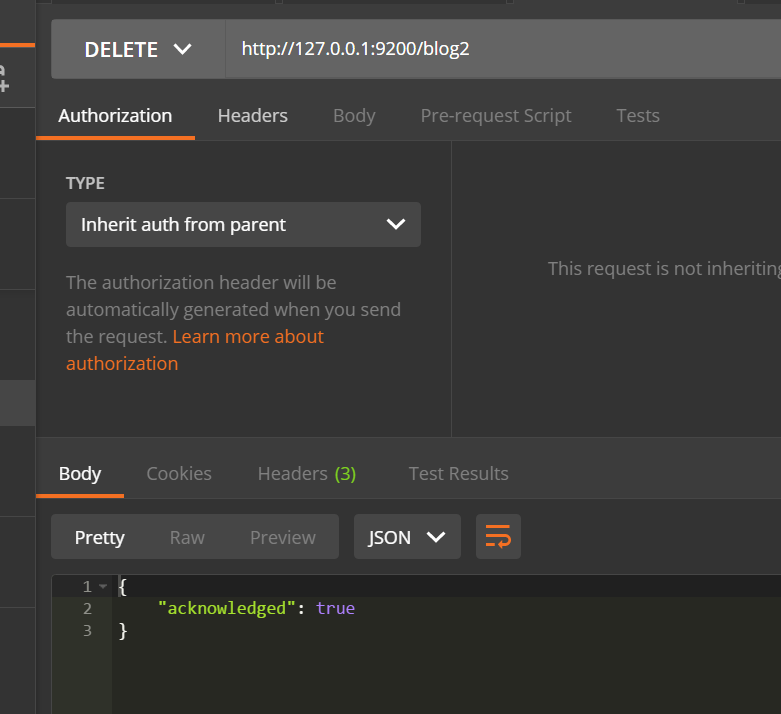
**}**

**article:**表示的是有多少个type

**properties:**表示的是有多少个document

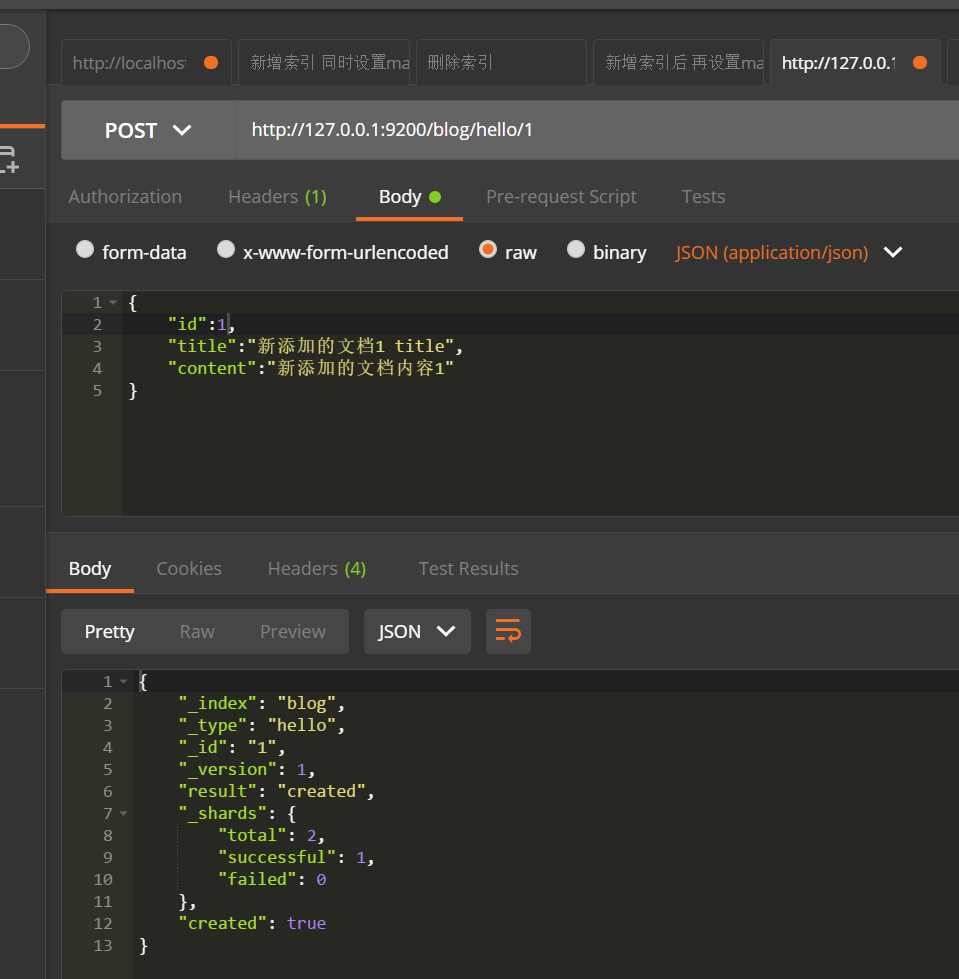
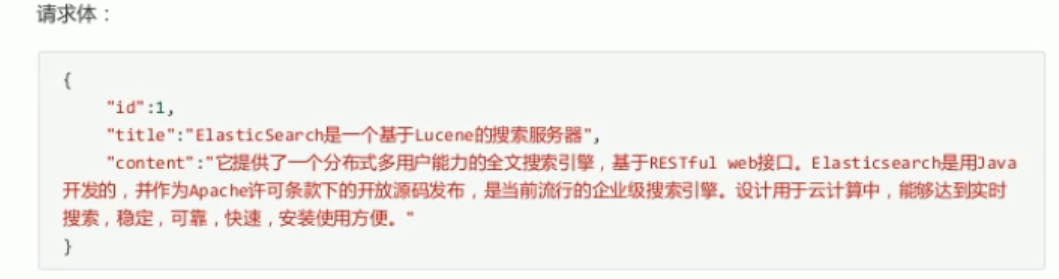
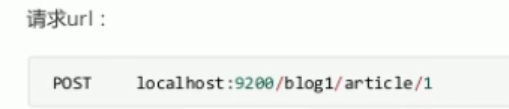
再往里就是Field

### 4.2.4 删除index



### 4.2.5 创建文档document到index

1是文档的id

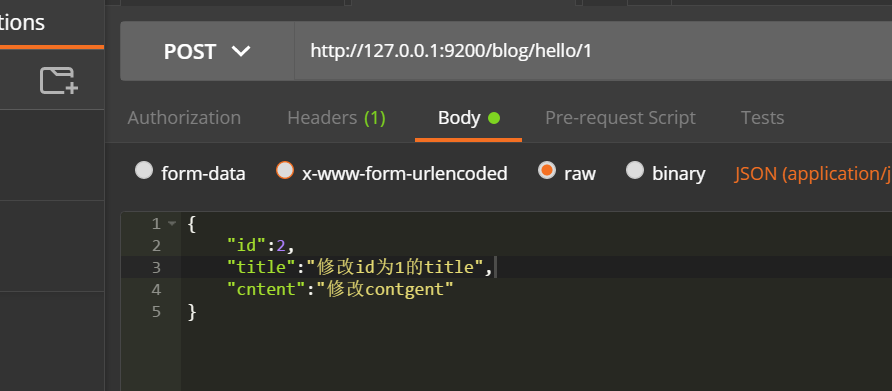


如果你不指定id，会自动生成随机的id

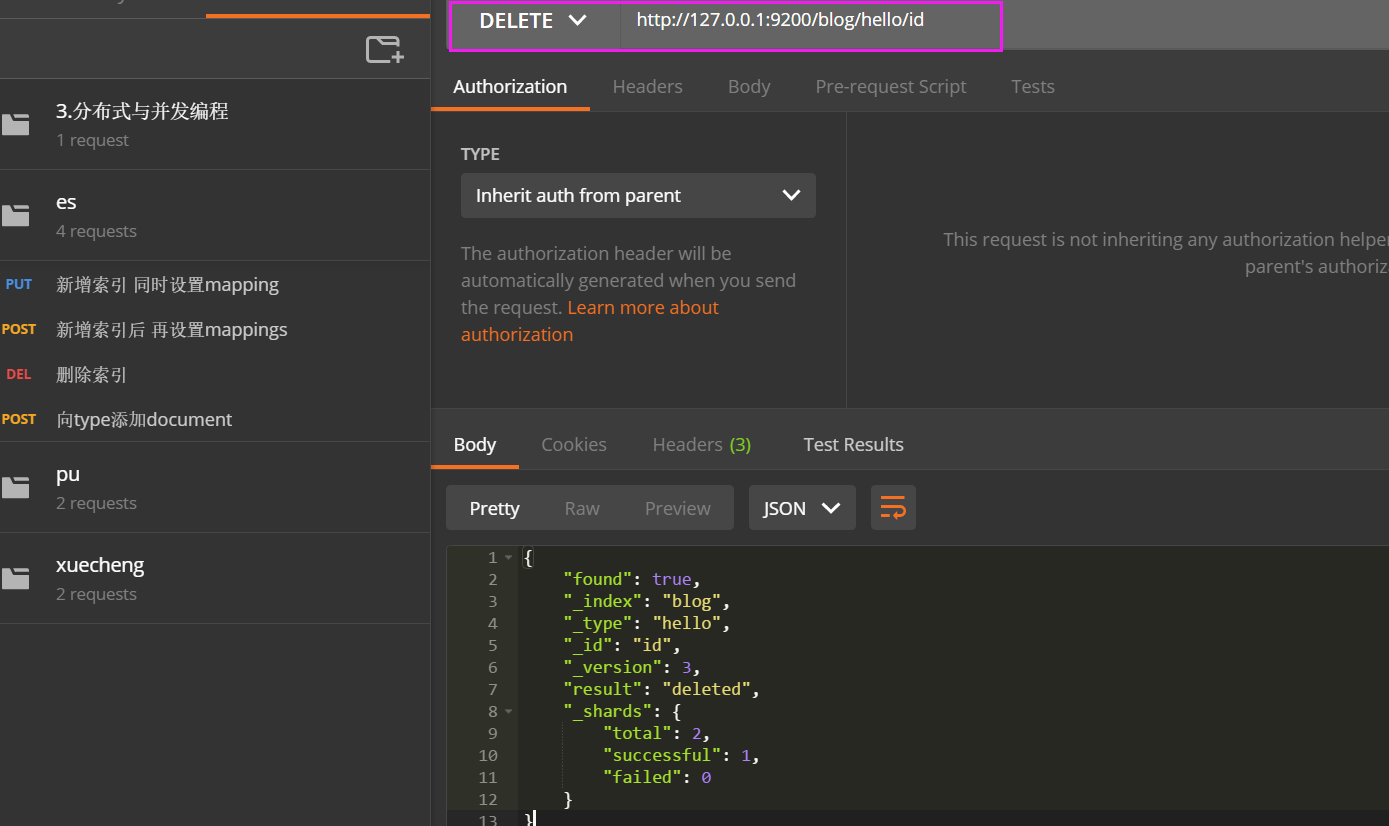
### 4.2.6 修改document

先删除后添加

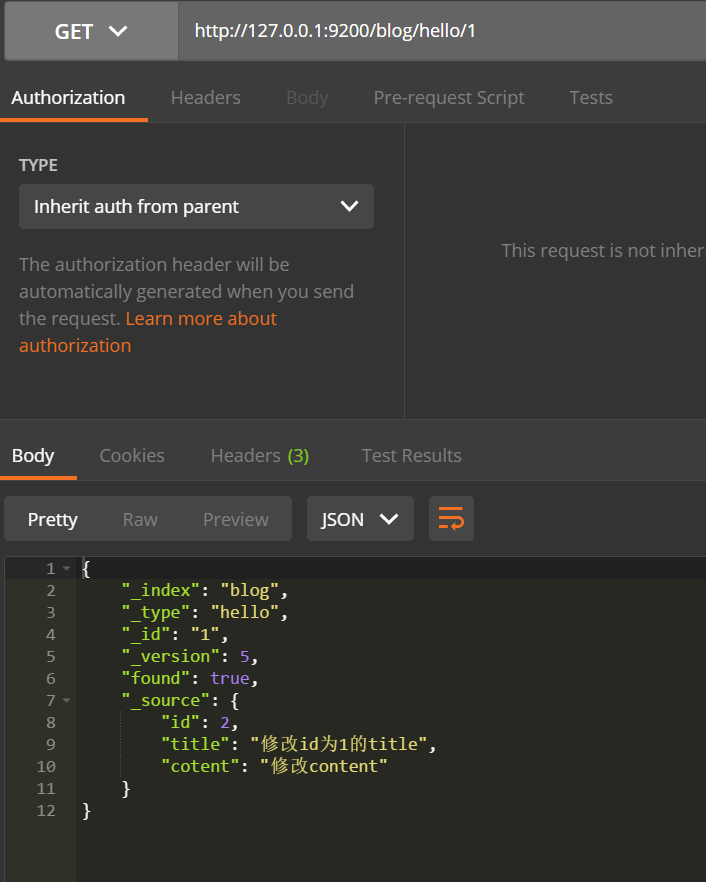
和添加是同一个操作



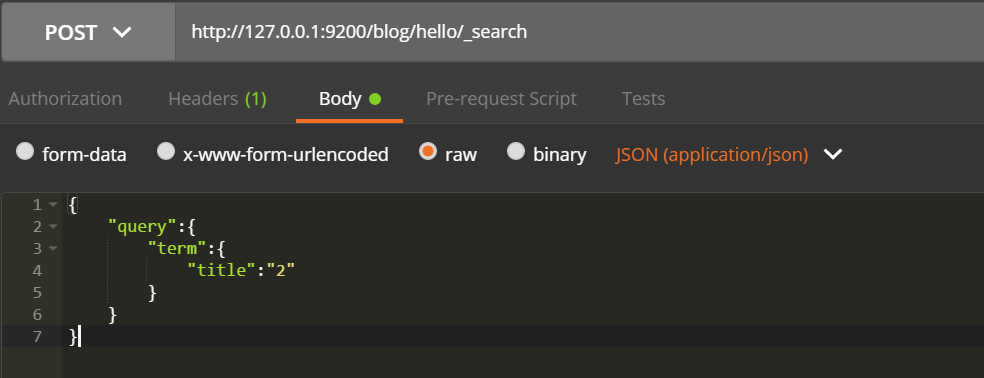
### 4.2.7 删除document



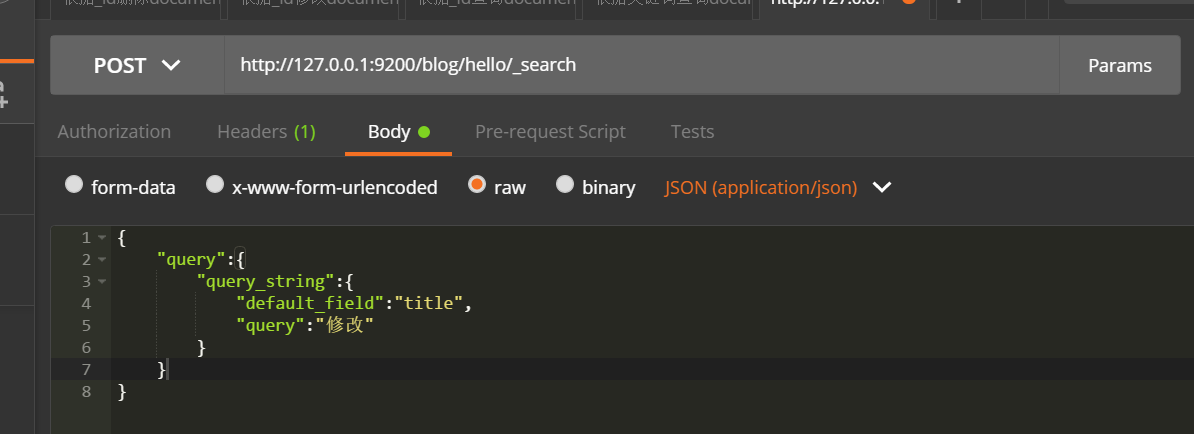
### 4.2.8 查询文档--根据id查询



### 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

### 5.3.2 测试

## 5.4 修改索引映射mapping

### 5.4.1 重建索引

### 5.4.2 再次测试querstring

### 5.4.3 再次测试term

# 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

### 6.2.2修改每台服务器的配置

### 6.2.3启动每个节点服务器

### 6.2.4集群测试

## 6.3 索引库的修改

修改的原理：先删除再添加

@Test  
public void updateDocument() throws Exception{  
 *//创建一个Indexwriter对象，需要使用IKAnalyzer作为分析器* Directory open = FSDirectory.*open*(new File("H:\\download\\baiduDiskDownload\\黑马JavaEE 57期\\00 讲义+笔记+资料\\流行框架\\61.会员版(2.0)-就业课(2.0)-Lucene\\index").toPath());  
 IndexWriter indexWriter = new IndexWriter(open, new IndexWriterConfig(new IKAnalyzer()));  
  
 *//创建一个Document 对象* Document d = new Document();  
 *//向document对象中添加域* d.add(new TextField("name","新更新的文件", Field.Store.*YES*));  
 d.add(new TextField("content","新更新的文件内容", Field.Store.*YES*));  
 d.add(new TextField("path","c://temp/helo", Field.Store.*YES*));  
  
 indexWriter.updateDocument(new Term("name","spring"),d);  
 *//关闭索引库* indexWriter.close();  
}

# 7 Lucene 索引库查询

## 7.1 TermQuery

参照6.。。

## 7.2RangeQuery数值范围查询

public class SearchIndex {  
  
 private IndexReader indexReader;  
 private IndexSearcher indexSearcher;  
  
 @Before  
 public void init() throws Exception{  
 Directory open = FSDirectory.*open*(new File("H:\\download\\baiduDiskDownload\\黑马JavaEE 57期\\00 讲义+笔记+资料\\流行框架\\61.会员版(2.0)-就业课(2.0)-Lucene\\index").toPath());  
 *//2、创建一一个IndexReader对象* indexReader = DirectoryReader.*open*(open);  
 indexSearcher = new IndexSearcher(indexReader);  
 }  
  
  
 @Test  
 public void testRangeQuery() throws Exception{  
 Query query = LongPoint.*newRangeQuery*("size", 0L, 100L);  
 this.printResult(query);  
 }  
  
  
  
 private void printResult(Query query)throws Exception{  
 TopDocs topDocs = indexSearcher.search(query, 10);  
 System.*out*.println("总记录数哦："+topDocs.totalHits);  
 ScoreDoc[] scoreDocs = topDocs.scoreDocs;  
 for (ScoreDoc scoreDoc : scoreDocs) {  
 *//取文档id* int docId = scoreDoc.doc;  
 *//根据文档id获取文档* Document document = indexSearcher.doc(docId);  
 System.*out*.println(document.get("name"));  
 System.*out*.println(document.get("path"));  
 System.*out*.println(document.get("size"));  
*// System.out.println(document.get("content"));* System.*out*.println("-----------------------寂寞的分隔说---------------------");  
 }  
 indexReader.close();  
 }  
}

## 7.3 使用queryparse查询

可以对要查询的内容先分词。然后基于分词的结果进行查询。

添加jar包

lucene-queryparser-7.4.0.jar

private void printResult(Query query)throws Exception{  
 TopDocs topDocs = indexSearcher.search(query, 10);  
 System.*out*.println("总记录数哦："+topDocs.totalHits);  
 ScoreDoc[] scoreDocs = topDocs.scoreDocs;  
 for (ScoreDoc scoreDoc : scoreDocs) {  
 *//取文档id* int docId = scoreDoc.doc;  
 *//根据文档id获取文档* Document document = indexSearcher.doc(docId);  
 System.*out*.println(document.get("name"));  
 System.*out*.println(document.get("path"));  
 System.*out*.println(document.get("size"));  
*// System.out.println(document.get("content"));* System.*out*.println("-----------------------寂寞的分隔说---------------------");  
 }  
 indexReader.close();  
 }  
  
  
 @Test  
 public void testQuerParser() throws Exception{  
 *//1)创建一个QuerParser对象，参数1：默认搜索域，参数2：分析器对象* QueryParser queryParser = new QueryParser("name",new IKAnalyzer());  
 Query query = queryParser.parse("lucene是一个Java开发的全文检索工具包\n");  
 printResult(query);  
 }